New Jersey Institute of Technology
Digital Commons @ NJIT

Dissertations | Theses and Dissertations

Summer 1998

Automatic analysis of electronic drawings using neural network

Yi Shi
New Jersey Institute of Technology

Follow this and additional works at: https://digitalcommons.njit.edu/dissertations

Part of the Electrical and Electronics Commons

Recommended Citation
Shi, Yi, "Automatic analysis of electronic drawings using neural network" (1998). *Dissertations*. 957.
https://digitalcommons.njit.edu/dissertations/957

This Dissertation is brought to you for free and open access by the Theses and Dissertations at Digital Commons @ NJIT. It has been accepted for inclusion in Dissertations by an authorized administrator of Digital Commons @ NJIT. For more information, please contact digitalcommons@njit.edu.

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

NJIT
New Jersey's Science &
Technology University

The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

ABSTRACT

AUTOMATIC ANALYSIS OF ELECTRONIC DRAWINGS USING NEURAL NETWORK

by
Yi Shi

Neural network technique has been found to be a powerful tool in pattern recognition. It captures associations or discovers regularities with a set of patterns, where the types, number of variables or diversity of the data are very great, the relationships between variables are vaguely understood, or the relationships are difficult to describe adequately with conventional approaches.

In this dissertation, which is related to the research and the system design aiming at recognizing the digital gate symbols and characters in electronic drawings, we have proposed: (1) A modified Kohonen neural network with a shift-invariant capability in pattern recognition; (2) An effective approach to optimization of the structure of the back-propagation neural network; (3) Candidate searching and pre-processing techniques to facilitate the automatic analysis of the electronic drawings.

An analysis and the system performance reveal that when the shift of an image pattern is not large, and the rotation is only by $n \times 90°$, (n = 1, 2, and 3), the modified Kohonen neural network is superior to the conventional Kohonen neural network in terms of shift-invariant and limited rotation-invariant capabilities. As a result, the dimensionality of the Kohonen layer can be reduced significantly compared with the conventional ones for the same performance. Moreover, the size of the subsequent neural network, say, back-propagation feed-forward neural network, can be decreased dramatically.

There are no known rules for specifying the number of nodes in the hidden layers of a feed-forward neural network. Increasing the size of the hidden layer usually improves the recognition accuracy, while decreasing the size generally improves generalization capability. We determine the optimal size by simulation to attain a balance between the accuracy and generalization. This optimized back-propagation neural network outperforms the conventional ones designed by experience in general.

In order to further reduce the computation complexity and save the calculation time spent in neural networks, pre-processing techniques have been developed to remove long circuit lines in the electronic drawings. This made the candidate searching more effective.

Blank Page

AUTOMATIC ANALYSIS OF ELECTRONIC DRAWINGS USING NEURAL NETWORK

by
Yi Shi

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

Department of Electrical and Computer Engineering

August 1998

Copyright © 1998 by Yi Shi

ALL RIGHTS RESERVED

APPROVAL PAGE

AUTOMATIC ANALYSIS OF ELECTRONIC DRAWINGS USING NEURAL NETWORK

Yi Shi

Dr. Yun Q. Shi, Dissertation Advisor — Date
Associate Professor of Electrical and Computer Engineering, NJIT

Dr. Edwin Hou, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, NJIT

Dr. Douglas D. C. Hung, Committee Member — Date
Associate Professor of Computer and Information Science, NJIT

Dr. Constantine N. Manikopoulos, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, NJIT

Dr. Wei Su, Committee Member — Date
Electronics Engineer of Intelligence and Information Warfare Directorate,
USA Army CECOM, Fort Monmouth, New Jersey

BIOGRAPHICAL SKETCH

Author: Yi Shi

Degree Doctor of Philosophy

Date: August 1998

Undergraduate and graduate Education:

- Doctor of Philosophy in Electrical and Computer Engineering,
  New Jersey Institute of Technology, Newark, NJ, USA, 1998

- Master of Science in Electrical Engineering,
  Beijing University of Aeronautics and Astronautics, Beijing, China, 1988

- Bachelor of Science in Electrical Engineering,
  Beijing University of Aeronautics and Astronautics, Beijing, China, 1983

Major: Electrical and Computer Engineering

Presentations and Publications:

Yi Shi, Yun Q. Shi, Constantine N. Manikopoulos, and Wei Su
"A Shift-Invariant Modified Kohonen Neural Network"
(Submitted to *Electronics Letters* of *IEE*)

X. Xia, Y. Q. Shi, and Y. Shi
"A Thresholding Hierarchical Block Matching Algorithm," *Journal of Computer Science and Information Management*, No. 2, 1998

Jiwu Huang, Yun Q. Shi, and Yi Shi
"Embedding Image Watermarks into DC Components" (Submitted)

Yi Shi
"A Design of EPROM Extension Card for APPLE-II Computer" *Radio*, No. 9, 1988.

Yi Shi, and Fan Renzhou
"Call and Parameters Transmission between C and Assembly Language", *Measurement & Control Technology*, No. 4, 1988, pp. 25-32.

Yi Shi
"A Design of Real-Time Control Operating System", Master Thesis, Beijing University of Aeronautics and Astronautics, 1988.

Yi Shi
"A Digital Serial Communication Telemetry and Telecontrol System for Nuclear Test", Bachelor Thesis, Beijing University of Aeronautics and Astronautics, 1983.

This work is dedicated to my beloved family

ACKNOWLEDGMENT

I would like to express my deep appreciation to Dr. Yun Q. Shi, who not only served as my research advisor, providing valuable, countless resources and insight, but also constantly gave me encouragement and reassurance.

Special thanks are given to Dr. Edwin Hou, Dr. Douglas D. C. Hung, Dr. Constantine N. Manikopoulos and Dr. Wei Su for their comments and suggestions to this dissertation and for actively participating in my committee.

I wish to thank Dr. Wei Su for his valuable advice regarding this research. Without the data and electronic drawings kindly supplied by him, this research would have not been able to be carried out. I am also grateful to my friend Yiwen Xu for the early cooperation in this research and his effort to supply the needed material.

Last but not least, heartfelt thanks to Dr. Constantine N. Manikopoulos, the Robert Van Houten Library and the Engineering Computing Lab at NJIT for their financial support during my Ph.D. program.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

CHAPTER 1

INTRODUCTION

1.1 Research Objective

Traditionally, paper documents are frequently utilized format for transmission and storage of information. In the last decade, the prevalence of fast computers with large memory space, and inexpensive scanners has fostered an increasing interest in the process and analysis of document images. With many paper documents being sent and received via fax machines and stored digitally in large document databases, the interest has grown to do more with these images than simply to view and print them. Just like humans extract information from these images, researches are being carried out and automatic recognition systems are being built up to read characters on pages, locate lines and recognize symbols on diagrams.

Document analysis has become more and more important than ever before. Look around our workplace, we can see stacks of paper. Some may be computer generated, but if so, inevitably by different computers and software such that even their electronic formats are incompatible. Some will include both formatted text and tables as well as handwritten entries. There are different sizes, from 3.5"×2" business cards to 34"×44" engineering drawings. In many businesses today, imaging systems are being used to store images of pages to make storage and retrieval more efficient. Future document analysis systems will recognize types of documents, enable the extraction of their functional parts, and facilitate translation from one computer-generated format to another. There are many other examples of the use of and need for document systems. Glance behind the counter

in a post office at the piles of letters and packages. In some US post offices, over a million pieces of mail must be handled each day. Machines to perform sorting and address recognition have been used, but there is a need to process more mail, more quickly and more accurately. As a final examine the stacks of a library, where row after row of paper documents are stored. Loss of material, misfiling, limited numbers of each copy, and even degradation of materials are common problems, and the problems may be alleviated by document analysis techniques. All of the above examples serve as applications ripe for the potential solutions of document image analysis.

Many researches are leading to new applications. For instance, millions of old books now in libraries will be replaced by computer files of page images that can be searched for content and accessed by many people at the same time and will never be misshelved. Business people will carry their file cabinets in their portable computers. Paper copies of new product literature, receipts, and other notes will be instantly filed and accessed in computers. Signatures will be analyzed by computers for verification and security access. Musical scores and other symbolic and diagrammatic documents will be read and their contents recognized and interpreted.

The ultimate solution would be for computers to deal with paper documents as they deal with other forms of computer media. That is, paper would be as readable by computers as magnetic and optical disks are now. If this were the case, then the major difference between paper documents and magnetic and optical disks would be that, unlike current computer media, paper documents could be read by both computers and human beings. This is, on the other hand, the major advantage of paper documents.

Nowadays the Department of Defense(DOD) maintains a large inventory of documentation of electronic systems. The existing documentation is in the form of circuit drawings and scanned image files, which cannot be directly used as an input to computer-aided design tools. In order to respecify or remanufacture an electronic part, board, or system, a considerable amount of time and human effort must be expended to collect and understand the circuit information from circuit drawings. On the other hand, VHSIC (Very High Speed Integrated Circuit) Hardware Description Language (VHDL) is an industry and DoD standard [1] for documentation, design, and simulation of electronic circuits. Most commercial computer-aided design (CAD) tools provide a fully automated and integrated manufacturing path from VHDL documentation to integrated circuit (IC) fabrication. Since VHDL provides manufacturing technology independent documentation of an IC, it is ideal to document Army systems in terms of VHDL so as to have full life support. Therefore, the automated generation of VHDL models is an urgent and important research topic, which is a research in the US Army Research Laboratory, Ft. Monmouth, NJ. In this research and system design, we address a part of their research: How to locate digital gate symbols' position and how to recognize various digital gate symbols and characters in electronic engineering drawings.

The objective of this research is to build a system which is a bridge connecting yesterday's documentation to today's modern computer and people. The proposed system will be used not only in the military but also in the civil industry of electrical manufacturing. We propose to develop a neural network as a recognition unit in the system. Neural network has been proved to be very capable in recognizing image patterns. The post offices use it to automatically recognize zip codes on daily basis. It

demonstrates that neural network is successful in the practical usage. Because the gate symbol patterns are larger than the zip codes, we proposed to design a larger neural network. Evidently, a large neural network needs more calculation time than a small one. Therefore, to use neural network efficiently is an important topic. We proposed not to use neural network to search and recognize gate symbols over the entire drawings, but to use it only to recognize just a few candidates on the entire drawings. We proposed and used some pre-processing techniques and developed some algorithms to find out gate symbol candidates. This makes the overall searching and recognition much more quickly.

In the early stage of this research, we did some survey (refer to the APPENDIX). In the survey, three techniques in the recognition of various gate symbols were investigated. Three types of neural network developed for recognition of characters and digits, and moments method used in image pattern recognition were studied as well. Of these three existing techniques [2–4], two of them are actually not practical yet. The practical one can work well only for well-standardized drawings. It involves feature extraction, template matching, and decision tree. Hence it is very complicated and can only be implemented in special hardware. It does not work well for noisy and deformed images. The moments method [5] is not practical. However, in practical setting, Fukushima's neocognitron [6] and AT&T's neural networks [7,8] can recognize characters and digits, respectively. Both of the neural networks can be implemented in software. We therefore propose to develop a NN to handle this part of the project. Considering that the neocognitron neural network needs complicated human efforts (hard work!) to define features and design templates in the training process, we decided to

combine the Kohonen neural network and the feed-forward back-propagation neural network as the main recognition unit instead of the neocognitron NN.

This system is divided into three parts. Refer to Figure 1.1. This thesis is organized in a very sequence in which the image pattern in the electronic drawing flows through the system.

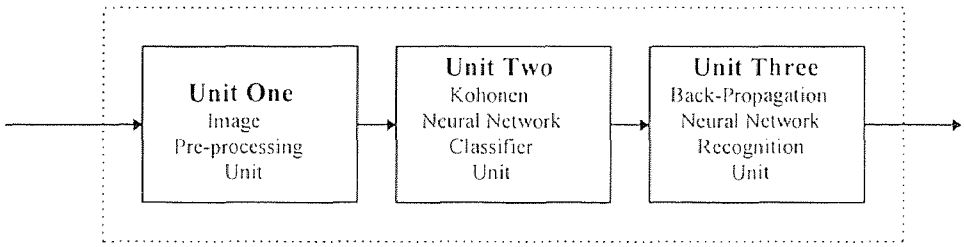


Figure 1.1 The Diagram of the System

In this chapter, Chapter 1, we briefly introduce this research topic, image pattern recognition and artificial neural network.

In Chapter 2, Image Pre-Processing Unit, we focus on how to pinpoint the candidate gate symbols. It is mainly for an effective usage of the neural network. Some pre-processing methodologies are described in this part.

The third chapter is a Kohonen Neural Network Classifier. It works as a bridge connecting between the Pre-Processing Unit and the Back-Propagation Neural Network (BPNN) Recognition Unit. Through this unit, the gate symbols and characters are roughly classified. The output of this unit is a categorized map. Because the map is of small dimension, the input layer of the subsequent BPNN can be small. Moreover, the overall dimension of the BPNN is much smaller than the dimension of the BPNN whose input is original image signals to be recognized.

In the fourth chapter, the BPNN Recognition Unit, we state about how to recognize various digital gate symbols by using BPNN and how to design and train a feedfoward neural network.

Chapter 5 is about neural network generalization and training with noise: A further training. After the further training, the overall capability of the neural network gets more powerful.

In the last chapter, we summarize all the contributions in our research and system design. We also raise a problem for the future research: how to verify the gate symbols which have been recognized? The motivation behind it originates from the fact that the correctness and the reliability of the recognition are more important than the processing speed of the proposed system.

1.2 Pattern Recognition

Traditional pattern recognition [9–12] techniques include statistical pattern recognition algorithms, probability and decision theory, and feature extraction. The latter techniques are concerned with the decomposition of an input into pattern primitives. These techniques overlap and are not really separable. As an example of a statistical approach, assume that a two-class problem is to be solved. That is, a given input pattern must be classified to belong to either class A or class B. For purposes of this example, assume that there are only two features that can be measured to determine classification with distributions as shown in Figure 1.2.

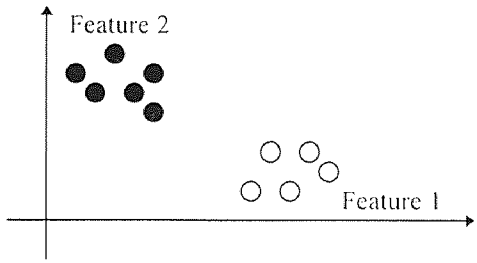


Figure 1.2 Two Class Problem

An example is taken here. Let's assume that the problem is to build an assembly line monitor that determines whether a product is to be shipped or is defective. Let's also assume that the product is a bottle that we must make sure is full and capped. To determine whether it is full we set up an optical beam to pass through the bottle at a point just below the required fill level. Again, for simplicity, assume that the liquid is basically opaque. If the bottle is full, the amount of light sensed by the photodetector will be less than that would be sensed if it were empty. The amount of light sensed by the photodetector will be the first feature, a measured parameter that enables the network to discriminate between the classes. Low values of this feature are associated with a full bottle (because the liquid blocks the light) and thus represent good bottles. To determine whether there is a cap on the bottle, a similar approach will be taken.

So let's take a beam of light and direct it at the top of the bottle. A strong reflectance off the top (resulting in a relatively large number for the second feature) would imply that the bottle has a top on it. This amount of light reflected off the top would be the second feature. In the feature space, good bottles, determined as being full and having a cap, are found in the upper left portion of the first quadrant of the feature space, denoted as solid circles. Bad bottles, determined by really being bad in the sense

that they are not full and do not have a cap, are found in the lower right portion of the first quadrant, denoted with circles.

A statistical approach to this problem is to find the K-nearest neighbors to the unknown input and check their respective class membership. This is done by first collecting a lot of bottles that have been observed to be either good or bad. Then, by placing these bottles through the measurement process, points in the feature apace are marked as being associated with either good or bad values. When a bottle is input that is not classified and must be tested to determine *goodness*, it is also mapped to the feature space. Measurements are made to determine which of the previous mapped points are closest to this unknown input. For example, some Minkowski distance could be used to measure the closeness to a previously classified point.

$$d^n(x,y) = \left(\sum_{i=0}^{s} |x_i - y_i|^n\right)^{1/n}, \text{ for } n \geq 0 \qquad (1.1)$$

Minkowski n-distance is defined as the nth root of the sum of the difference of feature, $|x_i - y_i|$, values raised to the nth power. Thus if $n = 1$ this is just the sum of the magnitude of the differences of the feature values; this is frequently known as taxi distance. If $n=2$, we have the square root of the sum of the squares of the differences of the feature values, Euclidean distance.

For example, the closest k points by a Minkowski metric could be determined. If the majority of the k-nearest neighbors are from some particular class, then the unknown input is also assumed to be of that class. To accomplish this, a large amount of labeled data is required. If we assume that the data are distributed in a Gaussian manner, then the means and standard deviations are computed for the classes of good and bottles. Then

when some unknown input is tested, a likelihood quantity can be calculated that maximizes the probability that the classification is correct. Of course, assumptions on the form of the density are questionable. One of the biggest potential advantages of artificial neural networks is the hope of eliminating the need for specific code development for a given discrimination task. Let the network make the probability estimates and computer the distance. We can show that when networks are trained to minimize the mean squared error, they become functionally equivalent to the common Bayes discriminant functions.

For this example, we've provided some obvious features. In feature extraction, a set of primitives is extracted from the data and compared to some internal representation of classes. These primitives, along with information about their relationships that distinguish the classes of interest, could be used for classification. The primitives could in fact be statistical measures. For example, the moments of the data could be calculated. For a two dimensional shape, $f(x, y)$, the mnth moment is defined below as mot_{mn}:

$$mot_{mn} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^m y^n f(x,y)dxdy \qquad (1.2)$$

Alternately there exist invariant moments that could have been calculated. The encoding of position, scale and rotation invariance in the extracted features might be preferable to making the artificial neural networks learn the invariance from the raw input images. The advantage of moments is that they can be computed optically, and in fact one may want to compute moments in both the Fourier and space domains to enhance recognition. Feature extraction is very important. Good features make good recognition.

Comparison of artificial neural network to other solution techniques is very instructive. It is easy to construct a simple neural network to solve the two-class problem.

The artificial neural network solution to the good or bad bottle problem could then be compared to the K-nearest neighbor to the density function estimator, the Bayesian classifier.

1.3 Artificial Neurons and Neural Networks

Artificial neurons and neural nets [13–16] are both aspects of a style of computing (some people call it neural computing) that attempts to mimic (to a greater or lesser extent) the activities of animal neural systems, like the human brain. The motivation for doing this varies from person to person. Some people use neural computing as a way to examine real neural systems. These people are not really interested in the artificial neural systems for their own sake; they are using artificial systems because the real animal neural systems are difficult to use in research. The real systems tend to be very large and complicated, with millions of interrelated neurons. Real neurons are so small and delicate that it is difficult to isolate them for study. Individual tasks done by real neural systems are often distributed in unknown ways throughout the entire system in such a fashion that it is almost impossible to tell which neurons are participating in the task and which are not. There are also legal and ethical problems involved in the study of living animal neural systems that make such research awkward. We cannot, for example, remove bits of a living human brain in order to see what functions of speech or cognition stop working. Other people want to get computers to perform actual tasks, doing things for us that we previously would have had to do ourselves. Without the techniques of neural computing, we can have the computer do only what we can describe algorithmically, that is, with a very complete and explicit set of instructions. As we shall see, neural computing allows

us to 'train' the computer to do tasks without us really having to understand how the tasks are being accomplished. There are products on the market today that employ neural computing. And, of course, there are other people who examine neural computation because they are interested in it for its own sake. They want to know its characteristics and limitations; the kinds of problems it can solve and the kinds of problems it can't solve. Their interest is like the theoretical chemist's interest in the interaction of matter; if it has a practical application down the road that's great, but the first interest is in the basic science of the field of study. Neural computing is usually considered to be a part of artificial intelligence.

Artificial Neurons An artificial neuron is a machine-based object that mimics both the internal and external behavior of a real animal neuron. It will have a variable number of inputs and outputs. The artificial neuron will have some process by which it combines all of the inputs with weights, called the integration step. The value resulting from this process is fed to a threshold function which will determine if the neuron produces output. Diagrammatically, we might draw a single neuron as follows:

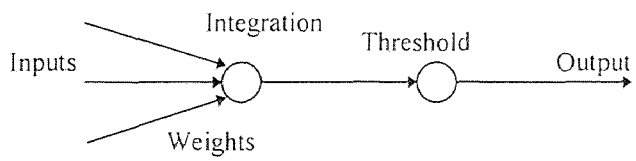


Figure 1.3 Single Neuron

However, we usually represent a neuron by a single dot. In most cases, a single neuron is so simple that it can't do very much. However, it is possible to get useful work even out of one neuron. Let's examine a neural system consisting of one neuron. The neuron will

have three inputs, the (x, y, z) coordinates of a point in three-dimensional space, and one output, +1 or −1. We want the neuron to tell us if the point given as input is above the plane $(x + y + z = 0)$ or below it by giving an output of +1 if the input point is above the plane and −1 if the input point is below the plane. In this case, the neuron's processing need only consist of only two steps. First, in the integration step, the input values are simply added together. The threshold function is uncomplicated. The resulting value is checked. If it is positive, then the neuron sets the output value to +1. If the result of the addition is negative, the neuron sets its output to −1. To the practical mind of the engineers in the crowd, it may seem that this simple one-neuron classifier doesn't really do anything useful. However, the three-dimensional space that it partitioned can represent any number of things. We could view the three dimensions as being the operating characteristics of some machine; temperature, pressure, and rotational speed. The plane represents the boundary between safe and unsafe conditions. The neuron's output could be connected to an emergency shutdown system. Whenever the combination of characteristics left the safe area, the neuron's output would change and activate the shutdown system. Of course, it may be that the boundary between safe and unsafe conditions may be too complicated to describe with such a simple classifier. In that case, either the neuron could have more complicated integration and threshold functions or we could add more neurons.

Neural Nets A neural net is a collection of some number of neurons along with the connections between them. Also referred to as connectionist architectures, parallel distributed processing, and neuromorphic systems, an artificial neural network (ANN) is an information-processing paradigm inspired by the way the densely interconnected,

parallel structure of the mammalian brain processes information. Artificial neural networks are collections of mathematica models that emulate some of the observed properties of biological nervous systems and draw on the analogies of adaptive biological learning. The key element of the ANN paradigm is the novel structure the information processing system. It is composed of a large number of highly interconnected processing elements that are analogous to neurons and are tied together with weighted connections that are analogous to synapses. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. This is true of ANNs as well. Learning typically occurs by example through training, or exposure to a truthed set of input/output data where the training algorithm iteratively adjusts the connection weights. These connection weights store the knowledge necessary to solve specific problems. Although ANNs have been around since the late 1950's, it wasn't until the mid-1980's that algorithms became sophisticated enough for general applications. Today ANNs are being applied to an increasing number of real world problems of considerable complexity. They are good pattern recognition engines and robust classifiers, with the ability to generalize in making decisions about imprecise input data. They offer ideal solutions to a variety of classification problems such as speech, character and signal recognition, as well as functional prediction and system modeling where the physical processes are not understood or are highly complex. ANNs may also be applied to control problems, where the input variables are measurements used to drive an output actuator, and the network learns the control function. The advantage of ANNs lies in their resilience against distortions in the input data and their capability of learning. They are often good at solving problems that are too complex for conventional technologies (e.g., problems that

do not have an algorithmic solution or for which an algorithmic solution is too complex to be found) and are often well suited to problems that people are good at solving, but for which traditional methods are not. There are different types of ANNs. Some of the more popular include the multilayer perceptron which is generally trained with the backpropagation of error algorithm, learning vector quantization, Hopfield, and Kohonen network. 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.

Implementation Each neuron could be represented entirely by specialized hardware (wires, silicon transistors, or chips) or entirely in software (a collection of data and the instructions to manipulate it) running on a general-purpose computer, or as some combination of the two. A hardware implementation is usually very fast but quite expensive to build, and it is difficult to change the structure of the net once it is built. Software implementations are usually comparatively slow, but the structure and characteristics of each neuron and of the net as a whole can be easily changed. Researchers tend to use software implementations almost exclusively. Engineers trying to build neural net controlled devices will also often use software implementations while

they are designing and testing the nets. When the design is finished and a software prototype tested, hardware versions can be built. For our purposes, it doesn't really matter how the neurons are implemented; hardware, software, or a combination. The capabilities of the system don't really change, just how fast it runs and how difficult it would be to build.

Comparison Finally, let's compare the ANN and the conventional computer. A serial computer has a central processor that can address an array of memory locations where data and instructions are stored. Computations are made by the processor reading an instruction as well as any data the instruction requires from memory. The instruction is then executed and the results are saved in a specified memory location as required. In a serial system, the computational steps are sequential and logical, and the state of a given variable can be tracked from one operation to another. In comparison, ANNs are not sequential but parallel. There are no complex central processors, instead there are many simple ones which generally do nothing more than take the weighted sum of their inputs from other processors. ANNs do not execute programmed instructions, they respond in parallel (either simulated or actual) to the pattern of inputs presented to it There are also no separate memory addresses for storing data. Instead, information is contained in to overall activation 'state' of the network. 'Knowledge' is thus represented by the network itself. ANNs can deal with `unseen' patterns and generalize from the training set. It is robust in the presence of noise, small changes in an input pattern will not drastically affect a node's output. ANNs are good at "perceptual" tasks and associative recall. These are just the tasks that the symbolic approach has difficulties with.

CHAPTER 2

CANDIDATE SEARCHING

The neural network is used in the gate symbol recognition in this system. The calculation of the neural network involves quite a lot floating point calculations and it takes longer time. On the other hand, there are only a few gate symbols on a diagram. It is not practical to use neural network to search the gate symbols all over the diagram. That will waste a lot of time and make the processing very slow.

In order to save the whole processing time, reduce the calculation time spent in neural network is crucial. We have two choices. One is to reduce the time spent in each neural network recognition. The other is to reduce the times using neural network. The latter one is suitable and easy to implement.

Before using neural network to recognize gate symbols, a pre-processing [17–20] can be implemented to pinpoint them. This procedure, known as candidate searching, is to use a few fast algorithms to quickly find out the gate symbol positions. After that, neural network will be connected to those areas one after another to detect whether they are gate symbols and what kind of gate symbols they are. In this way, the overall time spent in the gate symbol recognition is greatly lessened. The candidate searching consists of 3 parts. They are long line removal, thinning and pyramid algorithms.

2.1 Long Line Removal

In each digital diagrams, a part of them shown in Figure 2.1, 4 types of elements exist. They are straight lines, gates symbols, characters and dots (intersections points). Straight lines are divided into long lines and short lines. We define the long lines are those whose length is longer than the length or width (whichever is longer) of the gate symbols. Usually long lines may occupy up to 90% of "1" pixels. However, for the gate symbol recognition, lines are useless. In order to implement the succeeded algorithm, straight long lines have to be removed as more as possible. This makes the searching algorithm more accurate and fast.

Long line removal algorithm:

input: digital diagram image and the length of the gate

output: image without long lines

```
begin
    while pixel is on the image do
        detect the horizontal lines
        for each detected horizontal line do
            if the length of the horizontal line > length of the gate then
                make the marks b on each pixels
            end if
        end do
        detect the vertical lines
        for each detected vertical line do
```

```
            if the length of the vertical line > length of the gate then
              make the marks b on each pixels
            end if
        end do
    end do
    while pixel is on the image do
        if b then remove the line
        end if
    end do
end
```

Figure 2.2 shows the long line removed image.

2.2 Thinning

Thinning [19] is an operation in which binary-valued image regions are reduced to lines that approximate the center lines of the regions. The purpose of thinning is to reduce the image components to their essential information so that further analysis and recognition are facilitated. After thinning, it is easier and faster to process 1-pixel-wide lines than the wider ones.

In this system, the thinning is particularly important because this can make neural network input window and the neural network size smaller. Otherwise, the time used in the network training and network recognition would be much longer. If a supercomputer was used to do the training and recognition, a large size neural network could be designed to recognize arbitrary pixel-wide gate symbols.

The thinning requirements are stated as follows:

(1) Connected image regions must thin to connected line structures.

(2) The thinning results should approximate the medial lines.

(3) Approximate end line locations should be maintained.

The image pattern contour points are assumed to be "1" and the background points "0". The algorithm consists of successive passes of two basic steps applied to the contour points of the image, where a contour points is any pixel with value "1", having at least one 8-neighbor value "0". With reference to the 8-neighbor definition, refer to Figure 2.3, step 1 flags a contour

p9	p2	p3
p8	p1	p4
p7	p6	p5

Figure 2.3 8-neighbor definition

point p for the deletion if the following conditions are satisfied:

(a) 2 =<N(p1) =< 6;

(b) S(p1) = 1;

(c) p2 * p4 * p6 = 0;

(d) p4 * p6 * p8 = 0;

where N(p1) = p2 + p3 + p4 + p5 + p6 + p7 + p8 + p9; S(p1) is the number of 0-1 transitions in the ordered sequence of p2, p3, ... p8, p9,p2. In step two, conditions (a) and (b) remain the same, while conditions (c) and (d) are changed to

(c') p2 * p4 * p8 = 0;

(d') p2 * p6 * p8 = 0;

Step 1 is applied to every border pixel in the binary region under consideration. If one or more of conditions (a)-(d) are violated, the value of the point in question is not changed. If all conditions are satisfied, the point is flagged for deletion. However, the point is not deleted until all border pointed have been processed. This delay prevents changing the structure of the data during execution of the algorithm. After step 1 has applied to all border points, those that have been flagged are deleted. Then, step 2 is applied to the resulting data in exactly the same manner as step 1. Applying step 1 and 2 can satisfy the above 3 conditions to get the desired thinning image. These 2 steps are applied iteratively until no further points to be deleted in the image, at that moment the thinning process terminates, yielding the thinned image. Figure 2.4 shows the thinned image.

2.3 Pyramid (Multi-Resolution) Algorithm

Pyramid algorithm is an operation which lower the image resolution. It creates the new image's pixels by adding the values of 4 connected pixels in the higher resolution image. The new pixel corresponds to 4 pixels at the finer resolution image. Each time, the image size is reduced to 1/4 of the higher resolution image. After a specific times of operation, the gate symbols are shrunk to single pixels. Therefore, the central coordinates and the position of the candidate gate symbols are found.

Pyramid algorithm:

input: long line removed digital diagram image, the length of the gate output: low resolution $(1/4)^n$ sized of input image marked with candidate gate position

where n = 1, 2, 3, 4, 5, …

```
begin while n = n –1 do
    for each i, j do
      p[i][j] = p[2*i][2*j]+p[2*i+1][2*j]+p[2*i][2*j+1]+p[2*i+1][2*j+1]
    end do
    i=i/2;
    j=j/2;
  end do
  for each i, j do
     if p[i][j]> 3×length of gate then
     mark i, j as a candidate position in the low resolution image
   end do
end
```

Figure 2.5 illustrates the procedure of this algorithm.

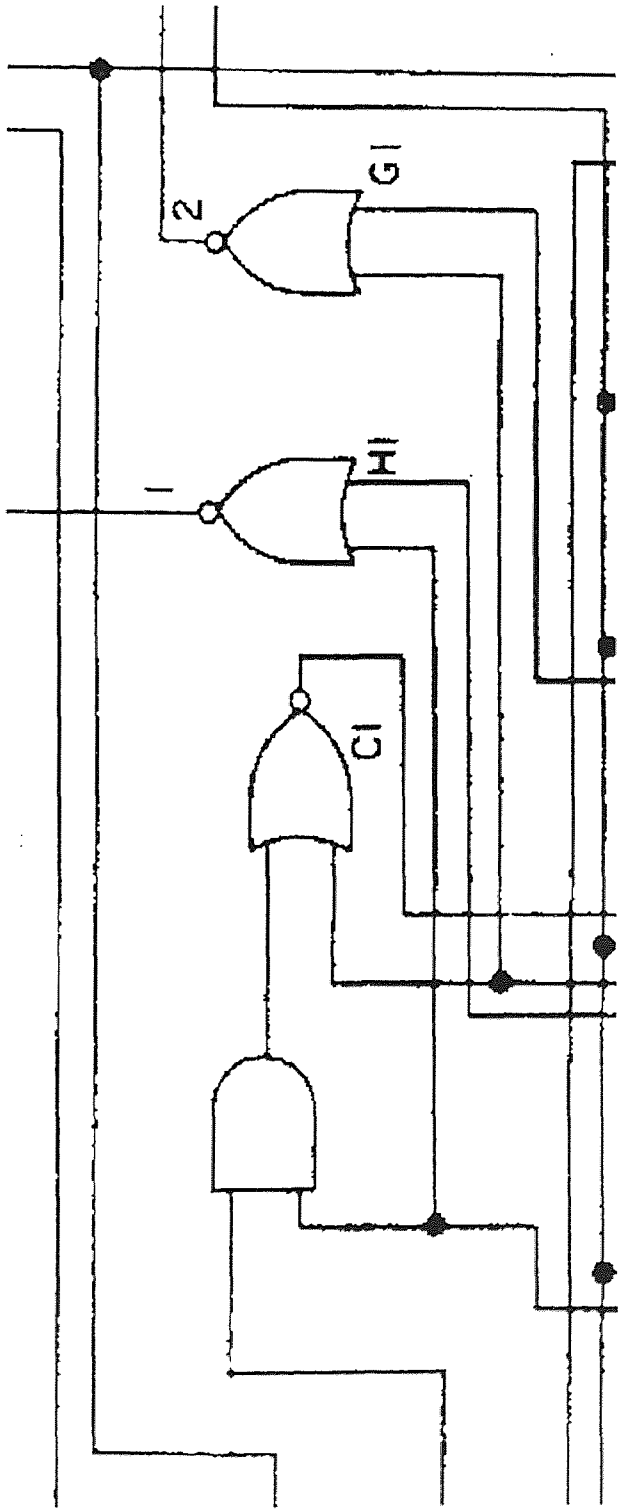


Figure 2.1 Part of a Digital Diagram

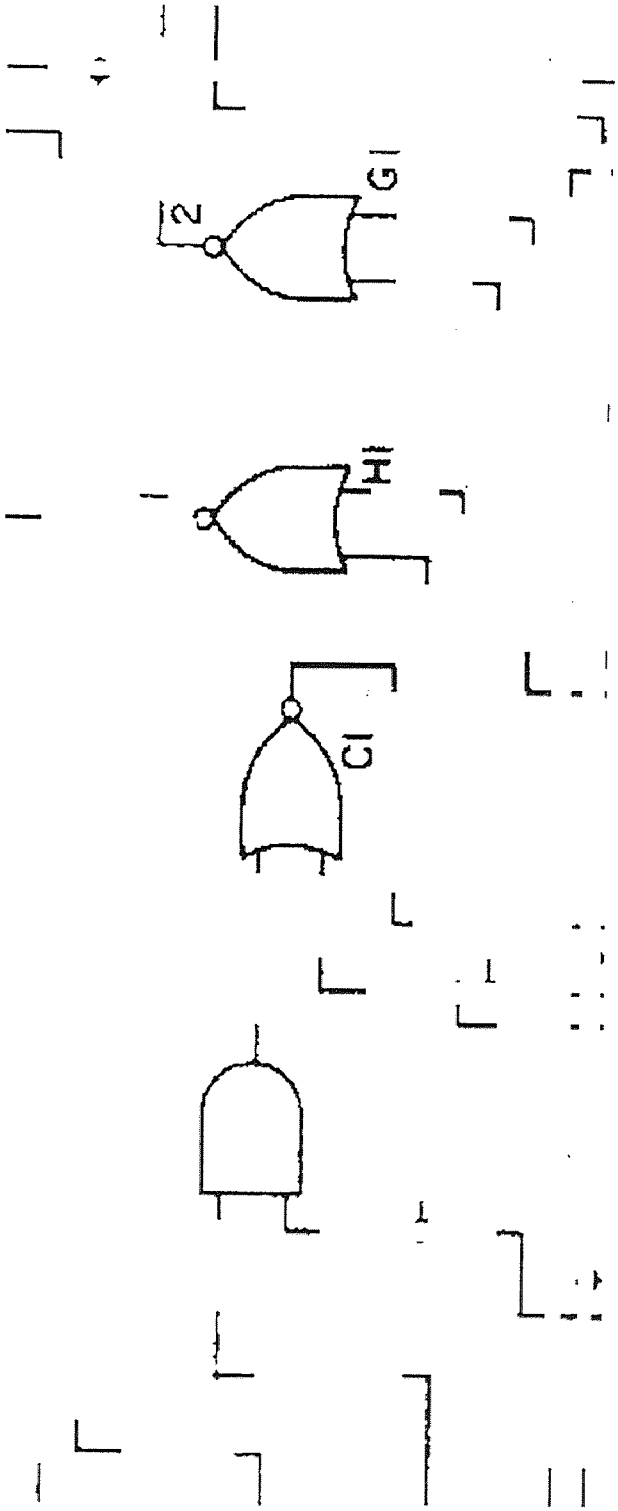


Figure 2.2 Long Line Removed Image

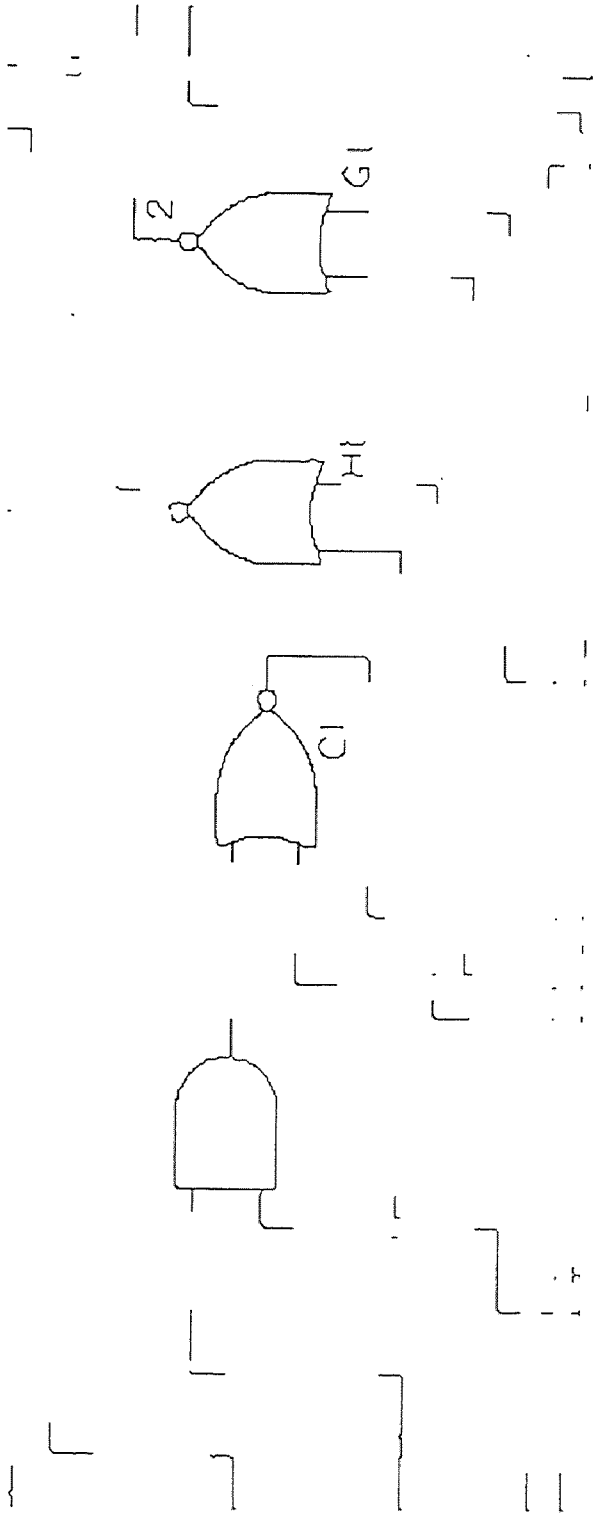


Figure 2.4 The Image After Thinning

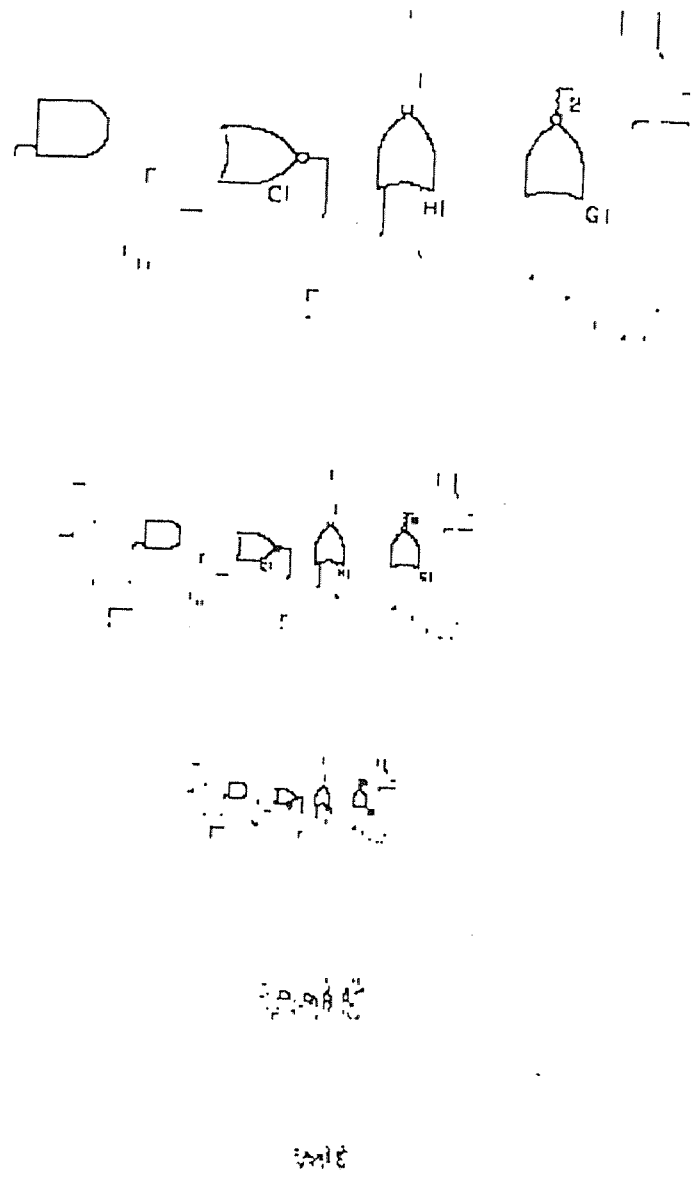


Figure 2.5 Images of Different Resolution

CHAPTER 3

A SHIFT-INVARIANT AND LIMITED ROTATION-INVARIANT MODIFIED KOHONEN NEURAL NETWORK

3.1 Introduction

An important feature of neural network is the ability to learn from their environment, and through learning to improve performance in some sense. In this chapter, we present a modified Kohonen neural network with shift-invariant and rotation-invariant capability. Kohonen neural network is a type of unsupervised neural network. It is [21] perhaps the simplest self-organization system [22], consists of a single layer of neurons (called Kohonen layer) and an input buffer layer that is fully connected to the neurons in the Kohonen layer through adjustable weights (see Figure 3.1).

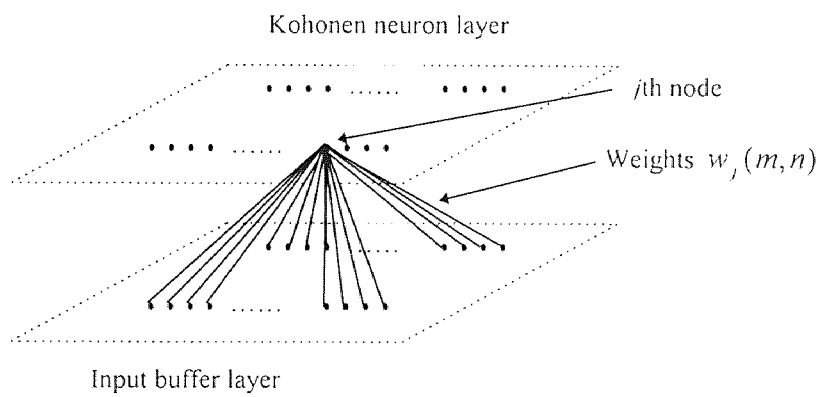


Figure 3.1 Structure of Kohonen Neural Network

Kohonen neural network has been successfully applied to speech recognition [21,23]. It is also utilized as a pre-processing layer in a more complicated neural network for image recognition [24]. A well trained Kohonen network, for example, can be used to classify English letters and other characters [25,26]. In our researches aiming at recognizing the

digital gate symbols and characters in electronic drawings, it was found, however, that when an object in an image is shifted horizontally and/or vertically, Kohonen network cannot classify the original image pattern and its shifted version into the same category. It is clear that the shift-invariant capability in image pattern recognition is desired in many applications. For this purpose, a modified shift-invariant Kohonen network with a two-dimensional correlation is proposed in this chapter. Owing to its shift-invariant capability, the dimension of the Kohonen neuron layer can be reduced dramatically. Moreover, the size of its subsequent neural network can be decreased significantly. An analysis of computational complexity and some experimental work are presented. The experiments have shown that when the shift in an image pattern is not large, the shift-invariant modified Kohonen neural network is superior to the original Kohonen neural network. The rotation-invariant capability is another modification to the original ones for the $n \times 90°$ rotations (n=1, 2, and 3).

In order to make the discussion subsequently more concrete, we shall consider how image information is captured and input to a neural network. Suppose we have a TV camera (monochrome for simplicity) which is viewing a picture that is to be used in training. The output from this is a picture where each point is represented by a continuously variable voltage (analogue quantity) so that shades of gray may be encoded accurately. For a node in Kohonen input buffer layer or a perceptron, however, we require a set of Binary values ('1', '0'). The conversion process is done by dividing the picture into a grid of picture elements or pixels each of which is allowed to take only one of two values black or white. To find the value for each pixel, the average value of the image in the pixel area is found and then threshold to determine whether it is white or black. We

now make the correspondence white = '1', and black = '0'. This array of Boolean quantities may now be stored in a special purpose computer memory. Typically the pixel grid may be 512 by 512 giving over 0.26 million pixels. Thus, the pattern space will have dimension 0.26 million. This is often reduced to make things more manageable for a neural network. In this research aiming at recognizing different digital gate symbols, the pixel grid is set to 32 by 32. The total number of pixels is 1024.

3.2 Conventional Kohonen Neural Network

Before we discuss conventional Kohonen neural network, let us observe the K-means algorithm [27] as the foundation. Pattern vector of n-dimensions may be considered as representing point within an n-dimensional Euclidean space. One of the most obvious means by which we may establish a measure of similarity between or among such pattern vectors is by means of their proximity to one another. The K-means algorithm is one of the many clustering techniques that shares this notion of clustering by minimum distance. Namely, vectors which identify points that are geometrically close together may be taken in some sense as belonging together. For presenting the operation of the K-means algorithm, a precise notion of distance metric is needed. The Euclidean norm of a vector, $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ is defined as follows:

$$\|\mathbf{x}\| = \left[\sum_{i=1}^{n} x_i^2 \right]^{1/2} \tag{3.1}$$

Equation (3.1) provides the length of the vector **x**. Since we desire the distance or length between two vectors within the pattern space, we need to apply equation (3.1) to the vector difference as follows:

$$\|\mathbf{x}-\mathbf{z}\| = \left[\sum_{i=1}^{n} (x_i - z_i)^2 \right]^{1/2} \tag{3.2}$$

where **x** and **z** are pattern vectors of order n.

Now that a measure of pattern similarity has been established, the task of establishing a procedure by which patterns are partitioned into cluster domains must be undertaken. That is, we require a procedure that will establish a set cluster with associated cluster centers such that the distance between an input vectors and the closet cluster center serves to classify the vector. The *K*-means algorithm represents one such method.

K-means makes the assumption that the number of cluster centers that will be required to adequately represent the sample space is known *a priori*. This assumption in itself somewhat limits the utility of the procedure. Other variations on minimum distance statistical clustering techniques lack this difficulty but may have other problems such as a sensitivity to the order in which input data are presented to the system.

Let $\mathbf{x}^{(P)}$ represent the pth space vector. The complete set of input vector will then be $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(P)}\}$. The vector **z** represents the cluster center for each of the *K* clusters. That is it points to the position in Euclidean space at which the cluster center is located. Since there are *K* cluster centers: Finally the notation, $S_j = \{ \mathbf{x} \mid \mathbf{x} \text{ is closest to cluster } j\}$ will be used to represent the set of samples that belong to the jth cluster center. The K-means algorithm is implemented in the following steps.

<u>Step 1. Initialization</u> choose the number of cluster *K*. For each of these *K* clusters choose an initial cluster center: $\{ z_1(l), z_2(l), \ldots, z_k(l)\}$, where $z_j(l)$ represents the value of the cluster center at the lth iteration. The starting values can be arbitrary but are generally taken to be the value of the first *K* of the sample vector.

Step 2. Sample distribution: Distribute all sample vectors. By this, each sample vector $\mathbf{x}^{(p)}$ is attached to one of the K clusters according to the following criteria:

$$\mathbf{x}^{(p)} \in S_j(l) \quad \text{if } \left\|x^{(p)} - z_j(l)\right\| < \left\|x^{(p)} - z_i(l)\right\| \tag{3.3}$$

for all $i = 1, 2, \ldots, K, \; i \neq j$

$S_j(l)$ represents the population of cluster j at iteration l.

Step 3. New cluster centers calculation: Using the new cluster membership sets established in step 2, recalculate the position of each cluster center such that the sum of the distances from each member vector to the new cluster center is minimized. Specifically we wish to minimize J_j where:

$$J_j = \sum_{x^{(p)} \in S_j(l)} \left\|x^{(p)} - z_j(l+1)\right\|^2 \tag{3.4}$$

where j=1, 2, …, K. The value of $\mathbf{z}_j(l + 1)$ which minimizes equation (3.4) is simply the mean taken over the samples of $S_j(l)$. Therefore the new cluster center is calculated using equation (3.5) as follows:

$$z_j(l+1) = \frac{1}{N_j} \sum_{x^{(p)} \in S_j(l)} x^{(p)} \tag{3.5}$$

where N_j is the number of sample vectors attached to S_j during step 2.

Step 4. Convergence checking: The condition for convergence is that no cluster center has changed its position during step 3. This condition can be expressed mathematically as follows:

$$z_j(l+1) = z_j(l) \qquad j = 1, 2, \ldots, K \tag{3.6}$$

If equation (3.6) is satisfied, then convergence has occurred. Otherwise iteration by going to step 2.

A number of factors may influence the behaviors of the K-means algorithm. Among these are the number of cluster centers, the choice of initial cluster centers, and the geometric properties of the input data. Some experiments with the choice of K and the initialization parameters may be required. Although no formal proof of convergence exists, the method can be expected to do well where the nature of the data is consistent with the assumption inherent in using the minimum distance as a similarity measure.

We begin by illustrating the ability of Kohonen network model to identify cluster centers just as the K-means algorithm did. The Kohonen network architecture consists of two layers, an input buffer layer and a Kohonen neuron layer (output layer). These two layers are fully connected. Each input layer neuron has a feed-forward connection to each output layer neuron. Refer to Figure 3.1. There the Kohonen neuron layer is of two-dimension. Actually one-dimensional and higher dimensional cases are possible. The input vectors $\boldsymbol{x}$ are required to normalize (i.e., $\| \boldsymbol{x} \|=1$). Inputs to the Kohonen neuron layer can be calculated conventionally using equation 3.7:

$$I_j = \sum_{i=0}^{n-1} (w_{ij} x_i) \tag{3.7}$$

Applying a winner-take-all paradigm, the winning output layer neuron will simply be the neuron with the biggest I_j. The output of the winning neuron will be 1. All other neurons in the Kohonen layer will output nothing. In effect equation 3.7 is the dot product between a neuron weight vector and the input vector. Thus this method chooses a winning neuron such that the angle between the winning neuron weight vector and the input vector will be smaller than the corresponding dot product for all other neurons. An equivalent method of choosing the winning neuron simply selects the neuron whose

weight vector has a minimum of the Euclidean distance from the input vector (i.e., d_j = $\|w_j - x\|$). In the next section we will focus on the training of Kohonen neural network.

3.3 The Training of Kohonen Neural Network

Let us consider the training [28–33] of a Kohonen neural network with an input buffer layer and a Kohonen neuron layer that are fully connected. An input vector is applied to the buffer layer, and its component vectors are transmitted to each neuron in the Kohonen layer through randomized connecting weights. The neuron in the Kohonen layer with the strongest response is declared the winner and its value is set equal to one. Then the weights connecting all component vectors from the buffer layer to winning neuron undergo training in accordance with the process shown in Figure 3.2.

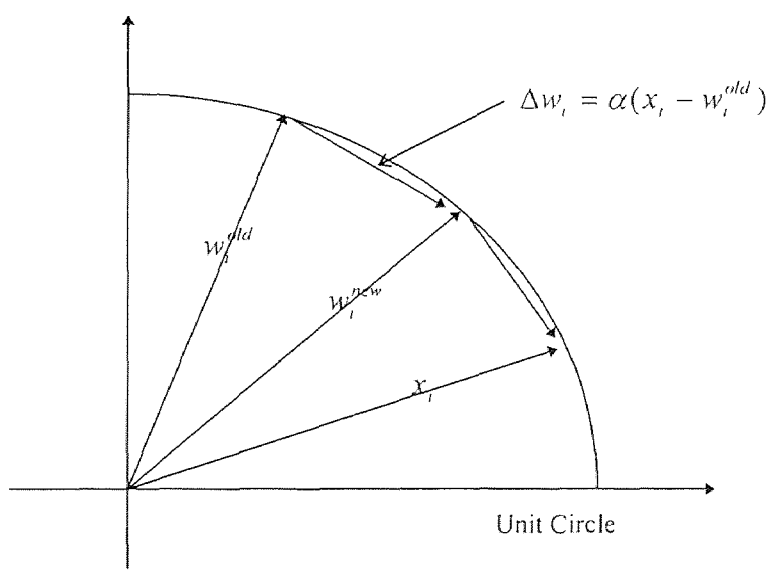


Figure 3.2 The weights update in Kohonen Network

Neurons adjacent to the winner are also allowed to undergo training. The second input vector is applied to the buffer layer, another neuron in the Kohonen layer is declared the

winner, This process continues until all the input data have been applied to the buffer layer and the distance between the input vectors and the weights are small enough. The training procedure is as follows:

(1) Initialization: assign small random values to all the weights and a specific value to the neighborhood size.

(2) Input: present an image array to the input buffer layer.

(3) compute distance to the jth node: $d_j = \sum_{i=0}^{N-1} (x_i - w_{ij})^2$

(4) Winner selection: select a node which corresponds to the highest correlation value as the winner.

(5) Weights update: update the weights of the winner and all the nodes within the neighborhood of the winner according to the following equation:

$$w_{ij}^{new} = w_{ij}^{old} + \alpha(x_i - w_{ij}^{old}) \tag{3.8}$$

where α is decreased over the learning as the size of the neighborhood decreases. Note that there is also a parameter that must be initialized, that is the neighborhood. The size of the neighborhood determines the size of the area of the Kohonen layer centerd about the winner whose weights will be updated for a given input. Initially the size of the neighborhood should be some significant fraction of the initial neighborhood. The training schedule includes the reduction of the size of these neighborhoods as well as a reduction in the leaming parameter α. The next step in the leaming algorithm is to provide an input to the network. Note that no classification is provided to this network, thus it is unsupervised learning. In the supervised learning paradigms, like the backward error propagation algorithm, the desired output for the network was provided. The next

step of the algorithm is to compute the distance of the input from all of the nodes in the Kohonen layer. If N is the number of inputs to the network, indexed by i, and j is the index over the output nodes, the two-dimensional Kohonen layer, the distance of some input from the jth Kohonen node is

$$d_j = \sum_{i=0}^{N-1} (x_i - w_{ij})^2 \tag{3.9}$$

This is a measure of the similarity between the input and the weights of a given node on the Kohonen output layer. The last step in the learning algorithm is to update the weights for the winning node, that node with the minimum distance. The weights are also updated for the nodes that lie within the neighborhood of the winning node. Finding the winning node is common in neural-processing systems and can be accomplished with lateral connections in the Kohonen layer. Quite often the Kohonen networks are drawn with connecting arcs drawn between the nodes on the output. Layer. These connections could accomplish the gaming weight adaptation and the picking of the winner. The weight update equation is:

$$w_{ij}^{new} = w_{ij}^{old} + \alpha(x_i - w_{ij}^{old}) \tag{3.10}$$

Let us think of the input vector, $\boldsymbol{x}$, as being the desired result for the weight vectors $\boldsymbol{w}$, being updated. This update equation pushes each of the vectors being updated toward the desired input, $\boldsymbol{x}$, along the vector that represents the vectoral difference. After all the weights within the neighborhood [34–40] of winning node are updated, another input is chosen and the process is repeated. For each repetition the learning parameter could be slightly reduced linearly. After some number of training inputs the neighborhood size could also be decreased. We commonly used a starting α of about 0.9, which is then

linearly reduced. For example, if training a 10×10 Kohonen output layer, starting with a initial neighborhood size of 7×7 might be appropriate, and after a few thousand training iterations, this neighborhood size might be reduced to 5×5. When the weights of the network are not changing significantly, and the distance between input patterns and the corresponding weights are small enough, the network is done with learning.

3.4 Similarity Measurement between Images

The two dimensional correlation function proposed in this research is defined as follows:

$$f(x,y) \circ g(x,y) = \sum_{m=0}^{P-1} \sum_{n=0}^{Q-1} f(m,n) \cdot g(x+m, y+n) \tag{3.11}$$

where the symbol "$\circ$" represents correlation, $f(m,n)$ is an unknown input image pattern array, and $g(m,n)$ is a reference image pattern or template array. Both of them are of size $P \times Q$, refer to Figure 3.3. It is assumed that $f(m,n)$ and $g(m,n)$ are equal to 0 if $m \geq P$ or $m < 0$, or $n \geq Q$ or $n < 0$, $-(P-1) \leq x \leq P-1$ and $-(Q-1) \leq y \leq Q-1$.

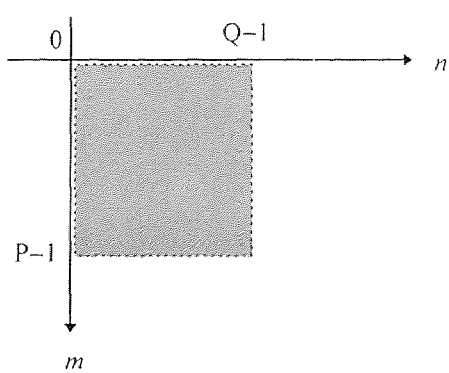


Figure 3.3 The image array of $f(m,n)$ and $g(m,n)$

From equation (3.11) we calculate the correlation starting with $f(m,n)$ and $g(m,n)$. We label the axes m and n, because we are going to sum over the plane, where the dummy variable m and n will disappear, and we wish to be left with a function of x and y. $g(x+m, y+n)$ is a replica of $g(m,n)$ but displaced by an amount y to the left and an amount x upward. We may represent this situation by copying the image $g(m,n)$ on a piece of transparency and displacing it. If we move the transparency on the top of the original and then displacing it, we have the value the value $g(x+m, y+n)$. It is multiplied by $f(m,n)$. We can imagine the product as a new function covering the (x, y)-plane. According to the defined equation, we perform the double summation, that is, to find the volume under the product function. That needs a lot of work involving multiplications over the area $0 \le m \le P-1$ and $0 \le n \le Q-1$, and followed by summing the products on the plane. Even so, the result is merely a single value of correlation, namely for the particular (x, y) describing the displacement. To get another value we have to displace the function to a new position, multiply throughout, and find the volume under the production again. To obtain the whole correlation function, we have to repeat over and over again until all the desired values of x and y, $-(P-1) \le x \le P-1$ and $-(Q-1) \le y \le Q-1$, are covered.

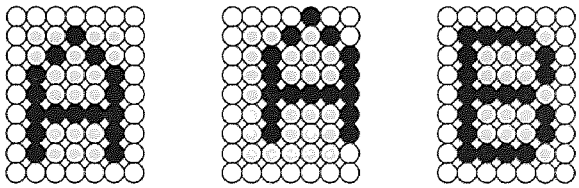
The correlation is utilized in image processing to evaluate the match between an unknown pattern and a reference pattern, where the problem is to find the closest match between an unknown image and a set of known images. One approach is to compute the correlation between the unknown image and each of the known images. The closest match can then be found by selecting the image that yields the correlation with the largest

value. Because the resultant correlation are two dimensional functions, this involves searching for the largest amplitude of each function.

Calculating the correlation defined in (3.11) for all the possible (x, y) is equivalent to shifting the reference pattern all over the possible locations on the unknown image pattern and find the similarity measurement. During the process, if and only if the two image patterns are identical and overlapped, $f \circ g$ yields the largest possible correlation value 1. Otherwise, the correlation between any two different patterns cannot reach this maximum value. In Figure 3.4, there are 3 English letter image patterns, letter "A", letter "B", and shifted letter "A". The calculation of correlation between them are shown in Table 3.2, 3.3 and 3.4. The maximum correlation values are shown in Table 3.1.

Table 3.1 Comparison between Euclidean distance and correlation

Measure between letters	Euclidean distance	Correlation
"A" and "A"	0.000	1.000
"A" and "B"	0.994	0.783
"A" and shifted "A"	1.500	1.000



(a) (b) (c)

Figure 3.4 (a) Letter "A", (b) Shifted letter "A", (c) Letter "B"

As a result, we can determine if the input pattern is identical to (Table 3.2) or is a shifted version (Table 3.4) of the reference pattern. That is, we can achieve the shift-invariant capability in recognition. Furthermore, if $f(m,n)$ is a shifted version of $g(m,n)$, the relative position (x, y) between f and g can be found. However, this is not our direct interest in this recognition process.

It is also noted that the 2-D correlation proposed in (3.11) is different from that defined in the typical digital signal and/or image processing texts, say in [10]. There, both $f(m,n)$ and $g(m,n)$ are first extended into 2-D arrays of $2P - 1$ by $2Q - 1$ for the sake of periodicity in both spatial and frequency domains. In our case, there is no such need. But in terms of recognition ability, these two correlation techniques are the same.

Euclidean distance is another way to measure the distance between two vectors: $\underline{x}$ and $\underline{w}$.. The definition is as follows:

$$d = \{\sum_{i=0}^{N-1}(x_i - w_i)^2\}^{1/2} \tag{3.12}$$

We will discuss it in detail in the next section.

It is worth noting that when using the Euclidean distance measure, the smaller value indicates the two images are more similar to each other. With the correlation, it is just the opposite: The larger the correlation, the more similar the two images are. It is easy to prove that zero Euclidean distance implies that the correlation equals to 1. In fact, having the Euclidean distance equal to 0 is just a special case of the correlation equals to 1 (when there is no shift).

Table 3.2 Correlation between letter "A" and itself

	y=6	y=5	y=4	y=3	y=2	y=1	y=0	y=-1	y=-2	y=-3	y=-4	y=-5	y=-6
x=8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
x=7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
x=6	0.00	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.00
x=5	0.00	0.00	0.00	0.06	0.06	0.06	0.00	0.06	0.06	0.06	0.00	0.00	0.00
x=4	0.00	0.00	0.06	0.06	0.06	0.12	0.19	0.12	0.06	0.06	0.06	0.00	0.00
x=3	0.00	0.00	0.12	0.06	0.12	0.12	0.38	0.12	0.12	0.06	0.12	0.00	0.00
x=2	0.00	0.00	0.19	0.19	0.19	0.19	0.38	0.19	0.19	0.19	0.19	0.00	0.00
x=1	0.00	0.00	0.25	0.19	0.12	0.25	0.50	0.25	0.12	0.19	0.25	0.00	0.00
x=0	0.00	0.00	0.31	0.12	0.25	0.25	1.00	0.25	0.25	0.12	0.31	0.00	0.00
x=−1	0.00	0.00	0.25	0.19	0.12	0.25	0.50	0.25	0.12	0.19	0.25	0.00	0.00
x=−2	0.00	0.00	0.19	0.19	0.19	0.19	0.38	0.19	0.19	0.19	0.19	0.00	0.00
x=−3	0.00	0.00	0.12	0.06	0.12	0.12	0.38	0.12	0.12	0.06	0.12	0.00	0.00
x=−4	0.00	0.00	0.06	0.06	0.06	0.12	0.19	0.12	0.06	0.06	0.06	0.00	0.00
x=−5	0.00	0.00	0.00	0.06	0.06	0.06	0.00	0.06	0.06	0.06	0.00	0.00	0.00
x=−6	0.00	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.00
x=−7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
x=−8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 3.3 Correlation between letter "A" and letter "B"

	y=6	y=5	y=4	y=3	y=2	y=1	y=0	y=-1	y=-2	y=-3	y=-4	y=-5	y=-6
x=8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
x=7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
x=6	0.00	0.00	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.00	0.00	0.00
x=5	0.00	0.00	0.11	0.06	0.06	0.06	0.17	0.06	0.06	0.06	0.06	0.00	0.00
x=4	0.00	0.00	0.17	0.11	0.17	0.22	0.45	0.17	0.11	0.06	0.11	0.00	0.00
x=3	0.00	0.00	0.22	0.17	0.17	0.17	0.34	0.17	0.17	0.17	0.11	0.00	0.00
x=2	0.00	0.00	0.28	0.17	0.17	0.17	0.45	0.17	0.17	0.17	0.17	0.00	0.00
x=1	0.00	0.00	0.28	0.17	0.22	0.34	0.78	0.22	0.17	0.06	0.22	0.00	0.00
x=0	0.00	0.00	0.28	0.22	0.22	0.28	0.50	0.28	0.17	0.22	0.17	0.00	0.00
x=−1	0.00	0.00	0.22	0.22	0.22	0.22	0.34	0.22	0.22	0.22	0.11	0.00	0.00
x=−2	0.00	0.00	0.17	0.17	0.28	0.34	0.56	0.22	0.22	0.06	0.11	0.00	0.00
x=−3	0.00	0.00	0.11	0.11	0.11	0.17	0.22	0.17	0.06	0.11	0.06	0.00	0.00
x=−4	0.00	0.00	0.06	0.11	0.11	0.11	0.06	0.11	0.11	0.11	0.00	0.00	0.00
x=−5	0.00	0.00	0.00	0.06	0.11	0.11	0.11	0.06	0.11	0.00	0.00	0.00	0.00
x=−6	0.00	0.00	0.00	0.00	0.06	0.06	0.06	0.06	0.00	0.00	0.00	0.00	0.00
x=−7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
x=−8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 3.4 Correlation between letter "A" and shifted letter "A"

	y=6	y=5	y=4	y=3	y=2	y=1	y=0	y=-1	y=-2	y=-3	y=-4	y=-5	y=-6
x=8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
x=7	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.06	0.00	0.00	0.00
x=6	0.00	0.00	0.00	0.00	0.06	0.06	0.06	0.00	0.06	0.06	0.06	0.00	0.00
x=5	0.00	0.00	0.00	0.06	0.06	0.06	0.12	0.19	0.12	0.06	0.06	0.06	0.00
x=4	0.00	0.00	0.00	0.12	0.06	0.12	0.12	0.38	0.12	0.12	0.06	0.12	0.00
x=3	0.00	0.00	0.00	0.19	0.19	0.19	0.19	0.38	0.19	0.19	0.19	0.19	0.00
x=2	0.00	0.00	0.00	0.25	0.19	0.12	0.25	0.50	0.25	0.12	0.19	0.25	0.00
x=1	0.00	0.00	0.00	0.31	0.12	0.25	0.25	1.00	0.25	0.25	0.12	0.31	0.00
x=0	0.00	0.00	0.00	0.25	0.19	0.12	0.25	0.50	0.25	0.12	0.19	0.25	0.00
x=−1	0.00	0.00	0.00	0.19	0.19	0.19	0.19	0.38	0.19	0.19	0.19	0.19	0.00
x=−2	0.00	0.00	0.00	0.12	0.06	0.12	0.12	0.38	0.12	0.12	0.06	0.12	0.00
x=−3	0.00	0.00	0.00	0.06	0.06	0.06	0.12	0.19	0.12	0.06	0.06	0.06	0.00
x=−4	0.00	0.00	0.00	0.00	0.06	0.06	0.06	0.00	0.06	0.06	0.06	0.00	0.00
x=−5	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.00	0.00	0.06	0.00	0.00	0.00
x=−6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
x=−7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
x=−8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

3.5 Shift-Invariant Modified Kohonen Network

In the Kohonen network, the square of the Euclidean distance [27] (referred to as the Euclidean distance measure in the rest of this thesis) is used to measure the distances between an input vector $\underline{x}$ and various weight vectors $\underline{w}_j$ in both the learning and classification processes:

$$d_j = \sum_{i=0}^{N-1} (x_i - w_{ij})^2 \tag{3.13}$$

where j is an index for neurons in the Kohonen neuron layer, x_i and w_{ij} are the ith components of $\underline{x}$ and $\underline{w}_j$ respectively. It works effectively when there is no relative shift between the input image pattern $\underline{x}$ and weight pattern $\underline{w}_j$. However, in the case when some shift does happen, this measure is not suitable. To see this, consider a simple example shown in Fig. 3.4 There the letter "A", the shifted letter "A" (1 pixel shift both horizontally and vertically), and the letter "B" are all shown with a dimension of 9×7.

These three binary images are normalized such that the sum of square of all the intensity values equals to 1. Under this circumstances, the Euclidean distance measure between letter "A" and its shifted letter "A" is as large as 1.500, whereas the distance between letter "A" and "B" is 0.994. Now refer to Table 1. Note that the Euclidean distance measure is zero when two patterns are identical. From this example, we see that the Euclidean distance measure between an object image and its shifted version is larger than that between two different object images. This is quite different from what the human vision system (HVS) perceives. In recognition, it is natural for the HVS to consider an object in an image and its shifted version in another to be identical. Obviously, it is using the Euclidean distance measure that causes this shift-variant problem. In order to make a neural network work in a more similar way to the HVS, we propose to use a 2-D correlation function to replace the Euclidean distance measure in the Kohonen network.

The 2-D correlation function proposed in our work is defined as follows:

$$f(x,y) \circ g(x,y) = \sum_{m=0}^{P-1} \sum_{n=0}^{Q-1} f(m,n) \cdot g(x+m, y+n) \tag{3.14}$$

where $f(m,n)$ is an unknown input pattern array and $g(m,n)$ is a reference pattern or template array. Both of them are of $P \times Q$. It is assumed that $f(m,n)$ and $g(m,n)$ are equal to 0 if $m \geq P$ or $m < 0$, or $n \geq Q$ or $n < 0$, $-(P-1) \leq x \leq P-1$ and $-(Q-1) \leq y \leq Q-1$. The correlation is utilized to evaluate the match between an unknown pattern and a reference pattern. Calculating the correlation defined in (3.14) for all the possible (x, y) is equivalent to shifting the reference pattern all over the possible locations on the unknown pattern and find the similarity measure. During the process, if and only if the two patterns are identical and overlapped, $f \circ g$ yields the largest possible correlation value 1.

Otherwise, the correlation between any two different patterns cannot reach this maximum value, as shown in Table 3.1. As a result, we can determine if the input pattern is identical to or is a shifted version of the reference pattern. That is, we can achieve the shift-invariant capability in recognition. Furthermore, if $f(m,n)$ is a shifted version of $g(m,n)$, the relative position (x, y) between f and g can be found. However, this is not of direct interest in this recognition process.

Network learning: The network we used is of the same structure as a conventional Kohonen network. The input buffer layer is of $P \times Q$, which is connected to the input image $f(m,n)$. The neuron layer consists of S nodes, each of which is connected to the input buffer layer through weights $w_j(m,n)$, where $j = 0, 1, 2, \ldots, S-1$ and m, n are spatial coordinates of pixels in the input image. In the learning, only the original patterns are required as input to the network. We do not input any shifted patterns to the network in the learning in order to minimize the number of patterns the neural network needs to memorize, thus decreasing the size of the neuron layer. The network learning algorithm consists of the following five steps: initialization, input, correlation, winner selection and weights update. They are the same as that for the conventional Kohonen network [23] except that in correlation, the Euclidean distance measure in (3.13) is replaced by the correlation defined in (3.14) with x and y equal to 0. The learning process continues until the correlation values between all the input patterns and their respective weight arrays, $f \circ w_j$, satisfy a certain accuracy criterion. The criterion value, α, is set to a number close to 1. That is, $\alpha=1-\beta$, where β is a small positive number. In our experiments, we let $0.01 \geq \beta \geq 0.001$.

Network classification: In the letter recognition, a part of which has been shown in Fig. 3.4, the correlation values between the input pattern $f(m,n)$ and the jth weight array $w_j(m,n)$, $j = 0, 1, 2, \ldots, S-1$, is calculated using (2) with $g(m,n)$ replaced by $w_j(m,n)$. Node j corresponding to the maximum correlation $\arg\max_{\forall j}\{f(x,y) \circ w_j(x,y)\}$ is selected as the winner of the network. The unknown input image pattern is then classified as the pattern class represented by the jth node.

Experimental results and discussion: In the experiments, all the capital English letters A, B, C ,..., Z and numeric characters 0, 1, 2,..., 9 form a set of training image patterns. The results show that in the pattern recognition, the proposed neural network outperforms the original Kohonen network in terms of shift-invariant classification capability. That is, our network successfully clusters letter "A" and shifted letter "A", shown in Figure 3.4, as the same pattern. Other images and their respective shifted versions are also correctly recognized as the same pattern class.

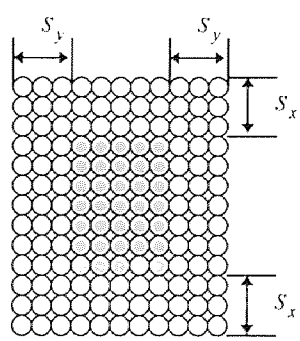


Figure 3.5 Maximum shifting with s_x and s_y equal to 3

By comparing (3.13) with (3.14), we observe that more computation is required for the correlation than for the Euclidean distance measure, especially when the shift quantities x and y are large. However, in practice, it is not necessary to compute the

correlation all over the theoretical range of $-(P-1) \le x \le P-1$ and $-(Q-1) \le y \le Q-1$ as defined in (3.14). Instead, the computation in (3.14) is only required to be conducted within a range of $-s_x \le x \le s_x$, and $-s_y \le y \le s_y$, where s_x, s_y are the expected maximum horizontal and vertical shifts of the object in an image, as shown in Fig. 3.5. Usually s_x and s_y are much smaller than P and Q, respectively. As a result, the time spent in correlation calculation does not increase drastically. To avoid large s_x and s_y, it is recommended that some pre-processing techniques be used to make sure that the unknown shifted image patterns are located within a reasonable range. These techniques have been developed in our research on digital gate symbol recognition and will be discussed in other publications.

Because the proposed network is shift-invariant, the shifted image patterns are no longer different pattern categories. Consequently, the number of neurons are decreased remarkably. If we assume that one extra node in the Kohonen neuron layer is needed for each horizontal and vertical shifted pattern, and N is the number of categories the network is expected to classify, then the total reduction in neurons by using our proposed network is: $N \cdot [(2s_x + 1) \cdot (2s_y + 1) - 1]$.

3.6 Rotation-Invariant Implementation

The proposed Kohonen network is capable of recognizing an image pattern rotated by $90° \times i$, i=0, 1, 2, 3. See Figure 3.6.

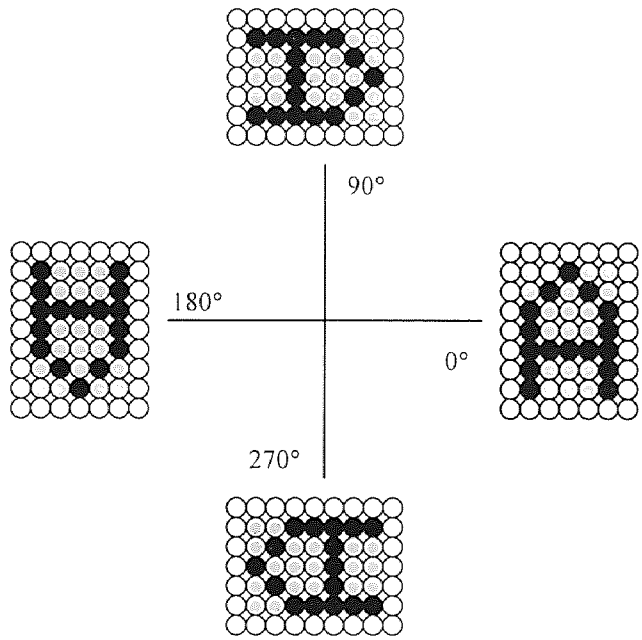


Figure 3.6 Rotation-invariant recognition of letter "A" with 0°, 90°, 180°, 270°

To achieve this type of rotation-invariance, we simply rotate $w_j(m,n)$ array by the same degree in the recognition procedure. This capability is useful in recognizing the digital gate symbols and characters in electronic drawings. In the electronic drawings, the gate symbols and characters may be in different orientation, which is 90°, 180°,or 270°rotation from the original form. The training of this shift-invariant Kohonen neural network is same as that of the shift-invariant network. We do not input any rotated patterns but the patterns with 0° to the network in the training in order to minimize the number of patterns the neural network needs to memorize, thus decreasing the size of the neuron layer. In the classification, the weight arrays are rotated instead of rotating the input pattern. Then, calculate the correlation between the input and the weight arrays. Finally pick up the largest correlation value which satisfies accuracy criterion. This value corresponds to the

found input pattern. As a result, the rotation-invariant capability is achieved while no additional neurons in the Kohonen neuron layer are increased. The total neuron reduction compares with the conventional Kohonen neural network with the same capability is 4 times $N \cdot [(2s_x + 1) \cdot (2s_y + 1) - 1]$.

3.7 Summary

The Euclidean distance measure is widely used in the Kohonen neural network which cannot classify an image pattern and its shifted version into the same category. In this chapter, a modified Kohonen neuron network using a 2-D correlation, its training and classification procedures are presented. Both the theoretical analysis and the experimental results demonstrate its shift-invariant capability in image pattern recognition. As a result, the modified Kohonen network and any subsequent network can be significantly simplified. It is also shown that the modified Kohonen network even possesses some limited (i.e., for 90°, 180°, 270° only) rotation-invariant recognition ability. This advancement is achieved at the expense of more computation. However, when the translation is not very large (which can be assured by using some pre-processing techniques), this increase in computation is minor.

CHAPTER 4

FEEDFORWARD NEURAL NETWORK WITH BACK-PROPAGATION LEARNING

4.1 Introduction

Since 1957, when psychologist Frank Rosenblatt proposed the "Perceptron" [41-47], a pattern recognition device with learning capabilities, the hierarchical neural network has been the most widely studied form of network structure. A hierarchical neural network is one that links multiple neurons together hierarchically. The special characteristic of this type of network is its simple dynamics. That is, when a signal is input into the input layer, it is propagated to the next layer by the interconnections between the neurons. Simple processing is performed on this signal by the neurons of the receiving layer prior to its being propagated on to the next layer. This process is repeated until the signal reaches the output layer completing the processing process for that signal.

The manner in which the various neurons in the hidden layers process the input signal will determine the kind of output signal it is transformed. Then hierarchical network dynamics are determined by the weight and threshold parameters of each of their units. If input signals can be transformed to the proper output signals by adjusting these parameters, then hierarchical networks can be used effectively to perform information processing.

Since it is difficult to accurately determine multiple parameter values, a learning method is employed. This involves creating a network that randomly determines parameter values. This network is then used to carry out input-to-output transformations for actual problems. The correct final parameters are obtained by properly modifying the

parameters in accordance with the errors that the network makes in the process. Quite a few such learning methods have been proposed. Probably the most representative of these is the error back-propagation learning method [48-52] proposed by D. E. Rumelhart et al. in 1986. This learning method has played a major role in the recent neural network development.

In our system, we used back-propagation algorithm for the feed-forward neural network. In the next several sections, we will introduce perceptrons, multilayer neural network, its designing, and the back-propagation algorithm. We mainly focus on the structure of this network: How many layer will it be, and how many nodes should be on each layer? Because there is no specific answer to these problems, we selected to use simulation method to get a better answer. The simulation results will be shown.

4.1.1 Artificial Neuron: Perceptron

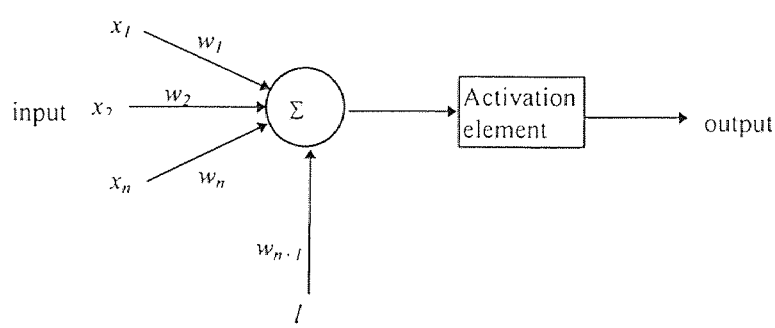


Figure 4.1 Percceptron

Perceptron (see Figure 4.1) is a basic element of the feed-forward neural network. It is originated by Rosenblatt in 1957, which caused significant excitement among researches of pattern recognition theory. The reason for the great interest in perceptron was

development of mathematical proofs showing that perceptrons, when trained with linear separable [53] training sets, would converge to a solution in a finite number of iterative steps. In its basic form, the perceptron learns a linear decision function that categorize linearly separable training sets.

Figure 4.1 shows schematically the proceptron model for two pattern classes. The response of this basic device is based on a weighted sum of its inputs, that is,

$$d(x) = \sum_{i=1}^{n} w_i x_i + w_{n+1} \tag{4.1}$$

which is a linear decision function with respect to the components of the pattern vectors. The coefficients w_i, $i = 1, 2, \ldots, n, n + 1$, called weights, modify the inputs before they are summed and fed into the threshold element. In this sense, weights are analogous to synapses in the human neural system. The function that maps the output of the summing junction into the final output of the device sometimes is called the activation function.

When $d(x) > 0$, the threshold element causes the output of the perceptron to be +1, indicating that the pattern x was recognized as belonging to class w_1. The reverse is true when $d(x) < 0$. When $d(x) = 0$, x lies on the decision surface separating the two pattern classes, giving an indeterminate condition. The decision boundary implemented by the perceptron is obtained by setting equation 4.1equal to zero:

$$d(x) = \sum_{i=1}^{n} w_i x_i + w_{n+1} = 0 \tag{4.2}$$

$$w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + w_{n+1} = 0 \tag{4.3}$$

which is the equation of a hyperplane in n dimensional pattern space. Geometrically, the first n coefficients establish the orientation of the hyperplane. Whereas the last

coefficient, w_{n+1} =0, the hyperplane goes through the origin of the pattern space. Similarly, if $w_j = 0$, the hyperplane is parallel to the x_j axis.

The output of the threshold element in Figure 4.1 depends on the sign of $d(x)$. Instead of testing the entire function to determine whether it is positive or negative, we could test the summation part of equation (4.1) against the term w_{n+1}, in which case, the output of the system would be

$$O = \begin{cases} +1, if \sum_{i=1}^{n} w_i x_i > -w_{n+1} \\ -1, if \sum_{i=1}^{n} w_i x_i < -w_{n+1} \end{cases} \tag{4.4}$$

Another formulation commonly found in practice is to augment the pattern vectors by appending an additional (n +1)st element, which is always equal to 1, regardless of class membership. That is, an augmented pattern vector y is created from a pattern vector x by letting $y_i = x_i$, $i = 1, 2, \ldots, n$, and appending the additional element y_{n+1} =1. Equation (4.1) then becomes

$$d(y) = \sum_{i=1}^{n+1} w_i y_i = \mathbf{w}^{\mathrm{T}}\mathbf{y} \tag{4.5}$$

where $y=(y_1, y_2, \ldots y_n, 1)^T$ is now an augmented pattern vector, and $\mathbf{w} = (w_1, w_2, \ldots, w_n, w_{n+1})^{\mathrm{T}}$ is called the weight vector. This expression is usually more convenient in terms of notation.

In the elementary perceptron, there are no hidden neurons. Consequently, it cannot classify input patterns that are not linearly separable. However, non-linearly separable [54] patterns are of common occurrence. For example, it arises in the XOR problem, which may be viewed as a special case of a more general problem. In the XOR problem,

we need consider only the four corners of the unit square that correspond to the input of patterns (0,0), (0,1), (1,1), and (1,0). The first and the third input patterns are in class 0, as shown by 0 XOR 0 = 0, and 1 XOR 1 =0. The input patterns (0,0) and (1,1) are at the opposite corners of the unit square, and they produce the identical output 0. On the other hand, the input patterns (0,1) and (1,0) are also at opposite corners of the square, but they are in class 1, as shown by 0 XOR 1 = 1 and 1 XOR 0 = 1.

We know that the use of a single neuron with two inputs results in a straight line for decision boundary in the input space. For all points on one side of this line, the neuron outputs 1, while for all points on the other side of the line, it outputs 0. The position and orientation of the line in the input space are determined by the weights of the neuron connected to the input nodes, and the threshold applied to the neuron. It is clear that we cannot construct a straight line for a decision boundary so that (0,0) and (1,1) lie in one decision region. Namely, an elementary perceptron cannot solve the XOR problem.

4.1.2 Multilayer Perceptrons

The perceptron learning rule [55] of Frank Rosenblatt were designed to train single-layer proceptrons. As we have discussed in the last section, these single-layer perceptrons suffer from the disadvantage that they are only able to solve linearly separable classification problems. For that reason, multilayer artificial networks are used for interesting problems that require greater discrimination. The key to these multilayer networks is the learning algorithms that allow the training of the weights on the hidden layers, the layers of neurons that are not directly connected to the inputs or the outputs.

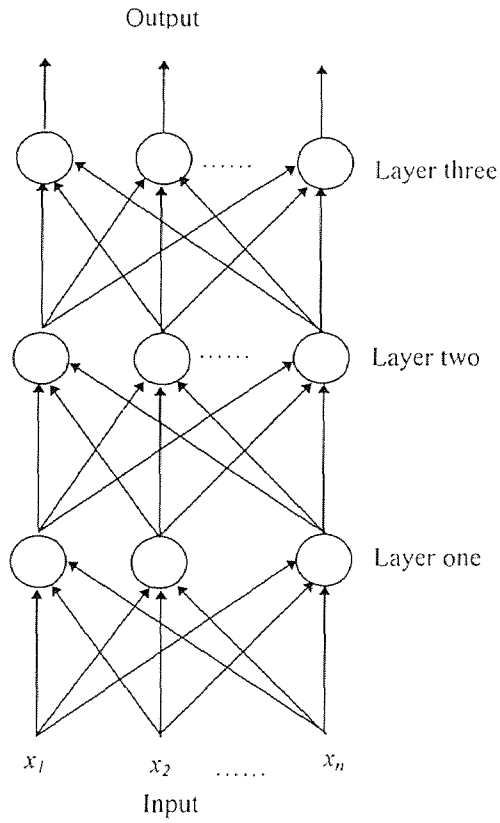


Figure 4.2 Three-layer Perceptron

Figure 4.2 is a three-layer perceptron structure. The nonlinearity normally used is the sigmoid. The simplicity of the derivative of the sigmoid provides a nice learning law. The reason that only three layers are shown is that three layers were believed to be the most required for any arbitrary classification problem. More recently it has been shown by Cybenko [56] and others that one hidden layer is sufficient for any arbitrary transformation, given enough nodes. Researchers still often use multiple hidden layers because they may sometimes provide advantages in quicker learning.

The reason to use multilayer networks is to allow the formation of complex decision regions in the feature space. The key, of course, is a learning algorithm to find correct set of weights. The most common algorithm used is called the backward error

propagation learning rule, sometimes just "back-propagation". We will discuss it in the next section.

4.2 Back-Propagation Training Algorithm

Back-propagation is a systematic method for training multilayer artificial neural network. It was invented independently by Bryson and Ho [57],Werbos [58], Parker [59], and Rumelhart, Hinton and Williams [60]. A closely related approach was proposed by Le Cun [61]. The training algorithm is an iterative gradient algorithm designed to minimize the mean square error between the actual output of a multilayer feed-forward perceptron and the desired output.

We analyze the delta rule with N nodes on the output layer of a network, the error has to be summed over all nodes The idea is to perform a gradient descent on the error considered as a function of the weights. All the weights are taken into account for both hidden and output nodes. The hidden nodes are in the intermediate layer(s) which we do not have direct access to for the purposes of training. The output nodes are the ones which tell us the net's response to an input and to which we may show a supervisory or target value during training.

The analysis for the output nodes is the delta rule given in the following equation:

$$\Delta w_i^j = \alpha\sigma'(a^j)(t^j - y^j)x_i^j \tag{4.6}$$

where a superscript is introduced to denote which node is being described. Because the gradient of the error with respect a weight on the jth node can only be affected by the part which contains reference to that node.

In order to see clearly, it is useful to split the right hand side of equation (4.6) in the following way. The term $(t^j - y^j)$ represents a measure of the error on the jth node. The term $\sigma'(a^j)$ relates to how quickly (rate of change or slope) the activation can change the output (and hence the error). If this is small, then we are on one of the ends of the sigmoid and changing the activation won't change the output much. If, however, it is large, then we can expect a rapid change for a given change in activation. The factor of x_i^j is related to the amount that the ith input has affected the activation. If it is zero then that input cannot be responsible for the error and so the weight change should also be zero. If on the other hand, it is large, then the ith input had a large contribution to the activation which gave the error and so the weight needs to be changed by a correspondingly larger amount.

To summarize: x_i^j tells us how much the ith input was `responsible for' the activation; $\sigma'(a^j)$ tells us how fast the output is changing in response to changes in the activation and $(t^j - y^j)$ is the error on the jth node. It is therefore reasonable that the product of these gives us something that is a measure of the rate of change (slope) of the error with respect to the weight w_i^j .Using these notations, we may combine two of these elements as follows:

$$\delta^j = \sigma'(a^j)(t^j - y^j) \tag{4.7}$$

The delta rule for output units may now be written:

$$\Delta w_i^j = \alpha \delta^j x_i \tag{4.8}$$

Consider now, the two layer net, in particular, the kth hidden node. The problem in assigning a set of weight changes to this type of node is related to how much influence

has this node had on the error. The resulting weight changes will be a result of including the right combination of 'responsibility' factors, rates of change and errors in the same way that these occurred for the output nodes. This however does not give insight. The purpose is just to see on where the resulting formula comes from. For the ith input to the hidden node, the value of the input will play a similar role as before so we might write:

$$\Delta w_i^j = \alpha \delta^j x_i \tag{4.9}$$

and the task now is to find what goes into the factor δ^k. For this, let us consider just a single output from the hidden node to an output node.

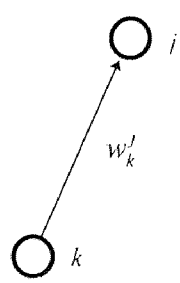


Figure 4.3 Diagram of kth hidden and jth output nodes

The effect this node has on the error depends on two things: first, how much it can influence the output of node j and, via this, how the output of node j affects the error. The more k can affect j, the greater the effect to be on the error, but this will only be significant if j is having some effect on the error at its output. The contribution that node j makes towards the error is, of course, expressed in the 'delta', for that node δ^j. The influence that k has on j is given by the weight w_k^j. Therefore we may expect the find the product $w_k^j \delta^j$ in the expression for δ^k. However, the kth node may be giving output to several nodes and so the contributions to the error from all of these must be taken into

account. Thus, we must sum these products over all j. Finally, the factor $\sigma'(a^k)$ will occur for exactly the same reasons that it did for the output nodes. This results in the following expression for δ^k:

$$\delta^k = \sigma'(a^k)\sum_{j \in I_k} \delta^j w_k^j \tag{4.10}$$

where I^k is the set of nodes which take an input from the hidden node k. This set is called the fan-out of k. Using this in equation (4.9) gives us a means for calculating the weight changes for the hidden nodes.

Next we develop a training algorithm using the rules we have developed. This basic iteration is as follows:

Repeat

  for each training pattern

    train on that pattern

  end for loop

until the error is acceptable low

Before examining the step `train on a pattern' a couple of points need comment. First, it is implied in the algorithm defined above that there is a fixed presentation sequence of training vectors. The alternative is to present vectors randomly. If we were to imagine our network in a real learning environment then this second option is more realistic. Empirically, however, it is often found that training is faster if the vectors are ordered in some way and that order is maintained in presentation. Second, what is an acceptable error? One possible definition for Boolean training sets might be to ensure that all output nodes had responses in the correct one of the pair of intervals [0, 0.1], [0.9, 1] as defined

by the target, since then, if we were to replace the sigmoid with a hard limiting threshold, the `correct' response would be guaranteed. Another might simply prescribe some low value like 0.001. Whatever approach is used, one has to interpret the significance of the criterion.

The main step of training on a pattern may now be expanded into the following steps.

(1) Present the pattern at the input layer.

(2) Let the hidden units evaluate their output using the pattern.

(3) Let the output units evaluate their output using the result in step (2) from the hidden units.

The steps (1) ~ (3) are collectively known as the forward pass.

(4) Apply the target pattern to the output layer.

(5) Calculate the δ's on the output nodes according to equation (4.7).

(6) Train each output node using gradient descent equation (4.8).

(7) For each hidden node, calculate its δ according to equation (4.10).

(8) For each hidden node, use the δ found in step (7) to train according to gradient descent equation (4.9).

Steps (4)~(8) are collectively known as the backward pass

Step (7) involves propagating the δ's from those output nodes in the hidden unit's fan-out back towards this node so that it can process them. This is where the name of the algorithm comes from.

4.3 The Network Capability and Its Structure

Multilayer perceptrons are feed-forward with one or more layers of nodes between the input and output nodes. These additional layers contain hidden units or nodes that are not directly connected to both the input and output nodes. A three-layer perceptron with two layers of hidden units is shown in Figure 4.2. Mulitilayer perceptrons overcome many of the limitations of single-layer perceptrons, but were generally not used in the past because effective training algorithms were not available. This recently changed with the development of new training algorithms [61]. Although it cannot be proven that these algorithms converge as with single layer perceptrons, they have been shown to be successful for many problems of interest [61].

The capabilities of multilayer perceptrons come from the nonlinearities used within nodes. If nodes were linear elements, then a single-layer net with appropriately chosen weights could exactly duplicate those calculations performed by any multilayer net. The capabilities of perceptrons with one, two, and three layers that use hard-limiting nonliearties are illustrated in Figure. 4.4. The second column in this figure indicates the types of decision regions that can be formed with different nets. The next two columns present examples of decision regions which be formed for the exclusive OR problem and a problem with meshed regions. The rightmost column gives examples of the most general decision regions that can be formed.

We have known that a single-layer perceptron forms halfplane decision regions. A two-layer perceptron forms any, possibly unbounded, convex region in the space spanned the inputs. Such regions include convex polygons sometimes called convex hulls, and the unbounded convex regions shown in the middle row of Figure 4.4. Here the term convex

means that any line joining points on the border of a region goes only through points within that region. Convex regions are formed from intersections of the half-plane regions formed by each node in the first layer of the multilayer perceptron. Each node in the first layer behaves like a single-layer perceptron and a "high" output only for points on one side of the hyperplane formed by its weights and offset. If weights to an output node from N_l first-layer nodes are 1.0, and the threshold in the output node is $N_l - \varepsilon$ where $0 < \varepsilon < 1$, then the output node will be "high" only if the outputs of all first-layer nodes are "high". This corresponds to performing a logical AND operation in the output node and results in a final decision region that is intersection of all the half-plane regions formed in the first layer. intersections of such half planes form convex regions as described above. These convex regions have at the most as many sides as there are nodes in the first layer.

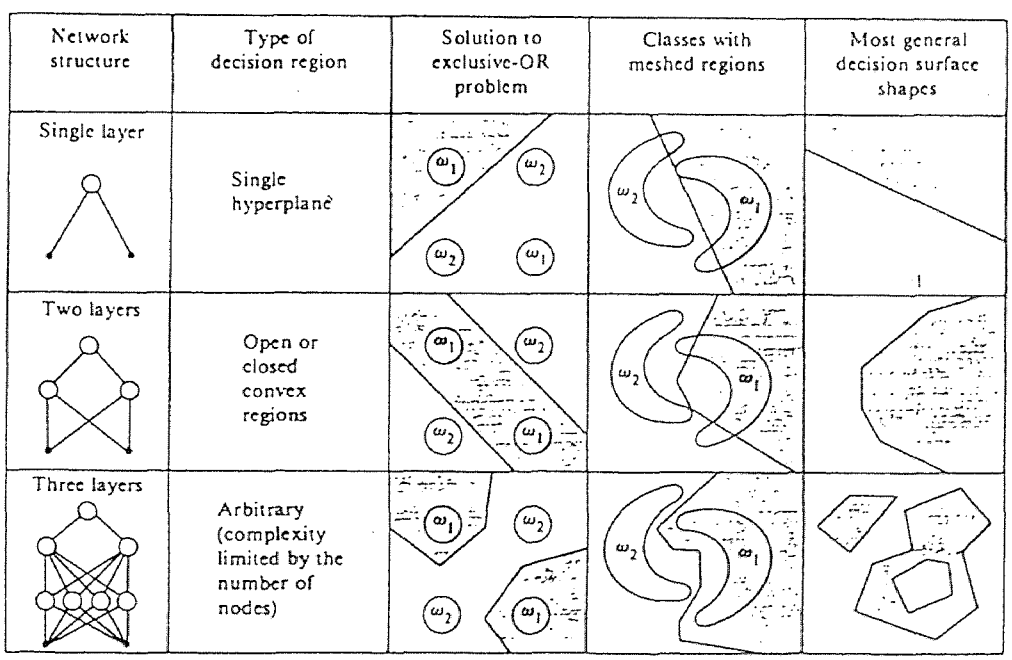


Figure 4.4 Type of decision regions of different layer network (Adapted from [42])

This analysis provides some insight into the problem of selecting the number of nodes to use in a two-layer perceptron. The number of nodes must be large enough to form a decision region that is as complex as is required by a given problem. It must not be so large that the many weights required can not be reliably estimated from the available training data. For example, two nodes are sufficient to solve the exclusive OR problem in the second row of Figure 4.4. No number of nodes, however, can separate the meshed class regions in Figure 4.4 with a two-layer perceptron.

A three-layer perceptron can form arbitrarily complex decision regions and can separate the meshed classes as shown in the bottom of Figure 4.4. It can form regions as complex as those formed using mixture distributions and nearest-neighbor classifiers. This can be proven by construction. The proof depends on partitioning the desired decision region into small hypercubes (squares when there are two inputs). Each hypercube requires $2N$ nodes in the first layer (four nodes when there are two inputs), one for each side of the hypercude, and one node in the second layer that takes the logical AND of the output from the first layer nodes. The outputs of second-layer nodes will be "high" only for inputs within each hypercube. Hypercubes are assigned to the proper decision regions by connecting the output of each second-layer node only to the output node corresponding to the decision region that nodes hypercube is in and performing a logic OR operation in each output node. A logical OR operation will be performed if these connection weights from second hidden layer to the output layer are one and thresholds in the output nodes are 0.5. This construction procedure can be generalized to

use arbitrarily shaped convex regions instead of small hypercubes and is capable of generating the disconnected and non-convex regions shown at the bottom of Figure 4.4.

The above analysis demonstrates that no more than three layers are required in perceptron-like feed-forward networks because a three-layer net can generate arbitrarily complex decision regions.

4.4 Layers and Nodes Specification

In this section, we involve the design of the neural network for our specific purpose. This network should be able to recognize eight different gate symbols (shown in APPENDIX B): AND, OR, XOR, NAND, NOR, XNOR, NOT, and Buffer. Therefore, the output layer of the network should have 8 neurons corresponding to those pattern classes.

From the discussion in the last section, we know that a three-layer network .can implement decision surfaces of arbitrary complexity. Therefore we selected a 3-layer structure in the system (refer to APPENDIX C).

The first layer is a Kohonen neural network. Its input buffer layer is identical to the size of input gate symbol patterns, which is of 32×32(1024 pixels). The size of the Kohonen neuron layer is selected 64 in order to make the classification space large enough. This is 8 times of the gate symbol type.

The third layer is the last layer. 8 neurons are specified in this layer corresponding to the number of pattern classes the network desired to recognize.

As for the number of neurons in the second layer (hidden layer), it is not easy to decide at the time when we began designing the neural network. Some people took the average number [10] of neurons in the input and output layers. Other people took the

square root [66] of that number. Still other people [33] suggested to make the number of neurons equal to about two-thirds of the number in the input layer. However, most of them were by experience. In fact, the number of neurons in the hidden layer may change the training time as well as the ability of the neural network to generalize. Often there is a wide range in the number of neurons in the hidden layer that can be used successfully. Therefore, we decided to utilize an optimization methodology for determining the optimal number of neurons in the hidden layer.

Usually, increasing the size of the hidden layer improves the networks accuracy on the training set. But decreasing the size of the hidden layer generally improves generalization and reduces the processing time. So an optimal size can be attained by a balance between the objectives of the accuracy and generation for the particular application.

We did a series of simulations to obtain the optimal size of the hidden layer. The network we worked on is a two layer feed-forward network. The input buffer is of 64 node, which comes from the output of the Kohonen neural network. The output layer is a 8-node layer corresponding to 8 different gate symbols we want to recognize.

The hidden layer nodes we selected for the simulation were 10, 15, 20, 25, 30, 35, 40 ,45, 50, 55, 60, 65. The simulation results are shown in Figure 4.5 ~ Figure 4.16. In the simulation, we used 40 patterns in the network training for each 8 different gate symbols, they were 320 in all. We also used 4 patterns in the testing for each 8 different gate symbols, they were 32 in all.

In the simulation, we randomly (not sequentially) presented the training patterns to the input layer to train the network. The iterative times were 200, 400, 800, 1,600,

4,000, 10,000 and 20,000, respectively. Back-propagation algorithm was used to adjusted all the weights. Then using these same patterns to test the percentage correct recognition rate, shown in each simulation result in Figure 4.5 ~ 4.16. Finally we input the testing patterns which had never appeared in the training to test the generalization capability of the network.

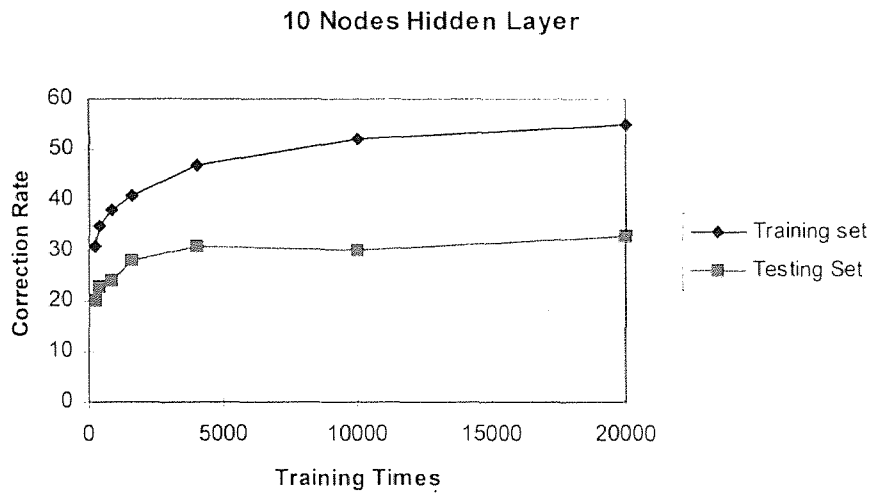


Figure 4.5 Simulation Result for 10 Nodes in the Hidden Layer

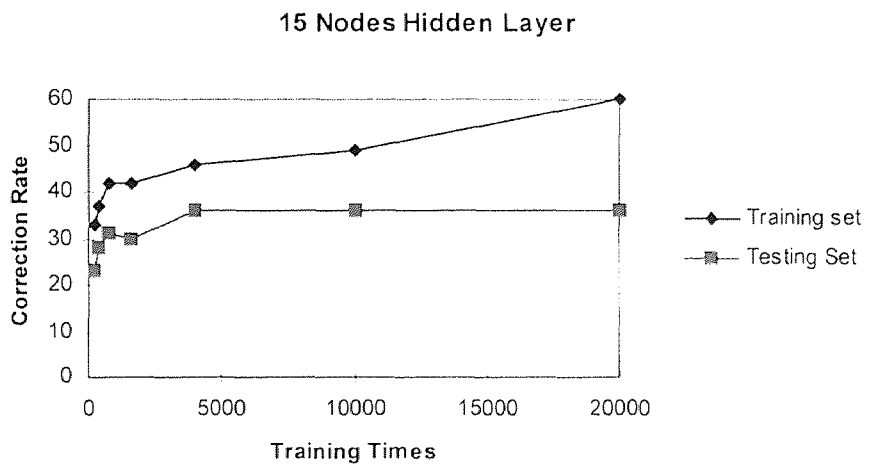


Figure 4.6 Simulation Result for 15 Nodes in the Hidden Layer

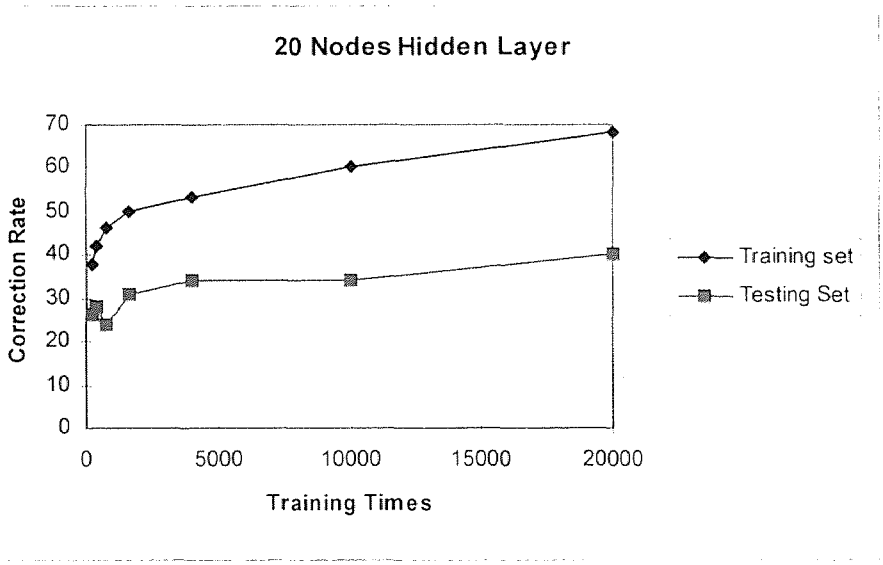


Figure 4.7 Simulation Result for 20 Nodes in the Hidden Layer

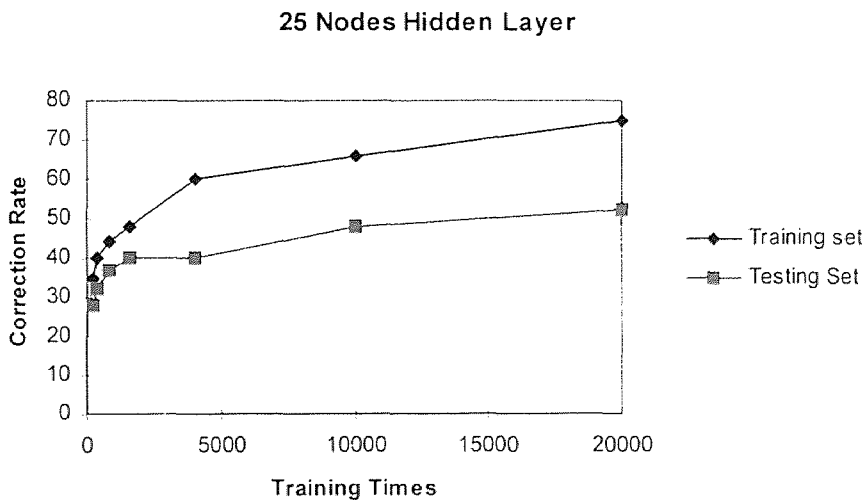


Figure 4.8 Simulation Result for 25 Nodes in the Hidden Layer

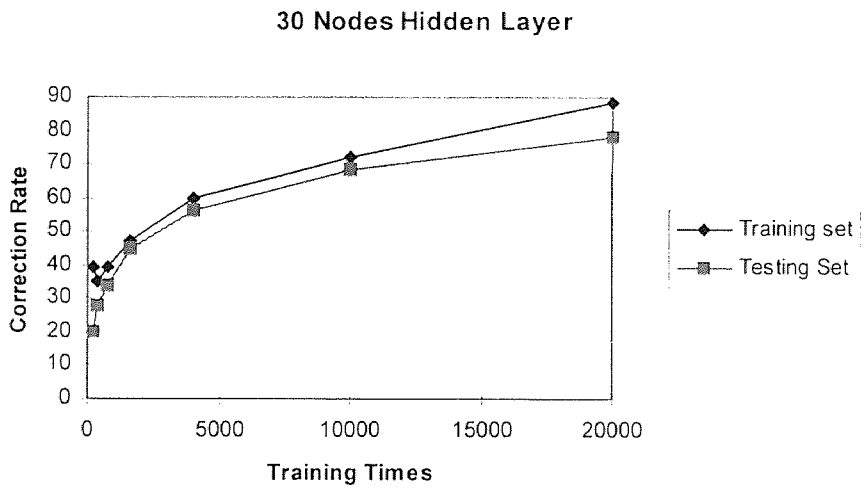


Figure 4.9 Simulation Result for 30 Nodes in the Hidden Layer

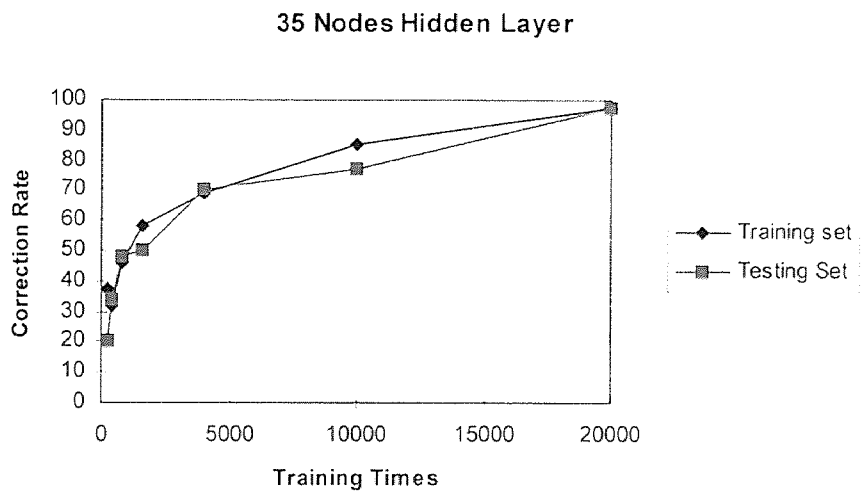


Figure 4.10 Simulation Result for 35 Nodes in the Hidden Layer

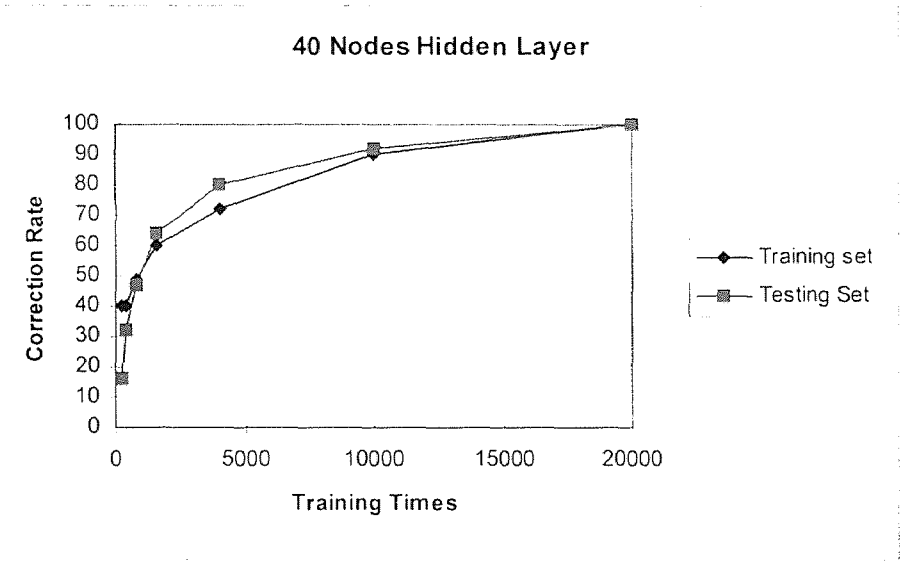


Figure 4.11 Simulation Result for 40 Nodes in the Hidden Layer

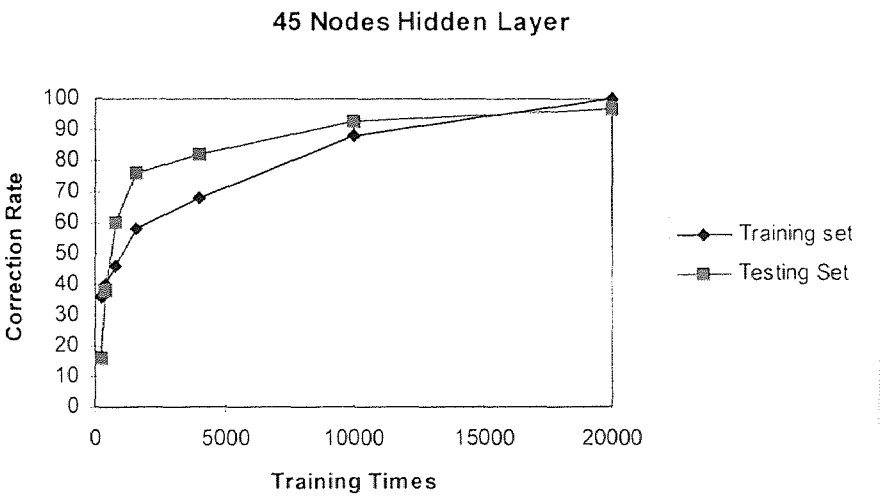


Figure 4.12 Simulation Result for 45 Nodes in the Hidden Layer

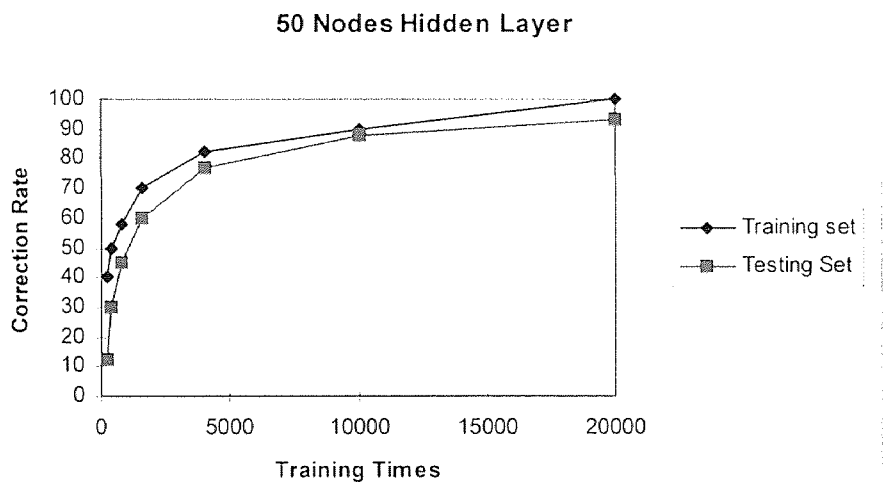


Figure 4.13 Simulation Result for 50 Nodes in the Hidden Layer

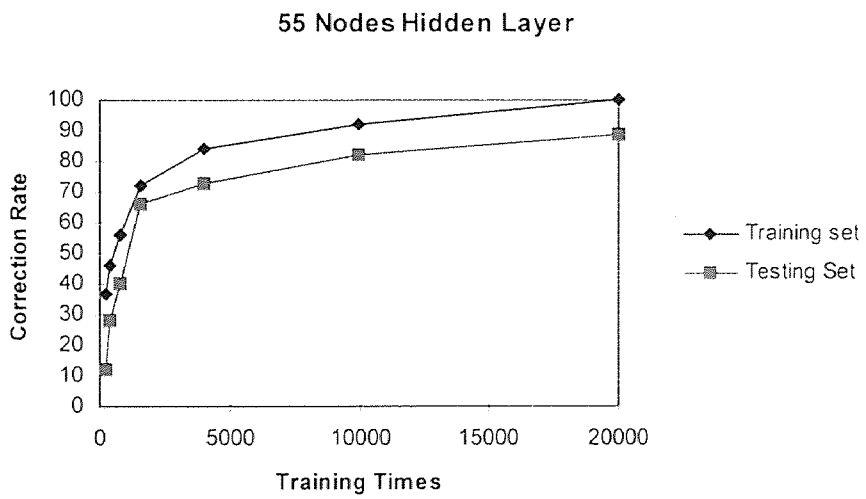


Figure 4.14 Simulation Result for 55 Nodes in the Hidden Layer

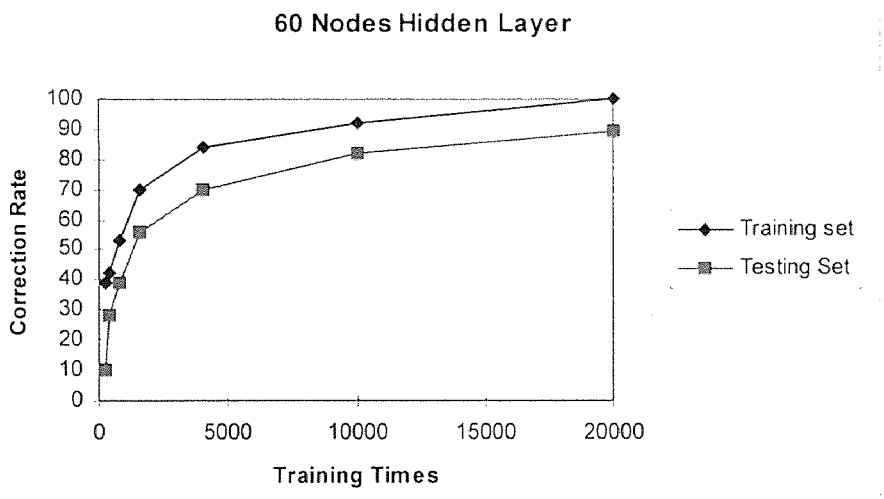


Figure 4.15 Simulation Result for 60 Nodes in the Hidden Layer

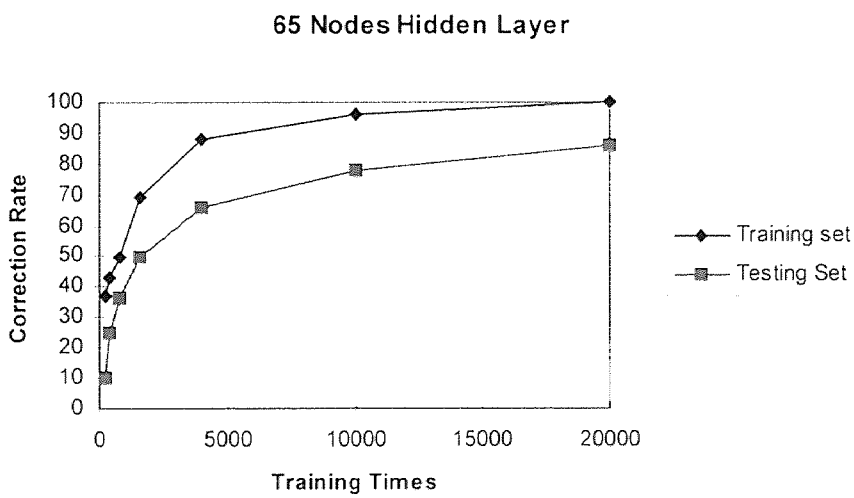


Figure 4.16 Simulation Result for 65 Nodes in the Hidden Layer

From these simulations, we can find that when we increase the nodes in the hidden layer, both the correction rate of the training set and the testing set increase, see Figure 4.5 ~ 4.11. This shows the importance of neurons as the basic unit to memorize what they have learned in the learning. In these figures, we can also find that the more times they learn, the better capability they have.

With the increase of the nodes in the hidden layer, the recognition accuracy and the generalization capability increase to the maximum 100% in Figure 4.11. Then, when the nodes are further increased, the correct recognition rate for the training set keeps the same, while that for testing set goes down. Refer to Figure 4.12 ~ 4.16. This indicates that with the further increase of the hidden nodes, the recognition accuracy does not change but the generalization capability decreases.

Based on the simulation, we tested the network with the hidden node around 40 and finally found the optimal point is that hidden nodes are equal to 42 in our specific system.

4.5 Summary

The number of neurons in the input and output layer are determined by the nature of the problem. In our neural network recognition system, we utilized a 64–node (output of the Kohonen network) as an input and it maps the inputs into 8 categories, thus 8 neurons in the output layer. Determining the proper number of the neurons for the hidden layer has to be accomplished through simulation. Too few neurons in the hidden layer prevent it from correctly mapping inputs to outputs while too many impede generalization and increase training and processing time. Too many neurons may allow the network to

memorize the patterns presented to it without extracting the pertinent features for generalization. Thus, when presented with new patterns, the network is unable to process them properly, because it has not discovered the underlying principles of the system. From the simulation, 42 hidden nodes are selected.

CHAPTER 5

TRAINING THE NETWORK WITH NOISE

Apply a little noise to the training set will generally produce a network that is robust to noise inputs. Although a network trained with no noise may still do well with noise inputs in practice, the one trained with an appropriate level of noise will do much better. In this chapter, we discuss the training with noise for the improvement of our neural network.

5.1 Generalization of the Neural Network

Generalization capability of a neural network is that it makes predictions for cases that are not in the training set. In most cases, the multilayer network is trained with a finite number of patterns. This training set is normally representative of a much larger class of possible input-output pairs. It is important that the network successfully generalize what it has learned to the total population. Consider the situation in pattern space shown in Figure 5.1.

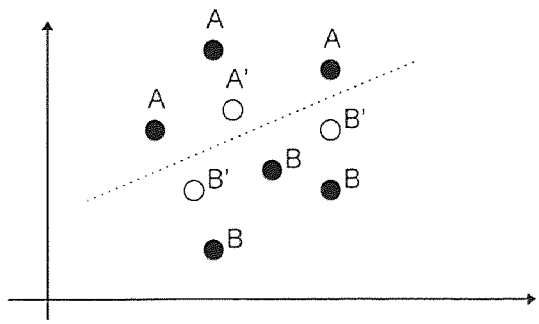


Figure 5.1 Two-Class Classification

The training pattern are shown by solid dots and there are two classes A and B. The circles in each class represent vectors which were not shown during training; these are test patterns. Representatives from each class of test data have been classified correctly, even though they were not seen during the training. This is the power of the network approach and one of the main reasons for using it. The net is said to have generalized from the training data.

Noise in the actual data is never a good thing, since it limits the accuracy of generalization that can be achieved no matter how extensive the training set is. On the other hand, injecting artificial noise into the inputs during training is one of several ways to improve generalization when you have a small training set. Certain assumptions about noise are necessary for theoretical results. Usually, the noise distribution is assumed to have zero mean and finite variance. Noise in the inputs limits the accuracy of generalization, but in a more complicated way than does noise in the targets. In a region of the input space where the function being learned is fairly flat, input noise will have little effect. In regions where that function is steep, input noise can degrade generalization severely. Furthermore, if the target function is $y = f(x)$, but you observe noisy inputs $x + d$, you cannot obtain an arbitrarily accurate estimate of $f(x)$ given $x + d$, no matter how large a training set you use.

5.2 Training the Neural Network with Uniform Distributed Noise

We applied some random noise which is uniformly distributed on the image patterns either to test or re-train the neural network. By testing the neural network, we may know the performance of a neural network. In the training, by adding some noise, the

generalization capability of a neural network will improve to some extent. In other words, If we have two cases with similar inputs, the desired outputs will usually be similar. That means we can take any training case and generate new training cases by adding small amounts of noise to the inputs. As long as the amount of noise is sufficiently small, we can assume that the desired output will not change enough to be of any consequence, so we can just use the same target value. The more training patterns are expected. This is one of the convenient ways to improve training. Obviously, too much noise will obviously produce garbage, while too little noise will have little effect.

For the performance testing, we generated 20 different noise level patterns for each 8 gate symbols, 160 in all. The noise levels are 2.5%, 5.0%, 7.5%, 10.0%, 12.5%, 15.0%, 17.5%, 20.0%, 30.0% and 40.0%, respectively. The noise is uniformly distributed over the each pattern. The neural network trained with noise free patterns was tested by the above noise patterns. The result is shown in Figure 5.2.

Then we generated noise pattern set for the training. Each gate symbols were superposed with different level uniform distributed noise. After the training by each level's noise, the testing sets were used to measure the performance of the network. The results are shown in Figure 5.3. We can find that the performance gets better and better with the noise level get higher until 10.0% level. Noise of 12.5% level makes the network performance become worse, refer to Figure 5.3. This indicates that the network cannot adapt itself sufficiently to the larger variations at the higher noise levels with the given training patterns.

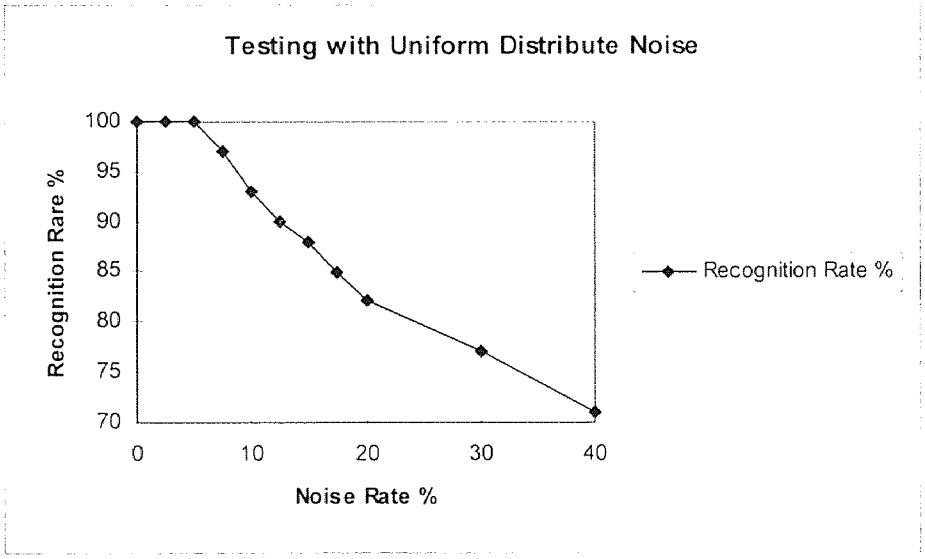


Figure 5.2 Uniform Distributed Noise Testing in Noise-Free Weight Network

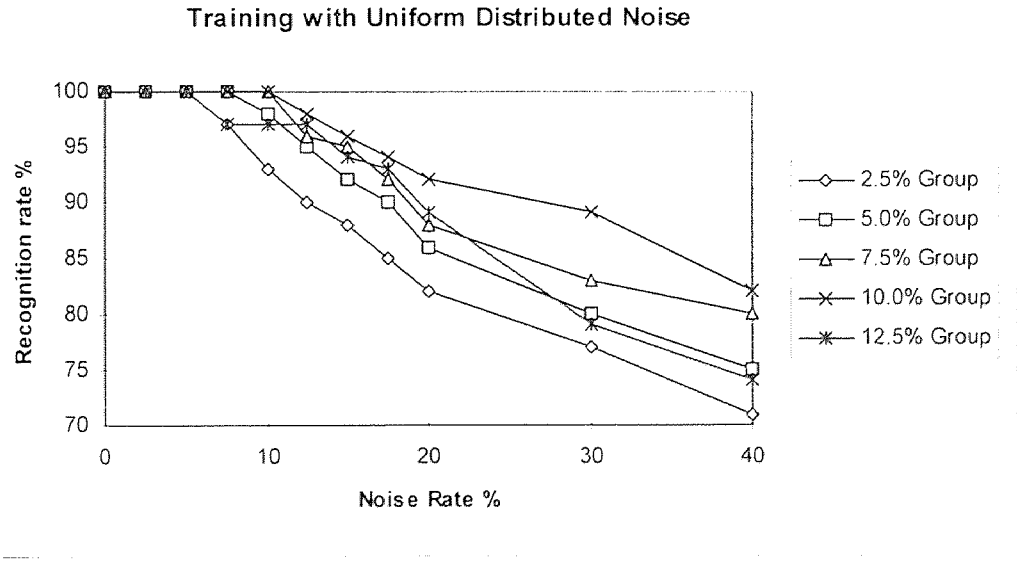


Figure 5.3 Training with Uniform Distributed Noise

5.3 Training the Neural Network with 8-Neighboring Noise

Unlike the training with the uniform distributed noise patterns in the previous section, we generated another kind of noise pattern which was superposed to the noise-free image patterns in the training. This type of noise pattern is quite similar to the distortion generated by the scanner when an electronic drawing is scanned.

We generated those patterns as follows. Each contour pixel in a noise-free shape was assigned a probability P of retaining its original coordinate in the image plane a probability $R = 1 - P$ of being randomly assigned to the coordinates of one of its eight neighboring pixels. The degree of noise increased by decreasing P (that is, increasing R). We generated 2 sets of noisy data. The first consisted of 20 noisy patterns of each class generated by 2.5%, 5.0%, 7.5%, 10.0%, 12.5%, 15.0%, 17.5%, 20.0%, 30.0% and 40.0%, giving a total of 160 patterns. This set, called the testing set, which was used to establish system performance after training.

We also generated several noisy data sets for training the system. The first set consisted of 10 samples for each class, generated by using $R = 0$, where R denotes a value of R used to generate training data. Starting with the weight vectors obtained in the noise-free training, the system was allowed to go through a learning sequence with the new data set. Because $R = 0$ implies no noise, this retraining was an extension of the earlier, noise-free training. Using the resulting weighing learned in this manner, the network was subjected to the test data set yielding the results shown by the curve labeled $R = 0$ in Figure 5.4. The number of correctly classified patterns divided by the total number of patterns tested gives the recognition rate, which is a measure commonly used to establish network performance.

Then we start with the weight vectors learned by using the data generated with R = 0, the system was retrained with a noisy data set generated with R = 2.5%. The recognition performance was then established by running the test samples through the system again with the new weight vectors. Note the significant improvement in performance. Figure 5.4 shows the results obtained by continuing this retraining and retesting procedure for R =0.5%, 7.5%, 10.0% and 12.5%. As expected if the system is learning properly, the recognition rate from the set increased as the value of R increased, because the system was being trained with noisier data for higher values of R. The one exception in Figure 5.5 is the result for R = 12.5%. The reason is the small number of samples used to train the system. That is, the network was not able to adapt itself sufficiently to the larger variations in shape at higher noise levels with the number of samples used.

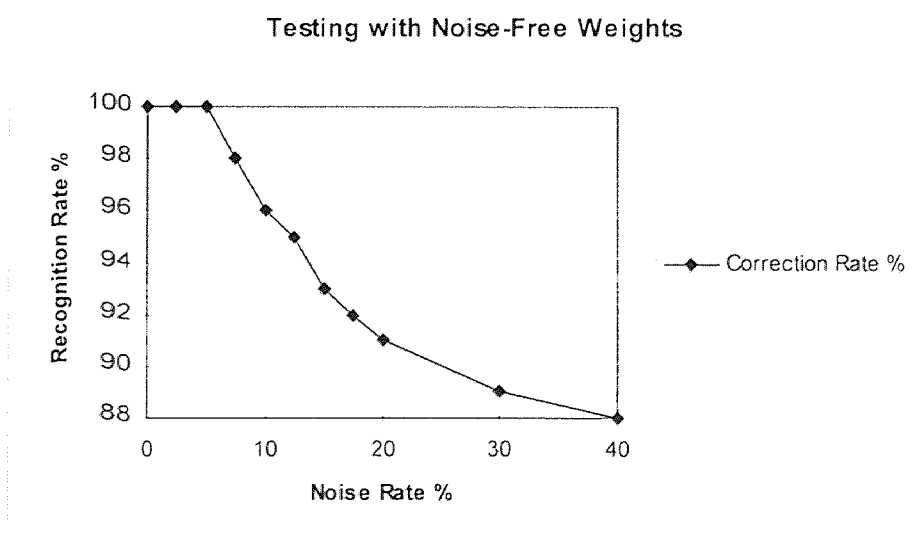


Figure 5.4 8-Neighbor Noise Testing in Noise-Free Weight network

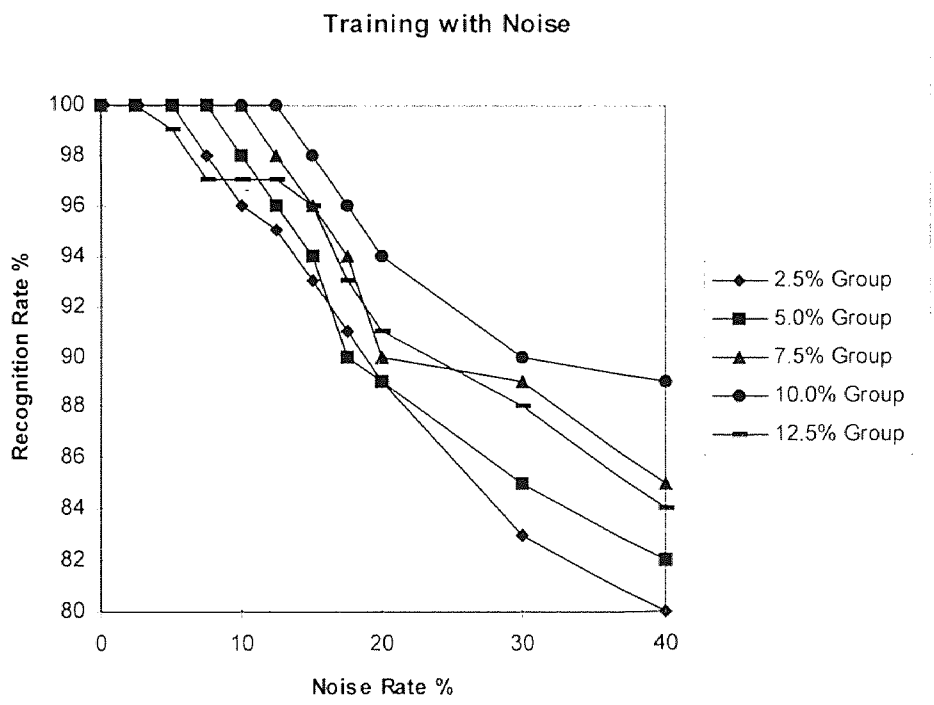


Figure 5.5 Training with 8-Neighbor Noise

CHAPTER 6

SUMMARY

This chapter gives a summary of our major contributions in this research, a review of some unsolved issues and a discussion of some possible directions for the future research.

6.1 Major Contributions

Neural network technique has been found to be a powerful tool in pattern recognition. It captures associations or discovers regularities with a set of patterns, where the types, number of variables or diversity of the data are very great, the relationships between variables are vaguely understood, or the relationships are difficult to describe adequately with conventional approaches.

In the research and the system (refer to APPENDIX C) design aiming at recognizing the digital gate symbols and characters in the electronic drawings, we have proposed: (1) A shift-invariant and limited rotation-invariant modified Kohonen neural network. (2) An effective approach to optimize the structure of the back-propagation neural network. (3) Candidate searching and pre-processing algorithms for the enhancement of the electronic drawings' recognition.

The Kohonen neural network is a type of unsupervised network. It has been successfully applied to speech recognition and used as a pre-processing layer, part of a neural network layer for image recognition. In our research, we found that when an object in an image is shifted horizontally and/or vertically, the Kohonen neural network cannot readily classify the original image pattern and its shifted version into the same category. Yet, this shift-invariant capability in image pattern recognition is crucial in many

applications. For this purpose, we developed a shift-invariant modified Kohonen network with a 2-D correlation. As a result, the dimension of this modified Kohonen neuron layer can be reduced dramatically compared with the conventional Kohonen network with the same capability. Moreover, the size of its subsequent neural network can be decreased significantly as well. An analysis and the system performance reveal that when the shift of an image pattern is not large, and the rotation is only n×90°, (n=1, 2, and 3) the shift-invariant and limited rotation-invariant modified Kohonen neural network is superior to the conventional Kohonen neural network. This new type of Kohonen neural network has been used successfully in our automatic analysis and recognition system.

The combination of Kohonen neural network and back-propagation feed-forward neural network is proved to be a better way to minimize the dimensionality of the overall recognition system. The optimized back-propagation neural network developed by us outperforms the conventional ones designed by experience. Therefore, the size and the computation can be minimized by simulation for any specific application. This optimized neural network has been proved to combine the good recognition accuracy and better generalization capability.

In order to further reduce the calculation time spent in neural network, pre-processing algorithm was developed to remove long circuit lines in the electronic drawings. Our experiments indicated that it can achieve very high erasing rate up to 90 percent of "1" pixels. As a result, the candidate searching can be more accurate and fast.

6.2 Major Unsolved Issues and Future Research

Automatic verification [67–70] for the recognition system is a very important issue. Till now this problem has not been solved in our system.

The reliability of an analysis and recognition system is frequently more important than the other performance factors. For an automated recognition and interpretation system, the rate of correct recognition is usually more important than the speed of calculations. Therefore, checking the correctness of recognition becomes the next procedure of the processing. Of cause, human power can be used to do this work and computer can also assist people to do this work. But is it possible for computers to do this work entirely and independently? This is an ultimate aim we are working for. For the first step, we hope to use computers to check the correctness of recognition just like we use computers to check the spellings in a word processor software. In the checking, wherever there is an error detected, computer should tell human beings about it. Human beings may then decide to correct it or keep it. It is hoped that all errors in recognition may be eliminated. It is expected that the future research in the automatic analysis system will be in this area.

APPENDIX A

EARLY RESEARCH SURVEY

There are several techniques that have been used in pattern recognition for decades. They are template, feature extraction, segmentation, contour tracking, structure analysis, moment, vectorization, decision tree, and conventional neural networks, etc. In certain circumstance, these techniques are efficient and convenient for implementation. Several researchers and companies have done some remarkable jobs towards the goal of automatic recognizing electronics drawing by a computer recognizer.

1. Conventional Pattern Recognition Techniques Used in Electronics Engineering Drawing Recognition

1.1 Template Matching, Feature Extraction and Decision Tree

In 1987, a group of researchers at Toshiba Research and Development Center [90] developed an automatic circuit diagram reader with loop-structure-based symbol recognition. This system has a high-performance logic circuit diagram reader for VLSI-CAD data input. The basic concept for this design is that almost all logic circuit symbols include, at least, one loop structure. The component labeling is used to find the loops that may represent logical gates. The gate recognition is achieved by two processes: symbol segmentation and symbol identification. In particular, symbol identification is implemented by a hybrid method, which uses heuristics to mediate image processing techniques between template matching and feature extraction. The entire symbol recognition process is carried out under a decision tree control strategy. After the loops

have been extracted, they are preprocessed to a normal size, while a set of templates is used to match the loops. The recognized loops are either gate components or gate bodies. All of the features of a gate are extracted. The decision-tree control strategy is then used to judge the extracted features and recompose the specific gate symbol. An A1-size drawing can be read within 30 min. with more than 95% recognition accuracy in this system. The misrecognition error rate is less than 1%.

However, there are some limitations in this system. 1) Only drawings with a predetermined drawing rule can be recognized. So, there is a limited flexibility of input patterns. 2) The processing algorithm is very complicated. A special hardware is designed to implement the processing, because using software is very time consuming. 3) The recognition accuracy is relatively low when the input patterns are not drawn by predetermined rule. 4) If a segmented symbol pattern does not preserve the correct topology or is severely smeared with noise, this will lead to misclassify or reject.

1.2 Contour Tracking, Feature Extracting and Structure Analysis

In 1990, a group of researchers at Hitachi Ltd reported an automatic recognition of logic-circuit diagrams [91]. They used the contour extracting technique to extract features such as end points, branch points, cross points, corner points and loops. These extracted feature components can be used by the system to analyze and recognize the gates on a drawing logically. Theoretically this system can be used for recognizing non-standardized electronic engineering drawings. However, this system is still very complex. A hardware is also required to implement the complicated algorithm. In the experiments, this system can not handle practical pattern recognition problems within a satisfied misclassify and rejection rate. It is not useful in practical applications yet. By these researcher's

comments, more extensive research must be conducted on image processing and image understanding technology, and more effective algorithms must be developed for their system.

1.3 Vectorization and Conventional Neural Networks Pattern Recognition

In 1991, two researchers in the GTX Corporation used vectorization and fuzzy logic to recognize the electronics gates as well as conventional neural networks to recognize characters [93]. They built a GTX 5000 CAD Conversion System. There are eight Motorola 68020 processors and 80 megabytes of RAM in the pattern recognition module for the input pattern processing. The processes include: 1) contour tracing, 2) distance transforms, 3) line tracking, 4) polygonal segmentation, and 5) thinning. Contour tracing tracks the boundary of an object and generates a chain code for it. Other processes are used to extract the features of the pattern and to construct the information into a feature vector of the pattern. Conventional feed forward neural network with vectorization processing is used to recognize the electronic engineering drawing and characters in it.

There are many limitations of this system. It needs complicated algorithm to vectorize the pattern. In order to shorten the processing time, more hardware is needed to process the software program. The vectorization processing is very sensitive to the noise. Most working drawings are noisy that may result in incorrect vectorization. In general, the algorithm relies on fuzzy logical to accommodate the usually noise data. The user may exert some degree of control over the sensitivity of this processing by varying the values of a small set of tolerance parameters. If the tolerances are set too fine, symbols are missed; if too coarse, misclassifications occur. This system has difficulties in vectorizing the characters. If the system cannot vectorize the characters correctly, neither fuzzy

logical nor conventional feed forward neural network can be used to recognize the character correctly. The reason is that the input of the conventional neural networks is the vectorized information of the characters. The system recognizing accurate rates are 90% between the samples of '5' and 'S' and 98% between the samples of '8' and 'B.' Other problems are the recognition of touching and broken characters as well as the lowercase characters. The touching characters are defined as that they must be broken apart before they can be individually recognized. Lowercase characters present special problem that is great number of alphabetic/number ambiguities (for example, numeral "6" versus lowercase "b") and the great importance of relative positioning of characters. It has been recommended by these researchers that the future work should be directed to the development of more intelligent context processing algorithms with a prior, top-down information.

2. Image Analysis by the Method of Moments

In 1988, two researchers at University of Wisconsin presented their work about image analysis using the method of moments [92]. The advantage of using moments to analyze an image is invariant under image translation, scaling, and rotation. They studied the regular moment, the lower and higher order moments and addressed some fundamental questions, such as image representation ability, noise sensitivity, and information redundancy. They found that the higher order moments such as Legendre, Zernik, and pseudo-Zernik moments are better than other types of moments in image analysis and representation, but the higher order moments are more vulnerable to noise as compared to lower order moments. The research result showed that the moment analysis can be used

for other image analysis and image representation. However, it performs poorly in pattern recognition and very sensitive to the noise. In some cases, its misclassifying and rejecting rate is up to 30%. It can not accomplish the task for electronic engineering drawing recognition.

3. Neocognitron

In 1990, Dr. Kunihiko Fukushima and other researchers presented a neural network model of visual pattern recognition called the neocognitron, which was previously proposed by Fukushima [94]. It has the capability of recognizing deformation-invariant visual pattern. They constructed a pattern recognition system that works with the mechanism of the neocognitron. During the simulation, the system is trained to recognize 35 hand-written alphanumeric characters. The system has a large power of generalization. After this system learnt by presenting only a few typical examples of deformed patterns (or features), it has enough power to recognize all the deformed versions of patterns that might appear during the process of inputting future. Therefore, this system can recognize input pattern robustly, with little effect from deformation, changes in size, or shifts in position. This system does not require any preprocessing such as normalization of the position, size, or deformation of input patterns. The structure of the network is illustrated in Figure 1. In Figure 1, each rectangle represents a two-dimensional array of neurons. The lowest stage of the network is the input layer, which consists of a two-dimensional array of receptor neurons. Each succeeding stage has a layer of neurons called S neurons followed by another layer of neurons called C neurons. In the whole neural network, layers of S neurons and C neurons are arranged alternately. S neurons are feature-

extracting neurons. The C neurons are inserted in the network to allow position error in the features. The layer of C neurons at the highest stage is the recognition layer, representing the final result of a pattern recognition by the neocognitron. The notation Usl and Ucl is used to denote the layers of S neurons and C neurons of the l th stage, respectively. Each layer of S neurons or C neurons is divided into subgroups, called neuron planes, according to the feature to which they respond. The neurons in each neuron plan are arranged in a two-dimensional array. After finishing the training, S neuron is activated only when a pattern feature is presented at a certain position in the input layer. The features that the S neurons extract are determined by training pattern given to the input layer. In the higher stages, features that are more global are extracted, for example, a part of a training pattern. Each C neuron receives signal from a group of S neurons that extract the same feature, but from slightly different positions. The C neuron is activated if at least one of these S neurons is active. Hence, the C neuron can get some shift invariance. The neocognitron system is trained by supervised learning. The variable input connections of the S neuron are reinforced by the training. Their initial values before training are all zero. Training is performed step by step from lower stages to the higher stages. All of the stages are trained with the same process. As a result of this learning, all the S neurons are in a neuron plane work as templates to extract the same feature at different locations. The special feature of this neural network is the ability to correctly recognize deformed characters, which depends highly on the choice of the training pattern set, and its special supervised training procedure. The advantage of the features of this neural network is a very short training time as compared to the back-propagation training. There are some drawbacks of this neural network. First, the

structure of the neural network is large and complex, because, if the input pattern is complex, the neural network needs more neuron plans to extract more features of the input pattern. Second, although a skillful choice of training patterns can make the neocognitron discriminate between similar patterns of different categories, a process of constructing a good training pattern set requires very skillful hard human labor with an increase in the number of characters to be recognized and the quality of the training pattern set will dominate the pattern recognition capability of the neural network. And third, the conventional technique of unsupervised learning for the neocognitron [95,96], with which all the training processes progress automatically, shows a somewhat lesser ability to recognize deformed pattern. It means that the intelligence of this neocognitron system is actually a storage of the human intelligence. The machine can not create intelligence information by itself. This is the major drawback of this system.

4. Template Matching and Conventional Neural network

In 1989, a group of researchers in AT&T Bell Laboratories demonstrated a recognition of 10 numbers of different post zip code using a neural network [97]. They constructed a neural network as a conventional neural network. However, instead of using conventional image processes to find the vectorized input pattern features, a set of input feature maps is designed and a set of templates is used to extract the features in different positions of the input pattern to form a feature vector. A set of templates is designed at attempting to remove the main sources of meaningless variation and extract the meaningful information. It is known from biological studies [98] that the cat vision system is sensitive to certain features that occur in images, particular the lines and the ends of lines.

They designed a set of 49 different feature extractor templates for such kind of features. The output of each feature template is stored separately. These outputs are called feature maps, since they have information about features and their distinct positions in the image. It is possible, indeed very likely, that several different features will occur in the same place.

They also designed a preprocessor for scaling and deskewing to normalize the original image, and then skeletonization was used to process this normalized image. The neural network they used is a two layers neural network. The first layer contains 40 neurons, each of which receives all information from the 49 feature maps as a feature vector. The second layer, the output of the neural network, has 10 neurons. Each neuron receives all information from the output of each neuron in the first layer. Each neuron in the second layer represents a different post zip code number. This neural network is trained by back-propagation process. After training, the performance of the neural network shows that, if 14% of the images are rejected as unclassifiable, 1% of the remainders are misclassified. If no images are rejected, approximately 6% are misclassified. The misclassifying and rejecting rates of the system are still high for a practical application. The features extracting templates are the key point for this system. The pattern recognition capability of this system depends on the quality of the set of templates. To design a good template set, it needs very experienced human effort to accomplish this task.

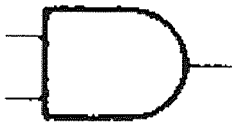
5. Weight Sharing Back-Propagation Neural Network (WSBPNN)

In 1990, a group of researchers of AT&T Bell Laboratories described a neural network for Handwritten Digit Recognition [99]. They designed this neural network with highly constrained weight sharing neural network architecture. Unlike the previous work on this subject [97], this neural network is directly fed with images, rather than feature vectors. Thus this neural network shows the ability to deal with large amounts of low level information. This neural network is designed at achieving a good generalization. In theory, a good generalization can only be obtain by designing a network architecture that contains a certain amount of a prior knowledge about the problem [89]. The basic principle of the design is to minimize the number of free parameters that must be determined by the learning algorithm, without reducing the computational power of the network. They use weight sharing neural network architecture to achieve this goal. They designed a neural network that has 3 layers. The first two layers used weight sharing principle, and each of them has a subsampling layer behind itself. Just as the neocognitron neural network, this design provides some shift invariance characteristics in the neural network. The first layer has four feature maps to extract the features of the input image. The difference of the first layer between this design and the previous one [97] is that the features extracted from the input image are determined by the templates designed by the designer's thought in the previous model, and in the new version the features extracted from the input image is determined by the trained neural network itself and the designer do not need to know what kind of features will be extracted by the neural network. The weight sharing principle is not used for output layer. It is a fully connected conventional neural network. The neural network is trained by back-propagation. After

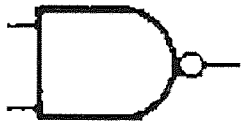
training, this neural network showed a good generalization property and obtained a better result than the previous model. This training database is the same database used in previous work. The test result of the new model is that the best percentage of rejections on the complete test set was 5.7% for 1% error. The original input image of this system has been normalized to a standard size followed by being inputted into the neural network. This system has several advantages. It has high variability to recognize an image pattern and runs at a reasonable speed on standard hardware (~10 characters/sec on a workstation) and high speed (~1000 characters/sec) on specialized hardware.

APPENDIX B

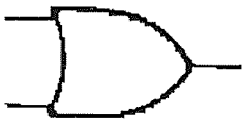
EIGHT GATE SYMBOLS (SCANNED IMAGES)



(a) AND

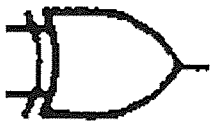


(b) NAND

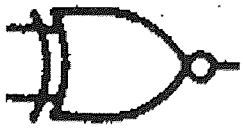


(c) OR

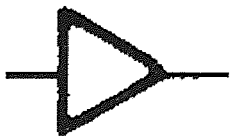
(d) NOR



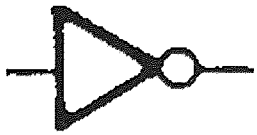
(e) X OR



(f) XNOR



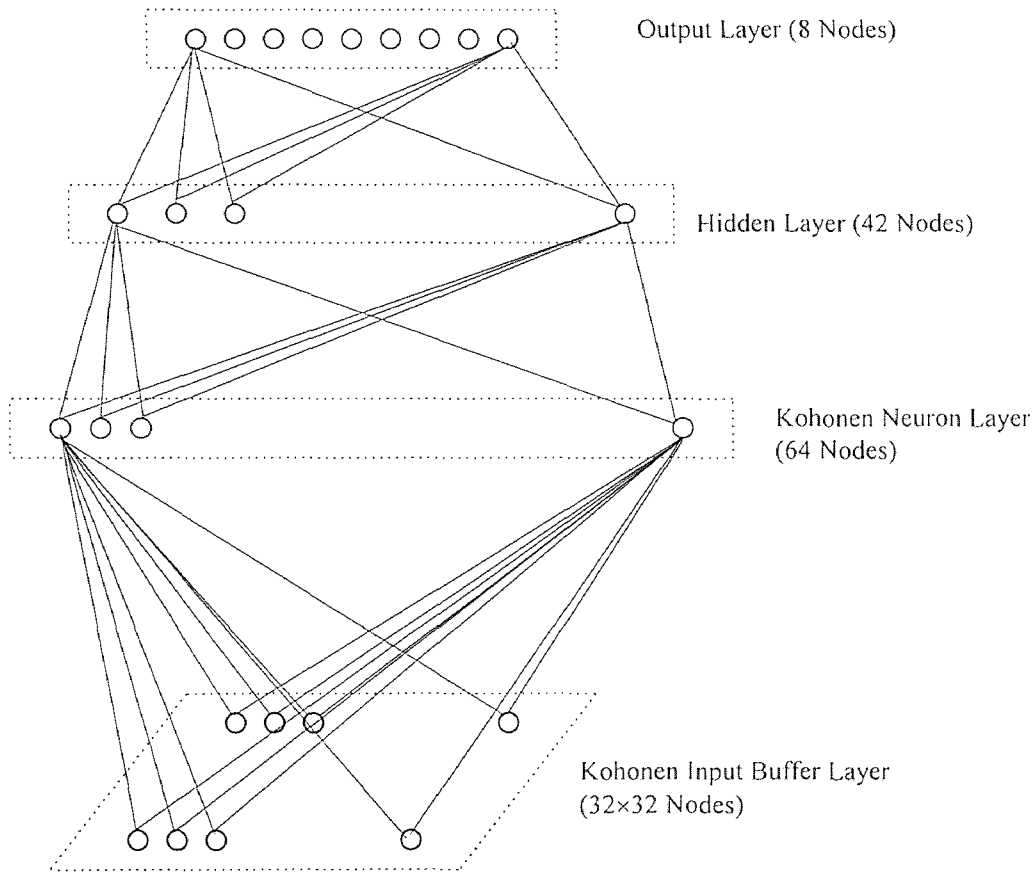
(f) Buffer



(g) NOT

APPENDIX C

NEURAL NETWORK DIAGRAM



Neural Network Recognition System (Kohonen + BP), Fully Connected Weights

REFERENCES

1. Wei Su and Gerald T. Michael, "A Simple and Quick Approach to Processing Drawing Images," *Proceeding of 1995, International Conference on Signal Processing Application & Technology*, Boston, MA., Aug. 1995.

2. Akio Okazaki et al., "An Automatic Circuit Diagram Reader with Loop-Structure-based Symbol Recognition," *IEEE, Transaction on Pattern Analysis and Machine Intelligence*, Vol. 10, pp. 331-341, May 1988.

3. Masakazu Ejiri, et al., "Automatic Recognition of Engineering Drawings and Maps," *Image Analysis Applications*, Rangachar, Kasturi and Mohan M. Trivedi, Eds., pp. 73-126, M. Dekker, NY., 1990.

4. Alan J. Filipski and Robert Flandrena. "Automated Conversion of Engineering Drawing to CAD form," *Proceeding of the IEEE*, Vol. 80, pp. 1195-1209, July 1992.

5. Cho-Huak Teh, Roland T. Chin, "On Image Analysis by the Methods of Moments," *IEEE Transaction on Pattern Analysis and, Machine Intelligence*, Vol. 10. No. 4, July 1988.

6. Kunihiko Fukushima and Nobuaki Wake, "Handwritten Alphanumeric Character Recognition by the Neocognitron," *IEEE, Transactions on Neural Network,* Vol. 2, pp. 119-130, May 1991.

7. Y. Le Cun, et al., "Handwritten Digital Recognition with a Back-Propagation Network," *Advances in Neural Network, Information Processing*, Vol. 2, pp. 396-404, D. S. Touretzky, Ed., Morgan Kaufmann, San Mateo, CA., 1990.

8. Y. Le Cun, et al., "Back-Propagation Applied to Handwritten Zipcode Recognition," *Neural Computation*, 1(4), pp. 541-551, 1989.

9. Robert Schalkoff, *Pattern Recognition: Statistical, Structural and Neural Approaches,* John Wiley & Sons, Inc., NY., 1992.

10. Rafael Gonzalez, Richard Woods, *Digital Image Processing*, Addison-Wesley, NY., 1993.

11. K. S. Fu, "Recent Developments in Pattern Recognition", *IEEE Transactions on Computers*, Vol. C-29, No. 10, pp. 845-857, Oct. 1980.

12. S. T. Bow, *Pattern Recognition,* Marcel Dekker, NY., 1984.

13. John Hertz, Andrers Krogh, Richard G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley Publishing Company, CA., 1991.

14. J. M. Zurada,, *Introduction to Artificial Neural Systems*, PWS Publishing Company, Boston, MA., 1992.

15. M. W. Roth, "Survey of Neural Network Technology for Automatic Target Recognition", *IEEE Transactions on Neural Networks*, Vol. 1, No. 1, p. 43, Mar. 1990.

16. R. D. Ripley, *Pattern Recognition and Neural Networks*, Cambridge University Press, Cambridge, UK., 1996.

17. R. M. Haralick, L. G. Shapiro, *Computer and Robot Vision*, Addison-Wesley, Reading, MA., 1992.

18. J. S. WesZka, "A Survey of Threshold Selection Techniques", *Computer Vision, Graphics, and Image Processing*, Vol. 7, No. 2, pp. 259-265, Apr. 1978.

19. Louisa Lam, Seong-Whan Lee, Ching Y. Suen, "Thinning Methodologies: A Comprehensive Survey", *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 14, No. 9, pp. 869-885, Sept. 1992.

20. Sebastiano Impedovo (Ed), *Fundamentals in Handwriting Recognition,* Springer Verlag, Germany, 1994.

21. Teuvo Kohonen, "The Self-Organizing Map", *Proc. IEEE,* Vol. 78 No. 9, pp. 1464-1480, Sept. 1990.

22. Lefteri Tsoukalas, *Fuzzy and Neural Approaches in Engineering*, John Wiley & Son, NY., 1997.

23. Steven K. Rogers, Matthew Kabrisky, *An Introduction to Biological and Artificial Neural Networks, for Pattern Recognition*, SPIE Optical Engineering Press, Bellingham, WA., 1991.

24. Steve Lawrence, C. Lee Giles and Ah Chung Tsoi, "What Size Neural Network Gives Optimal, Generalization?", Technical Report UMIACS-TR-96-22 and CS-TR-3617, Institute for Advanced, Computer Studies, University of Maryland, College Park, MD. 20742, June 1996.

25. Valluru B. Rao and Hayagriva V. Rao, *C++ Neural Networks and Fuzzy Logic*, MIS Press, NY., 1993.

26. Abhijit S. Pandya and Robert B. Macy, *Pattern Recognition with Neural Networks in C++*, CRC Press and IEEE Press, FL., 1996.

27. Simon Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan Publishing Company, CA., 1994.

28. L. Devroye, *A Probabilistic Theory of Pattern Recognition*, Springer, NY., 1996.

29. Martin T. Hagan, Howard B. Demuth, Mark Beale, *Neural Network Design*, PWS Publishing Company, MA., 1996.

30. B. Kosko, *Neural Networks and Fussy System,* Prentice-Hall, NJ., 1992.

31. T. Kohonen Neural Networks Research Center at: http://nucleus.hut.fi/nnrc/.

32. S. Amari, "Mathematical Foundations of Neurocomputing", *Proceedings of IEEE 78*, pp. 1443-1463, 1990.

33. P. D. Wasserman, *Advanced Methods in Neural Computing*, Van Nostrand Reinhold, NY., 1993.

34. Alan F. Murray (Ed), *Application of Neural Networks,* Kluwer Academic Publishers, Boston, MA., 1995.

35. Bart Kosko (Ed), *Neural Networks for Signal Processing*, Prentice Hall Inc., NJ., 1992.

36. Howard Demuth, Mark Beale, *Neural Network Toolbox for Use with MATLAB,* Version 3, The MATH WORKS Inc., MA., 1998.

37. N. M. Nasrabadi, Y. Feng, "Vector Quantization of Image Based upon Kohonen Self Organizing Feature Maps", *Proceeding IEEE Int. Conf. Neural Networks*, pp.1101-1108, 1988.

38. Ben Yuhas, Nirwan Ansari (Ed), *Neural Networks in Telecommunications*, Kluwer Academic Publishers, Boston, MA., 1994.

39. C. N. Manikopoulos, "Finite State Vector Quantization with Neural Network Classification of States", *IEE Proceedings-F*, Vol. 140, No.3, pp. 153-160, June 1993.

40. R. Hecht-Nielsen, *Neurocomputing*, Addison-Wesley, Reading, MA., 1990.

41. Bernard Widrow, and Michael A. Lehr, "30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation", *Proceedings of IEEE*, Vol. 78, No. 9, pp. 1415-1442, Sept. 1990.

42. Richard P. Lippmann, "An Introduction to Computing with Neural Nets", *IEEE ASSP Magazine*, pp. 4-22, Apr. 1987.

43. Yoh-Han Pao *Adaptive Pattern Recognition and Neural Network*, Addison-Wesley Publishing Company, Inc., NY., 1989.

44. Clifford Lau, *Neural Networks Theoretical Foundations and Analysis*, IEEE Press, 1992.

45. James D. McCafferty, *Human and Machine Vision Computing Perceptual Organization*, Ellis Horwood Limited, NY., 1990.

46. R. Rosenblatt, *Principles of Neurodynamics*, Spartan Books, NY., 1959.

47. M. Minsky, S. Papert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, MA., 1969.

48. D. E. Rumelhart, J. L. McClelland, "Learning Internal Representations by Error Propagation," *Parallel Distributed, Processing: Explorations in the Microstructures of Cognition, Vol. 1: Foundation*, 2nd ed., MIT Press, Cambridge, MA., 1986.

49. D. E. Rumelhart, J. L. McClelland, *Parallel Distributed, Processing: Explorations in the Microstructures of Cognition,* MIT Press, MA., 1986.

50. P. J. Werbos, *The Roots of Backpropagation*, John Wiley & Sons, NY., 1994.

51. Donald Tveter, "Backpropagation Review" at http://www.mcs.com/~drt/bprefs.html.

52. T. Masters, *Practical Neural Network Recipes in C++*, Academic Press, San Diego, CA., 1993.

53. R. O. Duda, P. E. Hart, *Pattern Classification and Scene Analysis*, John Wiley & Sons, NY., 1973.

54. B. Widrow, M. E. Hoff, "Adaptive Switching Circuits", *IRE WESCON Convention Record*, Part 4, pp. 96-104, 1960.

55. F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Sparton, Washington D. C., 1962.

56. G. Cybenko, "Approximation by Superpositions of a Sigmoidal Function", Research Note, Computer Science Department, Tufts University, Boston, MA., Oct. 1988.

57. A. E. Bryson, and Y. C. Ho, *Applied Optimal Control,* Blaisdell, NY., 1969.

58. P. Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences". Ph.D. Thesis, Harvard University, 1974.

59. D. B. Parker, "Learning Logic", Technical Report TR-47, Center for Computational Research in Economics and Management Science, Massachusetts Institute of Technology, Cambridge, MA., 1985.

60. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-Propagating Errors. *Nature*, No. 323, pp. 533-536, Reprinted in Anderson and Rosenfeld, 1988.

61. Le Cun, "Backpropgation Applied to Handwriitten Zip Code Recognition", *Neural Computation* No. 1, pp. 541-551, 1989.

62. C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, 1995.

63. K. Hornik, "Some New Results on Neural Network Approximation", *Neural Networks*, No. 6, pp. 1069-1072, 1993.

64. D. S. Touretzky, D. A. Pomerleau, "What's Hidden in the Hidden Layer?", *Byte*, No. 14, pp. 227-233, 1989.

65. Z. Luo, "On the Convergence of the LMS Algorithm with Adaptive Learning Rate for Linear Feedforward Networks", *Neural Computation*, No. 3, pp. 226-245, 1991.

66. MIT Neural Network FAQ at: ftp://rtfm.mit.edu/pub/usenet-by-group/news.answers/ai-faq/neural-nets/.

67. Rejean Plamondon, "A Model-Based Dynamic Signiture Verification System", Sebastiano Impedovo (Ed), *Fundamentals in Handwriting Recognition,* pp. 417-434, Springer Verlag, Germany, 1994.

68. Giuseppe Pirlo, "Algorithms for Signature Verification", Sebastiano Impedovo (Ed), *Fundamentals in Handwriting Recognition,* pp. 435-454, Springer Verlag, Germany, 1994.

69. Fathallah Nouboud, "Handwritten Signature Verification: A Global Approach", Sebastiano Impedovo (Ed), *Fundamentals in Handwriting Recognition,* pp. 455-459, Springer Verlag, Germany, 1994.

70. Plamondon R., Lorette G., "Automatic Signature Verification and Writer Identification. State of Art", *Pattern Recognition*, Vol. 22, No. 2, pp. 107-131, 1989.

89. Denker, J., Schwartz, D., Wittner, B., Solla, S., Howard, R., Jackel, L., & Hopfield, J., "Large Automatic Learning, Rule Extraction and Generalization", *Complex Systems*, Vol. 1, pp. 877-922, Complex Systems Publications Inc., Champaign, IL., 1987.

90. Akio Okazaki, Takashi Kondo, Kazuhiro Mori, Shou Tsunekawa, and Eiji Kawamoto, "An Automatic Circuit Diagram Reader with Loop-Structure-Based Symbol Recognition," *IEEE Transaction on Pattern Analysis and Machine Intelligence*, Vol. 10, pp. 331-341, May 1988.

91. Masakazu Ejiri, Shigeru Kakumoto, Takafumi Miyatake, Shigeru Shimada, and Kazuaki Lwamura, "Automatic Recognition of Engineering Drawings and Maps," *Image Analysis Applications*, Rangachar Kasturi and Mohan M. Trivedi, Eds., pp. 73-126, M. Dekker, New York, NY., 1990.

92. Cho-Huak The, and Roland T. Chin, "On Image Analysis by the Methods of Moments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10, pp. 496-513, July 1988.

93. Alan J. Filipski, and Robert Flandrena, "Automated Conversion of Engineering Drawing to CAD Form," *Proceedings of the IEEE*, Vol. 80, pp. 1195-1209, July 1992.

94. Kunihiko Fukushima and Nobuaki Wake, "Handwritten Alphanumeric Character Recognition by the Neocognitron," *IEEE Transactions on Neural Network*, Vol. 2, pp. 119-130, May 1991.

95. K. Fukushima, "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position," *Biol. Cybern.*, Vol. 36, pp.193-202, 1980.

96. K. Fukushima and S. Miyake, "Neocognitron: A New Algorithm for Pattern Recognition Tolerant of Deformations and Shifts in Position." *Pattern Recognition.*, Vol. 15, pp. 455-469, 1982.

97. Denker, J. S., Gardner, W. R., Graf, H.P., Henderson, D., Howard, R. E., Hubbard, W., Jackel, L. D., Baird, H. S., and Guyon, I., "Neural Network Recognizer for Hand-Written Zip Code Digits", *Neural Information Processing Systems*, Touretzky, D., Ed., Vol. 1, pp. 323-331, Morgan Kaufmann, San Mateo, CA., 1988.

98. D. H. Hubel and T. N. Wiesel, "Receptive Fields, Binocular Interaction and Functional Architecture in the Cat's Visual Cortex," *Journal of Physiology*, Vol. 160, pp. 106-154, 1962.

99. Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R.E., Howard, W. Hubbard, and L. D. Jackel, "Handwritten Digit Recognition with a Back-Propagation Network," in *Advances in Neural Information Processing*, D. S. Touretzky, Ed., Vol. 2, pp. 396-404, Morgan Kaufmann, San Mateo, CA., 1990.