

PERFORMANCE COMPARISON OF NEURAL NETWORK
ARCHITECTURES FOR HANDPRINTED
CHARACTER RECOGNITION

By

MAJED NASER

Bachelor of science

Oklahoma State University

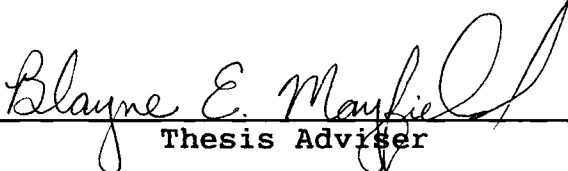
Stillwater, Oklahoma

1991

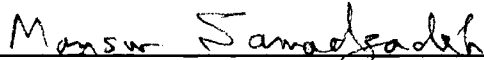
Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1995

PERFORMANCE COMPARISON OF NEURAL NETWORK
ARCHITECTURES FOR HANDPRINTED
CHARACTER RECOGNITION

Thesis Approved:




Thesis Adviser



Committee Member



Committee Member



Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to thank my advisor Dr. Blayne Mayfield for his constructive support, assistance, and patience. My sincere appreciation extends to my committee members Dr. M. Samadzadeh, and Dr. H. Lu, whose suggestions and guidance are also invaluable.

Moreover, I wish to express a special thanks to Ms. Amanda Smith and Mr. Ibrahim Alsqire for their strong encouragement, and invaluable suggestions during the whole process.

Finally, I would like to thank all the people who offered their friendship to me during the two years of study.

TABLE OF CONTENTS

Chapter	Page
I. INTRODUCTION	1
II. OPTICAL CHARACTER RECOGNITION (OCR)	3
2.1 OCR Application	3
2.2 OCR Techniques	4
2.2.1 Template Matching	4
2.2.2 Feature Analysis	6
III. ARTIFICIAL NEURAL NETWORKS	8
3.1 Definition of a Neural Network	8
3.2 Neural Network Connections	9
3.3 Neural Networks Learning Laws	10
3.4 Backpropagation Neural Networks	12
3.5 Self Organizing Map (SOM)	17
3.6 Counterpropagation Network (CPN)	24
3.7 Neural Network as Character Recognizer	28
IV. NETWORK IMPLEMENTATION AND TECHNIQUE	31
4.1 Neural Network Architecture and Learning Algorithms	31
4.2 Data Representation	31
4.3 Research Scope	34
4.4 Hardware and Software Requirement	34
4.5 Description of the Algorithms	34
4.6 Other Programs	35
4.7 File Formats	36
4.8 Stages Required in Evaluation A Network	37
V. NETWORK PERFORMANCE ANALYSIS	39
5.1 Row (Normal Data Representation)	39
5.2 Shifting Data	41
5.3 Crossing Technique	42
5.4 SOM Architecture Analysis	43
5.4.1 Radius Size	43
5.4.2 Kohonen Layer Size	44
5.4.3 Alpha Variable	44

5.5 Backpropagation Architecture Analysis	45
5.5.1 Hidden Layer Size	45
5.5.2 Alpha Variable	46
5.5.3 Network Overtraining	47
5.5.4 Network Convergence Rate	47
5.6 Counterpropagation Architecture Analysis	47
5.7 Feature Extraction Analysis	48
VI. CONCLUSION AND FUTURE RESEARCH	49
6.1 Backpropagation Research	49
6.2 Counterpropagation Research	50
6.3 Self-Organizing Map Research	46
REFERENCES	51
APPENDIXES	53
APPENDIX A--SAMPLE OF BATCH FILES	53
APPENDIX B--NETWORKS PARAMETERS	55
APPENDIX C--PERFORMANCE PROGRESS TABLES FOR THE ANNS	60

LIST OF TABLES

Table	Page
1. Networks Performance on Normal Data	39
2. Networks Performance on Data Shifting	41
3. Networks Performance on Crossing Data	42
4. CPN Performance	48
5. Performance of Feature Extraction	48

LIST OF FIGURES

Figure	Page
1. Simple Template for the Number 5	4
2. Varied Template Matching for Number Zero	5
3. Crossing Technique	7
4. Backpropagation Neural Network	12
5. Training Set Error vs. Test Set Error as a Function of the Training Cycles	14
6. SOM Architecture	17
7. Hexagonal and Rectangular Topologies	18
8. Mexican Hat Function	19
9. On-center/Off-surround Structure	19
10. Graphical Representation of Kohonen Layer Nodes and the Labels	24
11. Counterpropagation Network	24
12. Fonts Sample	29
13. Handwritten Character Grid for the Letter C	32
14. Character C After Data Shifting is Performed	33
15. Performance vs. Epochs for Normal Data	40
16. Performance vs. Epochs for Shifting Data	42
17. Radius of Neighborhood vs. Accuracy	44
18. SOM Performance vs. Epochs	45
19. Backpropagation Network Overtraining	46
20. BPNN Performance vs. Epochs	47

NOMENCLATURE

α	Learning Rate
ASCII	American Standard Character Information Interchange
ANSI	American National Standards Institute
BPNN	Backpropagation Neural Network
Ca	Grossberg Learning Rate
CPN	Counterpropagation Network
Intel	Registered Trade Mark For Intel Corporation
MB	Mega Byte
OCR	Optical Character Recognition
RAM	Random Access Memory
SOM	Self-Organizing Map

Chapter I

INTRODUCTION

Artificial neural networks (ANNs) is a subject that has captivated the interest of thousands of technologists, scientists, and mathematicians. The idea of training, instead of programming, a system to learn information processing functions has intrinsic appeal [Niel90].

Since the development of digital computers in mid 1940's, the basic approach of solving a problem using programs involved devising an algorithm and/or a set of rules for solving the problem [Niel90]. In order to accomplish such tasks, the problem must be described in terms of known algorithms or rules. If such rules or algorithms are not known, then they must be developed. In some cases the problem is so complicated that its solution requires a considerable amount of time. On the other hand, artificial neural networks don't require a set of rules or algorithms to be able to solve a problem; they can learn to solve a problem on their own. This approach can significantly reduce the quantity of software that must be developed in order to solve problems [Niel90].

One of the problem areas in which artificial neural networks have shown significant success is that of character recognition. The basic idea is to teach a neural network to recognize and classify written characters. The artificial

neural network can learn to recognize one or more sets of characters such as typed, or hand-written characters.

Chapter II discuss the Optical Character Recognition (OCR) applications, and the standard methods used in the recognition stage. Chapter III introduces the theory of Artificial Neural Networks: its definition, learning laws used, and architecture. Chapters IV, and V describe the steps taken to evaluate the performance of several neural network architectures and the methods used in such evaluations. Chapter VI, gives a conclusion and recommendations for future research.

Chapter II

OPTICAL CHARACTER RECOGNITION

The topic of character recognition is not a new research field. Relevant literature can be traced back as early as 1870 [Govi90]. The first successful application was developed by the Russian scientist Tyurin in 1900. In recent years, the developments of optical character recognition (OCR) applications expanded from English or Latin to Chinese, Japanese, Arabic, and the characters of many other languages [Govi90][Chou91][Upda92]. The next few sections discuss the topic of OCR in more detail, explaining its benefits and the methods it uses in recognizing characters.

2.1 OCR Application

OCR technology has many practical uses in the real world. For example, it can be used in reading tools especially for blind people, in telecommunication applications for the deaf, in postal departments for postal address reading, and as reader for handwritten and printed zip codes. It can be also used in the publishing industry and as a reader for data communication terminals. Moreover, it can be used in large-scale processing of document and in financial business applications such as cheque sorting, digital bar code reading, and health insurance data acquisition.

2.2 Standard OCR Techniques

Early attempts to develop OCR applications were encouraging for a limited set of characters. The principle incentive for the development of OCR systems is the need to cope with the enormous flood of paper such as bank checks, commercial forms, government records, credit card imprints, and mail generated by the expanding technologies in society [Govi90]. The two methods used to recognize characters were *template matching* and *feature analysis and match* [Govi90][Impe91].

2.2.1 Template Matching and Correlation Technique

Template matching and correlation method of OCR directly compares an input character to a standard set of prototypes stored in the system. Figure 1 shows a template of the number 5.

```
00000000
01111110
01000000
01000000
01111110
00000010
00000010
00000010
01111110
00000000
```

Figure 1. Simple template for number 5

The prototype that matches most closely with the input character will provide the corresponding character[Impe91].

However, the disadvantages of this technique are its sensitivity to noise and its failure to adapt to differences in writing style. Current OCR applications that use template matching rely on advanced template matching techniques such as the *variant template matching* [Impe91]. In this method, matching can be performed using logical rules. Figure 2 shows a varied template for the number zero. In this case a character will be recognized as zero if it satisfies the following rules:

(at least 5 'a's are '1' or at least 4 'e's are '1') and
 (at least 5 'b's are '1' or at least 4 'f's are '1') and
 (at least 5 'c's are '1' or at least 4 'g's are '1') and
 (at least 5 'd's are '1' or at least 4 'h's are '1') and
 (at least 3 'i's are '0') and
 (at least 3 'j's are '0').

Where '1' represent the dark pixels that form the character, and '0' represent the blank pixels.

```

0000000000
000aabb000
00aeffb000
0ae0000fb0
0ae0ii0fb0
0aeoii0fb0
0ae0000fb0
0cg0000hd0
0cg0jj0hd0
0cg0jj0hd0
0cg0000hd0
00cgghhd00
000cc0d000
0000000000
  
```

Figure 2. Varied Template Matching for Number Zero

2.2.2 Feature Analysis and Match

In the Feature Analysis and Match method, a specific number of features are extracted from the input character and then compared to the feature description of ideal characters.

The description that matches most closely provides recognition. A few of the problems such methods fail to adequately address are [Impe91]:

- A. Shape discrimination: A single character has a wide variety of fonts, e.g. Gothic, Elite and Orator. Moreover, handwritten characters have an infinite number of free styles.
- B. Deformation of the image: This problem can be classified into three parts. First, noise, such as holes or breaks in a line or isolated dots. Second, translation, e.g. the movement of an entire character or its components. Finally, Rotation, e.g. a change in the orientation of a character.
- C. Variation in sizes and pitch of the characters: The pitch of 10, 12, or 17 specifies that there are 10, 12, or 17 characters per inch (cpi), 10 pitch characters are usually larger in both width and height than those in 12 pitch.

One algorithm used in feature extraction is called the *Crossing technique* [Impe91]. In this technique, features are measured from the numbers of times the character shape is crossed by vectors along certain directions, e.g.,

0° , 45° , 90° , or 190° . Figure 3 illustrates a 45° crossing method for the number two [Impe91]. The advantage of such a technique is its tolerance to distortions and small stylistic variations.

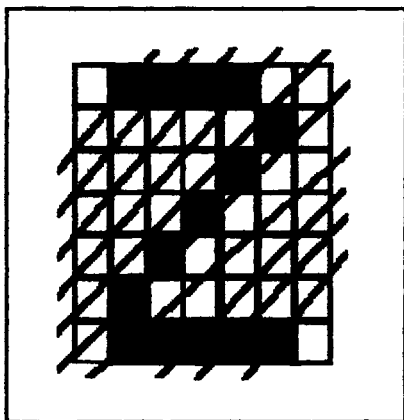


Figure 3. Crossing Technique [Impe91]

Several other techniques can be found in the literature and some are used widely in the OCR software industry [Impe91] [Govi90] [Guyo91] [Down91].

Chapter III

ARTIFICIAL NEURAL NETWORKS

Artificial neural networks (ANNs) are essentially simple structures. However, it is helpful to have a general idea about the fundamental structures of a neural network and a brief description of its components.

3.1 Definition of a Neural Network

A directed graph consists of a set of points (called nodes or vertices) along with a set of directed line (called links or edges) between them. An artificial neural network is a parallel distributed information processing structure in the form of a directed graph, with the following sub-definitions and restrictions [Niel90]. The nodes of the graph are called processing elements and the links of the graph are called connections. Each connection functions as an instantaneous unidirectional signal-conduction path which is assigned a value called *weight*. A processing element can receive any number of incoming connections (also called input connections) and can have any number of outgoing connections. The signals in all of the outgoing connections of a processing element must be the same. Processing elements can have local memory and a transfer function that uses (and alters) the local memory along with input signals to produce the processing element's output signal. In other words, the only inputs allowed to the transfer function are the values stored in the processing element's local memory

and the current values of the input signals in the connections received by the processing element. The only outputs allowed from the transfer function are values to be stored in the processing elements local memory and the processing element's output signal. Transfer functions can operate continuously or episodically. If they operate episodically, there must be an input called "activate" that causes the processing element's transfer function to operate on the current input signals and local memory values to produce an updated output signal (and possibly to modify local memory values). Continuous processing elements always operate. The "activate" input arrives via a connection from a scheduling processing element that is part of the network. Finally, input signals from outside the network arrive via connections that originate in the outside world. Outputs from the network to the outside world are connections that leave the network [Niel90].

3.2 Neural Network Connections

The data type of the signals carried by the connections of the various layers of a network can vary from one application to another. The signal type can be of an unlimited variety of mathematical data types (real numbers, complex numbers or integers, etc.).

3.3 Neural Networks Learning Laws

There are several classical neural network learning laws [Niel90]. These laws are written as equations that can be used in the training of a network. The two basic categories of learning for ANNs: supervised and unsupervised learning laws. There are several types of learning laws under each category.

The supervised learning implies that the network receives an input vector X and emits a vector Y . The network is trained by supplying it with a sequence of input/output pairs (X, Y') . X is the input vector and Y' is the desired or correct output vector. During training, the network compares its actual output Y with the output given to it by Y' . The network then computes the difference between y' and y and makes the adjustment in the weight vector(s) W of the processing connections. Unsupervised learning does not require the user to provide output vector Y' . Instead, the network modifies itself in response to the input vector X only. The process is called *Competitive Learning*, where a competition process involving some or all of the processing elements of the neural network takes place to determine which node will be able to change its weight vector to represent the input vector, before each episode of learning [Niel90].

The learning in ANNs is accomplished through the modification of the processing elements weight vectors using

the learning law of the processing element. In an ANN with N processing elements (assumed to be indexed from 1 to N) with its weight modified by the learning law, a *network weight vector*, is the vector formed by concatenating all of the weights of all of the individual processing elements of the network. The network weight vector can be written as:

$$\begin{aligned} \mathbf{w} &= (w_{11}, w_{12}, \dots, w_{1n}, w_{21}, w_{22}, \dots, w_{2n}, w_{n1}, w_{n2}, \dots, w_{nN}) \\ &= (w_1, w_2, \dots, w_N) \end{aligned}$$

Where w_{xy} is the weight associated with the connection between node x in layer 1 and node y in layer 2, and the vectors w_1, w_2, \dots, w_N are the processing element weight vectors of the processing elements 1, 2, ..., N , respectively.

3.3.1 Neural Networks Training and Test Sets

The input vectors used to train and test ANNs is divided into two sets. The first used contains input vectors and/or output vectors to train the network. This set is called the *training set*. In order to evaluate the networks performance after many epochs (training cycles) of training the network is supplied with another set of input vectors. This set is called the *test set*. The network then performs the computation required and produces its output. The network's output is compared with the actual output and the percentage of successful classifications by the network shows the level of the network's performance.

Some learning laws include a variable called the learning rate (usually denoted by α) with a recommended

value $0 < \alpha \leq 1$. The learning rate is used to control the change in weight vectors values during training. The learning law usually start with a high number ($\alpha > 0.70$) and then reduced to zero by time (in some networks such as BPNN the learning rate remains constant). The reason behind such values is that at the beginning the values of the weights are far from the optimal weights needed to train the network, therefore a giant changes are required (i.e., "coarse" adjustment). However, during many training cycles the weight vectors are close to the optimal value (Performance error is minimal) and therefore, a smaller changes in the weight vectors is required (i.e., "fine" adjustment).

3.4 Backpropagation Neural Network

The backpropagation neural network (BPNN) is one of the most widely used neural networks. The BPNN used in this research consists of an input layer, an output layer and one or more hidden layers (Figure 4). A hidden layer plays a particular role in the learning process. However, the user is not aware of its output signals. The only visible vectors are the input vectors and the output vectors of the output layer. Any other computation are "hidden" from the user. Therefore, they are hidden units. The dimensions of the input layer is equivalent in size to the input vector for the network plus the *threshold node*. The threshold unit is a unit with input signal set to 1, and a weight vector w .

The purpose of the threshold unit is to move the weight vector to achieve the desired classification. The output layer size is equal to the number of outputs the network must generate. The learning algorithm is called backpropagation and typically starts with a random set of

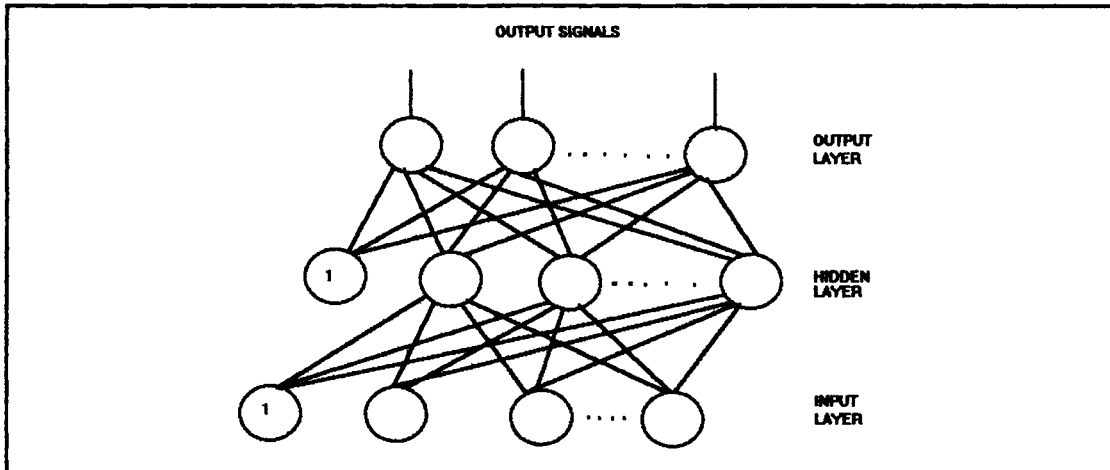
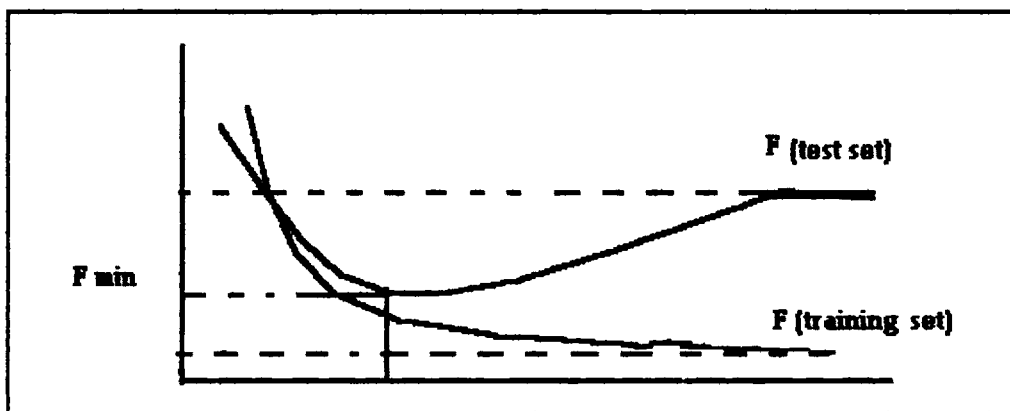


Figure 4. Backpropagation Neural Network [Rich91]

weights W_{1ij} (input to hidden layer weight vectors) and W_{2ij} (Hidden to output layer weight vectors). The weights are set to random values, the recommended values of the weights are between -0.1 and 0.1 . If the network consist of N ($N > 1$) hidden layers then the network consist of $N+2$ layers (input layer (layer 1) , output layer (layer $N + 2$) and N hidden layers. The weight vector W_n is the vector weight from layer n to layer $n+1$, for $n = 1$ to $N+1$. Each training cycle requires two stages: a forward pass and a backward pass. In the forward pass, an input-output (x,y)

pair is presented to the network, and the network propagates the input vector to the output layer. Each node computes its potential y (output signal) using the activation function: $y = 1 / (1 + e^{-\text{sum}})$, where sum is the weighted sum of the inputs to the unit ($\sum x.w$). In the backward stage, the network computes the error between the actual network output and the desired output. Then, the network adjusts the weights of the processing elements vectors to minimize the error.

The major disadvantage of BPNN architecture is overtraining the Backpropagation network (see figure 5) [Niel90][Zeid91].



(---) : Approximate performance level of network in operational environment if training is stopped.

Figure 5. Training Set Error vs. Test Set Error as a Function of the Number of Training Cycles

Figure 5 shows that as training of BPNN progresses, the network performance error decreases over both the training and test sets until it reaches a specific value then they level out. However, if the same set of training examples are presented repeatedly to the network, the performance error of the training set remains unchanged. However, the test set performance error starts to increase until it level out again. This phenomenon is called "memorization". To avoid such phenomenon, the training of the network should stop at the point that the performance error of the test set starts to increase and the train set performance error starts to increase. However, this strategy does not work all the time. In some cases the performance error of the test set starts to increase then, after more training cycles it starts to decrease again.

The supervised learning algorithm used in training the BPNN is as follows [Rich91]:

1. Let A be the number of units in the input layer; B the number of units in the hidden layer; and C the number units in the output layer (Figure 5). Note that one unit is added to both the input layer and the hidden layer to act as a threshold. Set the thresholds for the input layer x_0 and the threshold for the hidden layer h_0 to 1.

2. Initialize the weights

$W1_{ij}$ ($i = 0 \dots A$, $j = 1 \dots B$) and

$W2_{ij}$ ($i = 0 \dots B$, $j = 1 \dots C$), with random numbers between -0.1 and 0.1.

3. Choose an input-output pair (x_i , y_j).

4. Propagate the activation from the units in the input layer to the units of hidden layer using the activation function

$$h_i = \frac{1}{1 + e^{-\sum_{i=0}^A W1_{ij} x_i}}$$

where $j = 1, \dots, B$.

5. Propagate activation from the units in the hidden layer to the output layer using the activation function

$$o_i = \frac{1}{1 + e^{-\sum_{i=0}^B W2_{ij} h_i}}$$

where $i = 1, \dots, C$.

6. Compute the error vector δ_2 in the output layer units using the function

$$\delta_{2j} = o_j(1 - o_j)(y_j - o_j)$$

where $j = 1, \dots, C$.

7. Compute the error vector δ_1 in the hidden layer units using the function

$$\delta 1_j = h_j(1-h_j) \sum_{i=1}^C \delta 2_i \cdot W 2_{ji}$$

where $j = 1, \dots, B$.

8. Adjust the weights between the hidden layer and the output layer using the function

$$\Delta W 2_{ij} = \eta \cdot \delta 2_j \cdot h_i$$

where η is the learning rate, $i = 0, \dots, B$, and $j = 1, \dots, C$.

9. Adjust the weights between the hidden layer and the output layer using the function

$$\Delta W 1_{ij} = \eta \cdot \delta 1_j \cdot x_i$$

where η is the learning rate, $i = 0, \dots, A$, and $j = 1, \dots, B$.

11. go to step 3 and repeat. When all the input-output pair have been presented to the network, one epoch has been completed. Repeat as many times as desired.

3.5 Self-Organizing Map (SOM)

The Self-Organizing Map (SOM) is an artificial neural network architecture proposed by Teuvo Kohonen during the period 1979-1982 [Niel90][Koho89]. The Kohonen model of Self-Organization is based on the notion that the brain

tends to compress and organize sensory data spontaneously [Zeid91].

The architecture of the SOM used in this research is defined to be a two-dimensional map that, from a random starting point, can find the natural relations among patterns without any outside guidance [Hiot93]. The SOM consists of two layers, the input layer, and a the two-dimensional map (some SOM networks use one-dimension or more) also called Kohonen layer (see Figure 6). The SOM network is trained using an unsupervised learning algorithm.

Winning Node

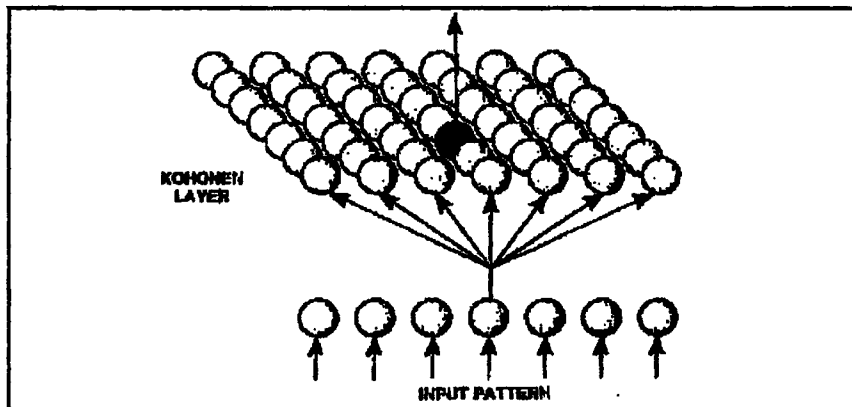


Figure 6. SOM Architecture [Caud93]

The two layers are fully connected [Caud93], [Hiot93]. The number of input nodes in a network determines the dimensions that the SOM has to work with. The two-dimensional Kohonen layer processing elements can be arranged in two ways, the

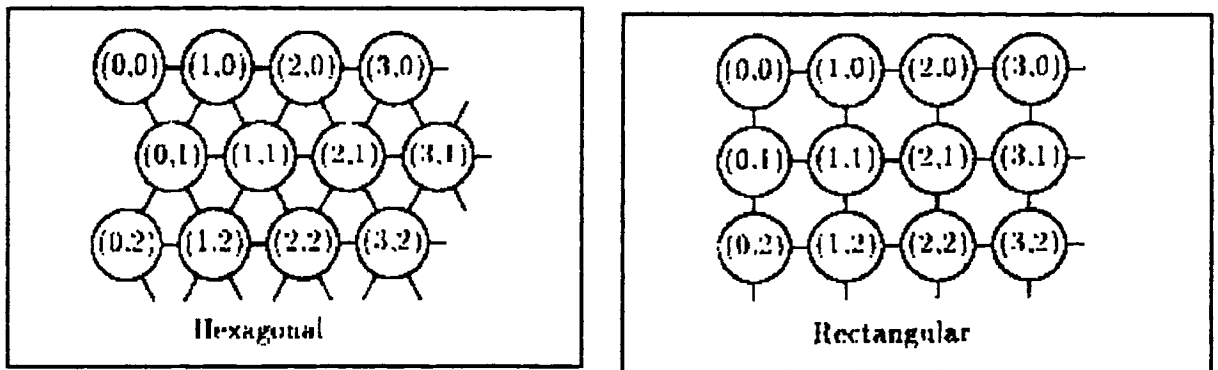


Figure 7. Hexagonal and Rectangular Topologies

Hexagonal or the Rectangular representations (see Figure 7). The hexagonal topology is more effective for visual display [Koho89]. This is true since the edges of the array of nodes ought to be rectangular rather than square.

Kohonen built his architecture from his observation that neural networks in the brain tend to consist of layers of neurons. The learning process of the layers comply with the *Mexican Hat* function (Figure 8). According to this function an excited unit tend to excite units of a certain distance from it, and inhibits those units that are not near. Implementing the "Mexican Hat" function as part of the learning algorithm for the SOM network is an expensive process in terms of computer time. Consequently, the SOM architecture takes a short cut to achieve the effect of the *Mexican Hat* function.

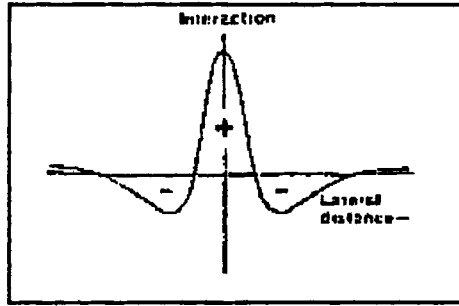


Figure 8. "Mexican Hat" function [Zeid91]

Figure 9 shows a commonly encountered architectural arrangement called an *on-center/off-surround* or *lateral-inhibition* structure. Each processing element (represented

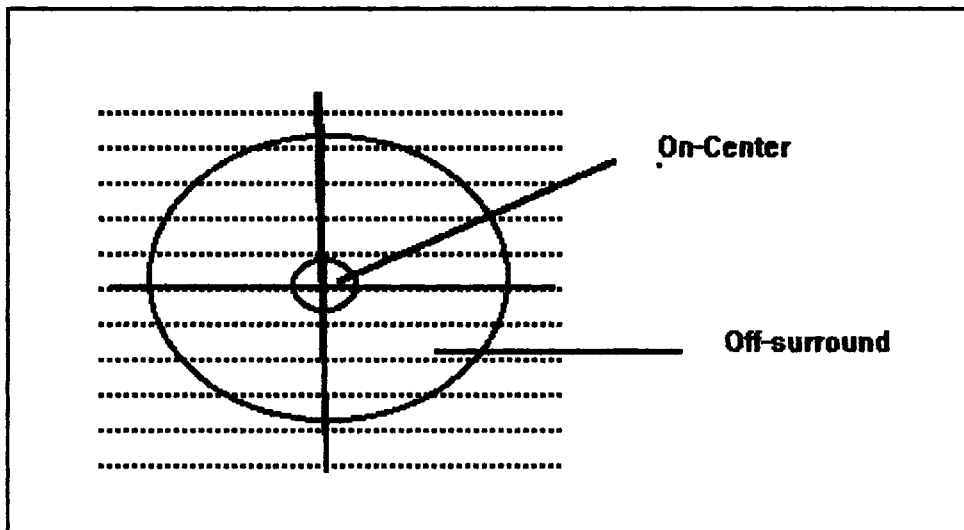


Figure 9. On-Center/Off-Surround Structure

by a dot) in this case, receives two different classes of input, "excitatory" class 1 inputs from nearby processing elements, and "inhibitory" class 2 inputs from more distance processing elements. The effects of a single processing element's output on surrounding processing (Neighborhood) elements comply or resemble the effect of the "Mexican Hat" function [Niel90][Zeid91]. In SOM architecture there is no lateral connections between the processing elements of the Kohonen layer. That is, there is no links between nodes and there is no propagation of signals between nodes of the Kohonen layer. Nevertheless, the weight update is modified to include neighborhood nodes as described in the *on-center/off-surround* structure [Hert91].

The advantage of using this type of architecture is that when a node is allowed to change its weight vector to close the Euclidean distance with the input vector, the nodes with the certain neighborhood distance will also change their weights. In this case more variation of the input vector are represented in the Kohonen layer.

During training, the nodes that are topologically close in the array up to a certain distance (radius) will activate each other to learn from the same input [Koho89]. This distance is called the *neighborhood Kernel*. The two popular functions used to determine this neighborhood are the "bubble" function and the Gaussian function. The bubble function refers to a neighborhood set of array nodes in the

Kohonen layer [Koho89]. Let N_c denote this set of nodes; then the neighborhood Kernel h_{ci} is defined as $h_{ci} = \alpha(t)$ if $i \in N_c$ and $h_{ci} = 0$, if $i \notin N_c$, where $\alpha(t)$ is some monotonically decreasing function of time ($0 < \alpha < 1$), i is a node in the Kohonen layer, c is a defined radius. The second function is the Gaussian function

$$h_{ci} = \alpha(t) * \exp \left(\frac{- ||r_c - r_i||^2}{2 \sigma^2(t)} \right)$$

where $\alpha(t)$ is the learning rate, and $\sigma(t)$ defines the width of the Kernel; r_i and r_c are the radius vectors of nodes c and i in the Kohonen layer, and $||r_c - r_i||$ is the Euclidean distance. The final result of training the SOM network is that similar input vectors are presented by one or more nodes in the Kohonen layer. Each node that represent a class of input vectors will be labeled by that class.

Studying the SOM architecture, several advantages can be found. First, the SOM has fewer parameters to adjust during the learning process than most other networks. Second, the SOM can use an unsupervised learning algorithm or a supervised learning algorithm called Learning Vector Quantization (LVQ). Third, learning time is several times faster than that of backpropagation since the learning laws of the SOM require the adjustment of fewer processing elements during training compares to backpropagation. The

backpropagation requires the adjustment of all weights of the processing elements in every layer. Fourth, when the SOM reaches its optimal performance within specified number of nodes, increasing the number of nodes will not decrease the performance of the network. This is also true for the Counterpropagation network but not for the Backpropagation network. Finally, the network learning laws tend to decrease the weight adjustment of its nodes to a very insignificant number when it reaches its equilibrium. As a result, the problem of overtraining the network will be eliminated.

Several learning algorithms have been proposed for training the SOM network [Caud93] [Hiot93] [Teno90]. The two basic learning algorithms are the unsupervised algorithm (Kohonen law) and the supervised learning algorithm known as Learning Vector Quantization (LVQ). The process of mapping input vectors to the Kohonen layer is usually done with the unsupervised learning algorithm. The process of learning in the SOM involves several steps. The following are brief points of the steps used in the unsupervised learning [Caud93] [Zeid91]:

0. Define X_i to be an input vector, and W_{ij} to be the weight vectors from the input layer to the Kohonen layer.

The weight vectors are normalized vectors with initial values set to random numbers. The recommended value is between 0.0 and 1.0.

1. The node with the smallest Euclidean distance from the input vector X_i is considered the best matching node. The equation used to compute the Euclidean distance I_i of neurode i is:

$$I_i = \sum_{j=1}^n W_{ij} X_i$$

2. The reference vectors of the winning node and those of neighboring nodes within a determined neighborhood will be modified. The equation to modify the weights of the nodes is:

$$W_{ij}^{new} - W_{ij}^{old} = \eta (X_i - W_{ij}^{old})$$

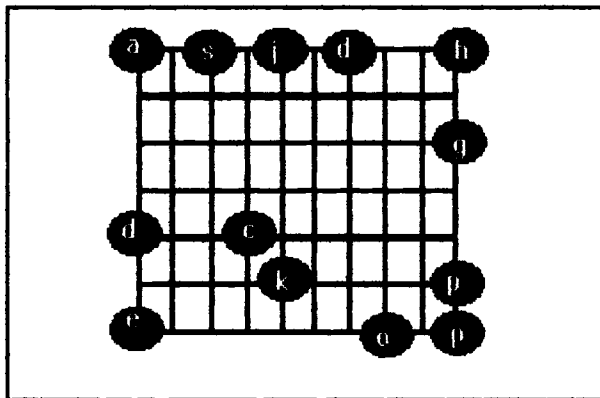
where η is the learning rate.

3. The learning rate and the neighborhood width are reduced proportionally with the length of the training elapsed thus far.
4. If the learning rate η reaches zero or a specified value then stop the learning process; otherwise, Go to step 1.

The LVQ algorithm is the same as the unsupervised algorithms except that the winning nodes are predetermined. As a result, the second step which determines the winning node will be eliminated. The LVQ learning algorithm is used when the classes of the input vectors are known in advance and a specific picture of which nodes will represent a specific class is desired. This algorithm is not used in

this paper, therefore, we will not go into detail in this subject.

At the end of training some nodes in the Kohonen Layer are labeled by one of the characters. A graphical visualization of the Kohonen layer can be displayed on the screen. A grid of the size used in the training is shown, and each node displays its label (the assigned character). Figure 10 shows an example of how the Kohonen layer will be graphically represented with the label of the nodes attached.



**Figure 10. Graphical Representation of Kohonen Layer
Nodes and the Labels**

3.6 Counterpropagation Network (CPN)

The idea of CPN is to combine the Kohonen layer with another layer employing Grossberg learning into one network (Figure 11) [Niel87].

The Grossberg's learning law equation is:

$$w_i^{new} = w_i^{old} + a [x_i y - w_i^{old}] U(x_i)$$

where $0 < a < 1$, and

$$U(s) = \begin{cases} 1 & \text{if } s > 0 \\ 0 & \text{otherwise} \end{cases}$$

x_i is the input vector, y_j is the desired output vector and w_i is the weight vector of processing element i .

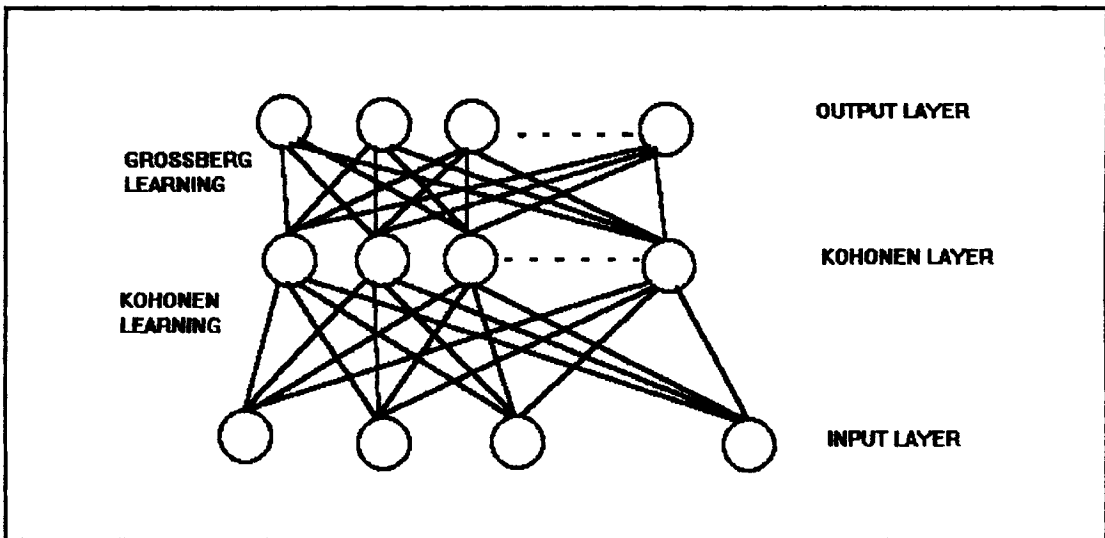


Figure 11. Counterpropagation Network [Niel87]

The network consists of three layers. An input layer (layer 1) containing n units that multiplex the input vector \mathbf{x} ($\mathbf{x} = x_1, x_2, \dots, x_n$) and m units that supply the "desired" output vector \mathbf{y} ($\mathbf{y} = y_1, y_2, \dots, y_m$). The second layer (layer 2) is a Kohonen layer that consists of N processing elements

that have the output signals vector z ($z = z_1, z_2, \dots, z_N$). The last layer (layer 3) has m processing elements that employ the Grossberg learning law. The network is trained by introducing it to a sequence of input-output pairs (x_n, y_n) . During training, the transfer function equation for layer 2 is:

$$z_i = \begin{cases} 1 & \text{if } i \text{ is the smallest integer for which} \\ & D(w_i^{\text{old}}, x) \leq D(w_j^{\text{old}}, x) \text{ for all } j \\ 0 & \text{Otherwise,} \end{cases}$$

where D is a distance metric. The winning node of layer 2 will adjust its weight using Kohonen learning laws (see section 5.2.2). After the completion of layer 2 computation, layer 3 receives the z signals from layer 2. Grossberg Laws are used to adjust the weights of all the processing elements of layer 3. To compute the actual output vector of the network the following equation is used:

$$y_i' = \sum_{j=1}^n w_{ij} z_j$$

The algorithm used to train the CPN works in two stages. The first stage will equilibrate the Kohonen layer with equiprobable weights. In the second stage, layer 3 learn to compute the average of the correct y vectors associated with each processing element weight vector w_i of layer 2. The algorithm used to train the CPN is as follows [Niel87]:

Step 1: Compute the Euclidean Distance for all the nodes in Kohonen layer using the equation:

$$I_i = \sum_{i=1}^n W_{ij} X_i$$

Where W_i the weight vector of node i in Kohonen layer, x is the input vector.

Step 2: The winning processing node has its output signal set z to 1, and the rest to 0.

step 3: The node with the smallest Euclidean distance I is allowed to change its weight using the equation:

$$W_{ij}^{new} - W_{ij}^{old} = \eta (X_i - W_{ij}^{old})$$

where w_i is the weight vector of the winning node i , x is the input vector, and η is the learning rate.

Step 4: Adjust the weight vectors of layer 3 nodes using the following equation (which include Grossberg learning law):

$$u_{ji}^{new} = u_{ji}^{old} + a [-u_{ji}^{old} + y_i] z_i$$

where u_i is the weight vector of the j^{th} processing element of layer 3, and a is the learning rate ($0 < a < 1$) of the Grossberg learning law.

Step 5: If more learning cycles are needed go to step 1; otherwise stop.

Several variations of the counterpropagation algorithm exist. Most of the proposed variants deal with the learning algorithm rather than the basic artificial neural architecture [Bord93]. The main difference between the traditional counterpropagation and the one proposed by Bord [Bord93] involves the strategy of assigning initial weights for the units. The alternative placement method for the weights is accomplished by setting the initial weight vector to each Kohonen neuron equal to the normalized average of the input vectors from each class [Bord93]. The performance of the network can exceed the performance of the traditional CPN. Improvement from a 77% successful classification rate for the random weights to around 97% for the variant method was achieved in Bord's research [Bord93].

3.7 Neural Networks as Character Recognizer

Hand-printed character recognition using neural networks is an area of great interest to researchers and businesses worldwide. In recent years, artificial neural networks have become increasingly popular for applications in character recognition and many papers have been published about this field. A multilayer neural network with a backpropagation learning algorithm is one of the most widely used and studied networks for character recognition. A few other neural network architectures have been suggested in recent years. Such as Self-Organizing Maps (SOM) and

counterpropagation neural networks [Bord93] [Guyo91] [Kner92].

Most of the neural network architectures used in character recognition can perform significantly well even with incomplete input data. As a result, neural networks can solve some problems encountered in character recognition applications. Shape discrimination is still a problem because a single handwritten character has infinite shapes and sizes depending on people's styles of writing (Refer to Figure 12). Another problem is the structure of the characters. Some characters may be broken, touching, or overlapping.

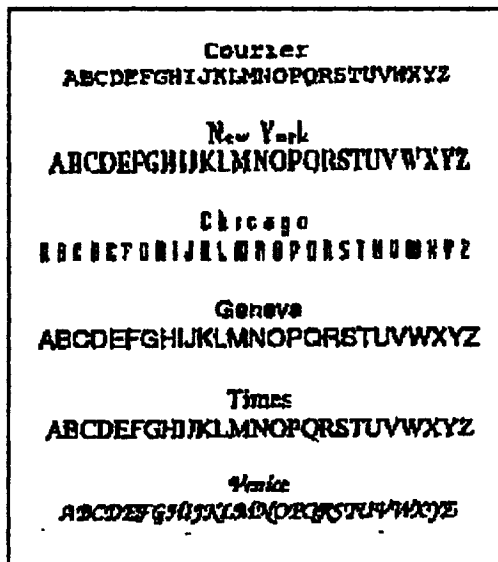


Figure 12. Fonts Sample [Bord93]

These problems in character recognition slow-down the research in developing OCR application using ANNs. Most research has concentrated on multilayer neural networks with

the backpropagation learning algorithm. It has been demonstrated that the backpropagation algorithm has an excellent capability for capturing generalization [Bord93]. Although results have been encouraging, the backpropagation algorithm has many disadvantages, including long training time, numerous parameters to adjust, and the lack of a graphical representation of the training results. Another problem is the choice of the number of nodes of the network's hidden layer. For example, if 10 hidden nodes can solve the problem, there is no guarantee that a network containing 20, 50, or 100 nodes will also solve it; chances are they will not, and the results will be worse [Hiot93].

Several neural networks have been proposed to create a character recognition application. New neural network architectures have been proposed by several researchers in the past few years [Bord93][Hert91][Koho89][Niel87]. However, few of these architectures were used in handwritten character recognition. The research presented here tests several neural network architectures using printed handwritten characters and compares their performances to the previously tested neural networks.

Chapter IV

NETWORK IMPLEMENTATION AND TECHNIQUES

The investigation process for handwritten character recognition is divided into three steps. The following is a brief introduction to each step.

4.1 Neural Network Architectures and Learning Algorithms

The main architectures that are investigated in this paper are the Self-Organizing Map in the form of Kohonen network (SOM), the Counterpropagation network (CPN), and the Backpropagation neural network (BPNN). The learning algorithms for the networks are the Kohonen Learning algorithm, the Supervised (Kohonen law) and the unsupervised learning algorithm (Grossberg), and the Backpropagation algorithm respectively.

4.2 Data representation

The handwritten characters used for the training and test sets are collected by an X-window function named *bitmap*. This function provide a grid of any size (for this research a 10x10 grid), the user then can draw the letter using a mouse (see Figure 13). The function stores the character as a Hexadecimal numbers. The dark pixels that form the letter are represented by ones and blank pixels by zeros. The artificial neural networks use the binary representation of the characters as the input data.

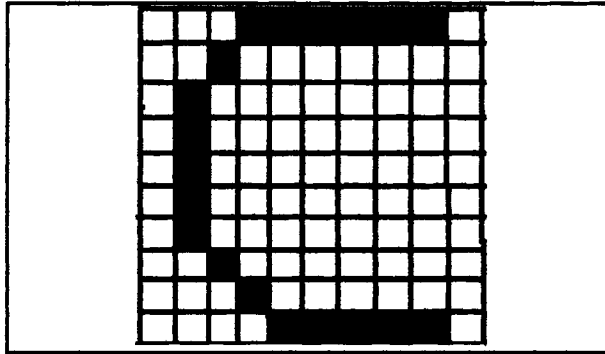


Figure 13. Handwritten Character Grid for the Letter C

The data is transformed in three ways:

1. As row data:- In this step the data is given to the neural networks without changing them in any form. In the case of the SOM network however, the data is normalized.
2. Shifting Data:- In this step each character is shifted so that the left most column and the top row contain at least one dark pixel. Figure 13 shows the letter C without shifting, then Figure 14 shows the letter after the shifting is performed.

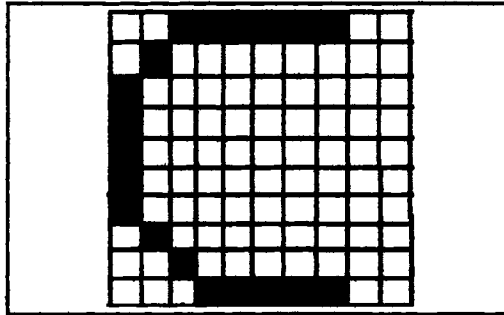


Figure 14. Character C after Data Shifting is Performed

3. Feature extraction is performed on each letter. Two feature extraction algorithms are used.

A. A BPNN is trained to extract 14 different features from the character (See Appendix D). The result of this network is used as input data only for the SOM network.

B. The crossing technique:- For an input vector of size 10x10, the crossing technique (described in section 2.2) will produce 19 input numbers. These numbers are used as the input for the three networks proposed in 4.1.

The training set consist of 120 data vectors. Test set 1 and test set 2 contain 64 and 120 input vectors respectively. The size of the input vectors vary depending on the data representation used. The normal and shifting data representation input vectors are of size 10x10. The crossing technique and the feature extraction

representations input vectors are of size 19 and 14 respectively. The number of training vectors is selected from the notion that most researches used such numbers in their papers.

4.3 Research Scope

All of the network architectures proposed (in section 4.1) are tested using the row data representations. The results obtained from each test are used as the basis of a performance comparison of the different architectures and data representations.

4.4 Hardware and Software Support

Data preprocessing and the training of the neural network architectures consume a large amount of CPU time. Training neural networks involves large amounts of floating point number manipulation. Therefore, a fast CPU with a math-coprocessor is recommended to perform the task in reasonable amount of time. The SOM network requires a graphic monitor to visualize the output of the training (see section 3.5).

The programs were implemented in Borland ANSI C language, which support graphics for the SOM network graphics. The programs run on an Intel 486DX-33MHZ computer with 5 MB of RAM.

4.5 Description of the Programs

To accomplish the proposed research, several programs were implemented for each artificial neural network

architecture. Each neural network program consists of three major parts. In addition, a few other minor programs had to be written. The three major files (programs) for each network are the header file, which contains all the functions and data structures required to train and test the neural network. The training program which contains the main functions of a program to train the network. The third program is the testing program which contains the main functions of a program to test the neural network.

The programs need various parameters; see Appendix B for the parameters of the programs. All the parameters that are required by any program are given as a command line options. The name of the option -- for example, the input layer dimension, specified as `-xdim` -- is followed by the value of that option. If one or more options are missing the program will terminate.

4.6 Other Programs

In addition to the standard three files, other programs were written to help with the evaluation of the networks.

4.6.1 Get_label

Some networks do not produce labels or letters as their output, such as Backpropagation and CPN. As a result, this program was written to read the `n` float numbers produced by the network and create a new file containing the matching label or letter. The two networks mentioned produce 26 floating numbers representing the 26 Alphabetical letters.

The highest value of the 26 numbers is considered the winner and its match label is printed. There are two reasons why this function was written separately. The first reason is modularity, the programs are written so each one perform only one task. The second reason is usage, some network's output is used as input for other networks. For example, the BPNN is trained to extract features from the input characters and produce output vectors of size 14 (See section 4.2).

4.6.2 Match Program

This program reads and matches labels of the input vectors and the labels of the network's output or the output of *get_label*. The program then matches the labels and computes the number of matches and the number of mismatches. It then displays the results on the monitor. The parameters required by this program are shown in Appendix B.

4.6.3 Randinit

This program is used by the SOM neural network program to create the weight file for the network. The program will produce numbers ranges between 0.0 and 1.0. The weight vector of each node in the Kohonen layers is equal to the a randomly selected normalized input vector. The parameters required by this program are listed in Appendix B.

4.6.4 Maninput

This program offers two options designated by the option *-tech*. The parameters *-tech* can have a value of 1 or

2. A value of 1 instructs the program to perform the shifting function on the input data vectors. A value of 2 instructs the program to perform a crossing technique on the input data vectors. The result is stored in another file. The name of the output file is set by the parameter -out (see Appendix B).

4.7 File Formats

All data files (input vectors, weight vectors, target vectors and output) are stored as ASCII files for their easy editing and checking. All the files, except output files, consist of specific size of floating-point numbers per line. Each line represents an entry. For example, the first line in the input files contains the 100 floating numbers that describe a written character. The only deviation from this format is required for the SOM network. In this case, each data line may have an optional qualifier that determine the usage of the data during training. The optional qualifier is the fixed point qualifiers (-fixed) which is used by the LVQ learning algorithm. The fixed point can be used to force the network to pick the specified node (Fixed=2,5 represents $x = 2$ and $y = 5$) as the winning node instead of the best matching node.

4.8 Stages Required in Evaluation of A Network

Each neural network architecture goes through at least three stages during its testing. Some networks will add one

or more stages in order to accommodate the requirements of its input vector representation.

4.8.1 First Stage - Weight Initialization

The weight vectors of the network are first initialized to random values. The SOM network uses the `Randinit` program. The random number generator used by the SOM can generate random numbers for one layer only (Kohonen Layer). Therefore, the CPN and Backpropagation networks contain their own random number generators in the header file.

4.8.2 Second Stage - Network Training

In this stage, the network is trained on the training set data. No evaluation of the network's performance in recognizing the input data is performed at this stage.

4.8.3 Third Stage - Network Testing

At this stage the network is tested using both the training data set and a new set called the test data set. The accuracy of the network in recognizing the input vectors are evaluated using the `match` program (see section 4.4.2).

Chapter V

NETWORKS PERFORMANCE ANALYSIS

In chapter III, the performance of three artificial neural network architectures was examined. Each architecture was tested using three data representations of the same data sets (training set and test set). The next sections will discuss the performance of each network architecture for the three data representations. Note that the tables presented in the next sections represent the highest performance of a network with the least amount of training cycles. Furthermore, the time required to perform one training cycles is the same for the SOM network and the BPNN. The CPN network required about twice the amount of time.

5.1 Row (Normal Data Representation)

Normal (row) data, as explained in section 3.3, is the 100 float-point numbers (1.0 or 0.0) where 1.0 represents a dark pixel (pixels forming the character) and 0.0 represents white pixels. The grid used to write each letter is 10x10 in size. The three architectures were trained and tested on the same data sets; Table 1 represents the best performance of the three proposed networks. It also includes the configuration of the network, the number of epochs needed, and several other variables needed by each network.

TABLE 1
NETWORKS PERFORMANCE ON NORMAL DATA REPRESENTATION

	Train Set	Test Set 1	Test Set 2	Ca	Alpha	Radius	Epochs	Config
SOM	100.0	67.5	77.8	N/A	0.85	3	4700	900
BPNN	100.0	72.5	79.7	N/A	0.90	N/A	16000	20
CPN	82.3	45.0	73.4	0.45	0.90	N/A	36000	90
BPNN & SOM	100.0	59.2	75.0	N/A	0.90	3	6700	900

Table 1, shows that all the networks with the exception of CPN, performed significantly well on the training data set. The SOM network required the least amount of training cycles to achieve this level of performance (67.5% and 55.8% for Test set 1 and Test set 2 respectively). On the other hand, it required a large size (900 nodes) in the Kohonen layer to accommodate for all the input vectors. The BPNN achieved a higher performance over the test sets (72.5% and 79.7% for Test set 1 and Test set 2 respectively), but required 16000 cycles to reach this performance. Moreover, this performance was reached with only 100, 20, and 26 nodes for the input, hidden, and output layer respectively, compared to the 1000 nodes (100 for input layer and 900 for Kohonen layer) for the SOM network. Figure 15 shows the performance of all three networks as a function of training epochs or cycles over test set 1.

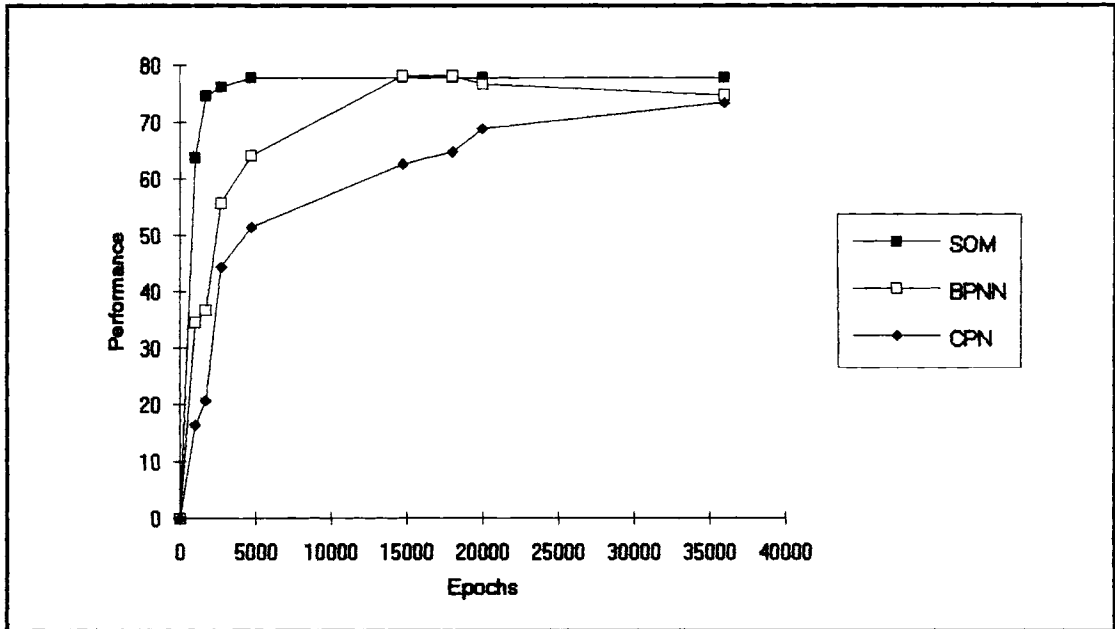


Figure 15. Performance vs. Epochs for Normal Data

5.2 Data Shifting

The data shifting is explained in section 4.3. The results obtained with this type of data representation are listed in Table 2. The BPNN architecture performance of 75.0% and 85.9% for test set 1 and test set 2 respectively, was the best for this data representation. However, the number of epochs required to train the network exceeded that of the SOM architecture, the performance of which was almost as good as that of BPNN.

TABLE 2
NETWORKS PERFORMANCE ON SHIFTING DATA REPRESENTATION

	Train Set	Test Set 1	Test Set 2	Ca	Alpha	Radius	Epochs	Config
SOM	100.0	75.0	82.5	N/A	0.80	3	3700	900
BPNN	100.0	75.0	85.9	N/A	0.90	N/A	10000	20
CPN	80.1	54.7	66.7	0.45	0.90	N/A	36000	90
BPNN & SOM	100.0	63.3	67.2	N/A	0.90	3	7000	900

The SOM network required 900 nodes for the Kohonen layer compared to BPNN with only 20 nodes in the hidden layer. The other two networks failed to reach the performance of the SOM and BPNN networks. Figure 16 represent the performance of the three networks on test set 1 over the learning cycles or epochs.

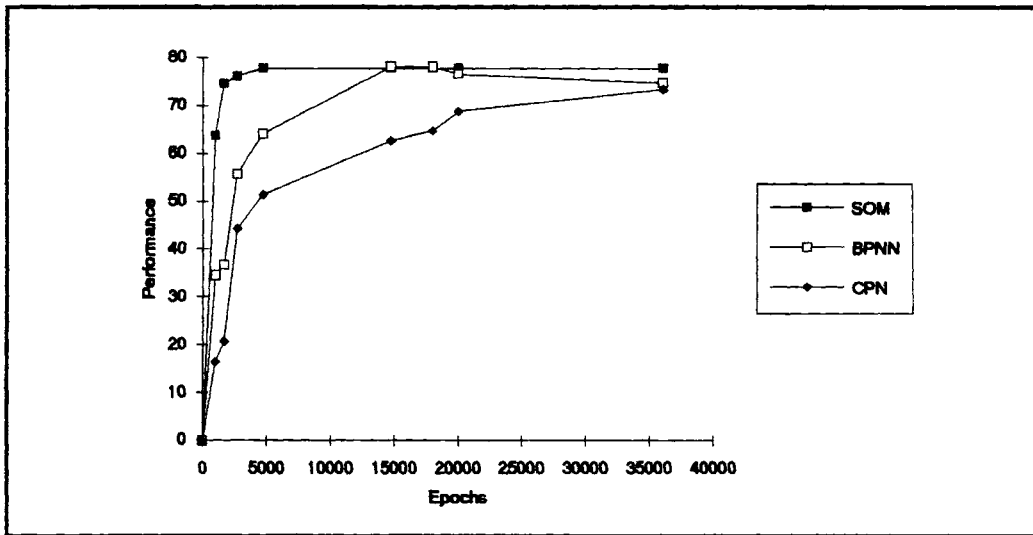


Figure 16. Performance vs. Epochs for Shifting Data

5.3 Crossing Technique

Crossing technique is explained in section 4.3. The results of such a representation are listed in Table 3.

**TABLE 3
NETWORKS PERFORMANCE ON CROSSING TECHNIQUE DATA
REPRESENTATION**

	Train Set	Test Set 1	Test Set 2	Ca	Alpha	Radius	Epochs	Config
SOM	100.0	54.2	52.4	N/A	0.80	3	4700	529
BPNN	100.0	53.3	52.4	N/A	0.65	N/A	54000	40
CPN	57.6	35.9	43.7	0.75	0.90	N/A	46000	90

Table 3 shows that the performance of the three architectures over the test sets did not reach that of the row data or data shifting representations. The reason behind such a decline in performance is due to the fact that no scaling of the input characters was performed. As a result, the Crossing technique failed to represent all the characters of the same class accurately. This problem did not effect the research progress since the goal of this research is to compare the performance of the networks over the same data sets, not the minute differences between the individual data sets.

5.4 SOM Architecture Analysis

Studying Tables 4, 5, and 6 (See Appendix C) a few conclusions can be reached.

5.4.1 Radius Size

The network performed the best with the neighborhood

radius set to 3 (see Figure 17). This is true for all three data representations used. The reason behind such an improvement in performance is that more than one node in the

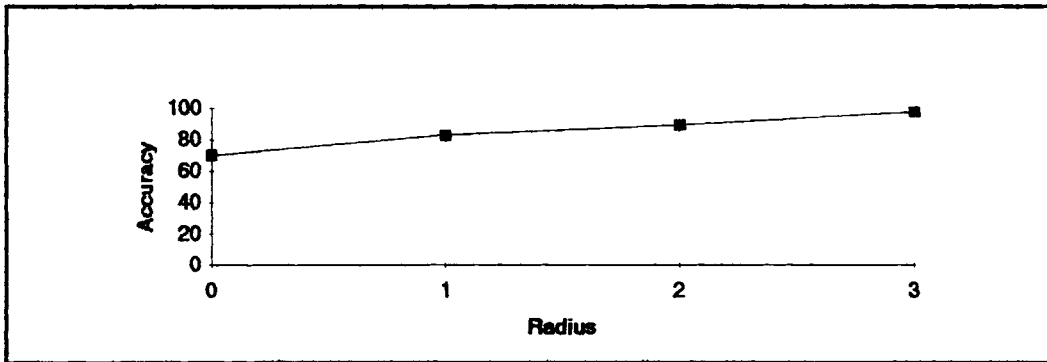


Figure 17. Radius of Neighborhood vs. Accuracy

Kohonen layer gets its weights adjusted (see Neighborhood Kernel in section 3.2). Therefore, more variations of the same letter can be represented in (or "remembered by") the Kohonen layer.

5.4.2 Kohonen Layer Size

Trying different sizes for the Kohonen layer lead to the conclusion that 30x30 or 900 nodes will lead to the best results of the network. Increasing the size of the Kohonen layer beyond this size does not improve the performance.

5.6.3 Alpha Variable

Changing the initial value of the learning rate (alpha variable) did not change the performance of the network significantly. However, most data representations (see

Table 4 and 5 in Appendix C) required a value of 0.8 or higher. On the other hand, shifting data representation required a value of 0.85. Since a mathematical formula for the value of alpha does not exist, there is no scientific explanation for such a difference.

The SOM network performance over test set 1 for all data representations is illustrated in Figure 18. It can be seen that normal data performed best.

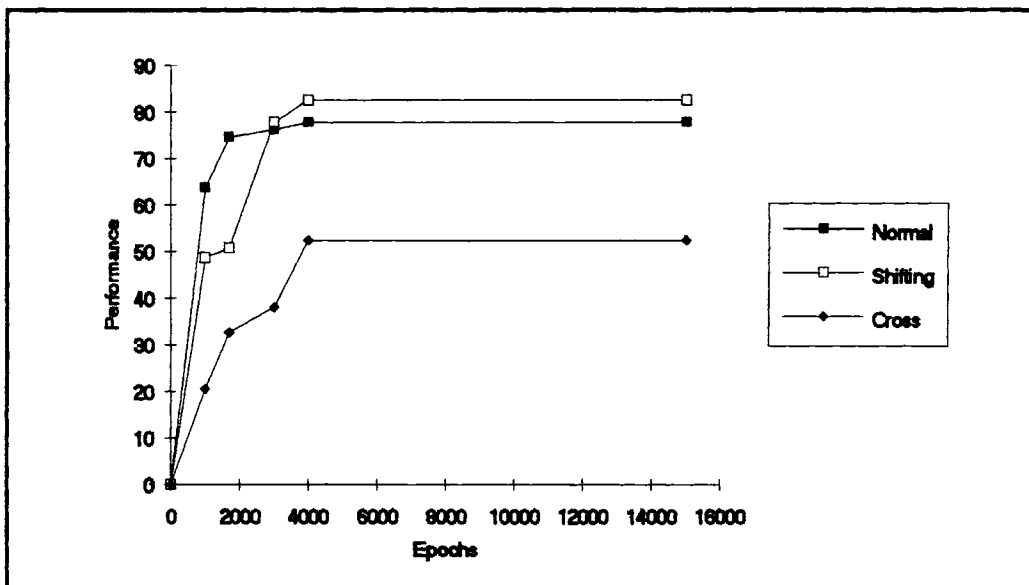


Figure 18. SOM Performance vs. Epochs

5.5 Backpropagation Architecture Analysis

The analysis of the BPNN can be divided into three parts.

5.5.1 Hidden Layer Size

Experimenting with different sizes of the hidden layer showed that a hidden layer with 20 nodes plus the threshold node configuration yielded a better performance than any other configuration (See Tables 7, 8 and 9 in Appendix C). The performance of the network ranged from 52.4% for the crossing technique (See Table 9) to 85.9% (See Table 8) for the test data set.

5.5.2 Alpha Variable

The value of alpha (learning rate) required by the network ranged from 0.65% for crossing technique data representations (See Table 9) to 0.90 for the two other data representation (See Tables 7 and 8). These values were obtained by training the network with different values of alpha.

5.5.3 Network Overtraining

One disadvantage of the BPNN architecture is overtraining which was explained in section 3.4. Studying figure 19 we can see that this problem was encountered in this research. Figure 19 shows that the network performance on the test data set starts to decrease after we exceed 12000 epochs.

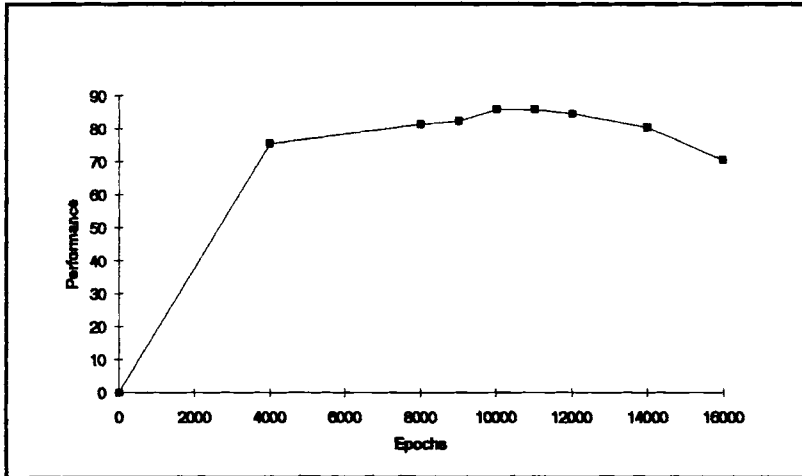


Figure 19. Backpropagation Network Overtraining

5.5.4 Network Convergence Rate

Figure 20 shows the performance of the network on test set 1 for all data representation over the number of training cycles.

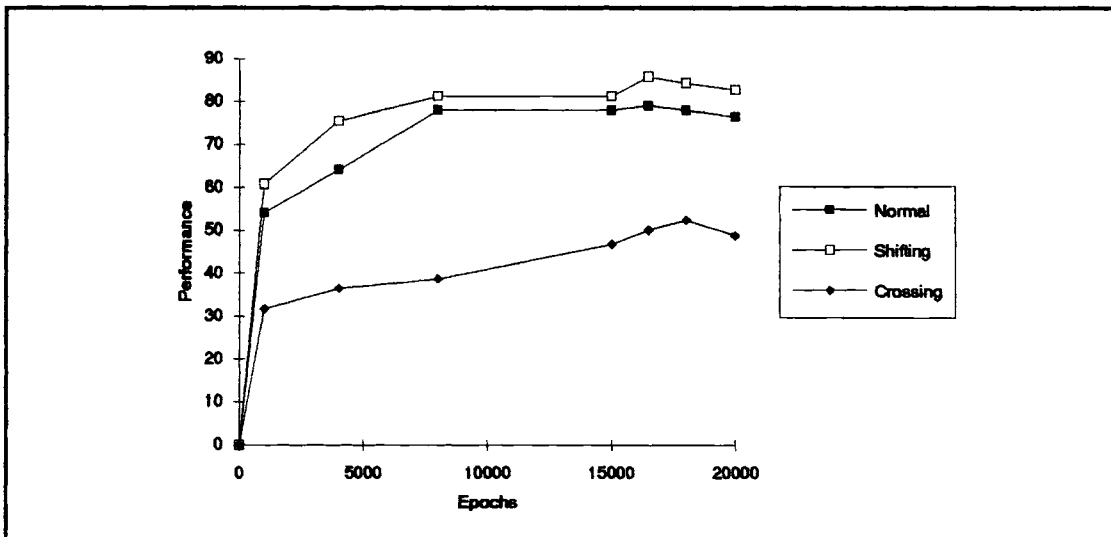


Figure 20. BPNN Performance vs. Epochs

5.6 Counterpropagation Architecture Analysis

The CPN network did not perform as expected. Table 4 shows that the network did not reach a 100.0% performance on the training set. The highest performance was 82.5%. Moreover, the CPN performance on the test data set ranged from 57.6% to 73.4%. The reason why the network failed to produce higher performance is that the network used a radius of 0 for the weight update algorithm. As a result, fewer variations of the letters were represented in the network, and the network failed to identify all the letters in the training data set. No attempt was made to include the radius parameter in the learning algorithm since the proposed CPN architecture with radius of zero is the most widely used architecture.

TABLE 4
CPN PERFORMANCE

Train Set	Test Set 1	Test Set 2	Ca	Alpha	Epochs	Config	Data Type
82.3	45.0	73.4	0.45	0.90	36000	90	Norm
80.1	41.7	66.7	0.75	0.90	46000	90	Shift
67.6	35.9	43.7	0.75	0.90	50000	90	Cross

5.7 Feature Extraction Using BPNN

A BPNN with 100-40-14 configuration used to extract 14 features from each letter and the output of the network was used as the input data set for the SOM network. The BPNN required 20000 epochs to yield an acceptable performance. The SOM network recognized 100.0% of the train data set.

However, it recognized 63.3% and 67.2% of the test data set 1 and test data set 2 respectively (See table 5). The performance difference can be traced to the problems of

**TABLE 5
PERFORMANCE FOR FEATURE EXTRACTION**

Train Set	Test Set 1	Test Set 2	Alpha	Radius	Epochs	Data Type
100.0	59.2	75.0	0.9	3	6700	Normal
100.0	63.3	67.2	0.9	3	7000	Shift

character recognition (See Chapter 1). Since the input data vectors varied in size and no scaling algorithms were used. The extraction network failed to capture all 14 features needed to recognize each character.

Chapter VI

CONCLUSION AND FUTURE RESEARCH

Neural network architectures are an approach to information processing that do not require specific algorithms or rule development to solve different tasks. Each architecture has its own unique mix of information processing capabilities, domains of applicability, techniques for use, required training data, training regimen, and so on [Niel90]. The networks employ different learning algorithms such as the Backpropagation algorithm, and supervised and unsupervised learning algorithms. This research focuses on a small subset of the networks and the data representations available in the research field. Modifications or additions to the proposed research could increase the performance of the networks and/or reduce the amount of time required to train such networks.

6.1 Research Findings

Let's define time complexity as the time required to train an ANN, and space complexity as the amount of storage required by an ANN (e.g., memory). In many situations it is desirable to have an ANN architecture that learn efficiently (time complexity) with minimum memory requirements (space complexity).

Of the three architectures tested in this research, none met both requirements at the same time. However, the

three architectures surpassed each other in one of the requirements.

The SOM architecture was the best architecture with regard to time complexity. It required on average 4000 epochs to reach its peak performance compared to 10000 and 30000 for both BPNN and CPN architecture, respectively.

When dealing with space complexity, the BPNN architecture surpassed the other two architectures with only 100-20-26 configuration. Compare that to 100-900 for the SOM architecture and 100-90-26 for the CPN architecture.

This research demonstrated that the ANN architectures (specifically SOM and BPNN) are suitable solutions to the problem of OCR. Note that the ANN architectures identified on average 80.0% of the normal and shifting data representations without the benefit of preprocessing the data (e.g., scaling, thinning, or noise elimination). It is possible to increase the performance by preprocessing the input data prior to training the network with the data.

6.2 Future Research

There are several modification that may be implemented that might improve the performance of the ANN architectures tested in this research. The next sections will explore some of these modifications in detail. Such modifications might involve the basic architecture of the network or the algorithms used to train the architecture.

6.2.1 Backpropagation Research

The BPNN used in this research contains one hidden layer only. The performance of the network might be improved by adding one or more of the following. First, Two or more hidden layers can be used instead of only one layer. Second, an improved learning algorithms for the training of the network can improve the network. Several learning algorithms for the BPNN can be found in the literatures that can lead to a better performance than previous algorithms.

6.2.2 Counterpropagation Network

The CPN used in this research employed one winner node in the Kohonen layer. Some researches suggested using more than one winner node in the Kohonen layer [Hech90]. Since the SOM network performed significantly better when the radius of the updated node increased from 0 to 3, taking advantage of the neighborhood update algorithm (see section 2.2.1) could improve the bad performance of the CPN architecture.

6.2.3 Self-Organizing Map

The neighborhood kernel used in this research is the bubble function (see section 3.5). However, another neighborhood kernel is the Gaussian function (see section 3.5). This function may improve the performance of the SOM network.

References

- [Bord93] Darren Bordeaux, Antonette Logar. *An Improved Counterpropagation Algorithm For Multi-Font Character Recognition*. AI Expert 1993.
- [Caud93] Maureen Caudill. *A little Knowledge is a Dangerous Thing*. AI Expert, pp 16-22, June 1993.
- [Cesa90] Michel Cesar. *An Algorithm for segmenting handwritten postal codes*. Man-Machine Studies. pp 63-80, 1990.
- [Chou91] S. Chou, W. Tsai. *Recognizing handwritten Chinese characters by stroke-segment matching using an iteration scheme*. International Journal of Pattern Recognition and Artificial Intelligence Vol. 5 No. 1 & 2 (1991) 175-197.
- [Down91] A. Downton, W. Tregidgo. *Recognition and verification of handwritten and hand-printed British Postal Address*. International Journal of Pattern Recognition and Artificial Intelligence. Vol. 5 No 1&2 (1991) 256-291.
- [Govi90] V. K. Govindan, A. P. Shavaprasad. *Character Recognition - A Review*. Pattern Recognition. Vol 23, No. 7, pp 671-683, 1990.
- [Guyo91] I. Guyon. *Application of neural networks to character recognition*. International Journal of Pattern Recognition and Artificial Intelligence. Vol. 5 No. 1 & 2 (1992) 353-382.
- [Hert91] J. Hertz, A. Krogh, R. Palmer. *Introduction to the theory of neural network computation*. Addison-Wesley (1991).
- [Hiot93] Andre Hiotis. *Inside a Self-Organizing Map*. AI Expert , pp 38-43, April 1993.
- [Impe91] S. Impedovo, L. Ottaviano, S. Occhinegro. *Optical Character Recognition - A survey*. International Journal of Pattern Recognition and Artificial Intelligence. Vol 5 No. 1 & 2 (1991) 1-24.
- [Kara93] Nicolaos Karayiannis, Anastasios Venetsanopoulos. *Artificial neural networks learning algorithms, performance evaluation, and application*. Kluwer Academic Publishers (1993).

- [Kner92] S. Knerr, L. Personnaz, G. Dreyfus. *Handwritten digit recognition by neural networks with single-layer training*. IEEE Transaction On Neural networks, Vol.3, No. 6 Nov 1992.
- [Koho89] Teuvo Kohonen. *Self-Organizing and associative memory*. Springer-Verlag, Berlin-Heidelberg-New York-Tokio, 3 edition, 1989.
- [Koho92] Teuvo Kohonen, J Kangas, J. Laaksonen. *SOM-PAK The self Organizing Map program Package*. Helsinki University of Technology, Finland, Nov. 2, 1992.
- [Mehr91] Kishan Mehrotra, Chilukuri Mohan, Sanjay Ranka. *Bounds on the Number of Samples Needed for Neural Learning*. IEEE Transaction Neural Networks, Vol 2, NO. 6, November 1991.
- [Niel87] Robert Hecht-Nielsen. *Counterpropagation networks*. Applied Optics, Vol. 26 (1987) 4979-4984.
- [Niel90] Robert Hect-Nielsen. *Neurocomputing*. Addison-Wesley (1990).
- [Paoy89] Yoh-Han Pao. *Adaptive Pattern Recognition and Neural Networks*. Addison-Wesley 1989.
- [Rich91] Elaine Rich, K. Knight. *Artificial Intelligence*. McGraw-Hill 1991.
- [Teno90] Manoel Tenorio. *Self-Organizing Network for Optimum Supervised Learning*. IEEE transaction on neural networks, Vol. 1 No. 1, March (1990).
- [Upda92] S. Upda, H. Al-Yousefi. *Recognition of Arabic Characters*. IEEE transaction on pattern analysis and machine intelligence, Vol. 14, No. 8, August (1992).
- [Zeid91] Matthew Zeidenberg. *Neural Network Models in Artificial Intelligence*, Ellis Horwood 1991.

Appendix A

Sample Of Batch Files

This is an example of using the program parameters to run various programs. The following code was used in a batch file to run the test for the SOM network.

REM Normal Data

** Call Randinit Program

```
randinit -vdim 100 -xdim 30 -ydim 30 -code cmap.dat -input  
train2.dat
```

** Call train program

```
train -vdim 100 -xdim 30 -ydim 30 -code cmap.dat -input  
train2.dat -len 3700 -alpha 0.85 -map cmap.dat -fixed 0 -rd  
3
```

** Call Test Program

```
test -vdim 100 -xdim 30 -ydim 30 -code cmap.dat -input  
train2.dat -label clabel.dat  
test -vdim 100 -xdim 30 -ydim 30 -code cmap.dat -input  
test2.dat -map cmap.dat -label clabel2.dat
```

** Call Match Program (Evaluate Performance for Training Set)

```
match -len 100 -in1 train2.dat -in2 clabel.dat
```

** Call Match Program (Evaluate Performance for Test Set)

```
match -len 100 -in1 test2.dat -in2 clabel2.dat
```

REM Test for shifting

** Call maninput (-tech1 Perform Data Shifting)

```
maninput -xdim 10 -ydim 10 -out cross.dat -input train2.dat  
-tech 1
```

** Call maninput (-tech 2 to perform Crossing Technique on Test Set)

```
maninput -xdim 10 -ydim 10 -out cross2.dat -input test2.dat  
-tech 1
```



```
randinit -vdim 100 -xdim 30 -ydim 30 -code cweight.dat
-input cross.dat
train -vdim 100 -xdim 30 -ydim 30 -code cweight.dat -input
cross.dat -len 23700 -alpha 0.8 -map cmap.dat -fixed 0 -rd 3
test -vdim 100 -xdim 30 -ydim 30 -code cmap.dat -input
cross.dat -label clabel.dat
test -vdim 100 -xdim 30 -ydim 30 -code cmap.dat -input
cross2.dat -map cmap.dat -label clabel2.dat
match -len 100 -in1 train2.dat -in2 clabel.dat
match -len 100 -in1 test2.dat -in2 clabel2.dat
```

REM For crossing technique

** Call maninput (-tech 2 to perform Crossing Technique on
Traning Set)

```
maninput -xdim 10 -ydim 10 -out cross.dat -input train2.dat
-tech 2
```

** Call maninput (-tech 2 to perform Crossing Technique on
Test Set)

```
maninput -xdim 10 -ydim 10 -out cross2.dat -input test2.dat
-tech 2
randinit -vdim 19 -xdim 30 -ydim 30 -code cmap.dat -input
cross.dat
train -vdim 19 -xdim 30 -ydim 30 -code cmap.dat -input
cross.dat -len 8000 -alpha 0.90 -map cmap.dat -fixed 0 -rd 3
test -vdim 19 -xdim 30 -ydim 30 -code cmap.dat -input
cross.dat -label clabel.dat
test -vdim 19 -xdim 30 -ydim 30 -code cmap.dat -input
cross2.dat -map cmap.dat -label clabel2.dat
match -len 100 -in1 train2.dat -in2 clabel.dat
match -len 100 -in1 test2.dat -in2 clabel2.dat
```

Appendix B

Networks Parameters

Counterpropagation Network Parameters

The CPN training and the testing programs require the following parameters:

- xdim Number of nodes in input layer.
- ydim Number of nodes in Kohonen layer.
- zdim Number of nodes in output layer.
- len Running length in training.
- alp Initial learning rate parameter for the supervised learning.
- code Name of weights (code) file.
- inp Name of input file (input vectors).
- out Name of output file.
- targ Name of the target file (Vector Y for output layer).
- ca Initial learning rate parameter for the unsupervised learning.

Self-Organizing Map Parameters

The SOM training and the testing programs require the following parameters:

- xdim Number of nodes in input layer.
- ydim Number of nodes in Kohonen layer.
- zdim Number of nodes in output layer.
- len Running length in training.
- alp Initial learning rate parameter for the supervised learning.
- code Name of weights (code) file.
- inp Name of input file (input vectors).
- out Name of output file.
- targ Name of the target file (Vector Y for output layer).

Backpropagation Network Parameters

The Backpropagation training and the testing programs require the following parameters:

- xdim Number of nodes in input layer.
- ydim Number of nodes in Kohonen layer.
- zdim Number of nodes in output layer.
- len Running length in training.

-alp Initial learning rate parameter for the supervised learning.
-code Name of weights (code) file.
-inp Name of input file (input vectors).
-out Name of output file.
-targ Name of the target file (Vector Y for output layer).

The Match program require the following parameters:

-xdim Size of the input vector.
-in1 Name of the input file (input vectors).
-in2 Name of the file produced by the network or get_label.

The Randinit program require the following parameters:

-xdim Number of units in the x-direction.
-ydim Number of units in the y-direction.
-din Name of input vector file.
-cout Name of the output file (weigh file).

The maninput program require the following parameters:

-xdim Number of units in x-direction
-ydim Number of units in y-direction
-out Name of output file
-input Name of input file
-tech technique reuired (1:shifting, 2:Crossing)

Appendix C

Performance Progress Tables for the ANNs

**TABLE 6
CROSSING TECHNIQUE PROGRESS FOR SOM NETWORK**

Train Sset	Test Set 1	Test Set 2	Alpha	Radius	Epochs	Kohonen Size
81.5	48.3	20.6	0.8	3	1000	529
91.9	50.8	32.7	0.8	3	2000	529
94.3	53.1	38.1	0.8	3	3000	529
100.0	54.2	52.4	0.8	3	4700	529
100.0	54.2	52.5	0.8	3	10000	529

**TABLE 7
NORMAL DATA PROGRESS FOR SOM NETWORK**

Train Set	Test Set 1	Test Set 2	Alpha	Radius	Epochs	Kohonen Size
81.5	58.3	63.7	0.85	3	1000	900
86.5	63.7	74.6	0.85	3	1700	900
100.0	65.8	76.2	0.85	3	2700	900
100.0	67.5	77.8	0.85	3	4700	900
100.0	67.5	77.8	0.85	3	14700	900

**TABLE 8
SHIFTING DATA PROGRESS FOR SOM NETWORK**

Train Set	Test Set 1	Test Set 2	Alpha	Radius	Epochs	Kohonen Size
82.9	70.3	50.8	0.85	3	1700	900
97.6	74.1	77.8	0.85	3	2700	900
100.0	75.0	82.5	0.85	3	3700	900
100.0	75.0	82.5	0.85	3	13700	900

**TABLE 9
NORMAL DATA PROGRESS FOR BPNN**

Train Set	Test Set 1	Test Set 2	Alpha	Epochs	Hidden Size
96.7	53.3	64.1	0.90	4000	20
100.0	65.8	78.1	0.90	8000	20
100.0	72.5	78.1	0.90	15000	20
100.0	72.5	79.7	0.90	16500	20
100.0	71.7	78.1	0.90	18000	20
100.0	70.3	76.6	0.90	19500	20

**TABLE 10
SHIFTING DATA PROGRESS FOR BPNN**

Train Set	Test Set 1	Test Set 2	Alpha	Epochs	Kohonen Size
96.8	66.7	75.5	0.90	4000	20
99.2	75.0	81.3	0.90	8000	20
100.0	74.2	81.3	0.90	9000	20
100.0	73.3	85.9	0.90	10000	20
100.0	73.3	85.9	0.90	11000	20
100.0	71.2	84.4	0.90	12000	20

**TABLE 11
CROSSING TECHNIQUE PROGRESS FOR BPNN**

Train Set	Test Set 1	Test Set 2	Alpha	Epochs	Hidden Size
61.3	33.3	36.5	0.65	4000	40
94.3	53.3	52.4	0.65	27000	40
100.0	52.5	50.0	0.65	30000	40
100.0	50.8	48.4	0.65	54000	40
100.0	49.7	46.9	0.65	60000	40

Appendix D

Feature Extraction List

The features extracted from the characters using a BPNN.

1. | Left Vertical Line
2. | Right Vertical Line
3. / Diagonal Right Line
4. \ Diagonal Left Line
5. - Middle Horizontal Line
6. - Top Horizontal Line
7. - Bottom Horizontal Line
8. O Circle
9. C C Shape
10. ⊃ Inverted C Shape
11. ^ Upper Diagonal Line
12. ` Lower Diagonal Line
13. ° Top Circle
14. V Upper v Shape

VITA

Majed Naser

Candidate for the Degree of

Master of Science

Thesis: PERFORMANCE COMPARISON OF NEURAL NETWORK ARCHITECTURES FOR HANDPRINTED CHARACTER RECOGNITION

Major Field: Computer Science

Biographical:

Education: Graduated from Abu Dhabi High School, Abu Dhabi, United Arab Emirates; received Bachelor of Science degree in Computer Information System from Oklahoma State University, Stillwater, Oklahoma in December 1991. Completed the requirements for the Master of Science degree with a major in Computer Information Systems at Oklahoma State University in May 1995.

Experience: Employed by Oklahoma State University, Wellness Center as a graduate research assistant; Oklahoma State University, wellness Center, 1993 to 1994.