NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Electrical and Electronic Engineering

PATTERN RECOGNITION USING NEURAL NETWORKS

Graduation Project
EE – 400

Student: Suleiman Al-derhalli (20010988)

Supervisor: Assoc. Prof. Dr. Adnan Khashman

Nicosia - 2005

ACKNOWLEDGMENTS

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

I would like to express my deep gratitude to my Supervisor Assoc.Prof. Dr. ADNAN KHASHMAN, for his great advices. And also, for his many constructive inputs and invaluable help.

I also wish to thank Mr. ALPER AKANSER, for his valuable and illuminated advices, and his full support and encouragement during the semester.

And finally, I would like to thank my family especially my father OTHMAN ALDERHALLI, for giving me the chance to complete my academic study and support me during the preparation of this project.

ABSTRACT

"Artificial neural networks" is a frequently used expression in everyday language, particularly among computer scientists. When you ask some people who are working in this area what they are really doing, you sometimes get the short answer: "We are simulating the brain".

In this project, I intend to demonstrate a general idea about character recognition and go in details toward Arabic character recognition. A back-propagation neural network with one hidden layer was used to create an adaptive Arabic character recognition system. The system was trained and evaluated with input vectors, as well as other different forms of noisy input vectors.

The objective of this project is to demonstrate a framework for giving good recognition accuracy to Arabic letter inputs. *Good recognition accuracy means that the system will* scale well for inputs, classify efficiently, and have the potential to be robust in the presence of noisy data input.

TABLE OF CONTENTS

INTRODUCTION

The human brain is organized as a huge network of numerous very simple computational units, called *neurons*. During the past half century, the study of *artificial neural networks*, modeled after those "natural prototypes" has become more and more popular. Particularly *after* the advent of VLSI technology, as computers became more powerful and running simulations of large artificial networks therefore became easier and faster, the field of *connectionism* (which is another name for the area of artificial neural networks) began to gain importance. The past decade has been the probably most significant one in the development of connectionist models.

In this project, I want to talk about this development in general and proceeds all the way to discuss in details one of its applications in a particular case. I chose this topic because of mainly two reasons. First of all, it is at the heart of present-day research, it represents a relatively young and progressive field, which makes it extremely interesting. Second, the connectionist approach is directly applicable to problems of the real world, a very impressive property for such a recent development in computer science.

Artificial neural networks might make it possible to solve interesting problems that could not be solved without them (at least not with an acceptable overhead). This refers to problems that are extremely hard to solve in the usual way with conventional computer programs, but very easy for the human brain, for instance tasks related to pattern or speech recognition, where in this project I intend to discuss a very related field under pattern recognition, where the recognition of Arabic characters using neural network implemented by the aid of Matlab is going to be our main topic underlying the main concept of pattern recognition. Furthermore, artificial neural networks allow the development of a totally new kind of algorithms, algorithms with a highly parallel structure.

A back-propagation neural network with one hidden layer was used to create an adaptive Arabic character recognition system. The system was trained and evaluated with input vectors, as well as other different forms of noisy input vectors. Experiments tested (1) the effect on recognition accuracy without noise, and (2) the effect on recognition accuracy with a Gaussian distributed noise added to the input vector. Results showed reduced accuracy in recognizing characters when training without noise and it illuminated the fact that a better performance and more accuracy of the system is achieved by training with noise, since noise is an unavoidable problem in the real world!!.

I start the actual project with a description of theoretical foundations of neural network in the first chapter, including a discussion of *training* an artificial neural network and an explanation of the basic concepts of the biological equivalent. Besides, I try to give a fair summary of the historical issue of the field.

Chapter two describes the formal introduction of the pattern recognition. Where it mainly concerns about character recognition in general and gives details about Arabic characters in particular. It also points to some desirable patterns, by presenting a bunch of interesting practical problems of pattern recognition.

Chapter three, discusses how Arabic character recognition can be done with a supervised learning backpropagation neural network. It determines the architecture of the neural network, and also examines the mechanism of training the neural network with noise free Arabic characters and enhances its performance by training it again with noisy inputs, and it ends up by showing the general performance for both types of training.

The aims of the work presented within this project are:

- To understand and to have a good insight of Neural Network in general.
- To investigate Pattern Recognition.
- To introduce the reader with some types of patterns in general and Arabic alphabet in particular.
- To Simulate a working Neural Network that is able to recognize Arabic alphabet.
- To be familiar with the Neural Network toolbox in MATLAB.

CHAPTER ONE

NEURAL NETWORKS

1.1 Overview

This chapter present Neural Networks in general, and tends to help the reader to understand what artificial Neural Networks are, how they work, including a discussion of training an artificial neural network and an explanation of the basic concepts of the biological equivalent in general.

1.2 What is a Neural Network?

Neural Networks have a large appeal to many researchers due to their great closeness to the structure of the brain, a characteristic not shared by more traditional system.

In an analogy to the brain, an entity made up of interconnected neurons, neural networks are made up of interconnected processing elements called units. Which respond in parallel to a set of input signals given to each. The unit is the equivalent of its brain counterpart, the neurons.

There are many definitions for a Neural Network:

1-*Neural network*: is a massively parallel-distributed processor that has a natural propensity for experiential knowledge and making it available for use.

2-*Neural network*: is machine that is designed to model the way in which the brain performs a particular task or function of interest, the network is usually implemented using electronic components or simulated in software on digital computers.

3-*Neural networks*: are also referred to in the literature as neurocomputers, connectionist network, parallel-distributed processors, etc.

4-*Neural networks*: are different paradigms for computing:

A- Von Neumann machines are based on the processing/memory abstraction of human information processing.

B- Neural networks are based on the parallel architecture of animal brains.

5-*Neural networks*: are forms of multiprocessor computer system, with

A- Simple processing elements.

B- A high degree of interconnection.

C- Simple scalar messages.

D- Adaptive interaction between elements.

6-*Neural network*: a mathematical model composed of a large number of processing elements organization in to layers.

1.3 Why are Artificial Neural Networks worth studying?

1. They are extremely powerful computational devices (Turing equivalent, universal computers).
2. Massive parallelism makes them very efficient.
3. They can learn and generalize from training data – so there is no need for enormous feats of programming.
4. They are particularly fault tolerant – this is equivalent to the "graceful degradation" found in biological systems.

5. They are very noise tolerant – so they can cope with situations where normal symbolic systems would have difficulty.
6. In principle, they can do anything a symbolic/logic/rule based system can do, and more. (Though, in practice, getting them to do it can be rather difficult.)

1.4 Benefits of Neural Networks

The use of neural networks offers the following useful properties and capabilities:

1-Nonlinearity: a neuron is basically a nonlinear device. Consequently, a neural network, made up of an interconnection of neurons is itself nonlinear.

2-Input – output mapping: popular paradigm of learning called supervised learning involves the modification of the synaptic weights of a neural network by applying a set of labeled training samples or task examples.

3-Adaptively: neural networks have a built-in capability to adapt their synaptic weights to changes in the surrounding environment.

4-Evidential response: in the context of pattern classification, a neural network can be designed to provide information may be used to reject ambiguous patterns, should they arise, and thereby improve the classification performance of the network.

5-Contextual information: knowledge is represented by the very structure and activation state of a neural network.

1.5 What are Artificial Neural Networks used for?

As with the field of AI in general, there are two basic goals for neural network research:

Brain modeling: The scientific goal of building models of how real brains work. This can potentially help us understand the nature of human intelligence, formulate better teaching strategies, or better remedial actions for brain damaged patients.

Artificial System Building: The engineering goal of building efficient systems for real world applications. This may make machines more powerful, relieve humans of tedious tasks, and may even improve upon human performance. These should not be thought of as competing goals. We often use exactly the same networks and techniques for both. Frequently progress is made when the two approaches are allowed to feed into each other. There are fundamental differences though, e.g. the need for biological plausibility in brain modeling, and the need for computational efficiency in artificial system building.

1.6 Brains versus Computers: Some numbers

There are approximately 10 billion neurons in the human cortex, compared with 10 of thousands of processors in the most powerful parallel computers. Each biological neuron is connected to several thousands of other neurons, similar to the connectivity in powerful parallel computers [1].
Lack of processing units can be compensated by speed. The typical operating speeds of biological neurons is measured in milliseconds (10-3 s), while a silicon chip can operate in nanoseconds (10-9 s).

The human brain (figure 1.1) is extremely energy efficient, using approximately 10-16 joules per operation per second, whereas the best computers today use around 10-6 joules per operation per second. Brains have been evolving for tens of millions of years; computers have been evolving for tens of decades.

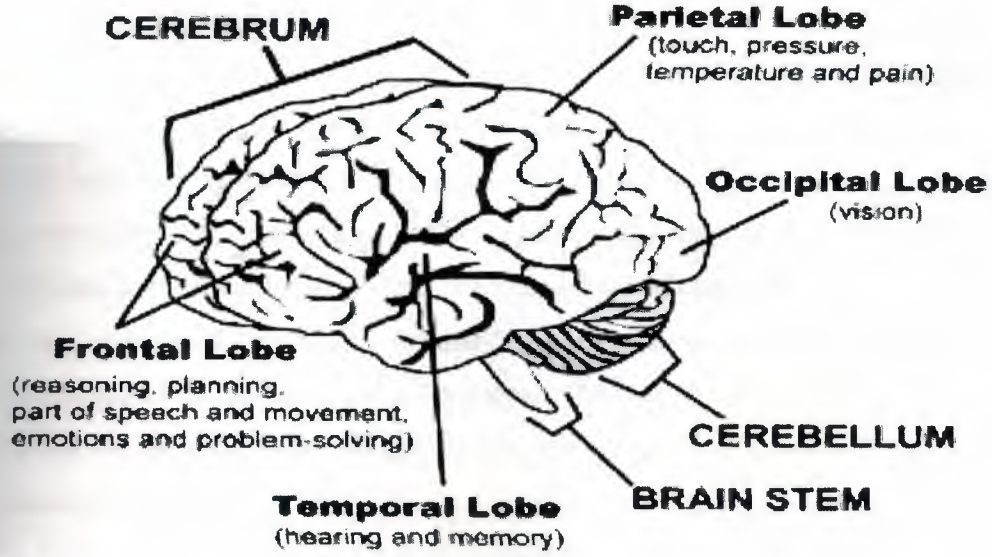


Figure 1.1 Layout of a Biological Neural Network [1]

1.7 The Biological Neuron

The brain is a collection of about 10 billion interconnected neurons. Each neuron is a cell (figure 1.2) that uses biochemical reactions to receive process, and transmit information.

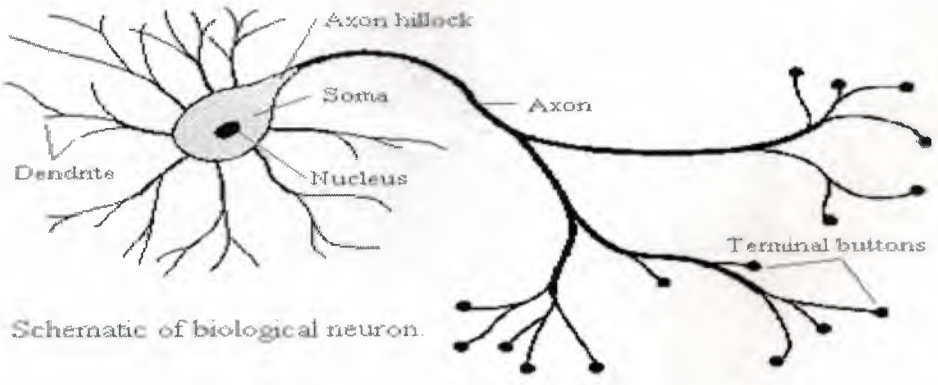


Figure 1.2 Schematic of biological neuron [2]

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.

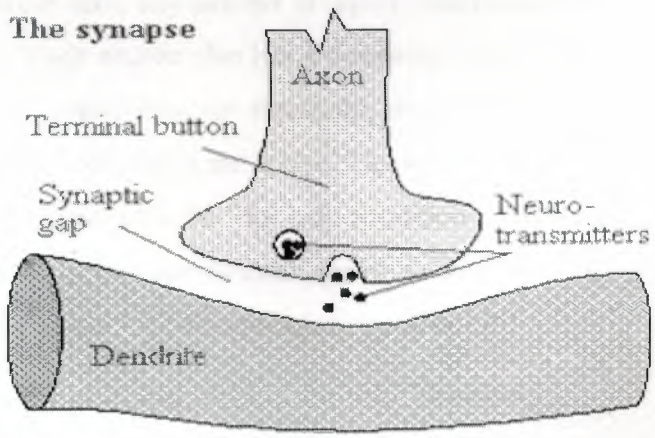
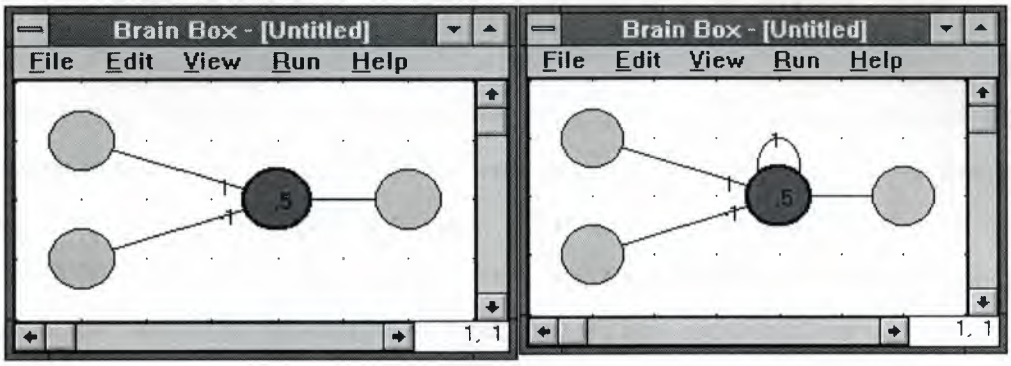


Figure 1.3 The synapse [2]

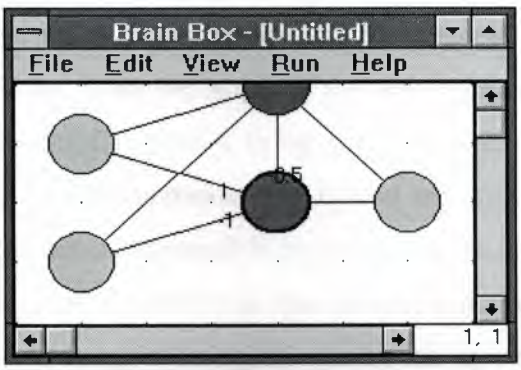
Each terminal button is connected to other neurons across a small gap called a synapse (figure 1.3). The physical and neurochemical characteristics of each synapse determine the strength and polarity of the new input signal. This is where the brain is the most flexible, and the most vulnerable. Changing 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 (acetylcholinesterase) 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 [2].

1.8 A Model of a Neuron

As complicated as the biological neuron is, it may be simulated by a very simple model (figure 1.4(a)). The inputs each have a weight that they contribute to the neuron, if the input is active. The neuron can have any number of inputs; neurons in the brain can have as many as a thousand inputs. Each neuron also has a threshold value. If the sum of all the weights of all active inputs is greater than the threshold, then the neuron is active. For example, consider the case where both inputs are active. The sum of the input's weights is 0. Since 0 is smaller than 0.5, the neuron is off. The only condition which would activate this neuron is if the top input were active and bottom one were inactive. This single neuron and its input weighting performs the logical expression *A and not B*.



(a) (b)



(c)

Figure 1.4 A model of the neuron

There is a variation on this model that sets the threshold to 0 on all neurons, and adds an extra input that is always active. The extra input is weighted to account for the missing threshold (figure 1.4(b)). The two models are mathematically identical. The advantage of the second version is that it simplifies the math involved in automatic learning and implementation, since there is only one type of variable to keep track of. Both of these simple models accurately simulate the most important aspects of the biological neuron, though they do leave out some features such as temporal summation. A more complicated model could easily account for these, but for most requirements the simple models suffice.

The previous examples showed that *A and not B* was solvable with a single neuron. This is a fairly obscure logical construct, and it leads to the question of what else a single neuron is capable of. The easiest way to find out is to play with a neural network computer program. A programming language such as BrainBox program [3] is a Windows application that allows one to watch and modify neural networks as they execute. It doesn't take long to find that of the 16 two-input logical functions, 14 of them can be constructed with a *single* neuron (XOR and XNOR both require two neurons). Since neurons are functionally complete, this means that in addition to logic, they are also capable of storing and retrieving data from `memory'. A neural network can store data in two formats. Permanent data (long term memory) may be designed into the weightings of each neuron. An example of this is the self-teaching network. Temporary data (short term memory) can be actively circulated in a loop, until it is needed again (figure 1.4(c)). In this example, activating the top input briefly will activate the neuron. Since the output of the neuron feeds back to itself, there is a self-sustaining loop that keeps the neuron firing even when the top input is no longer active. Activating the lower input suppresses the looped input, and the node stops firing. The stored binary bit is continuously accessible by looking at the output. This configuration is called a latch. While it works perfectly in this model, a biological neuron would not behave quite this way. After firing, a biological neuron has to rest for a thousandth of a second before it can fire again. Thus one would have to link several neurons together in a duty-cycle chain to achieve the same result.

1.9 Medium Independence

The neuron models are medium independent. They may be built out of organic materials, electrical components, lenses and mirrors, hydraulics or dominoes. The electronic implementation of a neuron (figure 1.5) is based on an operational amplifier (op-amp comparator).

These versatile components can determine which of two voltages is larger. They are accurate to about 0.001 volts. The three potentiometers (variable resistors) are used to set the neural weightings in this implementation. If one is set midway, then the associated

input would have no effect on the system because the voltage would be evenly applied between the two op-amp inputs, resulting in a weighting of 0.

Set the potentiometers towards the top, and the op-amp is positively biased. Since op-amps have a natural threshold of 0 volts difference, an extra bias input is required. Any number of additional inputs and potentiometers may be inserted. It is ironic that the lowly operational amplifier that was developed for the long-obsolete analog computers, is making a come-back as dedicated computer systems start to use neural networks instead of (or in addition to) conventional microprocessors.

To create a large neural network, one could either construct thousands of op-amp circuits in parallel, or one could merely simulate them using a program executing on a conventional serial processor. From a theoretical stance, the solutions are equivalent since a neuron's medium does not affect its operation. By simulating the neural behavior, one has created a virtual machine that is functionally identical to a machine that would have been prohibitively complex and expensive to build. A computer's flexibility makes the creation of one hundred neurons as easy as the creation of one neuron. The drawback is that the simulated machine is slower by many orders of magnitude than a *real* neural network since the simulation is being done in a serial manner by the CPU.

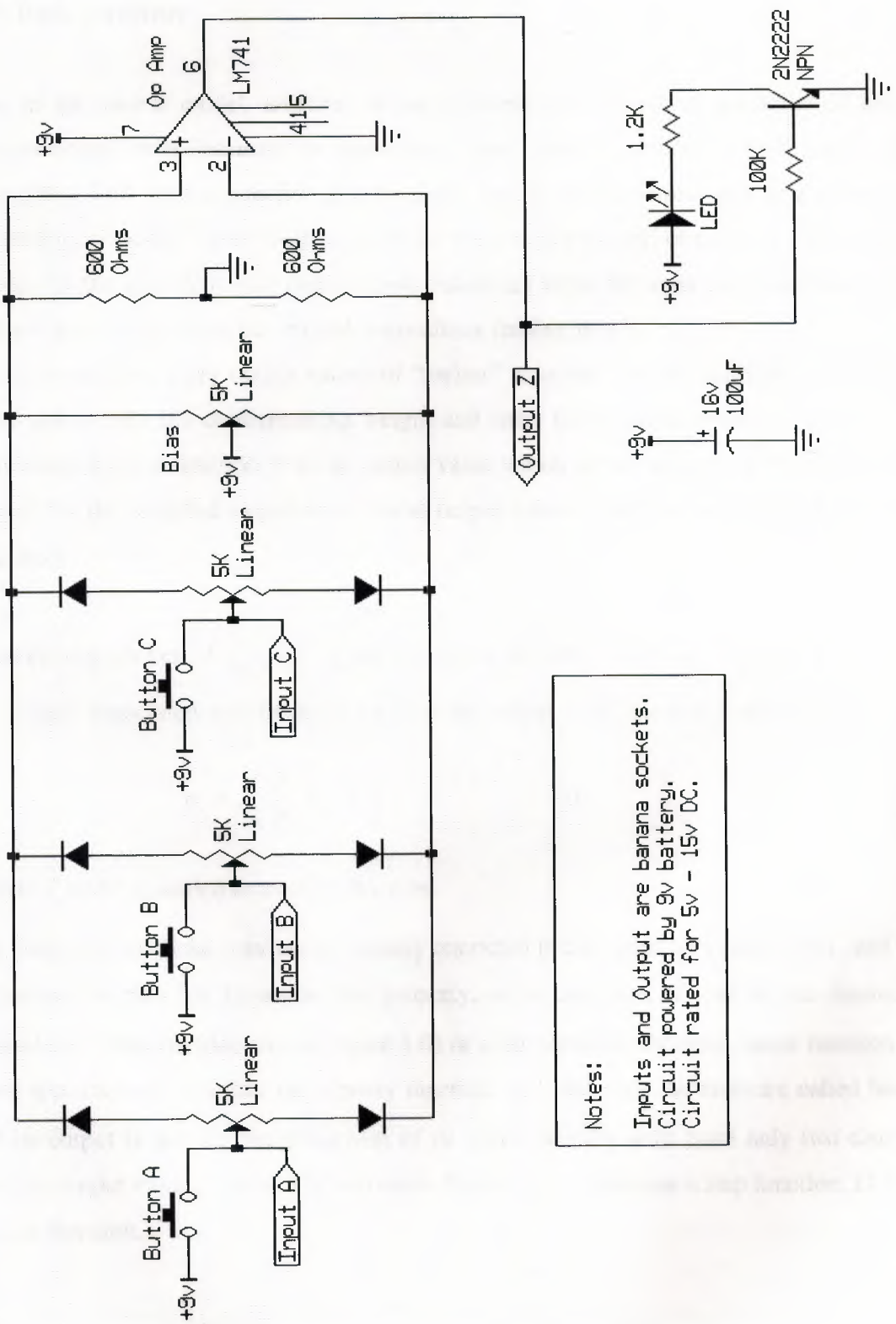


Figure 1.5 Electronic implementation of a neuron [2]

1.10 Basic Structure

Like in the natural model, artificial neural networks also consist of a number of simple computational units (neurons or sometimes also *nodes*), connected with each other. Associated with each connection is a so-called *weight* which corresponds to a synapse in the biological model. Those weights might be positive (excitatory) or negative (inhibitory). Except for the so-called *input units* (whose values are set to the input to the network), each of these artificial neurons has several connections feeding into it.

These connections carry output values of "earlier" neurons. The unit multiplies each of its input values with the corresponding weight and sums the resulting products. This sum is then mapped (by a function f) to an output value which serves as input to "later" neurons (except for the so-called *output units* whose output values represent the output of the entire network).

Theoretically spoken, if $i_1, i_2, ..., i_n$ are the inputs of unit j. And $w_{j1}, w_{j2}, w_{j3}, ..., w_{jn}$ are the weights associated with these inputs, then the output o_j of this unit is determined by

$$o_j = f\left(\sum_{k=1}^{n} w_{jk} \cdot i_k\right) \qquad (1)$$

where f is the so-called *activation function*.

The outputs of artificial neurons are usually restricted to the range between 0 and 1, and the activation function has to assure this property, so in this case, f can be the sigmoidal "squashing" function (depicted in figure 1.6) or a thresholded piecewise linear function. In some applications, f is just the identity function, in which case the units are called *linear* and its output is just the weighted sum of its inputs. Binary units have only two discrete possible output values, 0 or 1. The activation function is in this case a step function, like the signum function.

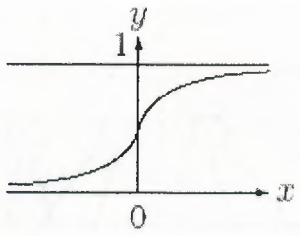


Figure 1.6 The sigmoidal "squashing" function

The nodes in an artificial neural network are usually grouped in *layers*, each consisting of one or more artificial neurons. There exist several types of topologies that determine how the interconnection networks look like. Common to all those network topologies is that there is one *input layer* and one *output layer*. The units of the input layer do not perform any computation; the values at their outputs are just set equal to their input values.

The simplest type of artificial neural network is a so-called *perceptron* which consists merely of an input layer, an output layer, and weighted connections in between (see for instance [4]).The perceptron is also the oldest type of artificial neural network. It can solve a bunch of interesting problems, but unfortunately, it is *not* applicable to many other problems. For many tasks, it is necessary that there are several (at least one) layers, called *hidden layers*, between the inputs and the outputs.

The probably most popular and most frequently used type of topology is represented by so-called *feed-forward networks*, where the units of each layer are connected only to units of "later" layers. In *recurrent networks*, there are also *feedback connections* to units in "earlier" layers or even *within* one layer. Figure 1.7 depicts a typical representative for each of the network types mentioned above.

There are some artificial models that differ from the network types presented in this subsection, such as the Hopfield nets or Boltzmann machines. For consistency reasons, in order to avoid confusion, I decided not to talk about them in detail. For further information, please see, for example, [5], [6], or [7].

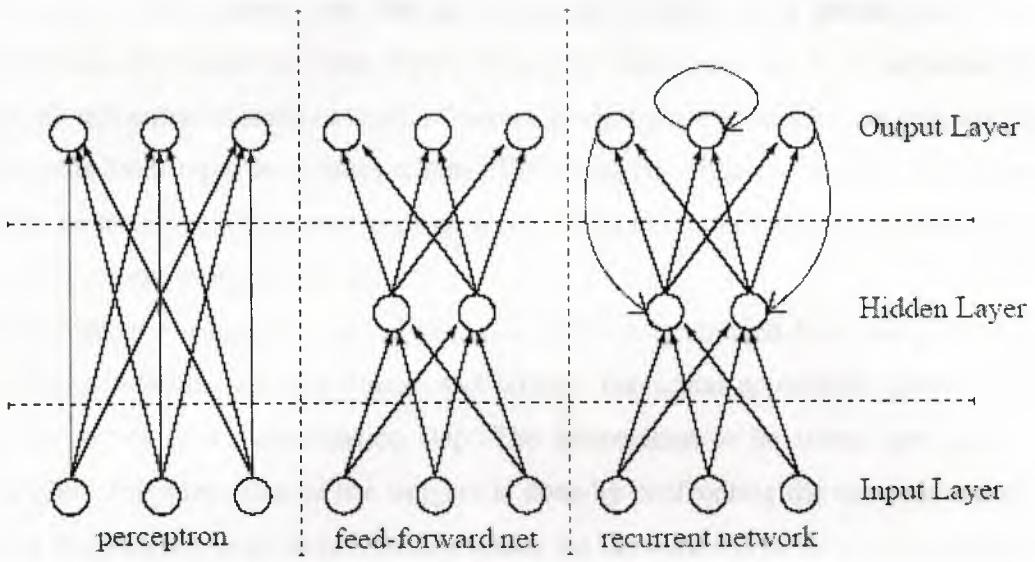


Figure 1.7 Different network topologies (all connections are weighted) [5]

In the remainder of this paper, biological neural networks are of minor interest. Therefore, I can get rid of the adjective "artificial". Whenever I use the notion "neural network" from this point on, I mean artificial ones. When I want to specifically mention the natural model, I will explicitly say so.

1.11 Learning and Training

One way to look at a neural net is to see it as a special kind of computer memory. Unlike the structure of conventional memory, there are no such things as memory cells that contain a bunch of binary bits and that can be accessed via some fixed address. "Neural memories" (i.e. memories realized by means of a neural net) are different in that they associate input patterns with certain output patterns. Thus, neural nets are often referred to as *associative memories*. In this sense, the inputs stand for an "address" and the corresponding outputs stand for the information stored at this address.

To "store" information, one has to adjust the weights in a neural net. The weights determine, from layer to layer, which units of a "later" layer are to be activated depending on the activation of units in "earlier" layers. So in total, the weights are responsible for the mapping from input to output patterns. Therefore, adjusting the weights, *adapting* them to react to certain input patterns appropriately, needs to be done before one can try to use the net to perform anything useful.

Therefore, the usage of a neural net is in most cases divided into two parts. First, a so-called *training* or *learning phase*, and second, the actual application phase. The training phase serves as an initialization step. The information to be stored gets coded into the weights. The adaptation of the weights is done by confronting the net over and over again with the patterns to be stored. In this sense, the network *learns* in a trial-and-error fashion by adjusting its weights.

Many neural networks continue adapting their weights during the application phase based on recently "learned" concepts in order to improve their performance. But in general, during the application phase, neural nets make use of the information stored in the weights in some way or another.

I now want to talk about the two basic methods used in the training phase: *supervised* and *unsupervised learning*.

1.11.1 Supervised

There are two kinds of supervised learning. The most popular and most widely used technique for training a neural net is called *learning by error correction*. The other branch of supervised learning is entitled *reinforcement learning*.

Learning by Error Correction If a network is trained using the method of *error correction*, the net is considered to learn certain mapping from inputs to outputs. The net is presented with a set of input patterns, which are entered at the input layer and "shuffled" through the net all the way up to the output layer.

A teacher (e.g. a human operator) compares the resulting output pattern with a desired or target output pattern. According to the current error (the deviation of the two patterns or *vectors* from each other), the weights in the neural network are adapted using one of several learning algorithms in order to reduce (or *correct*) the error.

One of the first learning algorithms is given by the *delta rule* (see [8]):

$$\Delta w_{jk} = \eta(t_j - o_j)i_k, \qquad (2)$$

$$w_{jk}^{new} = w_{jk}^{old} + \Delta w_{jk}. \qquad (3)$$

Here, w_{jk} is the synaptic weight from input neuron k to output neuron j, t_j is the *target output* of neuron j, o_j its actual output, i_k its actual output, k and η a *positive* factor of proportionality called *learning rate*. The rule tells us that we get the new value of a weight by adding a certain, weight-specific "delta".

In other words, the change of a weight feeding into an output unit is the bigger the more the actual output deviates from the target output. Besides it is proportionate to the value coming from input unit k. These both make sense. If the difference between actual and target output is zero, no changes need to be made to this weight. Secondly, the bigger the "flow" through this connection is (the bigger i_k), the bigger is the contribution of this weight to the error in o_j. That means for example, if i_k is zero, there is no justification for changing the weight w_{jk}, because this weight cannot be responsible for the error in o_j.

This rule works well for single-layer networks (i.e. perceptrons), but cannot be applied to general feed-forward nets, since there is no information concerning "targets" for hidden units, so the delta rule does not tell us how to change the weights feeding into these units. On the other hand, we cannot limit ourselves to networks *without* hidden units, since it has been proved that they are necessary for some tasks (see [8,10]).

To build general networks that are able to implement arbitrary types of mappings from input patterns to output patterns, there is no way to get around hidden units. However, the human operator does not know in advance how the net will implement the task at hand. Weights feeding into hidden layers correspond to internal representations which are important for the net, but not for the environment.

Rumelhart and his research group introduced in 1986 a generalization of the delta rule that works also for hidden units [8]. Their algorithm is called *backpropagation*, and it represents *the* state-of-the-art learning algorithm for multilayer feed-forward nets. As far as I know, there is almost no recent paper on artificial neural networks that does *not* mention backpropagation in some way or another. That is why I decided to dedicate the following subsection entirely to this algorithm.

Backpropagation does not only work for feed-forward nets, it can also be applied to recurrent networks, as Rumelhart shows in his original paper [8].

Genetic Algorithms In addition to the method of adapting the weights according to a certain learning rule, a more evolutionary approach can be used. *Genetic Algorithms* involve *creating* a "generation" of "individuals" (e.g. a set of neural networks with different topologies and/or different weights) and "producing" new generations which are hopefully "better" or "fitter". This is done by having the individuals *mutate* their features (e.g. change some of the weights in the network) or *reproduce* themselves and by allowing *crossovers* between two individuals, all this according to stochastic rules.

It is then determined which individuals of the new generation may survive and which of them have to die by means of evaluating a *fitness function*, which measures the quality of an individual (e.g. the conformity of the actual output of a neural net to the desired output)[6]. The whole process is then repeated generation by generation (and always the "fittest" are most likely to survive) until finally an "optimal" individual is found[7]. This "trial-and-error" approach imitates the "natural" evolution process to some extent.

Evaluating the fitness function is somewhat similar to calculating the deviation between actual and desired outputs, so *Genetic Algorithms* are related to *learning by error correction* in some sense. Genetic Algorithms represent a very interesting subject, but

unfortunately, it would exceed the range of this paper, if I discussed it in full detail. Hence, I have to refer the reader to the literature (e.g. [6, 7]).

Reinforcement Learning In *reinforcement learning*, there is no teacher telling the net how good its output is and how it has to be corrected. The only exterior feedback to the net is a signal given by a critic, telling if the current output is good or bad, right or wrong, desired or not. That means the net is only provided with scalar information, a single bit saying yes or no.
If the signal from the critic indicates that the current output is incorrect, the net does therefore *not* know *what* is wrong or *how far* its answer deviates from the desired one. The solution to this problem involves generating a teacher's signal from the critic's response and then using methods similar to those described above when talking about learning by error correction [4].

The major drawback of the reinforcement learning paradigm is that it is somewhat unrealistic in the sense that critics often provide *more* than just a scalar yes/no information (see [4]). The situation of a child trying to learn to ride a bike, for example, does definitely *not* belong to error correction learning, since the child is not told in detail which muscles to use or which movements to make. Rather, it is part of reinforcement learning, since the child notices what happens if certain movements are made in a certain order. This exterior information can be interpreted as the signal of a critic, and it is not at all scalar. The child not only learns whether or not it does the right things, it also learns *how* well it does.

1.11.2 Unsupervised

When there is no signal whatsoever providing the net with information about the quality of the current output, neither a teacher indicating a desired output nor a critic deciding whether or not the net works correctly, the learning is said to be *unsupervised*.

In this case, learning no longer means memorizing or storing input/output mappings, but discovering regularities in the input data. Thus, unsupervised learning cannot take place without those regularities. Here, the net can be really said to "program itself", it "acts" completely independently.

Some researchers think that unsupervised learning is the *only* biologically sound learning paradigm (see [10]), but on the other hand, supervised techniques are in widespread use and yield excellent results concerning technological problems.

Finally, I want to add that one approach to apply unsupervised learning involves, like in reinforcement learning, generating a target output in a certain way and thus providing the net with a teacher signal [4]. Then the adaptation of weights can again be done using "supervised" procedures like backpropagation.

1.12 Backpropagation

The most important and most popular learning algorithm was introduced by Rumelhart and his colleagues in 1986 [8, 9]. The algorithm works with a generalization of the delta rule to multi-layer feed-forward networks, that means in particular to networks with hidden units. The algorithm is called *backpropagation*, and it is based on the mathematical method of *gradient descent*. Rumelhart begins his original paper by presenting a derivation of the delta rule that shows that the delta rule also implements gradient descent.

The idea is to define an error function

$$E = \frac{1}{2}\sum_j (t_j - o_j)^2, \qquad (4)$$

Where j varies over all output units *and* all input/output patterns. This function is assumed to be a function of the weights, and the ultimate goal is to minimize this function (i.e. minimize the error) by adjusting the weights. There is no formula for the error function; the only known pieces of information are the current point in weight space and the current output, or its deviation from the desired output, respectively.

The great breakthrough now was to show that it is somehow possible to calculate the gradient of the error function, i.e. the partial derivatives of the error function with respect to each of the weights. The derivation of this result involves the one-dimensional and higher-dimensional chain rules, and I do not want to go into details (see [8]). What I only want to say is that in a backpropagation net, one first performs a forward pass, shuffling the input data through the network, and then a backward pass, computing, with the help of a recursive formula8, certain *error signals*, which are then used to calculate the partial derivatives. The backward pass with the error propagation provided the name of the algorithm.

To implement gradient descent, one has to move the weight vector in the direction of the steepest descent which is the direction of the negative gradient. And this is exactly the adaptation rule for backpropagation. The size of the step in this direction is determined by the learning rate which serves (like in the delta rule) as a factor of proportionality.

Moving the weight vector in the direction of the negative gradient means moving it towards a (local) minimum of the error function. However, to be mathematically precise, the step size should be infinitely small, but this would take infinitely long. So, backpropagation uses a bigger step size, thus only approximating gradient descent and risking to "jump over" the minimum in one step. To illustrate this, let me assume the one-dimensional case, where E is a function of only *one* independent variable w. Suppose, for simplicity that E is the parabola depicted in figure 1.8.

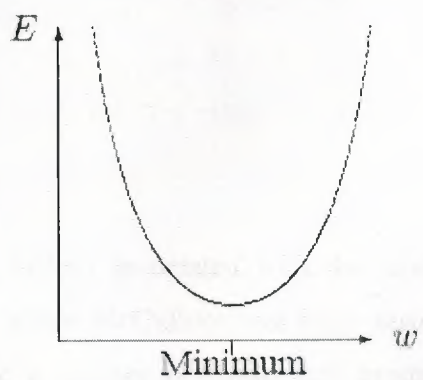


Figure 1.8 A one-dimensional error function

This special error function has a single local and global minimum. If the current value for w is too far left, the derivative $\frac{dE}{dw}$ is negative, and moving in the direction of the negative derivative means really marching right, i.e. adding a positive value, which takes w closer to the minimum (if the step size is small enough, so that w does not jump over the minimum). If w is too big, it is the same: moving towards the negative derivative means marching towards the minimum.

This adaptation method is applied iteratively, again and again for the input/output patterns to be learned, until the deviation of the actual output from the desired output is satisfyingly small, or until the weight vector converges to a local minimum. Although it may take very long until the process converges or until the "optimal" learning rate is found, and despite the problem with the local minima, the backpropagation algorithm often yields very good results, particularly since it can be applied very easily (and for all weights of one layer in parallel).

There are numerous learning algorithms that are based on backpropagation. Many of them just try to eliminate disadvantages, like for example *adaptive backpropagation*, where each weight has its own individual learning rate, and these learning rates are adapted during learning. Others seem to point in a to tally different direction, like the *scaled conjugate gradient algorithm* [11] or the *cascade-correlation algorithm* [12, 13].

But I claim that also these new developments would not have been possible without the invention of and inspiration by the backpropagation algorithm.

1.13 Historical Overview

One of the first points in history associated with the development of artificial neural networks is the year 1943, where McCulloch and Pitts introduced their *M-P neuron* (see [10]).These neurons receive a number of inputs and produce one single binary output depending on whether or not a certain threshold is reached or exceeded by the weighted sum of the inputs. The weights can only take on one of two values: +1 or –1.

An interesting application of this model is to use one single neuron as a perceptron to implement logic gates as mentioned before. For example, if there are m inputs, all of the weights are +1, and the threshold is set to m, then the output turns out to be the AND of the inputs. If the threshold is set to 1 with the same setting as far as weights are concerned, the output returns the logical OR of the inputs. Finally, if there is only one (binary) input whose weight is set to –1, and the threshold is set to 0, then the output represents the NOT of the input.

Hebb postulated in 1949 that an important property of the strengths of the connections in the biological neural network of the human brain is changing in time as the organism learns. Hebb's proposal triggered a lot of research effort in the field of adaptive neural networks, and a preliminary climax was reached when, in 1958, Rosenblatt showed how to train a network consisting of M-P neurons. In 1960, Widrow and Hoff proposed a variant of the perceptron: the *adaline* (an acronym for ADAptive LINear Element). They also introduced the now famous delta rule, which I already mentioned in the section about supervised learning.

Let me return to the idea of implementing logical functions with neural networks. Minsky and Papert proved in 1969 that it is not possible for elementary perceptrons to learn the XOR (exclusive or) function. They showed that one single M-P neuron is not enough and that hidden units are necessary to implement really *all* kinds of mappings. So the focus in the neural network community changed from elementary perceptrons to multi-layer feed-forward nets.

The problem with multi-layer networks is that there are no "target values" for hidden units. Therefore, the delta rule, which adjusts the weights feeding into a neuron proportionately to the deviation of its output from the target output, is not applicable. Thus, it was extremely difficult to train these networks for a relatively long period of time, because one simply was not sure how. In 1986, as already mentioned several times, a group around Rumelhart introduced a learning algorithm that generalizes the delta rule to general feed-forward networks. This algorithm, which is called *backpropagation* (for a description see the section on this learning method), has been since then the standard technique for training neural nets (see [10] for more details).

1.14 Summary

This chapter showed that biological neurons, where it is mainly formed of a cell body, axons, dendrites and synapses, are able to process and transmit neural activation. And the threshold Logic Unit is a crude approximation to real neurons that performs a simple summation and thresholding function on activation levels.

Appropriate mathematical notation facilitates the specification and programming of artificial neurons and networks of artificial neurons.

Also, explained the structure of the neural networks, and then learning processes was determined by different laws and was subdivided into the two basic methods used in the training phase : *supervised* and *unsupervised learning*.

CHAPTER TWO

PATTERN RECOGNITION AND ARABIC CHARACTERS

2.1 Overview

This chapter presents an overall overview of patterns, pattern recognition, and character recognition and goes all the way to give a general sight to the reader about Arabic characters, and also, stresses the aim of using neural networks in recognizing patterns and its motivation.
Also, in this chapter a discussion of a wide range of methods for pattern recognition by neural networks is provided.

2.2 What is a Pattern?

What is a pattern? A pattern is essentially an arrangement or an ordering in which some organization of underlying structure can be said to exist. We can view the world as made up of patterns. Watanabe (1985) [14] defines a pattern as an entity, vaguely defined, that could be given a name.

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

A pattern can be represented by a vector composed of measured stimuli or attributes derived from measured stimuli and their interrelationships. Often a pattern is characterized by the order of elements of which it is made, rather than the intrinsic nature of these elements. Broadly speaking, pattern recognition involves the partitioning or assignment of measurements, stimuli, or input patterns into meaningful categories. It naturally involves extraction of significant attributes of the data from the background of irrelevant details. Speech recognition maps a waveform into words. In character recognition a matrix of pixels (or strokes) is mapped into characters and words. Other examples of pattern recognition include: signature verification, recognition of faces from a pixel map, and friend-or-foe identification. Likewise, a system that would accept sonar data to determine whether the input was a submarine or a fish would be a pattern recognition system.

2.3 Pattern Recognition

Pattern recognition is the ability to categorize and identify images. When input enters one's eyes, one's brain instantly sorts and makes sense of a vast amount of visual data, allowing one to recognize a tall brown and green object as a tree. This is an amazingly complex operation requiring hundreds of thousands of neurons and a lifetime of experiences. For computer scientists, replicating this function has turned out to be overwhelmingly difficult.

One of the major features of modern computers (which will soon be revealed as a weakness) is their ability to execute sequential programs the same way each time. Each step is an exact instruction. Shown two identical photos of a tree, a computer can match them much faster than the brain by comparing the pixels in the image one by one. However, the simple true or false comparisons of the computer fail miserably when shown two photos of the same tree from two different angles. Any difference, no matter how slight, makes two images unequal. Obviously, no two images encountered in the real world will be exactly the same, but the human brain seems to effortlessly deal with this complication.

A pre-programmed algorithm fails for pattern recognition, so computer scientists have had to use different techniques. Fuzzy logic has been applied in this field, but by far the

most successful method has been neural networking. The brain has evolved over millions of years towards this purpose, so why not use it as a model? The reason that neural networks work so well is because of their parallel nature. Even if two patterns are not identical, if they have enough features in common, a neural network will be able to categorize them.

2.4 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.

2.5 Arabic Characters

2.5.1 Overview of Arabic Characters

Arabic is a language spoken by Arabs in over 20 countries, and roughly associated with the geographic region of the Middle East and North Africa, but is also spoken as a second language by several Asian countries in which Islam is the principle religion (e.g. Indonesia). However, non-Semitic languages such as Farsi, Urdu, Malay, and some West African languages such as Hausa have adopted the Arabic alphabet for writing [15]. Due to the cursive nature of the script, there are several characteristics that make recognition of Arabic distinct from the recognition of Latin scripts or Chinese (see Figure 2.1) [16]. The following section summarizes the nature of these differences.

Figure 2.1 Letters of the Isolated Arabic Alphabet

2.5.2 Arabic Alphabet

Arabic has 28 letters in the alphabet. It is based on 18 distinct shapes that vary according to their connection to preceding or following letters. Using a combination of dots and symbols above and below these shapes, the full complement of 28 consonants can be constructed.

Arabic is a cursive language. There are no capital letters and some letters are not connected to the letters that follow them (letters in blue in Figure 2).Thus, words cannot be segmented based on pen-up/pen-down information or space between letters. Block or hand printed letters do not exist in Arabic. Moreover, the cursive nature of the language makes recognition more difficult. In summary,

> *Many researchers have been working on cursive script recognition for more than three decades. Nevertheless, the field remains one of the most challenging problems in pattern recognition and all the existing systems are still limited to restricted applications [17].*

Arabic is written from right to left. Since the proposed application area provides letters in an isolated form, segmentation is assumed and direction of writing is not an issue.

However, if our system automatically segmented words for recognition, knowledge of the direction of writing would assist in segmentation and recognition.

Arabic has four forms for each letter depending on the position of the letter in each word. These are initial, medial, final and isolated (see Figure 2.2) [18]. A more generalized system would need to train 60 separate classes rather than 15 classes (for isolated letters) to accommodate all four forms for each letter.

Letter Name	*Isolated Form*	*Final Form*	*Medial Form*	*Initial Form*
Alef	ا	ا		
Ba	ب	ب	بـ	بـ
Ta	ت	ت	تـ	تـ
Tha	ث	ث	ثـ	ثـ
Jeem	ج	ج	جـ	جـ
Ha	ح	ح	حـ	حـ
Kha	خ	خ	خـ	خـ
Dal	د	د		

Figure 2.2 Samples of Various Arabic Letter Forms

A key difference between Latin scripts and Arabic is the fact that many letters only differ by a dot(s) but the primary stroke is exactly the same. Out of the 15 classes for isolated letters, 10 classes have 2 or more letters that vary by only a dot(s) or symbol. This highlights the need for a good feature extractor/classifier for the secondary stroke(s). The system detailed in this work addresses the recognition of primary strokes, and makes recommendations regarding the recognition of secondary strokes.

2.6 Motivation for ANN Approach

The development of a computer as something more than a calculating machine marked the birth of the field of pattern recognition. We have witnessed increased interest in research involving use of machines for performing intelligent tasks normally associated

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

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

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

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

The real problem of pattern recognition, however, is to generate a theory that specifies the nature of objects in such a way that a machine will be able to robustly identify them. A study of the way living systems operate provides great insight into addressing this problem.

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

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

2.7 Pattern Recognition and Neural Networks

As useful as back-propagation is, there are often easy ways to train a network. For specific-purpose networks, such as pattern recognition, the operation of the network and the training method are relatively easy to observe even though the networks might have hundreds of neurons. In its simplest form, pattern recognition uses one analog neuron for each pattern to be recognized. All the neurons share the entire input space. Assume that the two neuron network (figure 2.3) has been trained to recognize light and dark. The 2x2 grid of pixels forms the input layer. If answer #1 is the `light' output, then all of its dendrites would be excitatory (if a pixel is on, then increase the neuron's score, otherwise do nothing). By the same token, all of answer #2's dendrites would be inhibitory (if a pixel is off then increase the neuron's score, otherwise do nothing). This simply boils down to a count of how many pixels are on and how many are off. To determine if it is light or dark, pick the answer neuron with the highest score.

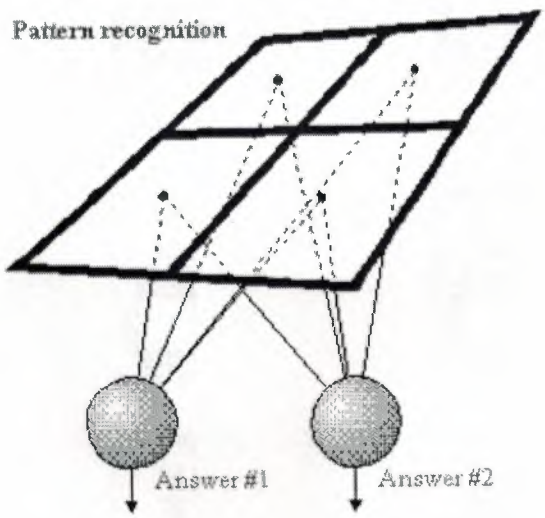


Figure 2.3 Two neuron network to recognize light and dark [2].

This example is a complete waste of a neural net, but it does demonstrate the principle. The output neurons of this type of network do not require thresholds because what matters is highest output. A more useful example which our system is going to thoroughly discuss in the next chapter of this paper would be a 7x9 grid of Boolean values that could recognize Arabic letters (figure 2.4).

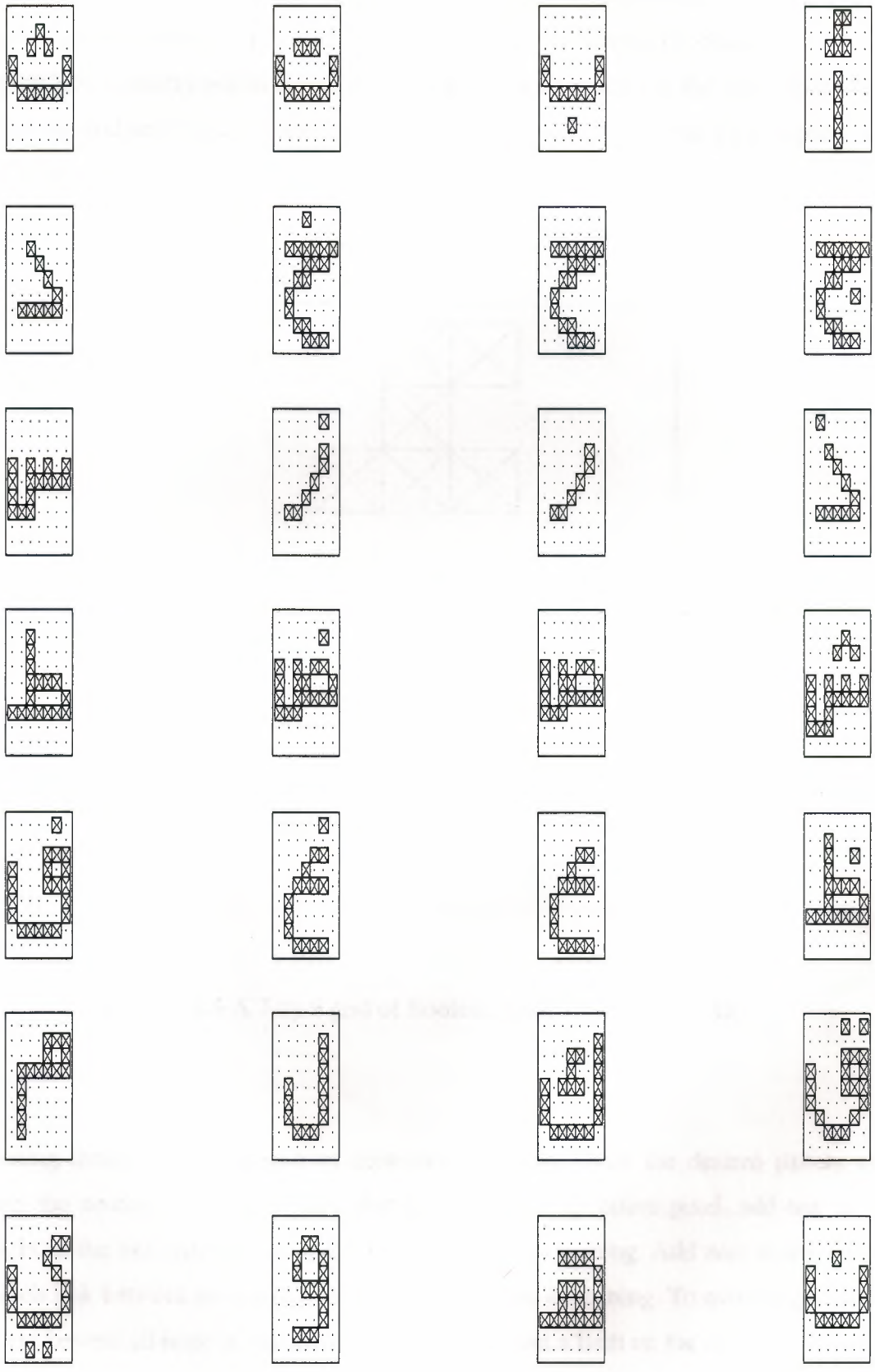


Figure 2.4 A grid configuration for the 28 Arabic alphabets.

One could have 28 neurons that all share the same 63 Boolean values input space. Each neuron would compute the probability of the inputs matching its character. The grid (figure 2.5) is configured to output the probability that the input is the letter 'Alef'. Each tile in the grid and those are not shown in the grid, being hidden from view, represents a link to the 'Alef' neuron.

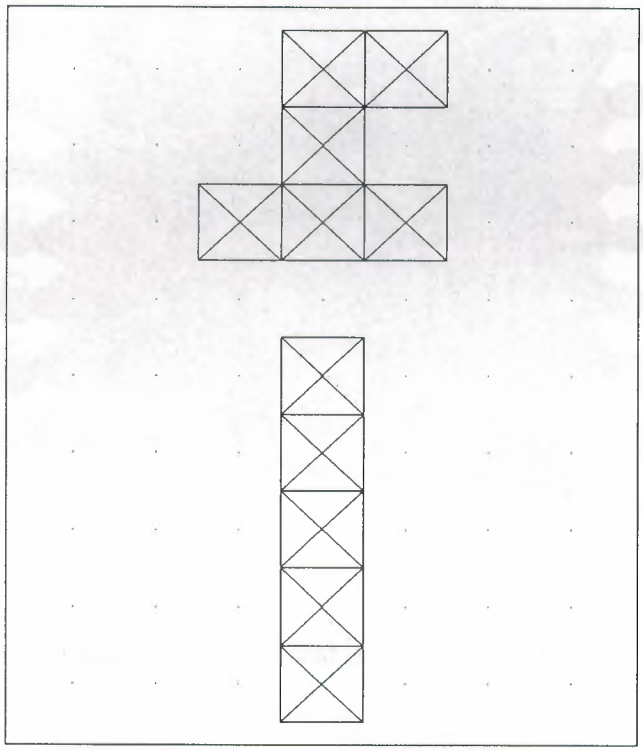


Figure 2.5 A 7 by 9 grid of Boolean value for the letter 'Alef'

Training these pattern recognition networks is simple. Draw the desired pattern and select the neuron that should learn that pattern. For each active pixel, add one to the weight of the link between the pixel and the neuron in training. Add zero to the weight of each link between an inactive pixel and the neuron in training. To avoid ingraining a pattern beyond all hope of modification, it is wise to set a limit on the absolute weights.

Another example is the network in (figure 2.6) was created by Recog [19] to recognize the digits from 0-9, and as it is clear we need 10 target neurons and the inputs depends upon the system identifier.

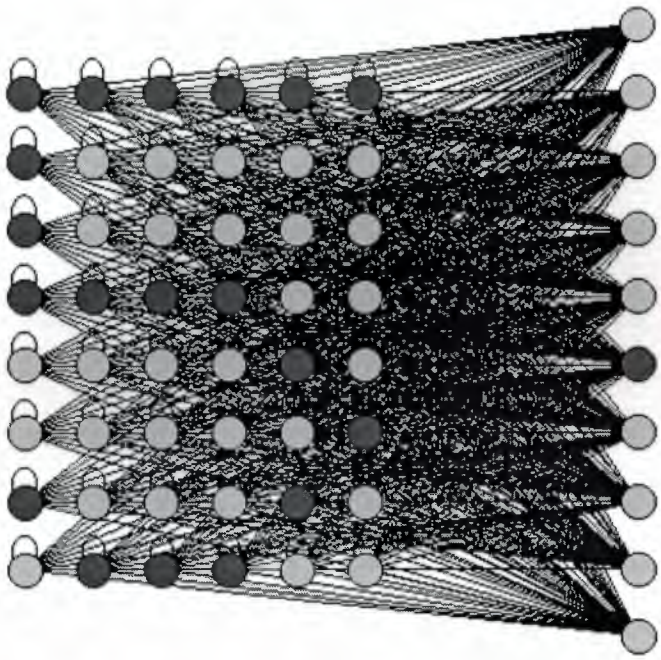


Figure 2.6 neural networks that is to recognize the digits from 0-9

A more sophisticated method of pattern recognition would involve several neural nets working in parallel, each looking for a particular feature such as "horizontal line at the top", or "enclosed space near the bottom". The results of these feature detectors would then be fed into another net that would match the best pattern. This is closer to the way humans recognize patterns. The drawback is the complexity of the learning scheme. The average child takes a year to learn the alpha-numeric system to competence.

2.8 Summary

The chapter, discussed the main topics underlying pattern recognition, including an overview of Arabic characters and how it can be shaped.

Different methods and pattern shapes for pattern recognition in line with neural network were also presented.

CHAPTER THREE

ARABIC CHARACTER RECOGNITION

3.1 Overview

This chapter demonstrates how Arabic character recognition can be done with a backpropagation network. The Purpose of the Neural Network, its architecture, and the training of this network with both noisy and noise free input vectors are going to be discussed.

Also, in this chapter, the performance of both training with and without noise is to be examined.

3.2 Problem Statement

A network is to be designed and trained to recognize the 28 letters of the Arabic alphabet. Assuming that our system segments the given words for recognition, and also 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 7 by 9 grid of Boolean values. For example, here is the letter "Alef" is depicted in figure 3.1. in the figure each grid stands for a binary bit, where the hidden (invisible) grids represent the availability of binary bit '0' in the input array of the Neural Network, and those can be seen as a crossed square (visible) represent a binary bit '1'.

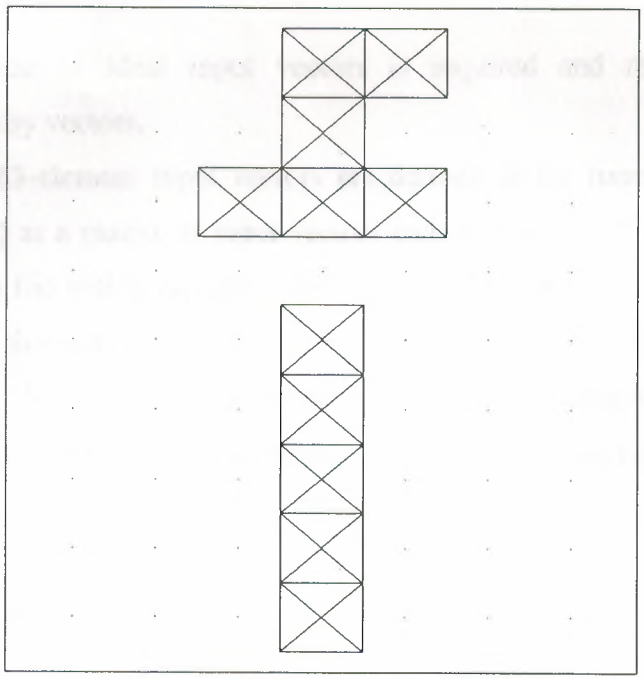


Figure 3.1 A noise free letter 'Alef'

However, the imaging system is not perfect and the letters may suffer from noise (figure 3.2).

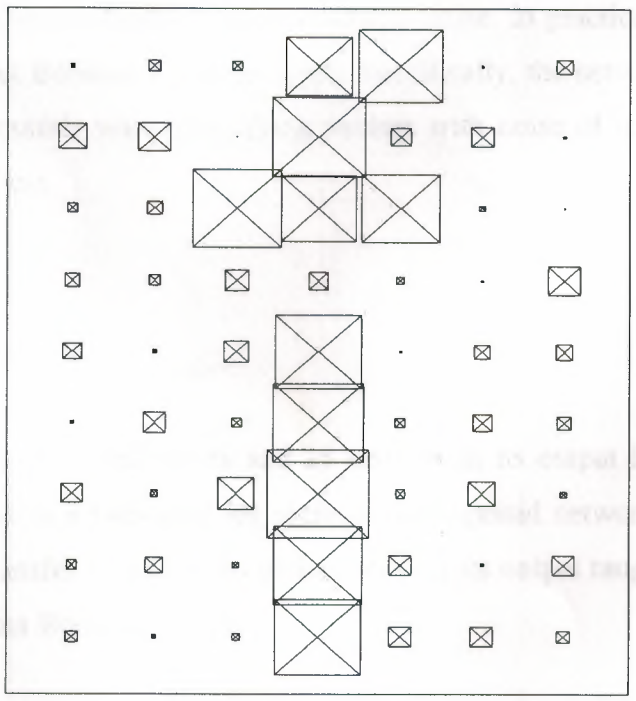


Figure 3.2 A noisy letter 'Alef'

Perfect classification of ideal input vectors is required and reasonably accurate classification of noisy vectors.

The twenty-eight 63-element input vectors are defined in the function prprob63 (see Appendix A (A.1)) 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 28-element vector with a 1 in the position of the letter it represents, and 0's everywhere else. For example, the letter "Alef" is to be represented by a 1 in the first element (as "Alef" is the first letter of the alphabet), and 0's in elements two through twenty-eight.

3.3 Neural Network

The network receives the 63 Boolean values as a 63-element input vector. It is then required to identify the letter by responding with a 28-element output vector. The 28 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.

3.3.1 Architecture

The neural network needs 63 inputs and 28 neurons in its output layer to identify the letters. The network is a two-layer log-sigmoid/log-sigmoid network [15] (figure 3.3). The log-sigmoid transfer function was picked because its output range (0 to 1) is perfect for learning to output Boolean values.

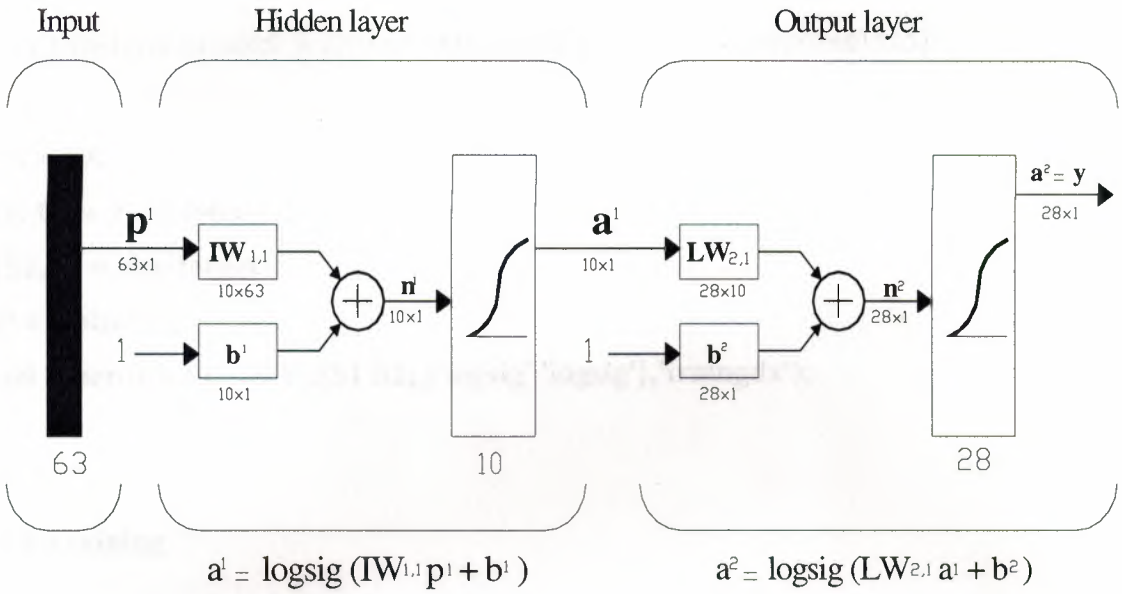


Figure 3.3 Architecture of the Neural Network

Where, IW: Input Weight Matrix, LW: Layer Weight Matrix.

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' (MATLAB command) [15]. 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.

3.3.2 Initialization

The two-layer network is created with 'newff' (MATLAB command) [15].

```
S1 = 10;
[R,Q] = size(alphabet);
[S2,Q] = size(targets);
P = alphabet;
net = newff(minmax(P),[S1 S2],{'logsig' 'logsig'},'traingdx');
```

3.4 Training

To create a network that can handle noisy input vectors it is best to train the network on both ideal and noisy vectors. To do this, the network is first trained on ideal vectors until it has a low sum-squared error.

Then, the network is trained on 10 sets of ideal and noisy vectors. The network is trained on two copies of the noise-free alphabet at the same time as it is trained on noisy vectors. The two copies of the noise-free alphabet are used to maintain the network's ability to classify ideal input vectors.

Unfortunately, after the training described above the network may have learned to classify some difficult noisy vectors at the expense of properly classifying a noise-free vector. Therefore, the network is again trained on just ideal vectors. This ensures that the network responds perfectly when presented with an ideal letter. All training is done using backpropagation with both adaptive learning rate and momentum with the function trainbpx.

3.4.1 Training without Noise

The network is initially trained without noise for a maximum of 10000 epochs or until the network sum-squared error falls beneath 0.05.

```
P = alphabet;
T = targets;
net.performFcn = 'sse';
net.trainParam.goal = 0.05;
net.trainParam.show = 20;
net.trainParam.epochs = 10000;
net.trainParam.mc = 0.95;
[net,tr] = train(net,P,T);
```

3.4.2 Training with Noise

To obtain a network not sensitive to noise, we trained with two ideal copies and two noisy copies of the vectors in alphabet. The target vectors consist of four copies of the vectors in target. The noisy vectors have noise of mean 0.1 and 0.2 added to them. This forces the neuron to learn how to properly identify noisy letters, while requiring that it can still respond well to ideal vectors.

To train with noise, the maximum number of epochs is reduced to 3000 and the error goal is increased to 0.1, reflecting that higher error is expected because more vectors (including some with noise), are being presented.

```
netn = net;
netn.trainParam.goal = 0.1;
netn.trainParam.epochs = 3000;
T = [targets targets targets targets];
for pass = 1:10
```

```
P = [alphabet, alphabet, ...
    (alphabet + randn(R,Q)*0.1), ...
    (alphabet + randn(R,Q)*0.2)];
[netn,tr] = train(netn,P,T);
end
```

3.4.3 Training without Noise Again

Once the network is trained with noise, it makes sense to train it without noise once more to ensure that ideal input vectors are always classified correctly. Therefore, the network is again trained with code identical to the Training Without Noise section.

3.5 System Performance

The reliability of the neural network pattern recognition system is measured by testing the network with hundreds of input vectors with varying quantities of noise. The script file appcr2 (MATLAB script) [15] tests the network at various noise levels, and then graphs the percentage of network errors versus noise. Noise with a mean of 0 and a standard deviation from 0 to 0.5 is added to input vectors. At each noise level, 100 presentations of different noisy versions of each letter are made and the network's output is calculated. The output is then passed through the competitive transfer function so that only one of the 28 outputs (representing the letters of the Arabic Alphabet), has a value of 1.
The number of erroneous classifications is then added and percentages are obtained.

The script appcr2 (see Appendix A (A.2)) demonstrates how Arabic character recognition can be done with a backpropagation network.

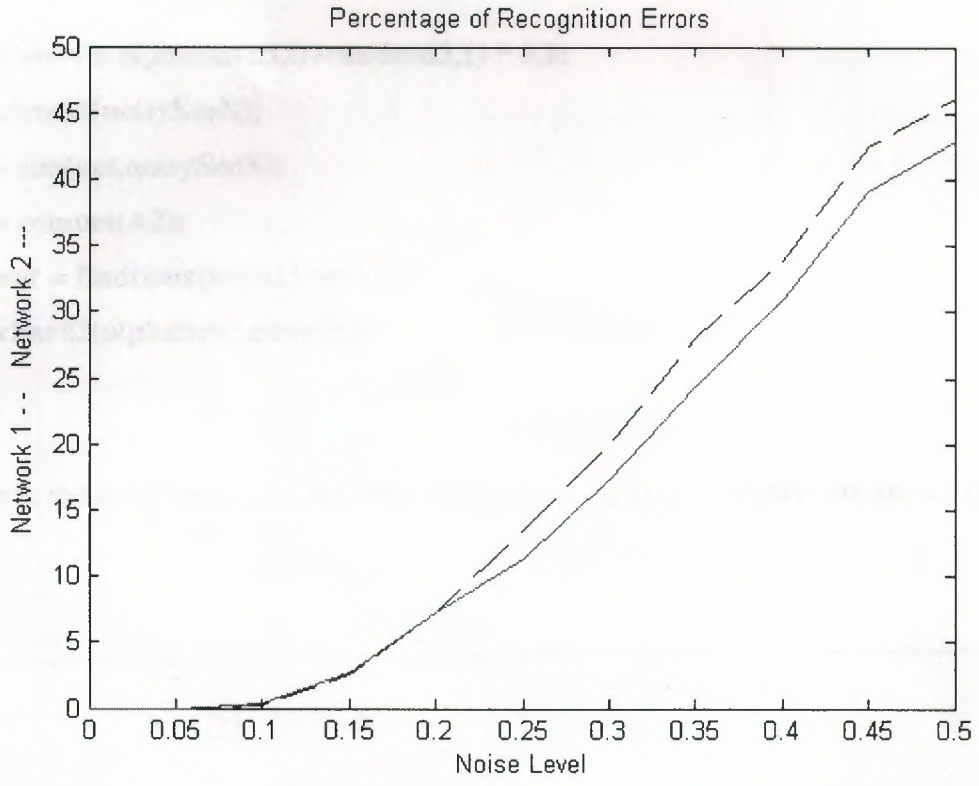


Figure 3.4 Reliability of the network to classify noisy vectors.

The solid line on the graph (figure 3.4) shows the reliability for the network trained with and without noise. The reliability of the same network when it had only been trained without noise is shown with a dashed line. Thus, training the network on noisy input vectors greatly reduces its errors when it has to classify noisy vectors. The network did not make any errors for vectors with noise of mean 0.00 or 0.05. When noise of mean 0.2 was added to the vectors both networks began making errors. If a higher accuracy is needed, the network can be trained for a longer time or retrained with more neurons in its hidden layer. Also, the resolution of the input vectors can be increased to a 10x14 grid. Finally, the network could be trained on input vectors with greater amounts of noise if greater reliability were needed for higher levels of noise. To test the system, a letter with noise can be created and presented to the network.

```
noisySeeN = alphabet(:,12)+randn(63,1) * 0.2;
plotchar63(noisySeeN);
A2 = sim(net,noisySeeN);
A2 = compet(A2);
answer = find(compet(A2) == 1);
plotchar63(alphabet(:,answer));
```

Here is the noisy letter and the letter the network picked (correctly) are shown in figure 3.5.

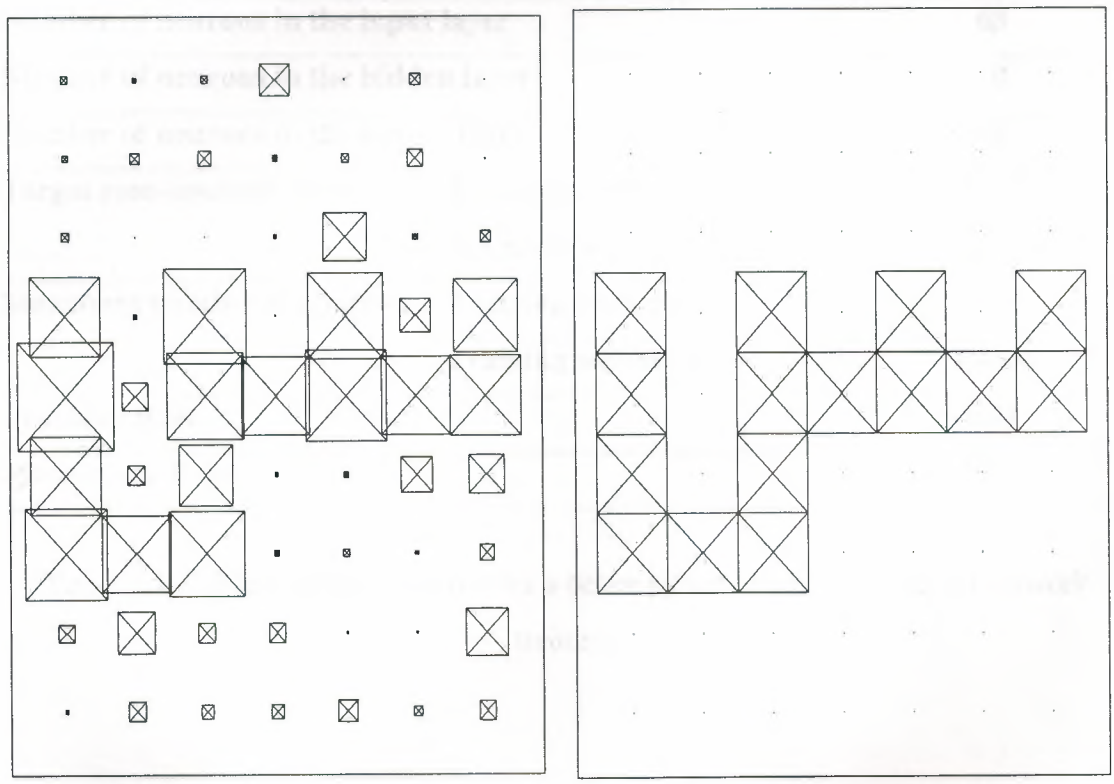


Figure 3.5 A noisy letter 'seen' (left) and the letter the network picked (correctly) (right).

3.6 Neural Network Training Parameters

After experiencing the network with different values for each of the learning rate, momentum rate, sum-squared error (for both training with and without noise) and the number of epochs required for fair and good results (also, for both training with and without noise) and after checking its performance, the last achieved values were tabulated below (table 3.1). Note that we have the values of both input and output neurons depending on the number of input vectors and output targets, and the number of neurons in the hidden layer was obtained mainly by guesswork and long working experience as mentioned earlier in this chapter.

Number of neurons in the input layer		**63**
Number of neurons in the hidden layer		**10**
Number of neurons in the output layer		**28**
Target sum-squared error	**Training with noise**	**0.05**
	Training without noise	**0.1**
Maximum number of epochs	**Training with noise**	**3000**
	Training without noise	**10000**
Learning Rate		**0.01**
Momentum Rate		**0.95**

Table 3.1 The last achieved values for a better performance of the neural network training.

3.7 Summary

This chapter demonstrates how a simple pattern recognition system can be designed. Note that the training process did not consist of a single call to a training function. Instead, the network was trained several times on various input vectors. In this case, training a network on different sets of noisy vectors forced the network to learn how to deal with noise, a common problem in the real world.

CONCLUSION

Today, neural networks can solve problems of economic importance that could not be approached previously in any practical way. One of the recent neural network applications is discussed in this project.

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.

In this project we designed a scheme able to recognize Arabic alphabet. We used a supervised neural network that we trained using the *Backpropagation algorithm.*
A back-propagation neural network with one hidden layer was used to create an adaptive Arabic character recognition system. The system was trained and evaluated with input vectors, as well as other different forms of noisy input vectors. Experiments tested (1) the effect on recognition accuracy without noise, and (2) the effect on recognition accuracy with a Gaussian distributed noise added to the input vector. Results showed reduced accuracy in recognizing characters when training without noise and it illuminated the fact that a better performance and more accuracy of the system is achieved by training with noise, since noise is an unavoidable problem in the real world!.
Note that the training process did not consist of a single call to a training function. Instead, the network was trained several times on various input vectors. In this case, training a network on different sets of noisy vectors forced the network to learn how to deal with noise.

And far from the training process and the achieved results the main objectives of this project were to introduce the reader with general definitions of Neural Network and its practical applications, to investigate pattern recognition, and to acquire a good command of Arabic characters. Also, in this project we aimed to simulate a working neural network that recognizes Arabic alphabet in which we tried to make use of MATLAB Neural Network toolbox as well.

REFERENCES

[1] HAYKIN, S., Neural Networks, 2nd edition 1999.

[2] HTTP://vv.carleton.ca/~neil/neural/neuron.

[3] HTTP://vv.carleton.ca/~neil/neural/brainbox.html.

[4] WEIGEND, A. S., Review of the Book *Introduction to the Theory of Neural Computation* by HERTZ, J. A., KROGH, A. S., & PALMER, R. G., Elsevier, Artificial Intelligence 62 (1993) pp. 93 – 111, Elsevier Science Publishers B. V., 1993

[5] LIPPMANN, R. P., *An Introduction to Computing with Neural Nets*, in: LAU, C. G. Y. (Ed.), *Neural Networks: Theoretical Foundations and Analysis*, pp. 5 – 23, IEEE Press, New York, 1992

[6] LIPPMANN, R. P., MOODY, J. E., TOURETZKY, D. S. (Eds.), *Advances in Neural Information Processing Systems 3*, Morgan Kaufmann Publishers, San Mateo, CA, 1991

[7] SANCHEZ-SINENCIO, E., LAU, C. G. Y. (Eds.), *Artificial Neural Networks: Paradigms, Applications, and Hardware Implementations*, IEEE Press, New York, 1992

[8] RUMELHART, D. E., HINTON, G. E., WILLIAMS, R. J., *Learning Internal Representations by Error Propagation*, ICS Report 8506, Sept. 1985, Institute for Cognitive Science, University of California, San Diego, published in: *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, Vol. I, RUMELHART, D. E., MCCLELLAND, J. L. (Eds.), Cambridge, MA, MIT Press, pp. 318 - 362, 1986

[9] RUMELHART, D. E., HINTON, G. E., WILLIAMS, R. J., *Learning Representations by Backpropagating Errors*, Nature 323: pp. 533 – 536, 1986

[10] VEMURI, V., *Artificial Neural Networks: An Introduction*, in: VEMURI, V. (ed.), *Artificial Neural Networks: Theoretical Concepts*, Computer Society Press, pp. 1 – 12, 1988

[11] MØLLER, M. F., *A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning*, Computer Science Department, University of Aarhus, Denmark, Preprint 11/13/1990

[12] FAHLMAN, S. E., LEBIERE, C., *The Cascade-Correlation Learning Architecture*, Technical Report # CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1990

[13] YANG, J., HONAVAR, V., *Experiments with the Cascade-Correlation Algorithm*, Technical Report # 91-16, Department of Computer Science, Iowa State University, Ames, Iowa, July 1991

[14] CHEN, C. H., *Statistical Pattern Recognition*, Hayden, Washington, D.C., 1973.

[15] MATLAB version 6.5.0.180913a Release 13, Neural Networks Toolbox.

APPENDIX A

A.1 The twenty-eight 63-element input vectors can be defined in the Matlab m.file prprob63 as follows:

```
function [alphabet,targets] = prprob63()
%prprob63 Arabic Character recognition problem definition
% [ALHABET,TARGETS] = prprob63()
% Returns:
%   ALPHABET - 63x28 matrix of 7x9 bit maps for each letter.
%   TARGETS  - 28x28 target vectors.

letterAlif = [0 0 0 1 1 0 0  0 0 0 1 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0
              0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0]';
letterBaa = [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 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 1 1 1
             1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]';
letterTaa = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 10 1 1 1 1
             1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]';
letterTh!aa = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1 1 1
               1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]';
letterJiim = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 0 0 0 1 1 0 0 0 0 1 0 0 0
              1 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 1 0]';
letterHaa= [0 0 0 0 0 0 0  0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 0 0 0 1 1 0 0 0 0 1 0 0 0
            0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 1 0]';
letterXaa = [0 0 0 1 0 0 0 0 0 0 0 0 0 0  0 1 1 1 1 1 1 0 0 0 1 1 1 0 0 0 1 1 0 0 0 0 1 0 0 0
             0 0 0 1 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 1 0]';
letterDaal = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
              1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0]';
letterThaal = [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
               0 1 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0]';
letterRaa = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0  0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0
             0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]';
```

```
letterZaay = [0000010 0000000000001000000100000010000011000000000000000000];
letterSiin = [00000000000000000000010101011011111101000011100000000000000000];
letterShiin = [00000000000010000010100000000101010110111110100001110000 0000000];
letterSaad= [00000000000000000000000101011010110011011111100000000000000000];
letterDaad = [0000000 0000010000000000 1010110 1011001101111111100000000000000000];
letterTaa = [0000000001 0000 0010000001000000011110001000111111110000000000000000];
letterThcaa = [0000000001 0000 0010010001 0000 0011110001000111111110000000000000000];
letterAyn= [0000000 0000000 0000110 0001000 0011110010000010000000011110];
letterGhayn = [0000010000000000 0000110000010000 0011110 0100000 0100000 0100000011110];
letterFaa = [0000010000000000000011110001011000111100000110000010111110000000];
letterQaaf = [0000101000000000 0000111100010110001111000001010001000111000000000];
letterKaaf= [0000000000000001000110100010011011101100000110000010111110000000];
letterLaam = [0000000000000100000010000000100100010010001001000100011100000000];
letterMiim = [0000000000000111 0000101011111110100000000 0100000 0100000 0100000000000000];
letterNuun = [0000000000000000 0001000100000011000000110000010111110 0000000000000000];
letterHaa= [0000000 0000000 0011110000000010011101001010111111110000000000000000];
letterWaaw = [0000000000011000010010001001000 0001110000000100001000111000000000];
```

```
letterYaa = [0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 1 1 0 0 0 0
             0 1 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0]';

alphabet = [letterAlif, letterBaa, letterTaa, letterTh!aa, letterJiim, letterHaa, letterXaa,
            letterDaal, letterThaal, letterRaa, letterZaay, letterSiin, letterShiin,
            letterSaad, letterDaad, letterTaa, letterThcaa, letterAyn, letterGhayn,
            letterFaa, letterQaaf, letterKaaf, letterLaam, letterMiim, letterNuun,
            letterHaa, letterWaaw, letterYaa];

targets = eye(28);
```

A.2 Arabic character recognition can be done with a backpropagation network by the Matlab script appcr2 as follows:

```
%APPCR2 Character recognition.

clf;
figure(gcf)
echo on
%   NEWFF   - Inititializes feed-forward networks.
%   TRAINGDX - Trains a feed-forward network with faster backpropagation.
%   SIM   - Simulates feed-forward networks.
%   CHARACTER RECOGNITION:
%   Using the above functions a feed-forward network is trained
%   to recognize character bit maps, in the presence of noise.
pause % Strike any key to continue...
%   DEFINING THE MODEL PROBLEM
%   ==========================
%   The script file PRPROB63 defines a matrix ALPHABET
%   which contains the bit maps of the 28 letters of the
%   alphabet.
%   This file also defines target vectors TARGETS for
%   each letter. Each target vector has 28 elements with
```

```
%   all zeros, except for a single 1.  'Alef' has a 1 in the
%   first element, 'Baa' in the second, etc.
[alphabet,targets] = prprob63;
[R,Q] = size(alphabet);
[S2,Q] = size(targets);
pause % Strike any key to define the network...
%   DEFINING THE NETWORK
%   ====================
%   The character recognition network will have 25 TANSIG
%   neurons in its hidden layer.
S1 = 10;
net = newff(minmax(alphabet),[S1 S2],{'logsig' 'logsig'},'traingdx');
net.LW{2,1} = net.LW{2,1}*0.01;
net.b{2} = net.b{2}*0.01;
pause % Strike any key to train the network...
%   TRAINING THE NETWORK WITHOUT NOISE
%   ==================================
net.performFcn = 'sse';      % Sum-Squared Error performance function
net.trainParam.goal = 0.05;    % Sum-squared error goal.
net.trainParam.show = 20;      % Frequency of progress displays (in epochs).
net.trainParam.epochs = 10000;  % Maximum number of epochs to train.
net.trainParam.mc = 0.95;      % Momentum constant.
%   Training begins...please wait...
P = alphabet;
T = targets;
[net,tr] = train(net,P,T);
%   ...and finally finishes.
pause % Strike any key to train the network with noise...
%   TRAINING THE NETWORK WITH NOISE
%   ===============================
%   A copy of the network will now be made.  This copy will
%   be trained with noisy examples of letters of the alphabet.
netn = net;
netn.trainParam.goal = 0.1;   % Mean-squared error goal.
```

```
netn.trainParam.epochs = 3000;  % Maximum number of epochs to train.
%    The network will be trained on 10 sets of noisy data.
pause % Strike any key to begin training...
%    Training begins...please wait...
T = [targets targets targets targets];
for pass = 1:10
  fprintf('Pass = %.0f\n',pass);
  P = [alphabet, alphabet, ...
     (alphabet + randn(R,Q)*0.1), ...
     (alphabet + randn(R,Q)*0.2)];
  [netn,tr] = train(netn,P,T);
  echo off
end
echo on
%    ...and finally finishes.
pause % Strike any key to finish training the network...
%    TRAINING THE SECOND NETWORK WITHOUT NOISE
%    ========================================
%    The second network is now retrained without noise to
%    insure that it correctly categorizes non-noizy letters.
netn.trainParam.goal = 0.1;    % Mean-squared error goal.
netn.trainParam.epochs = 500;  % Maximum number of epochs to train.
net.trainParam.show = 5;       % Frequency of progress displays (in epochs).
%    Training begins...please wait...
P = alphabet;
T = targets;
[netn,tr] = train(netn,P,T);
%    ...and finally finishes.
pause % Strike any key to test the networks...
%    TRAINING THE NETWORK
%    ====================
% SET TESTING PARAMETERS
noise_range = 0.05:0.05:0.5;
max_test = 100;
```

```
network1 = [ ];
network2 = [ ];
T = targets;
% PERFORM THE TEST
for noiselevel = noise_range
 fprintf('Testing networks with noise level of %.2f.\n',noiselevel);
 errors1 = 0;
 errors2 = 0;
 for i=1:max_test
  P = alphabet + randn(63,28)*noiselevel;
  % TEST NETWORK 1
  A = sim(net,P);
  AA = compet(A);
 errors1 = errors1 + sum(sum(abs(AA-T)))/2;
  % TEST NETWORK 2
  An = sim(netn,P);
  AAn = compet(An);
  errors2 = errors2 + sum(sum(abs(AAn-T)))/2;
  echo off
 end
 % AVERAGE ERRORS FOR 100 SETS OF 28 TARGET VECTORS.
 network1 = [network1 errors1/28/100];
 network2 = [network2 errors2/28/100];
end
echo on
pause % Strike any key to display the test results...
%   DISPLAY RESULTS
%   ===============
%   Here is a plot showing the percentage of errors for
%   the two networks for varying levels of noise.
clf
plot(noise_range,network1*100,'--',noise_range,network2*100);
title('Percentage of Recognition Errors');
xlabel('Noise Level');
```

```
ylabel('Network 1 - -  Network 2 ---');
%  Network 1, trained without noise, has more errors due
%  to noise than does Network 2, which was trained with noise.
echo off
disp('End of APPCR2').
```