NEAR EAST UNIVERSTIY

Faculty of Engineering

Department of Computer Engineering

IMPLEMENTATIONS OF NEURAL NETWORKS

Graduation Project
COM-400

Student : Yousef Ahmed Haif
Supervisor : Assoc.Prof.Dr.Adnan Khashman

Nicosia – 2003

ACKNOWLEDGMENT

First, I would like to thanks my supervisor Assoc.Prof.Dr Adnan khashman for his great advice and recommendations to finish this project properly.

Although I faced many problems collection data but he has guided me the appropriate references.

Second, I would like to thank my father and my brothers for supporting me to finish my school and giving me a lot of support to be a good man in this life.

Third, I thank all the staff of the Department of Computer Engineering for giving me the facilities to practice and solving any problem I was facing during working in this project.

Fourth, Thanks for all my friends for their advices and support me.

Finally I would like to dedicate this work to all those who are interested in Neural Network.

ABSTRACT

Work on artificial neural networks, commonly referred to as "neural networks", has been motivated right from its inception by the recognition that the brain computes in an entirely different way from the conventional digital computer.

When we are talking about a neural network, we should more properly say "artificial neural network" (ANN), Biological neural networks are much more complicated than the mathematical models we use for artificial neural networks. But it is customary to be lazy and drop the "A" or the "artificial".

Neural networks are computing devices that are loosely based on the operation of the brain. A neural network consists of a large number of simple processing units (or "neurons"), massively interconnected and operating in parallel.

This project describes what artificial neural networks are, how to use them, and where they are currently being applied. A brief overview of neural networks and their history is provided. The project describes the individual neurons that comprise an artificial neural network in detail, along with the most common training procedures in use.

Neural networks architectures and algorithms are presented. The project briefly summarizes application areas where neural networks are commonly applied. Finally, neural networks application in fraud detection is noted.

INTRODUCTION

Humans and other living beings manage to survive within a very complex world in their daily life. An uncountable number of decisions have to be taken on the way. Many of which are substantially based on existing knowledge and made by complex and discovered inference mechanism. Eventually, these decisions are derived either unconsciously or consciously. The ability of humans to make decisions has been created by an evolutionary selection process. The property is more powerful than any corresponding software implementation on existing system. Decisions, as a matter of fact, are taken mainly by actions of the brain and may be characterized by a complex hierarchical structure of associated subroutines. The decisions may solve, for instance, a pattern classification, recognition, or other problems. They may lead to certain control actions of the body, or they may not have any direct output.

System modeling has been an important issue in both engineering and non-engineering areas. Conventional approaches to system modeling rely heavily on mathematical tools that emphasize the precision and exact description of each quantity involved. The use of these mathematical tools such as (difference equations, transfer functions, relation models etc) is appropriate when the system is simple and/or well defined

To overcome the problems encountered by conventional modeling methods, **Neural Network** modeling have been proposed as viable alternatives and successfully employed in various areas where conventional approaches fail to provide satisfactory solutions.

Artificial Neural Networks are being touted as the wave of the future in computing. They are indeed self learning mechanisms which don't require the traditional skills of a programmer. But unfortunately, misconceptions have arisen. Writers have hyped that these neuron-inspired misconceptions have arisen. Writers have hyped that these neuron-

inspired misconceptions have arisen. Writers have hyped that these neuron-inspired their problems with neural networks.

Although Artificial Neural Networks (ANNs) have been around since the late 1950's, it wasn't until the mid-1980 that algorithms became sophisticated enough for general applications. Today ANNs are being applied to an increasing number of real-world problems of considerable complexity.

There are multitudes of different types of ANNs. Some of the more popular include the multilayer perceptron which is generally trained with the back propagation of error algorithm, learning vector quantization, radial basis function, Hopfield, and Kohonen, to name a few. Some ANNs are classified as feedforward while others are recurrent (i.e., implement feedback) depending on how data is processed through the network. Another way of classifying ANN types is by their method of learning (or training), as some ANNs employ supervised training while others are referred to as unsupervised or self-organizing. Supervised training is analogous to a student guided by an instructor. Unsupervised algorithms essentially perform clustering of the data into similar groups based on the measured attributes or features serving as inputs to the algorithms. This is analogous to a student who derives the lesson totally on his or her own. ANNs can be implemented in software or in specialized hardware.

In chapter one we have demonstrated a basic introduction to neural networks. Within the chapter we have explained that neural networks are groups of select neurons that are connected with one another and they are functional circuits in the brain that process information and create useful activities by sending outputs to the body. In chapter two we have discussed neural networks classified according to their learning processes into two types, supervised learning and unsupervised learning. Also this chapter discussed the various types of neural networks structures and algorithms. In chapter three we demonstrate applications of artificial neural networks in various fields. We have described briefly neural networks applications in language processing, character and pattern recognition, and servo control application. In chapter four we have demonstrate how to recognize the hand written digital recognition a using percepton neural network and using the back propagation algorithms.

The aim of this project is to investigate and demonstrate application of neural network in real life.

Various structures of neural networks will also be described .

CHAPTER ONE

INTRODUCTION TO NEURAL NETWORKS

1.1 Overview

This chapter intended to act a brief introduction to Artificial Neural Network technology and what Artificial Neural Networks are, how to use them, why they are important and who should know about Neural Networks. And will explain where Artificial Neural Networks have come from and presents a brief history of Neural Networks. Also this chapter discusses how they are currently being applied, and what types of application are currently utilizing the different structures. It will also detail why there has been such a large amount of interest generated in this are, and where the future of this technology.

1.2 Artificial Neural Networks

Artificial Neural Networks are relatively crude electronic models based on the neural structure of the brain. The brain basically learns from experience. It is natural proof that some problems that are beyond the scope of current computers are indeed solvable by small energy efficient packages. This brain modeling also promises a less technical way to develop machine solutions. This new approach to computing also provides a more graceful degradation during system overload than its more traditional counterparts.

These biologically inspired methods of computing are thought to be the next major advancement in the computing industry. Even simple animal brains are capable of functions that are currently impossible for computers. Computers do rote things well, like keeping ledgers or performing complex math. But computers have trouble recognizing even simple patterns much less generalizing those patterns of the past into actions of the future.

Now, advances in biological research promise an initial understanding of the natural thinking mechanism. This research shows that brains store information as patterns. Some of these patterns are very complicated and allow us the ability to recognize individual faces from many different angles. This process of storing information as patterns, utilizing those patterns, and then solving problems encompasses a new field in computing. This field, as mentioned before, does not utilize traditional programming but involves the creation of massively parallel networks and the training of those networks to solve specific problems. This field also utilizes words very different from traditional computing, words like behave, react, self-organize, learn, generalize, and forget.

Unfortunately, that confusion has come from the industry itself. An avalanche of articles has appeared touting a large assortment of different neural networks, all with unique claims and specific examples. Currently, only a few of these neuron-based structures, paradigms actually, are being used commercially. One particular structure, the feed forward, back propagation network, is by far and away the most popular. Most of the other neural network structures represent models for "thinking" that are still being evolved in the laboratories. Yet, all of these networks are simply tools and as such the only real demand they make is that they require the network architect to learn how to use them.

1.3 History of Neural Networks

The study of the human brain is thousands of years old. With the advent of modern electronics, it was only natural to try to harness this thinking process.

The history of neural networks can be traced back to the work of trying to model the neuron. The first model of a neuron was by physiologists, McCulloch and Pitts (1943) [1]. The model they created had two inputs and a single output. McCulloch and Pitts noted that a neuron would not activate if only one of the inputs was active. The weights for each input were equal, and the output was binary. Until the inputs summed up to a certain threshold level, the output would remain zero. The McCulloch and Pitts' neuron has become known today as a logic circuit.

The *perceptron* was developed as the next model of the neuron by Rosenblatt (1958) [2], as seen in Figure 1.2. Rosenblatt, who was a physiologist, randomly interconnected the perceptrons and used trial and error to randomly change the weights in order to achieve "learning." Ironically, McCulloch and Pitts' neuron is a much better model for the electrochemical process that goes on inside the neuron than the perceptron, which is the basis for the modern day field of neural networks (Anderson and Rosenfeld, 1987) [3].

The electrochemical process of a neuron works like a voltage-to-frequency translator (Anderson and Rosenfeld, 1987) [3]. The inputs to the neuron cause a chemical reaction such that, when the chemicals build to a certain threshold, the neuron discharges. As higher inputs come into the neuron, the neuron then fires at a higher frequency, but the magnitude of the output from the neuron is the same. Figure 1.2 is a model of a neuron. A visual comparison of Figures 1.1 and 1.2 shows the origins of the idea of the perceptron can be traced back to the neuron. Externally, a perceptron seems to resemble the neuron with multiple inputs and a single output. However, this similarity does not really begin to model the complex electrochemical processes that actually go on inside a neuron. The perceptron is a very simple mathematical representation of the neuron.

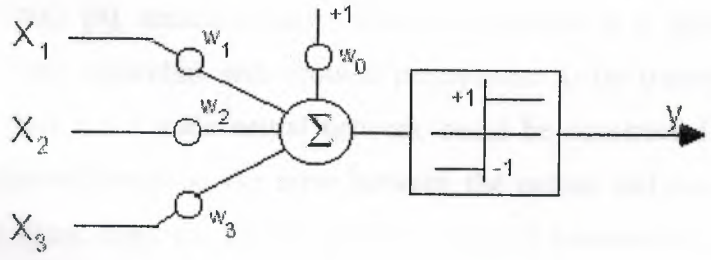


Figure 1.1 The Perceptron

Selfridges (1958) [4] brought the idea of the weight space to the perceptron. Rosenblatt adjusted the weights in a trial-and-error method. Selfridges adjusted the weights by randomly choosing a direction vector. If the performance did not improve, the weights were returned to their previous values, and a new random direction vector was

chosen. Selfridges referred to this process as climbing the mountain, as seen in Figure 1.3. Today, it is referred to as descending on the gradient because, generally, error squared, or the energy, is being minimized.

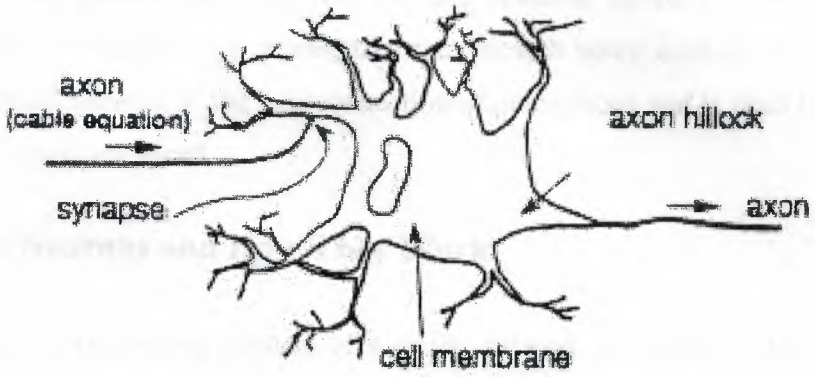


Figure 1.2 The Biological Neuron

After Perceptrons was published, research into neural networks went unfunded, and would remain so, until a method was developed to solve n-separable problems. Werbos (1974) [7] was first to develop the back propagation algorithm.

It was then independently rediscovered by Parker (1985) [8] and by Rumelhart and McClelland (1986) [9], simultaneously. Back propagation is a generalization of the Widrow-Hoff LMS algorithm and allowed perceptrons to be trained in a multilayer configuration, thus a n-1 node neural network could be constructed and trained. The weights are adjusted based on the error between the output and some known desired output. As the name suggests, the weights are adjusted backwards through the neural network, starting with the output layer and working through each hidden layer until the input layer is reached. The back propagation algorithm changes the schematic of the perceptron by using a sigmoidal function as the squashing function. Earlier versions of the perceptron used a signum function. The advantage of the sigmoidal function over the signum function is that the sigmoidal function is differentiable. This permits the back propagation algorithm to transfer the gradient information through the nonlinear squashing function, allowing the neural network to converge to a local minimum. Neuro

computing: Foundations of Research (Anderson and Rosenfeld, 1987) [3] is an excellent source of the work that was done before 1986. It is a collection of papers and gives an interesting overview of the events in the field of neural networks before 1986.

Although the golden age of neural network research ended 25 years ago, the discovery of back propagation has reenergized the research being done in this area. The feed-forward neural network is the interconnection of perceptrons and is used by the vast majority of the papers reviewed.

1.4 Artificial Neurons and How They Work

The fundamental processing element of a neural network is a neuron. This building block of human awareness encompasses a few general capabilities. Basically, a biological neuron receives inputs from other sources, combines them in some way, performs a generally nonlinear operation combines them in some way, performs a generally nonlinear operation on the result, and then outputs the final result. The figure shows the relationship of these four parts.

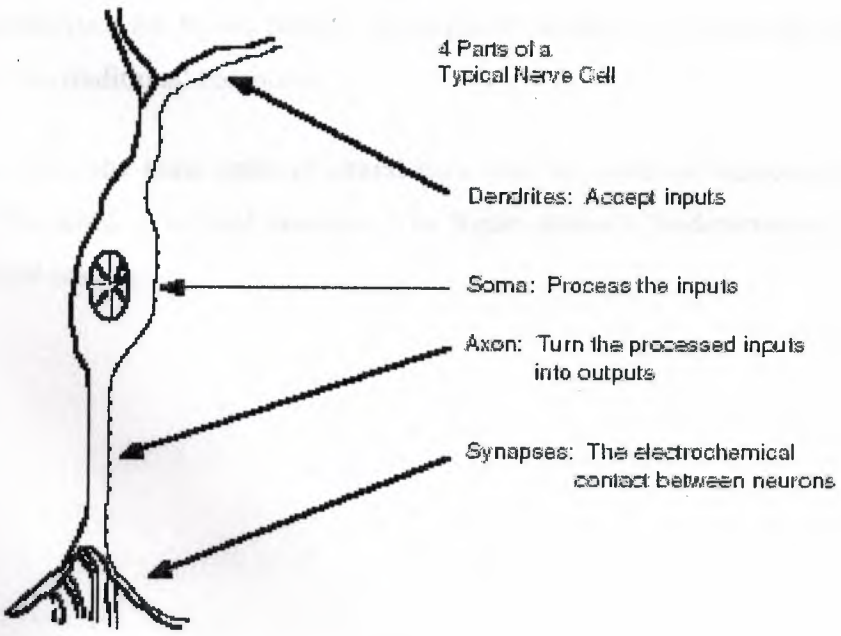


Figure 1.3 Simple Biological Neuron

Within humans there are many variations on this basic type of neuron, further complicating man's attempts at electrically replicating the process of thinking. Yet, all natural neurons have the same four basic components. These components are known by their biological names - dendrites, soma, axon, and synapses. Dendrites are hair-like extensions of the soma which act like input channels. These input channels receive their input through the synapses of other neurons. The soma then processes these incoming signals over time. The soma then turns that processed value into an output which is sent out to other neurons through the axon and the synapses.

Recent experimental data has provided further evidence that biological neurons are structurally more complex than the simplistic explanation above. They are significantly more complex than the existing artificial neurons that are built into today's artificial neural networks. As biology provides a better understanding of neurons, and as technology advances, network designers can continue to improve their systems by building upon man's understanding of the biological brain.

But currently, the goal of artificial neural networks is not the grandiose recreation of the brain. On the contrary, neural network researchers are seeking an understanding of nature's capabilities for which people can engineer solutions to problems that have not been solved by traditional computing.

To do this, the basic units of neural networks, the artificial neurons, simulate the four basic functions of natural neurons. The figure shows a fundamental representation of an artificial neuron.

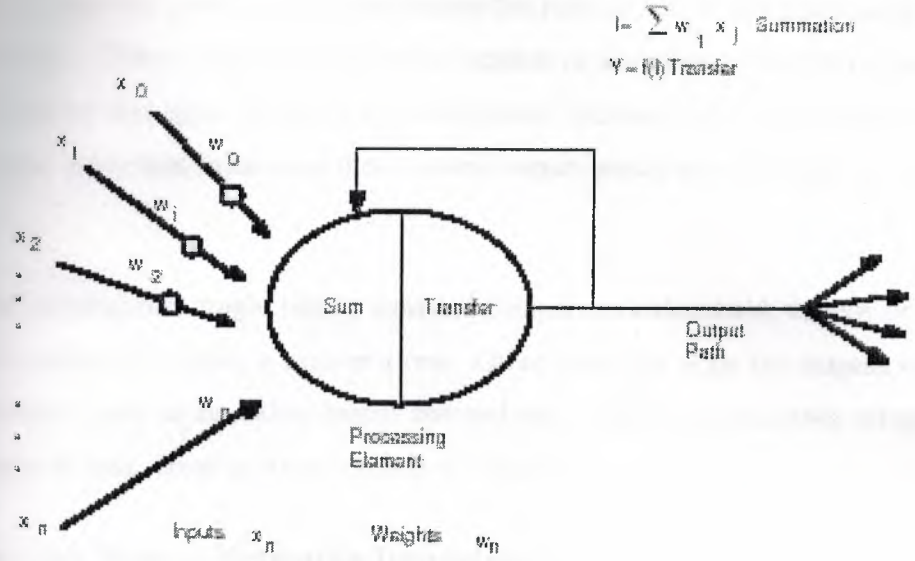


Figure 1.4 A Basic Artificial Neuron

In the figure above, various inputs to the network are represented by the mathematical symbol, x(n). Each of these inputs is multiplied by a connection weight. These weights are represented by w(n). In the simplest case, these products are simply summed, fed through a transfer function to generate a result, and then output. This process lends itself to physical implementation on a large scale in a small package. This electronic implementation is still possible with other network structures which utilize different summing functions as well as different transfer functions.

Some applications require "black and white," or binary, answers. These applications include the recognition of text, the identification of speech, and the image deciphering of scenes. These applications are required to turn real- world inputs into discrete values. These potential values are limited to some known set, like the ASCII characters or the most common 50,000 English words. Because of this limitation of output options, these applications don't always utilize networks composed of neurons that simply sum up, and thereby smooth, inputs. These networks may utilize the binary properties of ORing and ANDing of inputs. These functions, and many others, can be built into the summation and transfer functions of a network.

Other networks work on problems where the resolutions are not just one of several known values. These networks need to be capable of an infinite number of responses. Applications of this type include the "intelligence" behind robotic movements. This "intelligence" processes inputs and then creates outputs which actually cause some device to move.

Other applications might simply sum and compare to a threshold, thereby producing one of two possible outputs, a zero or a one. Other functions scale the outputs to match the application, such as the values minus one and one. Some functions even integrate the input data over time, creating time-dependent networks.

1.5 Why Are Neural Networks Important

Neural networks are responsible for the basic functions of our nervous system. They determine how we behave as an individual. Our emotions experienced as fear, anger, and what we enjoy in life come from neural networks in the brain. Even our ability to think and store memories depends on neural networks. Neural networks in the brain and spinal cord program all our movements including how fast we can type on a computer keyboard to how well we play sports. Our ability to see or hear is disturbed if something happens to the neural networks for vision or hearing in the brain.

Neural networks also control important functions of our bodies. Keeping a constant body temperature and blood pressure are examples where neural networks operate automatically to make our bodies work without us knowing what the networks are doing. These are called autonomic functions of neural networks because they are automatic and occur continuously without us being aware of them.

1.6 Electronic Implementation of Artificial Neurons

In currently available software packages these artificial neurons are called "processing elements" and have many more capabilities than the simple artificial neuron described above. Those capabilities will be discussed later in this report. The figure which shown below is a more detailed schematic of this still simplistic artificial neuron.

According to the inputs, inputs enter into the processing element from the upper left. The first step is for each of these inputs to be multiplied by their respective weighting factor (w(n)). Then these modified inputs are fed into the summing function, which usually just sums these products. Yet, many different types of operations can be selected. These operations could produce a number of different values which are then propagated forward; values such as the average, the largest, the smallest, the ORed values, the ANDed values, etc. Furthermore, most commercial development products allow software engineers to create their own summing functions via routines coded in a higher level language (C is commonly supported). Sometimes the summing function is further complicated by the addition of an activation function which enables the summing function to operate in a time sensitive way.

This sigmoid transfer function takes the value from the summation function, called sum in the figure below, and turns it into a value between zero and one.

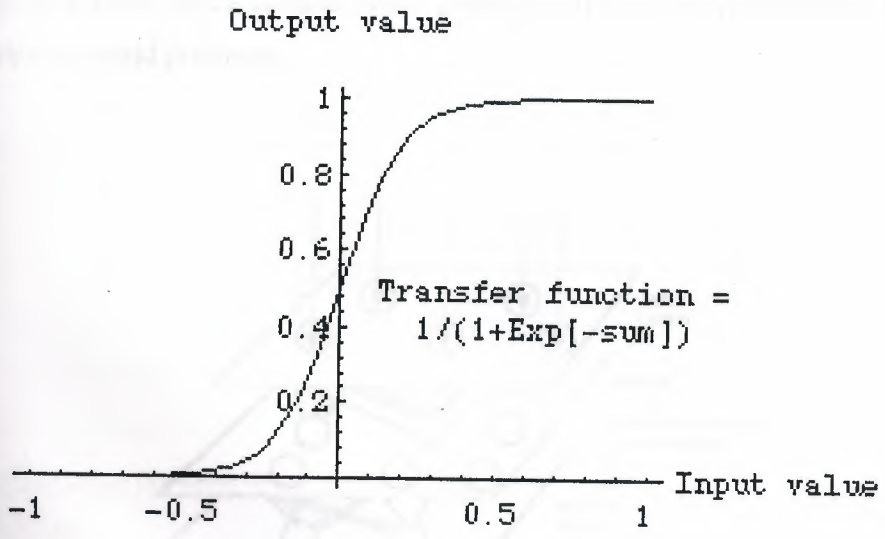


Figure 1.5 Sigmoid Transfer Function

Finally, the processing element is ready to output the result of its transfer function. This output is then input into other processing elements, or to an outside connection, as dictated by the structure of the network.

1.7 Artificial Network Operations

The other part of the "art" of using neural networks revolves around the myriad of ways these individual neurons can be clustered together. This clustering occurs in the human mind in such a way that information can be processed in a dynamic, interactive, and self-organizing way. Biologically, neural networks are constructed in a three-dimensional world from microscopic components. These neurons seem capable of nearly unrestricted interconnections. That is not true of any proposed, or existing, man-made network. Integrated circuits, using current technology, are two-dimensional devices with a limited number of layers for interconnection. This physical reality restrains the types, and scope, of artificial neural networks that can be implemented in silicon.

Currently, neural networks are the simple clustering of the primitive artificial neurons. This clustering occurs by creating layers which are then connected to one another. How these layers connect is the other part of the "art" of engineering networks to resolve real world problems.

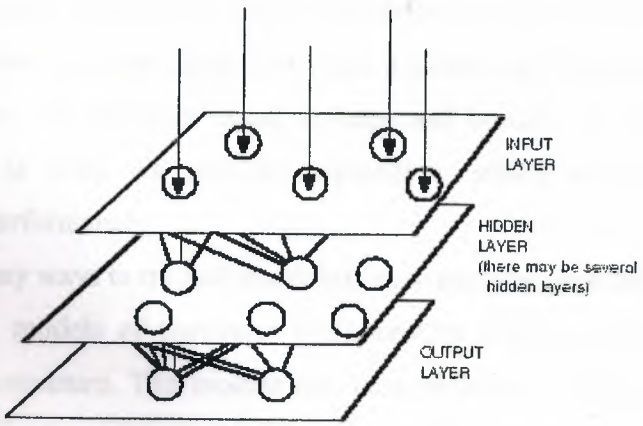


Figure 1.6 A Simple Neural Network Diagram

Basically, all artificial neural networks have a similar structure or topology as shown in the figure above. In that structure some of the neurons interface 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

1.8 The Future of Artificial Neural Networks

There is no doubt that neural networks are here to stay. There has been an intense amount of interest in them during recent years and as technology advances they will only become more valuable tools. There are a number of potential avenues that have, as yet, remained untapped, that will help to bring this technology to the forefront.

The first of these is the development of hardware acceleration for neural circuitry. It has been shown time and time again that when a technology begins to be supported by dedicated hardware that advances come in leaps and bounds. At present much of the work undertaken is done via software simulation, which obviously places severe restrictions upon performance.

There are many ways to try and attack this very complex problem, the first of which is to build neural models of particular brain centers without to much regard to the underlying neural structure. The models can then be tested against various behavioral paradigms like operant conditioning or classical conditioning. This technique is growing in its popularity with many neurophysiologists. An alternative approach is to attempt to

replicate as much of the neural substructure's complexity as possible. The problem with this approach is that you very rapidly run out of available processing power.

1.9 Summary

In this chapter we have demonstrated a basic introduction to neural networks. Within the chapter we have explained that neural networks are groups of select neurons that are connected with one another and they are functional circuits in the brain that process information and create useful activities by sending outputs to the body. As we have discussed in the sections, neural networks have had a unique history in the realm of technology and the earliest work in neural computing goes back to the 1940's when the first neural network computing model was developed. Also we have explained the definition of artificial neural networks as computing devices that are loosely based on the operation of the brain. Also we have considered the importance of neural networks, who should know about neural networks and their use. We have also explained where neural networks are being used giving some application of there use. Last but not least we have discussed the future of neural networks considering that there is a great deal of researches is going on in neural networks worldwide.

CHAPTER TWO

NEURAL NETWORKS ALGORITHMS

2.1 Overview

This Chapter presents a description of architectures and algorithms used to train neural networks. This chapter will explain the Model of a neuron and structures of neural networks including the single layer feed forward networks, multilayer feed forward networks, recurrent networks, and radial basis function networks. The sections below explains the artificial neural networks training and learning involved neural networks learning; supervised learning and unsupervised learning. Also this chapter discusses some advanced neural networks learning and problems using neural networks

2.2 Model of a Neuron

A *neuron* is an information-processing unit that is fundamental to the operation of a neural network. Figure 2.1 shows the model for a neuron. We may identify three basic elements of the neuron model, as described here:

1. A set of *synapses* or *connecting links*, each of which is characterized by a *weight* or *strength* of its own. Specially, a signal x_j at the input of synapse of j connected to neuron k is multiplied by the synaptic weight w_{kj}. It is important to make a note of the manner in which the subscripts of the synaptic weight w_{kj} are written. The first subscript refers to the neuron in question and the second subscript refers to the input end of the synapse to which the weight refers; the reverse of this notation is also used in the literature. The weight w_{kj} is positive if the associated synapse is excitatory; it is negative if the synapse is inhibitory.
2. An *adder* for summing the input signals, weighted by the respective synapses of the neuron; the operations described here constitutes *a liner combiner*.
3. An *activation function* for limiting the amplitude of the output of a neuron. The activation function is also referred to in the literature as a *squashing function* in that it

squashes (limits) the permissible amplitude range of the output signal to some finite value. Typically, the normalized amplitude range of the output of a neuron is written as the closed unit interval [0, 1] or alternatively [-1, 1].

The model of a neuron shown in Fig. 2.1 also includes an externally applied *threshold* θ_k that has the effect of lowering the net input of the activation function. On the other hand, the net input of the activation function may be increased by employing a *bias* term rather than a threshold; the bias is the negative of the threshold.

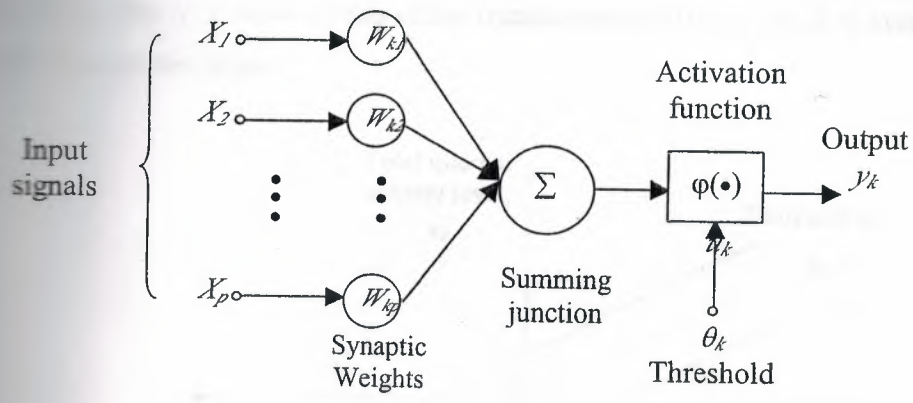


Figure 2.1 Nonlinear Model Of A Neuron.

In mathematical terms, we may describe a neuron k by writing the following pair of equations:

$$u_k = \sum_{j=1}^{p} w_{kj} x_j \qquad (2.1)$$

And

$$y_k = \varphi(u_k - \theta_k) \qquad (2.2)$$

Where $x_1, x_2, \ldots, x_p$ are the input signals; $w_{k1}, w_{k2}, \ldots, w_{kp}$ are the synaptic weights of neuron k; u_k is the *linear combiner output*, θ_k is the *threshold*, $\varphi(\cdot)$ is the *activation function*, and y_k is the output signal of the neuron. The use of threshold θ_k has the effect of

applying an *affine transformation* to the output u_k of the linear combiner in the model of Fig 2.2 as shown by

$$v_k = u_k - \theta_k \tag{2.3}$$

In particular, depending on whether the threshold θ_k is positive of negative, the relationship between the effective internal *activity level* or *activation potential* v_k of neuron k and the linear combiner output u_k is modified in the manner illustrated in Fig. 2.2. Note that as a result of this affine transformation, the graph of v_k versus u_k no longer pass through the origin.

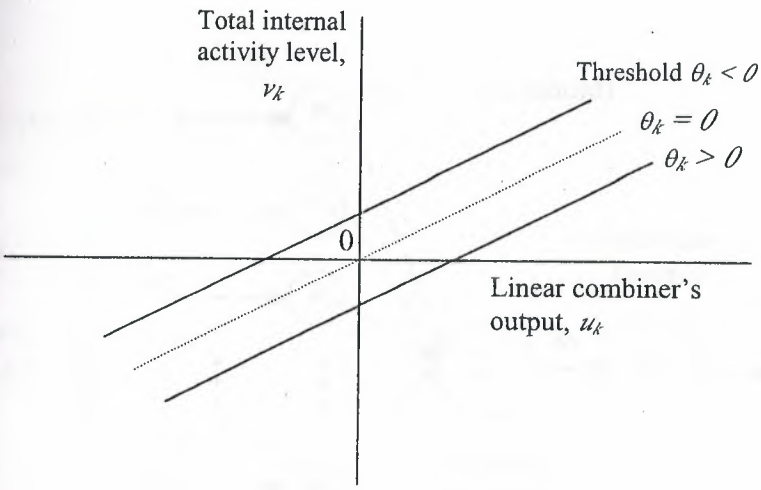


Figure 2.2. Affine Transformation Produced By The Presence Of A Threshold

The θ_k is an external parameter of artificial neuron k. We may account for its presence as in Eq. (2.2). Equivalently, we may formulate the combination of Eqs. (2.1) and (2.2) as follows:

$$v_k = \sum_{j=0}^{p} w_{kj} x_j \tag{2.4}$$

and

$$y_k = \varphi(v_k) \tag{2.5}$$

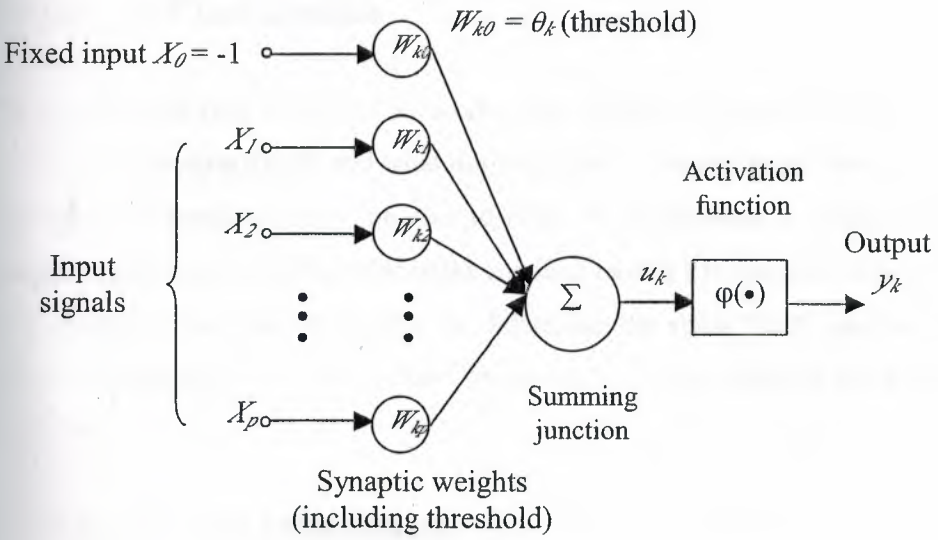
In Eq. (2.4) we have added a new synapse, whose input is

$$x_0 = -1 \tag{2.6}$$

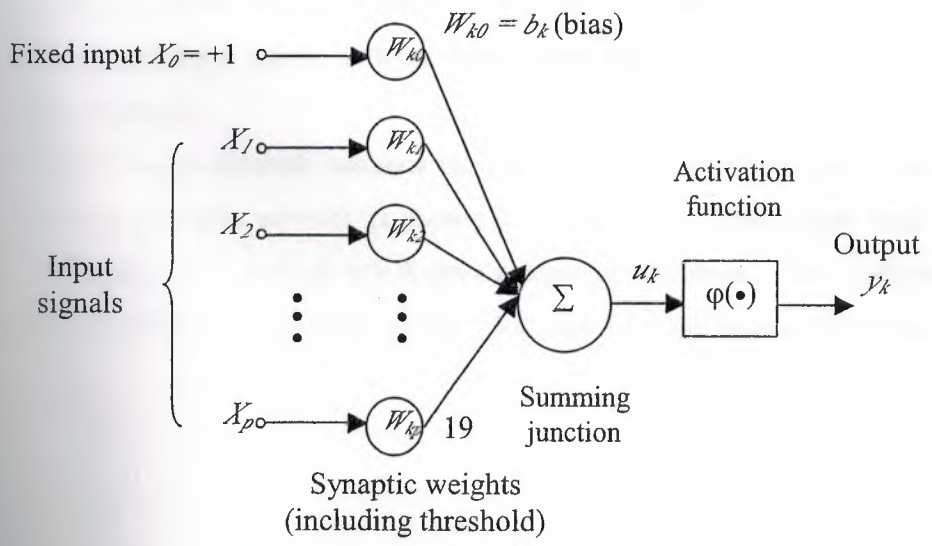
and whose weight is

$$w_{k0} = \theta_k \tag{2.7}$$

We may therefore reformulate the model of neuron k as in Fig. 2.3a. In this figure, the effect of the threshold is represented by doing two things: (1) adding a new input signal fixed at –1, and (2) adding a new synaptic weight equal to the threshold θ_k. alternatively, we may model the neuron as in Fig. 2.3b,



(a)



(b)

Figure 2.3. Two Other Nonlinear Models Of A Neuron

Where the combination of fixed input $x_0 = +1$ and weight $w_{k0} = b_k$ accounts for the *bias* b_k. Although the models in Fig. 2.1 and 2.3 are different in appearance, they are mathematically equivalent.

2.3 Network for Classification

The previous section describes networks that attempt to make projections of the future. But understanding trends and what impacts those trends might have is only one of several types of applications. The second class of applications is classification. A network that can classify could be used in the medical industry to process both lab results and doctor-recorded patience symptoms to determine the most likely disease. Other applications can separate the "tire kicker" inquiries from the requests for information from real buyers.

2.3.1. Learning Vector Quantization.

This network topology was originally suggested by Tuevo Kohonen in the mid 80's, well after his original work in self-organizing maps. Both this network and self-organizing maps are based on the Kohonen layer, which is Specifically, Learning Vector Quantization is a artificial neural network model used both for classification and image segmentation problems.

Topologically, the network contains an input layer, a single Kohonen layer and an output layer. An example network is shown in Figure 5.2.1. The output layer has as many processing elements as there are distinct categories or classes. The Kohonen layer has a number of processing elements grouped for each of these classes. The number of

processing elements per class depends upon the complexity of the input-output relationship. Usually, each class will have the same number of elements throughout the layer. It is the Kohonen layer that learns and performs relational classifications with the aid of a training set. This network uses supervised learning rules. However, these rules vary significantly from the back-propagation rules. To optimize the learning and recall functions, the input layer should contain only one processing element for each separable input parameter. Higher-order input structures could also be used.

2.4 Neural Network Structure

The manner in which the neurons of a neural network are structured is intimately linked with the learning algorithm used to train the network. We may therefore speak of learning algorithms (rules) used in the design of neural networks as being structured.
In general, we may identify four different classes of network architectures

2.4.1 Feed forward, Back-Propagation.

The feed forward, back-propagation architecture was developed in the early 1970's by several independent sources. This independent co-development was the result of a proliferation of articles and talks at various conferences which stimulated the entire industry. Currently, this synergistically developed back-propagation architecture is the most popular, effective, and easy to learn model for complex, multi-layered networks. This network is used more than all others combined. It is used in many different types of applications. This architecture has spawned a large class of network types with many different topologies and training methods. Its greatest strength is in non-linear solutions to ill- defined problems.

The typical back-propagation network has an input layer, an output layer, and at least one hidden layer. There is no theoretical limit on the number of hidden layers but typically there is just one or two. Some work has been done which indicates that a maximum of four layers (three hidden layers plus an output layer) are required to solve problems of any complexity. Each layer is fully connected to the succeeding layer, as shown in the figure. (Note: all of the drawings of networks in section 5 are from

NeuralWare's NeuralWorks Professional II/Plus artificial neural network development tool.)

The in and out layers indicate the flow of information during recall. Recall is the process of putting input data into a trained network and receiving the answer. Back-propagation is not used during recall, but only when the network is learning a training set.

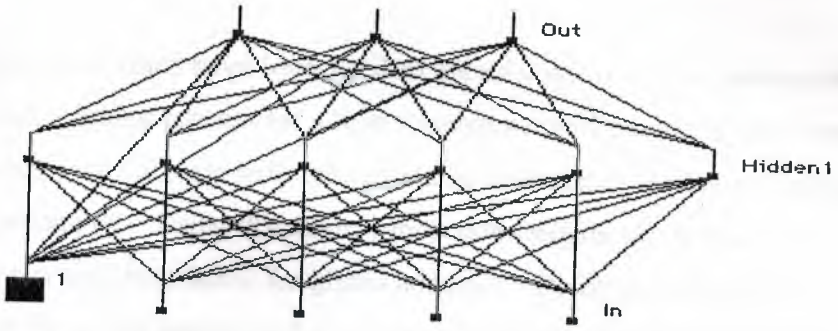


Figure 2.4 Example Feed Forward Back-propagation Network.

The number of layers and the number of processing elements per layer are important decisions. These parameters to a feed forward, back-propagation topology are also the most ethereal. They are the "art" of the network designer. There is no quantifiable, best answer to the layout of the network for any particular application. There are only general rules picked up over time and followed by most researchers and engineers applying this architecture to their problems.

Rule One: As the complexity in the relationship between the input data and the desired output increases, then the number of the processing elements in the hidden layer should also increase.

Rule Two: If the process being modeled is separable into multiple stages, then additional hidden layer(s) may be required. If the process is not separable into stages, then additional layers may simply enable memorization and not a true general solution.

Rule Three: The amount of training data available sets an upper bound for the number of processing elements in the hidden layer(s). To calculate this upper bound, use the number of input- output pair examples in the training set and divide that number by

the total number of input and output processing elements in the network. Then divide that result again by a scaling factor between five and ten. Larger scaling factors are used for relatively noisy data. Extremely noisy data may require a factor of twenty or even fifty, while very clean input data with an exact relationship to the output might drop the factor to around two. It is important that the hidden layers have few processing elements.

Too many artificial neurons and the training set will be memorized. If that happens then no generalization of the data trends will occur, making the network useless on new data sets.

Once the above rules have been used to create a network, the process of teaching begins. This teaching process for a feed forward network normally uses some variant of the Delta Rule, which starts with the calculated difference between the actual outputs and the desired outputs. Using this error, connection weights are increased in proportion to the error times a scaling factor for global accuracy. Doing this for an individual node means that the inputs, the output, and the desired output all have to be present at the same processing element. The complex part of this learning mechanism is for the system to determine which input contributed the most to an incorrect output and how does that element get changed to correct the error. An inactive node would not contribute to the error and would have no need to change its weights.

To solve this problem, training inputs are applied to the input layer of the network, and desired outputs are compared at the output layer. During the learning process, a forward sweep is made through the network, and the output of each element is computed layer by layer. The difference between the output of the final layer and the desired output is back-propagated to the previous layer(s), usually modified by the derivative of the transfer function, and the connection weights are normally adjusted using the Delta Rule. This process proceeds for the previous layer(s) until the input layer is reached.

2.4.2 Higher Order Neural Network or Functional Link Network

Either name is given to neural networks which expand the standard feedforward, back-propagation architecture to include nodes at the input layer which provide the network with a more complete understanding of the input. Basically, the inputs are

transformed in a well understood mathematical way so that the network does not have to learn some basic math functions. These functions do enhance the network's understanding of a given problem. These mathematical functions transform the inputs via higher-order functions such as squares, cubes, or sine's. It is from the very name of these functions, higher-order or functionally linked mappings, that the two names for this same concept were derived.

This technique has been shown to dramatically improve the learning rates of some applications. An additional advantage to this extension of back- propagation is that these higher order functions can be applied to other derivations - delta bar delta, extended delta bar delta, or any other enhanced feed forward, back-propagation networks.

There are two basic ways of adding additional input nodes. First, the cross-products of the input terms can be added into the model. This is also called the output product or tensor model, where each component of the input pattern multiplies the entire input pattern vector. A reasonable way to do this is to add all interaction terms between input values. For example, for a back-propagation network with three inputs (A, B and C), the cross-products would include: AA, BB, CC, AB, AC, and BC. This example adds second-order terms to the input structure of the network. Third-order terms, such as ABC, could also be added.

2.4.3 Training an Artificial Neural Network

Once a network has been structured for a particular application, that network is ready to be trained. To start this process the initial weights are chosen randomly. Then, the training, or learning, begins.

There are two approaches to training - supervised and unsupervised. Supervised training involves a mechanism of providing the network with the desired output either by manually "grading" the network's performance or by providing the desired outputs with the inputs. Unsupervised training is where the network has to make sense of the inputs without outside help.

2.4.3.1 Supervised Training.

In supervised training, both the inputs and the outputs are provided. The network then processes the inputs and compares its resulting outputs against the desired outputs. Errors are then propagated back through the system, causing the system to adjust the weights which control the network this process occurs over and over as the weights are continually tweaked. Training sets need to be fairly large to contain all the needed information if the network is to learn the features and relationships that are important. Not only do the sets have to be large but the training sessions must include a wide variety of data. If the network is trained just one example at a time, all the weights set so meticulously for one fact could be drastically altered in learning the next fact. The previous facts could be forgotten in learning something new. As a result, the system has to learn everything together, finding the best weight settings for the total set of facts. For example, in teaching a system to recognize pixel patterns for the ten digits, if there were twenty examples of each digit, all the examples of the digit seven should not be presented at the same time.

If a network simply can't solve the problem, the designer then has to If a network simply can't solve the problem, the designer then has to per layer, the connections between the layers, the summation, transfer, and training functions, and even the initial weights themselves. Those changes required to create a successful network constitute a process wherein the "art" of neural networking occurs.

2.4.3.2 Unsupervised or Adaptive Training.

The other type of training is called unsupervised training. In unsupervised training, the network is provided with inputs but not with desired outputs. The system itself must then decide what features it will use to group the input data. This is often referred to as self-organization or adoption.

At the present time, unsupervised learning is not well understood. This adaptation to the environment is the promise which would enable science fiction types of robots to continually learn on their own as they encounter new situations and new environments. Life is filled with situations where exact training sets do not exist. Some of these situations involve military action where new combat techniques and new weapons might

be encountered. Because of this unexpected aspect to life and the human desire to be prepared, there continues to be research into, and hope for, this field. Yet, at the present time, the vast bulk of neural network work is in systems with supervised learning. Supervised learning is achieving results.

Unsupervised learning is the great promise of the future. It shouts that computers could someday learn on their own in a true robotic sense. Currently, this learning method is limited to networks known as self-organizing maps. These kinds of networks are not in widespread use. They are basically an academic novelty. Yet, they have shown they can provide a solution in a few instances, proving that their promise is not groundless. They have been proven to be more effective than many algorithmic techniques for numerical aerodynamic flow calculations. They are also being used in the lab where they are split into a front-end network that recognizes short, phoneme-like fragments of speech which are then passed on to a backend network. The second artificial network recognizes these strings of fragments as words.

This promising field of unsupervised learning is sometimes called self-supervised learning. These networks use no external influences to adjust their weights. Instead, they internally monitor their performance. These networks look for regularities or trends in the input signals, and makes adaptations according to the function of the network. Even without being told whether it's right or wrong, the network still must have some information about how to organize itself. This information is built into the network topology and learning rules.

2.4.3.3 Learning Rates

The rate at which ANNs learn depends upon several controllable factors. In selecting the approach there are many trade-offs to consider. Obviously, a slower rate means a lot more time is spent in accomplishing the off-line learning to produce an adequately trained system. With the faster learning rates, however, the network may not be able to make the fine discriminations possible with a system that learns more slowly. Researchers are working on producing the best of both worlds.

Generally, several factors besides time have to be considered when discussing the off-line training task, which is often described as "tiresome." Network complexity, size, paradigm selection, architecture, type of learning rule or rules employed, and desired accuracy must all be considered. These factors play a significant role in determining how long it will take to train a network. Changing any one of these factors may either extend the training time to an unreasonable length or even result in an unacceptable accuracy.

Most learning functions have some provision for a learning rate, or learning constant. Usually this term is positive and between zero and one. If the learning rate is greater than one, it is easy for the learning algorithm to overshoot in correcting the weights, and the network will oscillate. Small values of the learning rate will not correct the current error as quickly, but if small steps are taken in correcting errors, there is a good chance of arriving at the best minimum convergence.

2.4.3.4 Learning Laws

Many learning laws are in common use. Most of these laws are some sort of variation of the best known and oldest learning law, Hebb's Rule. Research into different learning functions continues as new ideas routinely show up in trade publications. Some researchers have the modeling of biological learning as their main objective. Others are experimenting with adaptations of their perceptions of how nature handles learning. Either way, man's understanding of how neural processing actually works is very limited. Learning is certainly more complex than the simplifications represented by the learning laws currently developed. A few of the major laws are presented as examples.

Hebb's Rule: The first, and undoubtedly the best known, learning rule as introduced by Donald Hebb. The description appeared in his book *The Organization of Behavior* in 1949 [14]. His basic rule is: If a neuron receives an input from another neuron, and if both are highly active (mathematically have the same sign), the weight between the neurons should be strengthened.

Hopfield Law: It is similar to Hebb's rule with the exception that it specifies the magnitude of the strengthening or weakening. It states, "If the desired output and the

input are both active and both inactive, increment the connection weight by the learning rate, otherwise decrement the weight by the learning rate." [15].

The Delta Rule: This rule is a further variation of Hebb's Rule. It is one of the most commonly used. This rule is based on the simple idea of continuously modifying the strengths of the input connections to reduce the difference (the delta) between the desired output value and the actual output of a processing element. This rule changes the synaptic weights in the way that minimizes the mean squared error of the network. This rule is also referred to as the Widrow-Hoff Learning Rule and the Least Mean Square (LMS) Learning Rule.

The way that the Delta Rule works is that the delta error in the output layer is transformed by the derivative of the transfer function and is then used in the previous neural layer to adjust input connection weights. In other words, this error is back-propagated into previous layers one layer at a time. The process of back-propagating the network errors continues until the first layer is reached. The network type called Feedforward, Back-propagation derives its name from this method of computing the error term.

When using the delta rule, it is important to ensure that the input data set is well randomized. Well ordered or structured presentation of the training set can lead to a network which can not converge to the desired accuracy. If that happens, then the network is incapable of learning the problem.

The Gradient Descent Rule: This rule is similar to the Delta Rule in that the derivative of the transfer function is still used to modify the delta error before it is applied to the connection weights. Here, however, an additional proportional constant tied to the learning rate is appended to the final modifying factor acting upon the weight. This rule is commonly used, even though it converges to a point of stability very slowly. It has been shown that different learning rates for different layers of a network help the learning process converge faster. In these tests, the learning rates for those layers close to the output were set lower than those layers near the input. This is especially important for applications where the input data is not derived from a strong underlying model.

Kohonen's Learning Law: This procedure, developed by Teuvo Kohonen, was inspired by learning in biological systems. In this procedure, the processing elements

compete for the opportunity to learn, or update their weights. The processing element with the largest output is declared the winner and has the capability of inhibiting its competitors as well as exciting its neighbors. Only the winner is permitted an output, and only the winner plus its neighbors are allowed to adjust their connection weights.

Further, the size of the neighborhood can vary during the training period. The usual paradigm is to start with a larger definition of the neighborhood, and narrow in as the training process proceeds. Because the winning element is defined as the one that has the closest match to the input pattern, Kohonen networks model the distribution of the inputs. This is good for statistical or topological modeling of the data and is sometimes referred to as self-organizing maps or self-organizing topologies.

2.5 Advanced Neural Networks

Many advanced algorithms have been invented since the first simple neural network. Some algorithms are based on the same assumptions or learning techniques as the SLP and the MLP. A very different approach however was taken by Kohonen, in his research in self-organizing networks.

2.5.1 Kohonen Self-Organizing Networks

The Kohonen self-organizing networks have a two-layer topology. The first layer is the input layer; the second layer is itself a network in a plane. Every unit in the input layer is connected to all the nodes in the grid in the second layer. Furthermore the units in the grid function as the output nodes.

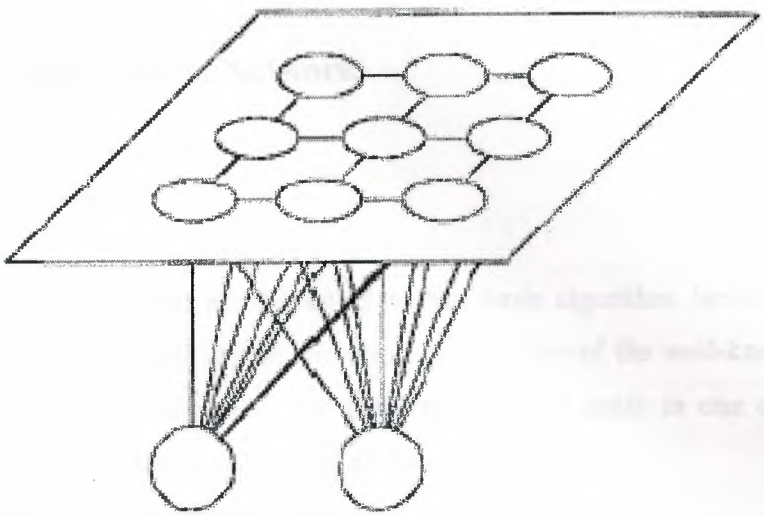


Figure 2.5 The Kohonen Topology

The nodes in the grid are only sparsely connected. Here each node has four immediate neighbors.

2.5.2 Algorithm

The network (the units in the grid) is initialized with small random values. A neighborhood radius is set to a large value. The input is presented and the Euclidean distance between the input and each output node is calculated. The node with the minimum distance is selected, and this node, together with its neighbors within the neighborhood radius, will have their weights modified to increase similarity to the input. The neighborhood radius decreases over time to let areas of the network be specialized to a pattern.

The algorithm results in a network where groups of nodes respond to each class thus creating a map of the found classes.

The big difference in the learning algorithm, compared with the MLP, is that the Kohonen self-organizing net uses unsupervised learning. But after the learning period when the network has mapped the test patterns, it is the operator's responsibility to label the different patterns accordingly.

2.6 Problem using neural Network

2.6.1 Local Minimum

All the NN in this paper are described in their basic algorithm. Several suggestions for improvements and modifications have been made. One of the well-known problems in the MLP is the *local minimum.* The network does not settle in one of the learned minima but instead in a local minimum in the Energy landscape

Approaches to avoid local minimum:

- The *gain term* in the weight adaption function can be lowered progressively as the network iterates. This would at first let the differences in weights and energy be large, and then hopefully when the network is approaching the right solution, the steps would be smaller. The tradeoff is when the gain term has decreased the network will take a longer time to converge to right solution.
- A local minimum can be caused by a bad internal representation of the patterns. This can be aided by the adding more internal nodes to the network.
- An extra term can be added to the weight adaption: the *Momentum term.* The Momentum term should let the weight change be large if the current change in energy is large.
- The network gradient descent can be disrupted by adding random noise to ensure sure the system will take unequal steps toward the solution. This solution has the advantage that it requires no extra computation time.

A similar problem is known in the Hopfield Network as *metastable states.* That is when the network settles in a state that is not represented in the stored patterns. One way to minimize this is by adjusting the number of nodes in the network (N) to the number of patterns to store, so that the number of patterns does not exceed 0.15N. Another solution is to add a probabilistic update rule to the Hopfield network. This is known as the Boltzman machine.

2.6.2 Practical Problems

There are some practical problems applying neural networks to applications. It is not possible to know in advance the ideal network for an application. So every time a NN is to be built in an application, it requires tests and experiments with different network settings or topologies to find a solution that performs well on the given application. This is a problem because most NN requires a long training period – many iterations of the same pattern set. And even after much iteration there is no way other that testing to see whether the network is efficiently mapping the training sets. A solution for this might be to adapt newer NN technologies such as the bump tree which need only one run through the training set to adjust all weights in the network. The most commonly used network still seems to be the MLP and the RBF[3] even though alternatives exist that can drastically shorten processing time.

In general most NN include complex computation, which is time consuming. Some of these computations could gain efficiency if they were to be implemented on a parallel processing system, but the hardware implementation raises new problems of physical limits and the NN need for changeability

2.7 Summary

As we have discussed neural networks classified according to their learning processes into two types, supervised learning and unsupervised learning. Also this chapter discussed the various types of neural networks structures and algorithms. The most commonly used neural network configurations known as multilayer perceptron (MLP) are described. Other structures discussed in this chapter include recurrent (feedback) neural networks and radial basis function (RBF) network. Also we have explained a brief description of Kohenon self-organizing networks and Hopfield networks. Finally we have discussed the problems using neural networks including the local minimum and practical problems

CHAPTER THREE

NEURAL NETWORKS APPLICATIONS

3.1 Overview

This chapter presents a brief description of some artificial neural networks applications. The section below provides an understanding of how neural networks are currently being used and the researches area in artificial neural networks. The applications that artificial neural networks cover in this chapter such as language processing, character recognition, servo control and pattern recognition are described briefly. Also there will be a sufficient description about neural networks applications in image compression, and some applications area in medicine and business, also the applications in arts and telecommunications. Last section presents a determination if an application is a neural network candidate and how to determine it.

3.2 How Artificial Neural Network Are Being Used

Artificial neural networks are undergoing the change that occurs when a concept leaves the academic environment and is thrown into the harsher world of users who simply want to get a job done. Many of the networks now being designed are statistically quite accurate but they still leave a bad taste with users who expect computers to solve their problems absolutely. These networks might be 85% to 90% accurate. Unfortunately, few applications tolerate that level of error.

While researchers continue to work on improving the accuracy of their "creations", some explorers are finding uses for the current technology.

In reviewing this state of the art, it is hard not to be overcome by the bright promises or tainted by the unachieved realities. Currently, neural networks are not the user interface which translates spoken works into instructions for a machine, but some day they will. Someday, VCRs, home security systems, CD players, and word processors

will simply be activated by voice. Touch screen and voice editing will replace the word processors of today while bringing spreadsheets and data bases to a level of usability pleasing to most everyone. But for now, neural networks are simply entering the marketplace in niches where their statistical accuracy is valuable as they await what will surely come.

Many of these niches indeed involve applications where answers are nebulous. Loan approval is one. Financial institutions make more money by having the lowest bad loan rate they can achieve. Systems that are "90% accurate" might be an improvement over the current selection process. Indeed, some banks have proven that the failure rate on loans approved by neural networks is lower than those approved by some of their best traditional methods. Also, some credit card companies are using neural networks in their application screening process.

This newest method of seeking the future by analyzing past experiences has generated its own unique problems. One of those problems is to provide a reason behind the computer-generated answer, say as to why a particular loan application was denied. As mentioned throughout this report, the inner workings of neural networks are "black boxes." Some people have even called the use of neural networks "voodoo engineering." To explain how a network learned and why it recommends a particular decision has been difficult. To facilitate this process of justification, several neural network tool makers have provided programs which explain which input through which node dominates the decision making process. From that information, experts in the application should be able to infer the reason that a particular piece of data is important.

Besides this filling of niches, neural network work is progressing in other more promising application areas. The next section of this chapter goes through some of these areas and briefly details the current work. This is done to help stimulate within the reader the various possibilities where neural networks might offer solutions, possibilities such as language processing, character recognition, image compression, pattern recognition among others.

3.3 Language Processing

Language processing encompasses a wide variety of applications. These applications include text-to-speech conversion, auditory input for machines, automatic language translation, secure voice keyed locks, automatic transcription, aids for the deaf, aids for the physically disabled which respond to voice commands, and natural language processing.

Many companies and universities are researching how a computer, via ANNs, could be programmed to respond to spoken commands. The potential economic rewards are a proverbial gold mine. If this capability could be shrunk to a chip, that chip could become part of almost any electronic device sold today. Literally hundreds of millions of these chips could be sold.

This magic-like capability needs to be able to understand the 50,000 most commonly spoken words. Currently, according to the academic journals, most of the hearing-capable neural networks are trained to only one talker. These one-talker, isolated-word recognizers can recognize a few hundred words. Within the context of speech, with pauses between each word, they can recognize up to 20,000 words.

Some researchers are touting even greater capabilities, but due to the potential reward the true progress and methods involved, are being closely held. The most highly touted, and demonstrated, speech-parsing system comes from the Apple Corporation. This network, according to an April 1992 Wall Street Journal article, can recognize most any person's speech through a limited vocabulary.

This works continues in Corporate America (particularly venture capital land), in the universities, and in Japan.

3.4. Character Recognition

Character recognition is another area in which neural networks are providing solutions. Some of these solutions are beyond simply academic curiosities. HNC Inc., according to a HNC spokesman, markets a neural network based product that can recognize hand printed characters through a scanner. This product can take cards, like a credit card application form, and put those recognized characters into a data base. This product has been out for two and a half years. It is 98% to 99% accurate for numbers, a little less for alphabetical characters. Currently, the system is built to highlight characters below a certain percent probability of being right so that a user can manually fill in what the computer could not. This product is in use by banks, financial institutions, and credit card companies.

Electronic Engineering Times has also proved capable of recognizing characters, including cursive. This capability utilizes Odin's proprietary Quantum Neural Network software package called, QNspec. It has proven uncannily successful in analyzing reasonably good handwriting. It actually benefits from the cursive stroking.

The largest amount of research in the field of character recognition is aimed at scanning oriental characters into a computer. Currently, these characters require four or five keystrokes each. This complicated process elongates the task of keying a page of text into hours of drudgery. Several vendors are saying they are close to commercial products that can scan pages.

3.5 Image Compression

A number of studies have been done proving that neural networks can do real-time compression and decompression of data. These networks are auto associative in that they can reduce eight bits of data to three and then reverse that process upon restructuring to eight bits again. However, they are not lossless. Because of this losing of bits they do not favorably compete with more traditional methods.

Computer images are extremely data intensive and hence require large amounts of memory for storage. As a result, the transmission of an image from one machine to

another can be very time consuming. By using data compression techniques, it is possible to remove some of the redundant information contained in images, requiring less storage space and less time to transmit. Neural networks can be used for the purpose of image compression.

Neural network architecture suitable for solving the image compression problem is shown below. This type of structure--a large input layer feeding into a small hidden layer, which then feeds into a large output layer, is referred to as a bottleneck type network. The idea is this: suppose that the neural net shown below had been trained to implement the identity map. Then, a tiny image presented to the network as input would appear exactly the same at the output layer.

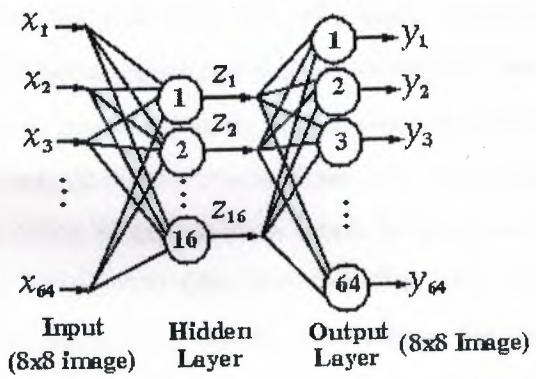


Figure 3.1 Bottleneck-type Neural Net Architecture For Image Compression

In this case, the network could be used for image compression by breaking it in two as shown in the Figure below. The transmitter encodes and then transmits the output of the hidden layer (only 16 values as compared to the 64 values of the original image).The receiver receives and decodes the 16 hidden outputs and generates the 64 outputs. Since the network is implementing an identity map, the output at the receiver is an exact reconstruction of the original image.

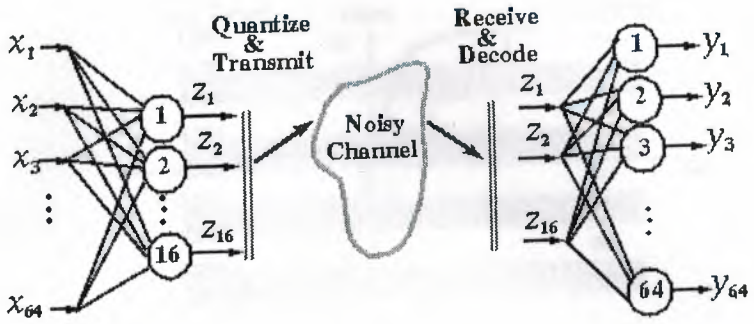


Figure 3.2 The Image Compression Scheme Using The Trained Neural Net

Actually, even though the bottleneck takes us from 64 nodes down to 16 nodes, no real compression has occurred because unlike the 64 original inputs which are 8-bit pixel values, the outputs of the hidden layer are real-valued (between -1 and 1), which requires possibly an infinite number of bits to transmit. True image compression occurs when the hidden layer outputs are quantized before transmission. The Figure below shows a typical quantization scheme using 3 bits to encode each input. In this case, there are 8 possible binary codes which may be formed: 000, 001, 010, 011, 100, 101, 110, 111. Each of these codes represents a range of values for a hidden unit output. For example, consider the first hidden output. When the value of is between -1.0 and -0.75, then the code 000 is transmitted; when is between 0.25 and 0.5, then 101 is transmitted. To compute the amount of image compression (measured in bits-per-pixel) for this level of quantization, we compute the ratio of the total number of bits transmitted: to the total number of pixels in the original image: 64; so in this case, the compression rate is given as bits/pixel. Using 8 bit quantization of the hidden units gives a compression rate of bits/pixel.

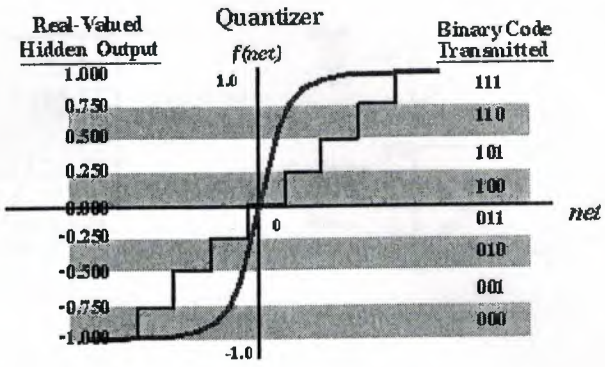


Figure 3.3 The Quantization Of Hidden Unit Outputs

The training of the neural net proceeds as follows, a 256x256 training image is used to train the bottleneck type network to learn the required identity map. Training input-output pairs are produced from the training image by extracting small 8x8 chunks of the image chosen at a uniformly random location in the image. The easiest way to extract such a random chunk i s to generate a pair of random integers to serve as the upper left hand corner of the extracted chunk. In this case, we choose random integers i and j, each between 0 and 248, and then (i,j) is the coordinate of the upper left hand corner of the extracted chunk. The pixel values of the extracted image chunk are sent (left to right, top to bottom) through the pixel-to-real mapping shown in the Figure below to construct the 64-dimensional neural net input. Since the goal is to learn the identity map, the desired target for the constructed input is itself; hence, the training pair is used to update the weights of the network.

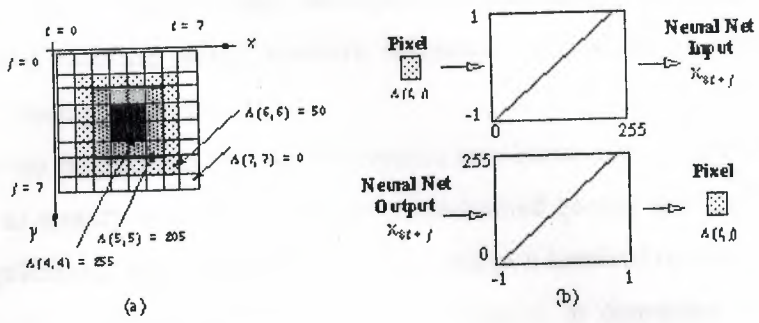


Figure 3.4 The Pixel-to-Real and Real-to-Pixel Conversions

Once training is complete, image compression is demonstrated in the recall phase. In this case, we still present the neural net with 8x8 chunks of the image, but now instead of randomly selecting the location of each chunk, we select the chunks in sequence from left to right and from top to bottom. For each such 8x8 chunk, the output the network can be computed and displayed on the screen to visually observe the performance of neural net image compression. In addition, the 16 outputs of the hidden layer can be grouped into a 4x4 "compressed image", which can be displayed as well.

3.6 Pattern Recognition

Recently, a number of pattern recognition applications have been written about in the general press. The Wall Street Journal has featured a system that can detect bombs in luggage at airports by identifying, from small variances, patterns from within specialized sensor's outputs. Another article reported on how a physician had trained a back-propagation neural network on data collected in emergency rooms from people who felt that they were experiencing a heart attack to provide a probability of a real heart attack versus a false alarm. His system is touted as being a very good discriminator in an arena where priority decisions have to be made all the time.

Another application involves the grading of rare coins. Digitized images from an electronic camera are fed into a neural network. These images include several angles of the front and back. These images are then compared against known patterns which represent the various grades for a coin. This system has enabled a quick evaluation for

about $15 as opposed to the standard three-person evaluation which costs $200. The results have shown that the neural network recommendations are as accurate as the people-intensive grading method.

Yet, by far the biggest use of neural networks as a recognizer of patterns is within the field known as quality control. A number of automated quality applications are now in use. These applications are designed to find that one in a hundred or one in a thousand part that is defective. Human inspectors become fatigued or distracted. Systems now evaluate solder joints, welds, cuttings, and glue applications. One car manufacturer is now even prototyping a system which evaluates the color of paints. This system digitizes pictures of new batches of paint to determine if they are the right shades.

Another major area where neural networks are being built into pattern recognition systems is as processors for sensors. Sensors can provide so much data that the few meaningful pieces of information can become lost. People can lose interest as they stare at screens looking for "the needle in the haystack." Many of these sensor-processing applications exist within the defense industry. These neural network systems have been shown successful at recognizing targets. These sensor processors take data from cameras, sonar systems, seismic recorders, and infrared sensors. That data is then used to identify probable phenomenon.

Another field related to defense sensor processing is the recognition of patterns within the sensor data of the medical industry. A neural network is now being used in the scanning of PAP smears. This network is trying to do a better job at reading the smears than can the average lab technician. A missed diagnosis is a too common problem throughout this industry. In many cases, a professional must perceive patterns from noise, such as identifying a fracture from an X-ray or cancer from a X-ray "shadow." Neural networks promise, particularly when faster hardware becomes available, help in many areas of the medical profession where data is hard to read.

3.7 Signal Processing

Neural networks' promise for signal processing has resulted in a number of experiments in various university labs. Neural networks have proven capable of filtering

out noise. Widrow's MADALINE was the first network applied to a real-world problem. It eliminates noise from phone lines.

Another application is a system that can detect engine misfire simply from the noise. This system, developed by Odin Corp, works on engines up to 10,000 RPMS. The Odin system satisfies the California Air Resources Board's mandate that by 1994 new automobiles will have to detect misfire in real time. Misfires are suspected of being a leading cause of pollution. The Odin solution requires 3 kbytes of software running on a Motorola 68030 microprocessor.

3.8 Financial

Neural networks are making big inroads into the financial worlds. Banking, credit card companies, and lending institutions deal with decisions that are not clear cut. They involve learning and statistical trends.

The loan approval process involves filling out forms which hopefully can enable a loan officer to make a decision. The data from these forms is now being used by neural networks which have been trained on the data from past decisions. Indeed, to meet government requirements as to why applications are being denied, these packages are providing information on what input, or combination of inputs, weighed heaviest on the decision.

Credit card companies are also using similar back-propagation networks to aid in establishing credit risks and credit limits.

In the world of direct marketing, neural networks are being applied to data bases so that these phone peddlers can achieve higher ordering rates from those annoying calls that most of us receive at dinner time. (A probably more lucrative business opportunity awaits the person who can devise a system which will tailor all of the data bases in the world so that certain phone numbers are never selected).

Neural networks are being used in all of the financial markets - stock, bonds, international currency, and commodities. Some users are cackling that these systems just make them "see green," money that is. Indeed, neural networks are reported to be highly successful in the Japanese financial markets. Daiichi Kangyo Bank has reported that for

government bond transactions, neural networks have boosted their hit rate from 60% to 75%. Daiwa research Institute has reported a neural net system which has scored 20% better than the Nikkei average. Daiwa Securities' stock prediction system has boosted the companies hit rate from 70% to 80%.

3.9 Neural Network in Medicine

Artificial Neural Networks are currently a 'hot' research area in medicine and it is believed that they will receive extensive application to biomedical systems in the next few years. At the moment, the research is mostly on modeling parts of the human body and recognizing diseases from various scans (e.g. cardiograms, CAT scans, ultrasonic scans, etc.).

Neural networks are ideal in recognizing diseases using scans since there is no need to provide a specific algorithm on how to identify the disease. Neural networks learn by example so the details of how to recognize the disease are not needed. What is needed is a set of examples that are representative of all the variations of the disease. The quantity of examples is not as important as the 'quantity'. The examples need to be selected very carefully if the system is to perform reliably and efficiently.

3.10 Electronic Noses

The two main components of an electronic nose are the sensing system and the automated pattern recognition system. The sensing system can be an array of several different sensing elements (e.g., chemical sensors), where each element measures a different property of the sensed chemical, or it can be a single sensing device (e.g., spectrometer) that produces an array of measurements for each chemical, or it can be a combination. Each chemical vapor presented to the sensor array produces a signature or pattern characteristic of the vapor. By presenting many different chemicals to the sensor array, a database of signatures is built up. This database of labeled signatures is used to train the pattern recognition system. The goal of this training process is to configure the

recognition system to produce unique classifications of each chemical so that an automated identification can be implemented.

The quantity and complexity of the data collected by sensors array can make conventional chemical analysis of data in an automated fashion difficult. One approach to chemical vapor identification is to build an array of sensors, where each sensor in the array is designed to respond to a specific chemical. With this approach, the number of unique sensors must be at least as great as the number of chemicals being monitored. It is both expensive and difficult to build highly selective chemical sensors.

Artificial neural networks (ANNs), which have been used to analyze complex data and to recognize patterns, are showing promising results in chemical vapor recognition. When an ANN is combined with a sensor array, the number of detectable chemicals is generally greater than the number of sensors [22]. Also, less selective sensors which are generally less expensive can be used with this approach. Once the ANN is trained for chemical vapor recognition, operation consists of propagating the sensor data through the network. Since this is simply a series of vector-matrix multiplications, unknown chemicals can be rapidly identified in the field.

Electronic noses that incorporate ANNs have been demonstrated in various applications. Some of these applications will be discussed later in the paper. Many ANN configurations and training algorithms have been used to build electronic noses including back propagation-trained, feed-forward networks; fuzzy ARTmaps; Kohonen's self-organizing maps (SOMs); learning vector quantizes (LVQs); Hamming networks; Boltzmann machines; and Hopfield networks. Figure 3.5 illustrates the basic schematic of an electronic nose.

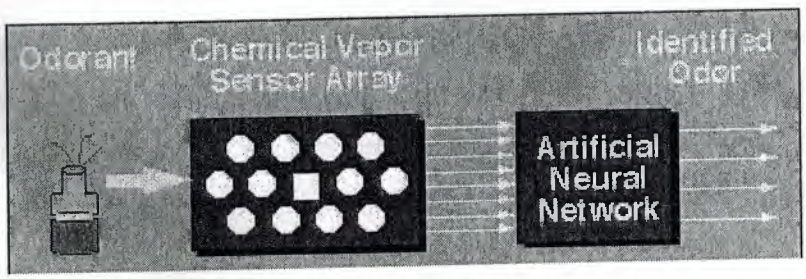


Figure 3.5 Schematic Diagram Of an Electronic Nose

Because the sense of smell is an important sense to the physician, an electronic nose has applicability as a diagnostic tool. An electronic nose can examine odors from the body (e.g., breath, wounds, body fluids, etc.) and identify possible problems. Odors in the breath can be indicative of gastrointestinal problems, sinus problems, infections, diabetes, and liver problems. Infected wounds and tissues emit distinctive odors that can be detected by an electronic nose. Odors coming from body fluids can indicate liver and bladder problems. Currently, an electronic nose for examining wound infections is being tested at South Manchester University Hospital [23].

A more futuristic application of electronic noses has been recently proposed for telesurgery [24]. While the inclusion of visual, aural, and tactile senses into telepresent systems is widespread, the sense of smell has been largely ignored. An electronic nose will potentially be a key component in an olfactory input to telepresent virtual reality systems including telesurgery. The electronic nose would identify odors in the remote surgical environment. These identified odors would then be electronically transmitted to another site where an odor generation system would recreate them.

3.11 Applications in the Arts

We now turn to the artistic uses of NNs. Currently, this is a wide-open field; exploration has just begun in most cases, and we've barely scratched the surface of possibilities. The ideas below are mostly speculations on what networks could do, the sorts of tasks they could be applied to in the arts, sometimes based on applications that have already been done in scientific or engineering domains, and sometimes just based on imaginative speculation. As such, these ideas are intended to spark people's imaginations further in the search for innovative uses of this powerful and flexible new technology.

The main place where neural networks have been put to creative and artistic use so far is in *music,* as witnessed by the recent publication of the book, Music and Connectionism Several applications have been done in this area, ranging from psychological models of human pitch, chord, and melody perception, to networks for algorithmic composition and performance control. Generally speaking, the applications here (and in other fields) can be divided into two classes: "input" and "output". The input

side includes networks for recognition and understanding of a provided stimulus, for instance speech recognition, or modeling how humans listen to and process a melody. Such applications are useful for communication from human to machine, and for artistic analysis of a set of inputs. The output side includes the production of novel works, applications such as music composition or drawing generation. "Input" tasks tend to be much more difficult than "output" tasks (compare the state-of-the-art in speech recognition versus speech production by computers), so most of the network applications so far have focused on creation and generation of output, but continuing research has begun to address this imbalance.

On the "input" side in musical applications, Sano and Jenkins have modeled human pitch perception; Bharucha have modeled the perception and processing of harmony and chords; Gjerdingen has explored networks that understand more complex musical patterns; and Desain and Honing have devised a network for looking at the quantization of musical time and rhythm. Dolson has also suggested some approaches to musical signal processing by neural networks, including instrument recognition, generation, and modification. In this regard, musical applications of networks have much to gain from the vast literature on networks for speech processing (primarily recognition--see Lippmann,

On the "output" side, several network models of music composition have been devised. Todd and Mozer use essentially the dynamic sequential network approach mentioned earlier, in which a network is trained to map from one time-chunk of a piece of music to the following time-chunk (e.g. measure N as input should produce measure N+1 as output). The network's outputs are then connected back to its inputs for the creation phase, and a new measure 1 is provided to begin the network down a new dynamic path, creating one measure after another, and all the while incorporating the sorts of features it learned from its training examples. In this way, new pieces that have a sound like Bach or Joplin (or a combination of both!) can be created, if the network is first trained on these composers. But the problems mentioned earlier of lack of higher-level structure emerge, and these compositions tend to wander, having no clear direction, and seldom ending up anywhere in particular. Approaches for learning and using hierarchical structure are being devised, and Lewis describes one such method, in which

the inputs to a network, rather than the weights in the network, are modified during a learning stage, to produce an input which has a specified form or character. Kohonen present still another method of composition, which uses a network-style approach to build up a context-free grammar that models the music examples it's trained on.

Networks can also be used to generate musical performance parameters and instructions, as Sayegh demonstrates in his paper on a network method for choosing correct chord fingering for a simple melody. Many other musical performance applications are possible, from synthesizer control to automatic rhythmic accompaniment generators; Todd discusses some of these possibilities along with further ideas for musical applications of neural networks.

3.12 Neural Networks in Telecommunications

The IEEE Communications Society is active in developing a list of state-of-the-art topics in communications. Some of these are areas in which neural networks have a rôle, such as signal processing for beam forming, adaptive antennas, consumer communications, radio resource management and mobility management.

Beam forming employs signal processing in transmitting information over multiple antennas. It is also used for receivers to create steerable arrays. The purpose of beamforming is to minimize interference whether this is caused by fading, reflections or the effects of multi-user interference. If the channel is unknown or is changing, an adaptive antenna system will prove to be an advantage. Adaptive antennas can also offer capacity enhancements or allow higher bit-rates to be used.

Consumer products will soon have the capability of high-speed communications. This requires low cost and low power electronics. However, the domestic environment may not be RF-friendly so that an intelligent and adaptive receiver can improve the throughput without requiring an increase in transmitter power. One such wireless communications standard is Bluetooth; Bluetooth has to compete with IrDA (Infrared Data Association) which is a line-of-sight system, whereas Bluetooth is not.

Wireless systems are demanding higher spectrum efficiency as applications become more bandwidth-hungry. Radio resource management is essential and requires dynamic

channel assignment, interference avoidance, propagation prediction and automated planning techniques which are conventional neural network applications. Handoff requires a decision which is similar to a fuzzy logic rule.

When a user moves between a fixed and mobile platform, it will be essential that this user can enjoy the same services and applications transparently. Research continues into intelligent systems to implement dynamic routing, wireless ATM and location prediction.

3.13 Summary

This chapter demonstrates applications of artificial neural networks in various fields. We have described briefly neural networks applications in language processing, character and pattern recognition, and servo control application. Also we have discussed the neural networks application in image compression and application fields in medicine and business includes some examples, in addition to applications in arts and telecommunication. Finally we have discussed how to determine if the application is a neural network candidate.

CHAPTER FOUR

DIGIT RECOGNITION USING NEURAL NETWORK

4.1 Overview

By reducing the need for human numeral recognition system can speed up jobs such as reading income tax returns, sorting inventory, and routing mail. Several steps are necessary to achieve this. A recognition system must first capture digital snapshots of handwritten numerals. Before attempting to classify the numerals, some preprocessing of the snapshots might be necessary. If a string of adjacent numerals is captured at once, for example, the system has to decide on boundaries between digits. An algorithm must then classify each hand written numeral as one of the ten decimal digits. This information can then be stored in a database, passed on to mail sorting robots, or utilized in some other fashion.

Although a qualitative description of this process is straightforward, it cannot be easily reduced to a few simple mathematical rules. The difficulty results from the natural variation in human handwritten. A useful recognition system must be robust to alteration in size, shape, orientation, thickness, etc. Closed form mathematical models tend to be inadequate for such a task because of the many possible representations of the same image. To be sufficiently robust such models require a very large number of rules.

Humans, on the other hand can accurately classify all but the most severely distorted numerals. One natural idea then is to emulate in software the process that people use to recognize numerals. As a child is growing up he learns to read by seeing images (letters, numbers) and having a teacher(someone who already can read) tell him what the images represented. After sufficient training the child becomes able to read without a teacher. His education also generalizes so that he can even read unfamiliar handwritten. As an example consider figure 1. Although the numerals in each pair look significantly different, few readers will have difficulty identity them.

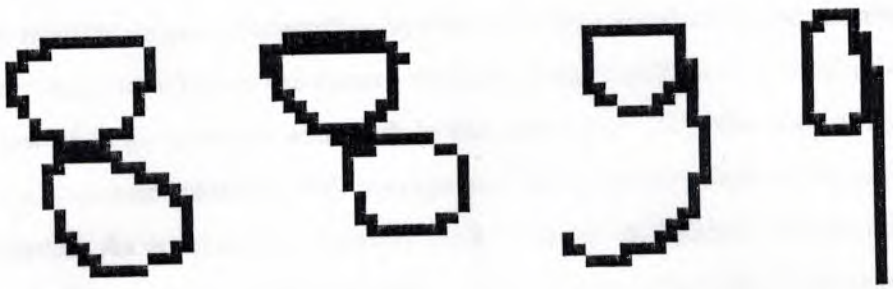


Figure 4.1 hand written numerals from different sources

Human learn to classify fuzzy patterns like these, and neural networks are well suited for mapping this human style of recognition into a procedure implementable in software.

4.2 Training

Neural networks are highly nonlinear, so the only feasible method of setting the neuron weights is to train the network. Initially the weights are set to small random values. Then an input pattern is fed into the network, and the outputs from the neurons in the final layer are observed. An error signal is calculated at each output neuron by subtracting the observed output from the desired output. These errors are fed back into network and used to update the neuron weights in each layer. The standard technique used to do is the back propagation algorithm.

This process can be repeated with many input output pairs. Eventually the neural network will learn to associate certain inputs with certain outputs. Additionally, the network behavior will generalize to some extent. That is given an input that it has not seen before, the network will match it with a similar input that it has seen before and will produce a corresponding output. This is the characteristics of neural networks that make them suitable for classifying patterns such as those in figure 1. A network is first trained by feeding in many different images of hand written numerals and telling the network what the output should be (i.e. what digit the input represents). After sufficient training the network will be able to classify new images based on their similarity to images used for training.

4.3 Training and Test Data

A reliable numeral recognition system must be trained on a large number of samples. With a large training set the system will get to many different versions of each digit, and the law of large numbers will work to our advantage. The effects of minor variations in the hand written numerals will average out, while the key features of each digit will be reinforced. As an example, consider the two"eight" in figure 1. Notice that one of them learns a little to the left while the other leans a little to the right. If the network is trained with lots of eights, some learning one way and some learning the other, it will conclude that this is not an important feature in determine " eight ness" . All eights consist of two touching ovals, however so the network will learn that this is a critical feature for all eights. If the training set is too small the network will not able to figure this out and therefore will not generalize well.

The database used in this chapter was used originally assembled by Alpaydin and Kaynak at Bogazici University in Turkey. The database and some background information are available online [4]. It consists of one of about 3800 samples from 30 people, and a second set of about 1800 samples from 13 different people. Each sample is 32x32 pixel binary images scanned from preprinted forms filled out by these 43 people. Each digit zero through nine is represented an equal number of times.

Alpaydin and Kaynak also scaled and normalized the image using standard routine from the National Institute of Standard and Technology (NIST). This preprocessing in numeral independent and serves only to center the numerals is the 32x32 grids and stretch, them to have consistent width and height. This is illustrate on figure 2, where the one (originally a this line) was widened to be about as wide as the eight.

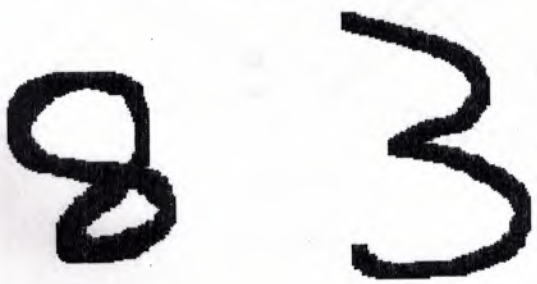


Figure 4.2 Normalized images

4.4 The Backpropagation Algorithm

The Error back propagation (or simple, backpropagation) algorithm is the most important algorithm for the supervised training of multilayer feed forward ANNs. It derives its name from the fact that error signals are propagated back ward through the network on a layer by layer basis.

The back propagation algorithm is based on the selection of suitable error function or cost function whose values are determined by the actual and desired outputs of the network and which is also dependent on the network parameters such as the weights and the thresholds. The basic idea is that the cost function has a particular surface over the weight space and therefore an iterative process such as the gradient descent method can be used for its minimization. The method of gradient descent is based on the fact that, since the gradient of a function always points in the direction of maximum increase of the function then, by moving to the direction of the negative gradient induces a maximal "downhill" movement that will eventually reach the minimum of the function surfaces over its parameter space. This is a rigorous and well establishes technique for minimization of functions and has probably been the main factor behind the success of backpropagation

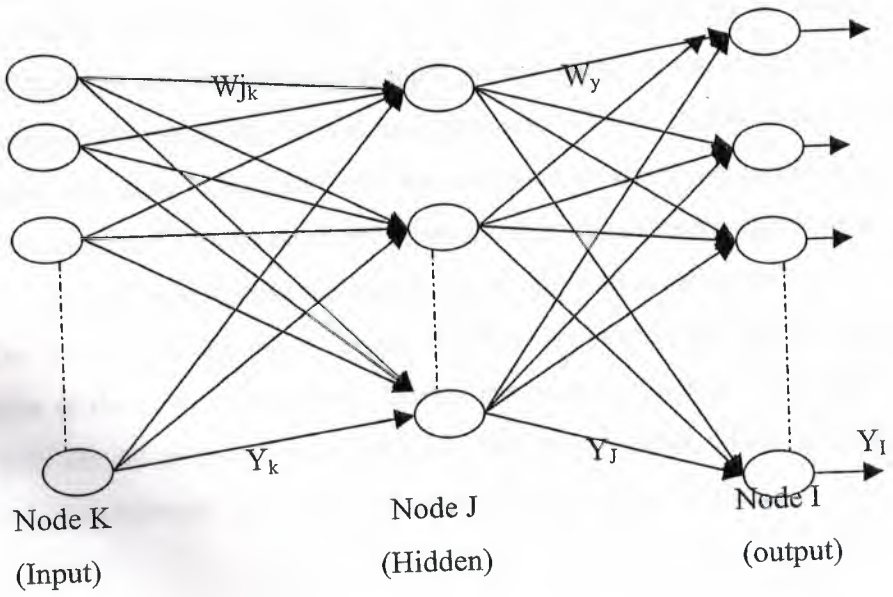


Figure 4.3 typical multilayer feed forward

A typical multilayer feed forward ANN is shown in figure 4.3. This type of network is also known as a Multilayer Perceptron (MLP). The units (or nodes) of the network arte nonlinear threshold units described by equations. The units are arranged in layers and each unit in a layer has all its inputs connected to the units of a preceding layer (or to the inputs from the external world in the case of the units in the first layer), but it dose not have any connections to units of the same layer to which it belongs. The layers are arrayed one succeeding the other so that there is an input layers multiple intermediate layers and finally an output layer. Intermediate layers that is those that have no inputs or outputs to the external world, are called hidden layer fig 4.3 show a MLP with only one hidden layer. Backpropagation neural networks are usually fully connected. This means that each unit is connected to every output from the preceding layer (or to every input from the external world if the unit is in the first layer) as well as to a bias signal which is common to all the units. Corresponding each unity has its output connected to every unit in the succeeding layer. Generally, the input layer is considered as just a distributor of the signals from the external world and is not there fore counted as layer. We will retain this convention in our analysis and hence in the case of figure 4.3 the hidden layer is the first layer of the network.

The back propagation training consists of two passes of computations: a forward pass and a backward pass. In the forward pass an input pattern vector is applied to the sensory nodes of the network that is to the units in the input layer. These signals from the input layer propagate to the units in the first layer and each unit produces an output according to equations. The outputs of these units are propagated to units in subsequent layers and this process continuous unit; finally the signals reach the output layer where the actual response of the network to the input vectors is obtained. During the forward pass the synaptic weights of the network are fixed. During the backward pass, on the other hand synaptic weights are all adjusted in accordance with error signal which is propagate backward through the network against the direction of synaptic connection.

Forward Pass

a) Initialization: assign random values between -1 and +1 to the weights between input and hidden layers W_{ij}, the weights between hidden and output layers V_{ji} and thresholds for hidden and output layers neurons θ_j , Υ_i.

b) Train the networks by randomly selecting a pair of patterns $A_k=(a^k_1,a^k_2$ ....., $a^k_n)$, $Y_k=(y^k_1,y^k_2$ ....., $y^k_n)$ for training set.

c) Compute the input of hidden layer neurons S_j by using input pattern A_k, weights W_{ij} and threshold θj, compute the hidden layer neuron activation b_j by sigmoid activation function.

$$Sj=\sum_{i=1}^{n} W_{ij} * a_i - \theta_j$$

$$b_j= f(s_j) \quad j=1,2,.....p$$

d) Compute the input of output layer neurons L_t based on hidden layer neuron activation b_j weights V_{ij} and threshold Υ_i computer the output layer activation C_t by sigmoid activation function.

$$Lj=\sum_{j=1}^{p} V_{ji} * b_i - \Upsilon_j$$

$$t= 1,2,.....,q \quad C_t=f(L_t) \quad t=1,2,.....q$$

e) Compute then output layer error d^k_i by using the observed outputs C_t and target output Y_k.

$$d^k_i = (d^k_i - C_t)* C_t (1- C_t) \quad t=1,2,.....q$$

f) Compute the hidden layer error e^k_j based on weights V_{ij} output layer error d_t hidden layer output b_t

$$e^k_j =(\sum_{i=1}^{q} V_{ji} * d_t) b_i(1- b_j) \quad j=1,2,.....p$$

Backward Pass

g) Adjust the weights V_{ji} and threshold Υ_t by hidden layer error a^k_t hidden layer output b_j

$$V_{ji}(N+1) = V_{ji}(N) + a^* a^k_t + b_j \quad j=1,2,\ldots\ldots\ldots\ldots p \quad t=1,2,\ldots\ldots\ldots q$$

$(0<\beta<1)$

$$\Upsilon_j(N+1) = \Upsilon_j(N) + a^* a^k_t \quad t=1,2,\ldots\ldots\ldots\ldots\ldots\ldots q$$

h) Adjust the weights W_{ij} and threshold θ_j by using hidden layer error a^k_j input layer A_t

$$W_{ij}(N+1) = W_{ij}(N) + \beta^* a^k_i * e^k_j \quad i=1,2,\ldots\ldots\ldots\ldots\ldots\ldots n \; j=1,2\ldots\ldots\ldots p$$

$$\theta_j(N+1) = \theta_j(N) + \beta^* e^k_j \quad j=1,2,\ldots\ldots\ldots\ldots p$$

4.5 Network Convergence

i) Select the next pair of patterns from training set randomly trains the network from the step (c) until all patterns are used for network training.

ii) Reselect a pair of patterns from training set randomly, train the network from the step (c) until the output layer error is within the specified tolerance for each pattern and neuron.

The algorithm tested for this project is illustrated in this figure:

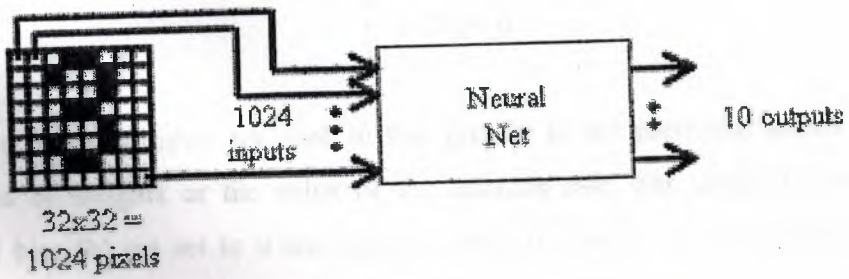


Figure 4.4 Tested Algorithms

The input to the neural network consists of all 1024 pixel amplitudes, and these are one output corresponding to each digit 0-9. The desired response is +1 at the output corresponding to the identified digit and -1 at all the other outputs. The desired response

for the input shown in figure 4.4 then would is +1 at output 3 and -1 at all other outputs. Typically the outputs will not by exactly +/-1, but we hope that they are close say +/-0.95 or better. Some experiments were run to determine what type of neural network topology worked best for this algorithm. Ultimately a 1024-100-10 network (i.e. 1024 inputs, 100 neurons in layer 1, 10 neurons in layer 2) was found to work best.

4.6 Method

For the experimental purpose of our project we have selected and trained a single layer percetron to recognize handwritten digits. The training data set we used is obtained from UCI machine learning repository (Courtesy of UCI machine learning repository) which consists 1.934 cases of numeric digits of '0', '1', '2', '3', '4', '5', '6', '7', '8', '9'. Each training case is in digitized format see the figure of 0s and 1s at a resolution of 32X32. Based from this input format we designed the perceptron at accept 1,024 input units 32X32=1024) and 10 output units see the next figure. The weights are fully connected between input and output units. We chose a modified version of binary step function to our perceptron activation function .the output of the perceptron is based from this activation function given the input y.

$$F(y)= \begin{cases} 1 \text{ if } y > 0 \\ 0 \text{ if } y = 0 \\ -1 \text{ if } y < 0 \end{cases}$$

The perceptron algorithm we used in this project is not particular sensitive to the initial values of weights or the value of the learning rate. For simplicity the default weights and bias (b) are set to 0 and learning rate (a) is set to 1. The following is the modified version of perceptron algorithm based on our initial setting:

W_{ij} : weight associated with input unit I and output unit j

S_i : raw data for input unit i.

X_i : input unit i.

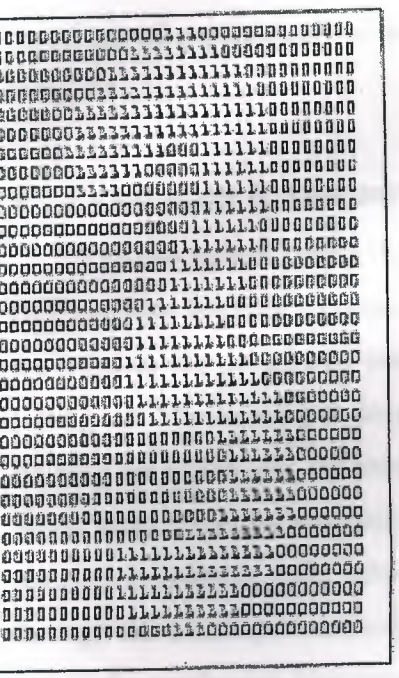
T_i : true pattern.

Y_in: summation of weighted inputs

Y: perceptron output computed by activation function

a: learning rate set to 1

b: bias set to 0



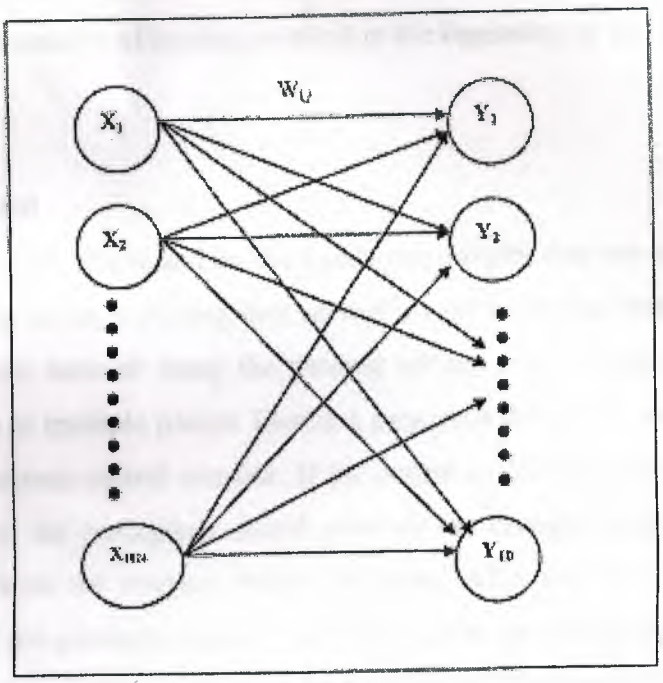


Figure 4.5 a) Digitized numeric digit 3 **b) perceptron with 1024 input 10 output unit and full connected weights**

initial weights and bias

do {

$X_i = S_i$

$Y\_in = \sum XW$

$$y = \begin{cases} 1 \text{ if } y\_in > 0 \\ 0 \text{ if } y\_in = 0 \\ -1 \text{ if } y\_in < 0 \end{cases}$$

if y = 1 then // update weight

$W_{i,j}$ (new)= $W_{i,j}$(old) + $T_f X_i$

Else // no change

$W_{i,j}$ (new)= $W_{i,j}$(old)

} while (at least one weight has been updated)

We also modified the training set which w obtained from UCI machine learning repository all parameters (except number of training pattern) in the beginning of file were removed.

4.7 Evaluation & Discussion

The perceptron neural network was tested on the handwritten digits data set in the UCI machine learning repository. Given a training data set and a separate testing data set, we training the perceptron neural network using the training set and test it against the testing set. The training is done in multiple passes. During a pass each data point inn the training set is fed into the perceptron neural network. If the output is different from the target label then the weights of the perceptron neural network are changed using the perceptron learning rule, otherwise the weights remain the same. After we run a pass through the training set we test the perceptron neural network against the testing set and record the erro0r rate. One question arises is when to stop the training. We given a plot of training and testing error rate for 50 passes (see the figure3) .here we use heuristic based on the observation that when a learning algorithm starts over fitting the error rate on the testing will go up. Our c is to run for 5 passes so that the algorithm wills stabilize and then monitor the testing error rate; if the testing error rate starts going upon we stop. We use th testing error rate at point A.

To get an unbiased estimated of the prediction error rate of the algorithm, we use ten fold cross validation to evaluate the predicate error rate. The whole data set is divided into 10 bins(more generally the data set can be divided into k bins in which case we have the k fold cross validation). Each bin will be used as the testing set in turn when the remaining 9 bins are used as the training set. The prediction error rates are averaged give as estimated of the prediction error rate on the algorithm for future unseen data. In the following we report the result in terms of cross validation testing error rate.

There are two formats of data set. The first one is the original 32x32 bitmap format. n the second data format the 32x32 bitmap is divided into non overlapping blocks of 4x4 nd the number of ON pixels is counted. It is very interesting to compare the relative dvantages and disadvantages of two formats. The preprocessed 8x8 blocks formats educes the dimensions from 1024(32x32) to 64(8x8). This will speed up computation. Iowever it is debatable whether it can improve accuracy on the one hand it gives nstance to small distortions but on the other hand it loses some original information. Iere are the results for 32x32 bitmap format the error rate is 9.98% for 8x8 blocks ormats the error rate is 11.7%. We are not sure whether this difference of 1^ is ignificant if yes we can say the 8x8 blocks formats really loses much of the information n the original 32x32 bitmap imply that the W0 term is an unnecessary degree of freedom ero threshold is adequate for this data set of handwritten digits.

We also look at some variations on the algorithm. For example in our basic lgorithm we do not include the W0 term in the weight vector which in effect fix nreshold to zero. We had expected to get a lower error rate when we include the W0 erm for non zero threshold. But to turns out that this does not help. In fact the error rate n the 32x32 bitmap format increasing slightly from 9.98% to 10.69% and from 11.07% 12.10% on the 8x8 block format. We were also curios about the capability of erceptron. We believed based from the design of perceptron architecture that it will cognize other pattern as well provided with sufficient amount of training set. As an xtension top our project we have created a small training set at a lower resolution (6x6) ith three distinct patterns: A.F, Chinese's character for day (see figure 6).as we have xpected the preceptron recognizes the patterns once it has been properly trained.

000100

001010
010010
011111
010001
100001

Figure 4.4 Characters 'A'

4.8 Results

Metrics of interest

Three outcomes are possible when testing this algorithm given any input the network can either make a correct identification an incorrect identification or no identification. The first case is of course desirable. A three is correctly identity as a three. The second outcome occurs when the network makes a mistakes – a three is identity assume other digit. The third case occurs when the network is not sufficient in the output to make a guess. My algorithm makes no identification if either all outputs are negative or more than one output is positive. It will only venture guess when exactly one output is greater than zero. This approach is often better than guessing with a high probability of error. In sorting mail for example it would be better to send unreadable addresses to a person clarification rather than a route them to the wrong destination.

Classes of data

The data used for testing can be divided into three categories. The first set consists of samples actually used in training. The network should identity these digits with near 100% accuracy since it has seen them before. If it does not the amount of training was probability adequate. The second class of data is composed of additional samples from people who contributed to the training set. These samples are likely to resemble training sample very closely but they were not themselves used for training. Finally the system should be tested with samples from people not use in training. Real systems must handle this type of data robustly so accuracy on unfamiliar data is the most important metric.

We can see this software as an example for our project.

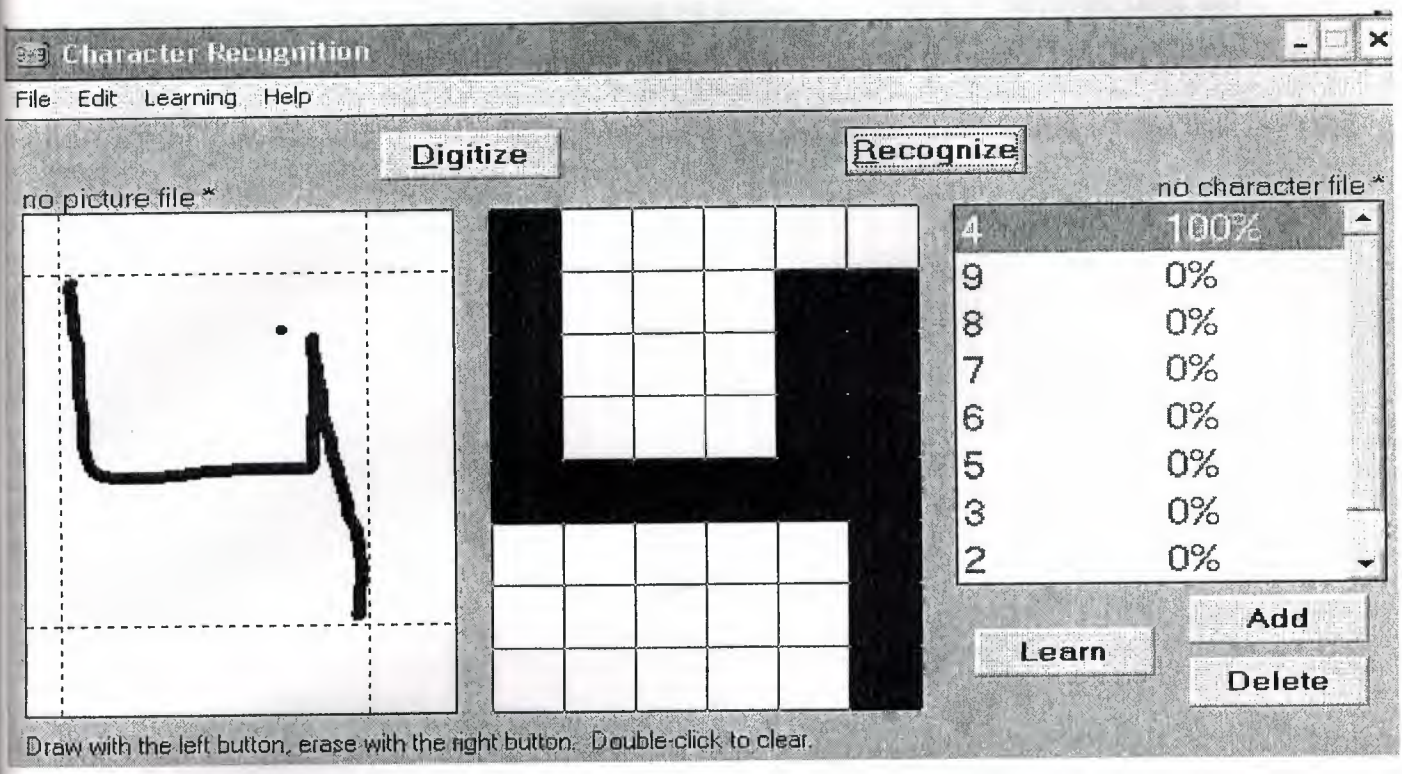


Figure 4.5 Examples for Recognition

4.9 Summary

In this chapter we have successfully demonstrated preceptron learning algorithm to the problem of handwritten digits recognition. The cross validated error rate was 9.98%. The algorithm runs best (in terms of lowest error rate) when setting threshold to zero and training directly on the original 32x32 bitmap format. Allowed none zero thresholds does not improve perceptron learning nor does dimension reduction using the 8x8 blocks format although the latter speed up computation. The perceptron model can recognize other character data set besides hand written digits.

Automatic recognition of hand written numerals has many applications but designed reliable system is challenging because of the neural variations in human handwritten. One way to solve this problem is to use a neural network that learns to identify numerals much

like a person learns to read. If the training set for such a network is sufficient large and diverse, the network will generalize top recognize hand written numerals from unfamiliar sources.

CONCLUSION

Neural networks are developed with the goal of modeling information processing and learning in the brain applied to a number of practical applications in various fields, including computational molecular biology.

Artificial neural networks are one of the promises for the future in computing. They offer an ability to perform tasks outside the scope of traditional processors. They can recognize patterns within vast data sets and then generalize those patterns into recommended courses of action. Neural networks learn, they are not programmed.

Yet, even though they are not traditionally programmed, the designing of neural networks does require a skill. It requires an "art." This art involves the understanding of the various network topologies, current hardware, current software tools, the application to be solved, and a strategy to acquire the necessary data to train the network. This art further involves the selection of learning rules, transfer functions, summation functions, and how to connect the neurons within the network.

Then, the art of neural networking requires a lot of hard work as data is fed into the system, performances are monitored, processes tweaked, connections added, rules modified, and on and on until the network achieves the desired results.

These desired results are statistical in nature. The network is not always right. It is for that reason that neural networks are finding themselves in applications where humans are also unable to always be right. Neural networks can now pick stocks, cull marketing prospects, approve loans, deny credit cards, tweak control systems, grade coins, and inspect work.

Yet, the future holds even more promises. Neural networks need faster hardware. They need to become part of hybrid systems which also utilize fuzzy logic and expert systems. It is then that these systems will be able to hear speech, read handwriting, and formulate actions. They will be able to become the intelligence behind robots that never tire nor become distracted. It is then that they will become the leading edge in an age of "intelligent" machines.

The aim of this project was to present an understanding to the broad subject of neural networks explaining the implementations of neural networks.

Chapter one described a general introduction of neural networks, the definition of artificial neural and the history of neural networks from 1940s when the first neuron

was developed. The differences between neural computing and traditional computing were presented. Also it was explained how neural networks are being used and where the future of neural networks technology may lie.

Chapter two was about neural networks architectures and algorithms. Single-layer and multilayer feedforward networks, recurrent networks and radial basis function networks were described. Supervised and unsupervised learning were also explained.

Chapter three was aimed to present real applications to let the reader to enter the world of neural networks as they are used. Neural networks applied in vast amounts of field, in medicine, business, pattern recognition, image compression arts and telecommunications. These applications were discussed.

Chapter four was aimed to show the important application of neural networks in fraud detection concentrating on credit card fraud detection and how to use unsupervised neural networks in fraud detection. In this project we have successfully applied preceptron learning algorithm to the problem of handwritten digits recognition. Our cross validated error rate 9.98%. We find the algorithm runs best (in terms of lowest error rate) when setting threshold to zero and training directly on the original 32x32 bitmap format. Allowed none zero thresholds does not improve perceptron learning nor does dimension reduction using the 8x8 blocks format although the latter speed up computation. Our perceptron model can recognize other character data set besides hand written digits. Automatic recognition of hand written has much application, but designing reliable systems is challenging because of the natural variations is human handwriting. One way to solve this problem is to use a neural network that learns to identify numerals much like a person learns to read. If the training set for such a network is sufficient large and diverse, the network will generalize to recognize hand written numerals from unfamiliar sources.

REFERENCES

[1] McCulloch, W. S. and Pitts, W. H. *"A logical calculus of the ideas immanent in nervous activity"*, Bulletin of Mathematical Biophysics, 5:115-133, .1943.

[2] Rosenblatt, F.Rosenblatt, *"The perceptron: A probabillstic model for information storage and organization in the brain"*, Psychological Review, 65:386-408, 1958.

[3] Anderson, J. A., and Rosenfeld, E. (Eds.), *"Neurocomputing: Foundations of Research"*, Cambridge, MA: MIT Press, 1988

[4] Selfridge, O. G., "Pandemonium: a paradigm for learning. Mechanisation of Thought Processes", *Proceedings of a Symposium Held at the National Physical Laboratory*, 1958.

[5] Widrow, B and Hoff, "Adaptive Switching Circuits", *In 1960 IRE WESCON Convention Record*, pages 96 - 104. IRE., 1960.

[6] Minsky, M. and Papert, S., "*Perceptrons*", MIT Press, Cambridge. 1969

[7] Werbos, P.J., "*The Roots of Back propagation*", NY: John Wiley &Sons, 1974/1994.

[8] Parker, D.B., "*Learning-Logic*", MIT Center for Computational Research in Economic and Management Science, Cambridge, MA, 1985

[9] Rumelhart, D., J. McClelland & the PDP Research Group, "*Parallel Distributed Processing: Explorations in the Microstructure of Cognition V. 1/2*", MIT Press, Cambridge, MA, 1986.

[10] Churchland, P. and Sejnowski, T, "*The Computational Brain*", MIT Press Cambridge, 1992.

[11] Haykin, S. and Li, L., "Nonlinear adaptive prediction of no nstationary signals". *IEEE Transactions on Signal Processing*, 43(2):526-535, 1995.

[12] Bishop, C. M., "*Neural Networks for Pattern Recognition*". Oxford University Press, 1995.

[13] Kohonen, T., "An adaptive associative memory principle". *IEEE Transactions on Computers*, C-23:444-445, 1974.

[14] Hebb, D. O., "*The Organization of Behavior*". Wiley, 1949.

[15] Hopfield, J. J., "Neural networks and physical systems with emergent computational abilities", *Proceedings of the National Academy of Sciences*, 79:2554, 1982

[16] Kashman A., Neural Networks: "Lecture Notes of COM420", Near East University, Nicosia, 2002.

[17] Scheff, K. and Szu, H. "Gram-Schmidt Orthogonalization Neural Networks for Optical Character Recognition", *Journal of Neural Network Computing*, Winter, 1990.

[18] Paul Watta, Brijesh Desaie, Norman Dannug, Mohamad Hassoun, "*Image Compression using Backprop*", 1996.

[19] Schalkoff, R. J., "*Pattern Recognition: Statistical, Structural, and Neural Approaches*", John Wiley & Sons, New York, NY, 1992.

[20] Hutchison, W.R. & Stephens, K.R., "The Airline Marketing Tactician (AMT): a commercial application of adaptive networking". *Proceedings of the first International Conference on Neural Networks*, 4: 753-756. IEEE Press. 1987.

[21] Robert Hecht-Nielsen. , "*Neurocomputing*". Addison-Wesley, 1989

[22] B.S. Hoffheins, Using Sensor Arrays and Pattern Recognition to Identify Organic Compounds. *MS-Thesis*, the University of Tennessee, Knoxville, TN, 1989.

[23] K. Pope, "Technology Improves on the Nose As Science Tries to Imitate Smell", *Wall Street Journal*, pp. B1-2, 1995.

[24] P.E. Keller, R.T. Kouzes, L.J. Kangas, and S. Hashem, "*Transmission of Olfactory Information for Telemedicine*", IOS Press, Amsterdam, , pp. 168-172, 1995.

[25] Pacific Northwest National Laboratory (PNNL)
"http://www.emsl.pnl.gov:2080/proj/neuron/neural ".

[26] Todd, P.M., and D.G. Loy (Eds.), "*Music and Connectionism*". Cambridge, MA: MIT Press, 1991

[27] Lippmann, R.P., "Review of neural networks for speech recognition". Neural Computation, 1 1-38, 1989

[28] Brause, R., Langsdorf T. and Hepp M., "Neural Data Mining for Credit Card Fraud Detection". *Proceedings. 11th IEEE International Conference on Tools with Artificial Intelligence*, 1999.

[29] Hassibi, K., "*Detecting Payment Card Fraud with Neural Networks. Business Applications of Neural Networks*". P.J.G. Lisboa, A.Vellido, B.Edisbury Eds. Singapore: World Scientific, 2000.