NEURAL NETWORK STRUCTURE MODELING: AN APPLICATION TO FONT RECOGNITION

by

MING-CHIH YEH LEE, B.A., M.S.

A THESIS

IN

COMPUTER SCIENCE

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements for
the Degree of

MASTER OF SCIENCE

Approved

Accepted

December, 1988

T3
1988

ACKNOWLEDGMENTS

To accomplish this task, I am deeply indebted to Dr. Oldham, the chairman of my committee. His expertise, encouragement, guidance, and patience have supported me throughout the process and will never be forgotten.

My sincere gratitude is also expressed to Dr. Marcy and Dr. Gustafson for their support. Special thanks go to Mrs. Weiner for her help in the various phases of this research as well as in my academic years in the Computer Science Department.

Finally, I would like to thank my parents and my husband, Yung-Hui, for their support throughout the study.

CONTENTS

ACKNOWLEDGMENTS ii

ABSTRACT v

LIST OF TABLES vi

LIST OF FIGURES vii

CHAPTER

I. INTRODUCTION 1

Scope 1
Problem Definition 3
Specific Tasks 5

II. BACKGROUND INFORMATION AND LITERATURE REVIEW 7

Neural Network Model 7
ANS Description 7
Features of ANS 8
Mathematical Model of ANS 10
Applications of ANS 11
Various Neural Network Models 14
Hopfield Models 14
Carpenter and Grossberg Models 17
Single Layer Perceptron 18
Anderson Model 19
Hogg and Huberman Models 20
Summary 27

III. METHOD AND APPROACH 28

Problem Formulation 28
Software Development 29
Encoding Scheme Development 31
Normalizing Fonts and Generating Character Matrices 31
Selecting Character Properties and Generating Property Matrices 32
Computing Filter Matrix 35
Extracting Properties and Construct Character Vectors 36

Application of Models Through Simulation 42
Model Comparisons 47

IV. SIMULATION RESULTS 50

Characteristics of Collected Characters 50
Principal Component Analysis 51
Behavior of Model H-H1 56
Behavior of Model H-H2 71
Model Comparisons 86
Model H-H1 versus Model H-H2 86
Hogg and Huberman versus Cash and Hatamian 92
Hogg and Huberman versus Fujii and Morita 93

V. CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE RESEARCH 95

Conclusions 95
Recommendations for Future Research 98

BIBLIOGRAPHY 100

APPENDICES

A MODEL H-H1 102

B MODEL H-H2 113

ABSTRACT

Two neural network models, Model H-H1 (Hogg and Huberman, 1984) and Model H-H2 (Hogg and Huberman, 1985) have been successfully applied to the font recognition problem and were used to recognize 26 English capital letters, each with six font representations. Recognition rate, memory space requirement, learning speed, and recognition speed were used to measure the models' performances. Model parameters such as memory array size, Smin_Smax, and Mmin_Mmax were varied to elucidate the models' behavior.

As a result, both models achieved a 100% recognition rate when all six fonts were used as the training as well as the recognition set. When three out of six fonts were used for training, Model H-H1 achieved a maximum recognition rate of 87.82% and Model H-H2 achieved a maximum recognition rate of 89.10%. This shows that the basins of attractor states existed for the letters in most of the various font presentations. Model H-H2 significantly outperformed Model H-H1 in terms of recognition rate, use of memory space, and learning speed when all six fonts were used as the training set. This was supported by the results of the Pairwised T Test.

LIST OF TABLES

1. Assignment of Weighting Factor 41

2. Levels of Parameters 48

3. Principal Component Analysis of the Character Vectors 53

4. Performance Comparison of 1x9 and 1x14 Input Code 55

5. An Example Output (Model H-H1) 58

6. Model Performance (Model H-H1; Trained 3 Fonts) 60

7. Analysis of Variance of Performance Data of Model H-H1 62

8. Model Performance (Model H-H1; Trained 6 Fonts) 70

9. An Example Output (Model H-H2) 73

10. Model Performance (Model H-H2; Trained 3 Fonts) 75

11. Analysis of Variance of Performance Data of Model H-H2 77

12. Model Performance (Model H-H2; Trained 6 Fonts) 85

13. Performance Differences of Model H-H1 and Model H-H2 (Trained 3 fonts) 87

14. Pairwised T Test of Performance of Model H-H1 and Model H-H2 (Trained 3 fonts) 89

15. Performance Differences of Model H-H1 and Model H-H2 (Trained 6 fonts) 90

16. Pairwised T Test of Performance of Model H-H1 and Model H-H2 (Trained 6 fonts) 91

LIST OF FIGURES

1. The Architecture of a Typical ANS 9

2. The Non-linearity Functions 12

3. Model Classifications 15

4. The Hogg and Huberman Model 22

5. Basins of Attractor Change After the Coalescence Process 25

6. The 156 Target Alphabet 30

7. The Courier 'A' Font and Its Matrix Representation 33

8. The Selected Character Properties and Their Matrix Representations 34

9. The Search Matrix and the Windows 38

10. Location Assignment in the Character Matrix 40

11. An Example of the Decision Tree 44

12. Frequency Occurrence of the Extracted Properties 52

13. Input and Output Relationships of Model H-H1 57

14. Recognition Rate as a Function of Memory Matrix Size, Smin_Smax, and Mmin_Mmax (Model H-H1) 64

15. Memory Space as a Function of Memory Matrix Size, Smin_Smax, and Mmin_Mmax (Model H-H1) 66

16. Learning Speed as a Function of Memory Matrix Size, Smin_Smax, and Mmin_Mmax (Model H-H1) 67

17. Recognition Speed as a Function of Memory Matrix Size, Smin_Smax, and Mmin_Mmax (Model H-H1) 69

18. Input and Output Relationships of Model H-H2 72

19. Recognition Rate as a Function of Memory Matrix Size, Smin_Smax, and Mmin_Mmax (Model H-H2) 79

20. Memory Space as a Function of Memory Matrix Size, Smin_Smax, and Mmin_Mmax (Model H-H2) 81

21. Learning Speed as a Function of Memory Matrix Size, Smin_Smax, and Mmin_Mmax (Model H-H2) 82

22. Recognition Speed as a Function of Memory Matrix Size, Smin_Smax, and Mmin_Mmax (Model H-H2) 84

CHAPTER I

INTRODUCTION

Scope

Artificial Neural Net systems (ANS) is a rapidly growing research area because of its promise to solve problems that have confounded computer science and artificial intelligence for over 30 years (Hecht-Nielsen, 1987). Primarily, ANS technology is being applied to the development of information processing systems which perform tasks similar to those that the human brain does. For example, ANS technology has enhanced the processing capabilities of high-performance pattern recognition process.

In the past, many pattern recognition problems have not been solved successfully. Although many traditional computer science approaches have been applied, methods for carrying them out affordably with sufficient speed and with adequate robustness have not been available (Hecht-Nielsen, 1987). Artificial neural net models are an alternative which employs massive parallelism to complete the job in a reasonable time. One of the major reasons that artificial neural nets out-perform other approaches in solving the recognition problem is that the structures of the models are developed based on the understanding of the biological

nervous system (McClelland, Rumelhart, and Hinton, 1987). Other ANS properties include the following:

(1) The existence of attractor states: An attractor state is the state of a system that is at least quasi-stable (Oldham, 1986). It has been proven that attractor states exist in discrete systems such as computer models (Hogg and Huberman, 1985). This is an important feature since the attractor states can be used to store information. The information is introduced into the attractor by training the system using examples. Moreover, computing with attractive fixed points can lead to reliable behavior (Hogg and Huberman, 1984). This feature makes the neural net more attractive to the pattern recognition application.

(2) A learning and recognition capability: Learning and recognition are the basic processes involved in pattern recognition. Some artificial neural networks, such as the Hogg and Huberman models, can be trained by repeatedly presenting example patterns from an environment. The elements of the memory matrix which represents the model are adjusted accordingly in discrete time steps until the system converges according to some criteria. The system is said to be trained or to have converged if the values of the memory matrix reach stable values. If the system is properly trained, it will then recognize a pattern with which it was trained.

(3) A fault tolerant capability: Because of the existence of attractor states, the overall performance of the system tends to be insensitive to partial internal failures or input data errors. This is because the learned states are attractor states and the system is capable of evolving to the attractor states based on only partial or approximate information.

(4) High performance speed: The massive parallelism (parallel input channels and parallel output channels) enables the system to process with satisfactory speed. As mentioned previously, pattern recognition involves the learning and the recognition phases. In the learning phase, the time complexity depends on the number of iterations before the system converges; in the recognition phase, the time is independent of the code complexity.

(5) VLSI implementation: The structure of the network makes it easy to implement in VLSI technology.

In this research, the modeling of neural networks was studied and their capabilities in pattern recognition were explored. The study has made a substantial contribution to the understanding of neural networks through the application of the model to the area of pattern recognition.

Problem Definition

From among the various neural net models, two models that were developed by Hogg and Huberman in 1984 and 1985

(Model H-H1 and Model H-H2) were chosen as the focus of this study. They were examined, elucidated, and evaluated when applied to font recognition. Basically, Model H-H1 and Model H-H2 are fairly simple and easy to implement. Properties that make these models interesting are:

(1) The models are quite simple in presentation and can be easily implemented in software.

(2) Some non-determinism is injected via the memory update.

(3) All calculations can be done in parallel as a pipeline.

(4) Since the input and output contact the external world at the edges of the system, the models can be easily implemented by VLSI technology (Oldham, 1986).

(5) Although the topologies of these two models are the same, their output functions and learning rules are different.

The behavior of Model H-H1 has been investigated by several studies. These studies revealed that this model has a self-repairing capability (Hogg and Huberman, 1984) and conditional learning capability (Hogg and Huberman, 1985).

Model H-H2 can dynamically modify the basins of attractions to include or exclude a particular set of inputs by using the coalescence process or the dissociation process. The coalescence process is used to produce the

same outputs when the corresponding inputs are originally associated with different outputs; the dissociation process is used to differentiate the outputs when the input are initially mapping into the same outputs. To be able to manipulate the basins of attractors of the model is a very desirable feature in recognition application.

The basins of attractions of Model H-H2 can be manipulated to couple the desired grouping of inputs into specific outputs. This feature is particularly useful in the area of font recognition (Hogg and Huberman, 1985). In summary, the study was conducted to better understand, elucidate, and evaluate the models developed by Hogg and Huberman. The goals were accomplished by applying the models to font recognition.

Specific Tasks

The study was accomplished by the following steps:

(1) Problem formulation: Each English letter has many font representations. Although each font differs from the others, their overall images should be treated the same. In the study, the models were used to recognize the English characters independent of their font representations.

(2) Software Development: To simulate the behavior of these models, programs were developed in PL/I and were executed on a VAX 8650.

(3) Encoding scheme development: To obtain reliable recognition, an encoding system was developed and the attributes of the English characters were derived. Six fonts for each of the 26 English capital letters (156 characters in total) were encoded.

(4) Application of models through simulation: Half of the codes (the training set) were submitted to each model during the training phase. After each model was trained, the testing set was presented to it and the behavior of the model was recorded. The testing set consisted of all 156 characters. To explore the behavior of the models, their parameters were varied and different simulation data were generated.

(5) Model comparisons: The performance of these models were determined by comparing their recognition rates, memory space requirements, learning speeds, and recognition speeds. Also, the performances of the models were compared with the results of Cash and Hatamian's study (1987) and the result of Fujii and Morita's study(1971).

CHAPTER II

BACKGROUND INFORMATION AND LITERATURE REVIEW

In this Literature Review, the first section provides a review of the basic structures and functions of an Artificial Neural System (ANS). The second section introduces various neural net models. The Hogg and Huberman models which are the focus of this study are illustrated in the third section and was followed by a summary.

Neural Network Model

ANS Description

An artificial neural network is a network that consists of a set of nodes, the interconnections among nodes, learning rules, and input/output data (Oldham, 1986). The nodes, also called neurons or processing elements (PEs), represent particular conceptual objects and perform relatively simple computational processes. A node's job is to receive inputs from its neighbors, to compute the output according to the computation rules, and to send the output value to its neighbors. Thus, each node has little information stored internally and works as a short term working memory (Fahlman and Hinton, 1987).

Each node is connected to one or more other node(s) via an interconnection weight. In general, the connection weight can be positive or negative. Positive weights are excitory; negative weights are inhibitory. The weights are modified as a function of experience by the adaptive or learning rules and the information is stored in the connectivity pattern. In other words, the connection weights determine the long term storage of information (Fahlman and Hinton, 1987). Figure 1 illustrates the architecture of a typical ANS (Oldham, 1986).

As previously mentioned, changing the knowledge stored in the network involves modifying the pattern of connectivity and the weights. Many, but not all, learning rules can be considered as the variation of the Hebbian learning rule developed in 1949 (McClelland, Rumelhart, and Hinton, 1986). The basic principal for the Hebbian rule is that, if a pair of neighboring nodes are both highly active, the weight between them should be strengthened.

Overall inputs and outputs take place at the edges of the network. The inputs for a node are the outputs of its neighboring nodes. Similarly, the output from a node is input for other nodes.

Features of ANS

Artificial neural net models are parallel, distributed process models. They are inherently parallel since a large

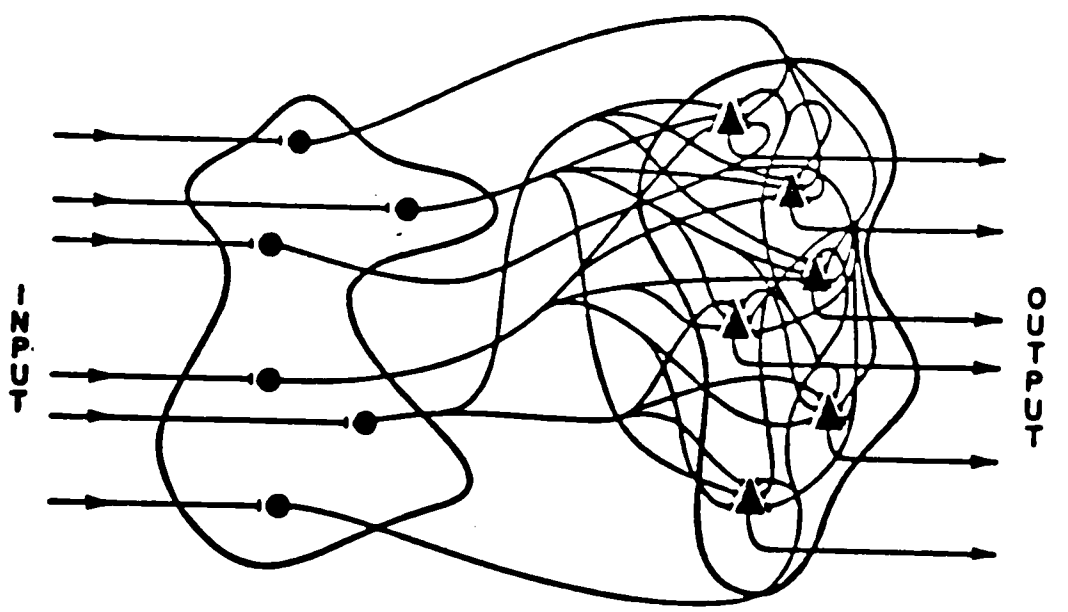


Figure 1: The Architecture of
a Typical ANS

number of the process elements perform their computations simultaneously and independently. They are inherently distributed because each process element is assigned a very tiny sub-task and stores little information.

The distributed feature enables the artificial neural net to be fault tolerant in regards to both internal failures and input data errors. This feature is drawn from the cognitive research idea that a given neuron is involved in many processes and many neurons participate in many decisions or processes (Oldham, 1986). As a result, a given neuron may play an insignificant role in the entire storage process. Consequently, a degree of fault tolerance is achieved by the process.

The parallel feature provides the models with computational efficiency because that all neurons perform simple operations, and a large number of operations occur concurrently.

Mathematical Model of ANS

Symbolically, a neural net can be viewed as G (I, O, n, f(n), C). Where I is input, O is output, f(n) is function operating at the nth node, and C is a connectivity matrix. The function f can be either linear or non-linear. Linear models have limitations because they can not mix or combine basic states to generate new states that correspond to learning or creativity (Oldham, 1986). They can only

accomplish superficial mixing of learning. Therefore, non-linear models were studied in this research.

Non-linear models are difficult to analyze. Even the simplest non-linear mechanism with very few nodes, is extremely complicated and intractable. Currently, the only method of analysis is accomplished by computer simulation (Oldham, 1986).

The most frequently used non-linearity functions are describe mathematically below:

(1) hard limiter: $f(x)= -1$, if $x<0$

$f(x)= 1$, if $x>0$

(2) threshold logic element:

$f(x)= 0$, if $x<0$

$f(x)= x$, if $0<x<c$

$f(x)= a$, if $x>c$

where c is the threshold

and a is a constant

(3) sigmoidal non-linearity: $f(x)=1+\tanh(x)$

Figure 2 presents these non-linearity functions graphically (Lippmann, 1987).

Applications of ANS

Neural net models have been applied to the following problem areas:

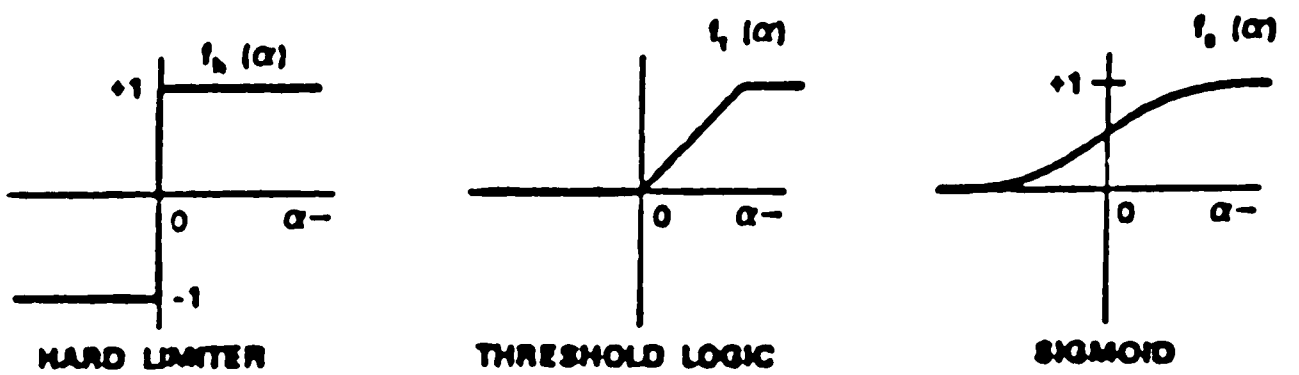


Figure 2: The Non-linearity Functions

(1) Classifier: Determining which one of M classes is the most representative of an unknown input pattern. (Lippmann, 1987). Neural net models can identify which class best represent a input pattern. This is a classical decision theory problem.

(2) Pattern Associator: The goal of pattern association is to build up an association between patterns defined in one set and patterns defined in another set. After the connections between sets are established, the associated pattern will appear on the second set if a particular pattern reappears on the first set (Lippmann, 1987). An auto-associator is one such that the output pattern is associated with the same as the input. Thus, whenever a portion of the input pattern is presented, the remainder of the pattern is to be filled in or completed. The content-addressable memory (also called associative memory) that can recall total storage information based on partial or noisy input is an example of auto-associator. A hetero-associator is one in which a pattern in the first set is associated with a different pattern in the second set.

(3) Regularity Discovery (feature detector): The models learn to respond to "interesting" pattern in their inputs (McClelland, Rumelhart and Hinton, 1987) and will divide the N input patterns into M classes.

Various Neural Network Models

An ANS is specified by the net topology, node characteristics, and training or learning rules (Lippmann, 1987). As mentioned previously, the pattern recognition process involves a training phase and a recognition phase. In the training phase, some models are trained with supervision while others are not. When a model is trained with supervision, it is provided the side information or label that specifies the correct class for the new input patterns. Whereas, nets trained without supervision, no information concerning the correct class is provided. Among those models that are trained with supervision, Hopfield models were used as auto-associators while the Perceptron and Anderson models were used as hetero-associators. Among those models that are trained without supervision, the Carpenter and Grossberg model was used as an auto-associator and the Hogg and Huberman models were used as hetero-associators. Figure 3 presents the model classifications.

Hopfield Models

The various Hopfield models, which are highly connected with strong feedback, can be used as an associative memory or can be used to solve optimization problems. In the models that can be used as associative memories, the inputs and outputs are binary values which can take on +1 or -1.

	Supervised	Non-supervised
Auto-associator	Hopfield Models	Carpenter/Grossberg Models
Hetero-associator	Perceptron Anderson Model	Hogg/Huberman Models

Figure 3: Model Classifications

The output of each node is fed back to all other nodes via weights. Since the model is trained with supervision, examplars are provided. Following the initialization, the net iterates in discrete time steps. The pattern specified by the node outputs, after convergence, is the net output. It has been proven that this net converges to stable final states when the weights are symmetric (Hopfield, 1982). This model has two major disadvantages when use as a content addressable memory. First, there is a limitation in the number of patterns that can be stored and accurately recalled. Second, the examplar pattern is unstable if it shares many bits in common with another examplar pattern.

Also, Hopfield models can solve optimization problems such as the classical Traveling-Salesman problem, analog to digital conversion problems, and linear programming problems. The stable states of the network containing N neurons are the local minima of the energy function E and the circuit operates over the interior of a hypercube defined by the input patterns. The minima only occur at the corners of the hypercube; stable states correspond to the 2^N corners of the hypercube that minimizes E. These optimization problems can be solved by the following steps: (1) Choosing the connectivities; (2) Choosing the input bias circuits, which appropriately represent the function to be minimized; (3) Providing an initial set of inputs that cause the system to converge to a stable state which represents

the minimum of the function; (4) Interpreting the solution from the stable final state (Hopfield and Tank, 1985).

Carpenter and Grossberg Models

Carpenter and Grossberg introduced the adaptive resonance theory (ART) which forms clusters and is trained without supervision (Lippmann, 1987). ART embedded a competitive learning model into a system that can solve the stability-plasticity dilemma (Carpenter and Grossberg, 1988). Stability is essential for the system to remain unchanged in response to irrelevant events. Whereas plasticity is essential for the system to learn in response to significant new events. The stability-plasticity dilemma says that the system must employ some mechanism to distinguish between signal inputs and noisy inputs so that it knows how to switch between its stable and its plastic modes.

The ART I deals with binary inputs, while ART II deals with analog inputs. These two models are rather complicated and involve many parameters and learning rules. Among these parameters, the vigilance parameter, which ranges between 0.0 and 1.0, determines how close a new input pattern must be to a stored examplar to be considered similar. High vigilance forces the system to search for new categories in response to small differences between input and expectation.

Thus the system classifies input patterns into a large number of categories. On the other hand, low vigilance enables the system to tolerate large mismatches and thus groups input patterns into a small number of categories. It has been mathematically proven that an ART I architecture is capable of stably learning a recognition code in response to an arbitrary sequence of binary input patterns, until it utilizes its full memory capacity (Carpenter and Grossberg, 1987).

Single Layer Perceptron

The original perceptron was developed by Rosenblatt (Lippmann, 1987). perceptrons are able to decide whether an input belongs to one of two classes (Class A or Class B) that are separated by a decision region created in the multidimensional space spanned by the input variables. The basic idea is to compute a weighted sum of input elements, subtract a threshold, and pass the result through a hard limiting non-linearity function such that the output is either +1 or -1. In the training phase, both inputs and desired outputs are provided. The connection weights are adapted only when the actual output differs from the desired output. The decision rule is to respond Class A if the output is +1 and Class B if the output is -1. The Perceptron forms two decision regions separated by a hyperplane.

Single layer perceptrons can be expanded to multi-layer perceptron by introducing hidden layers. A three-layer perceptron can form arbitrarily complex decision regions. Although it can not be proven that this model will converge, it has been shown to be successful for many problems of interest.

Anderson Model

The Brain In a Box (Anderson model) was developed by Anderson in 1977. This model performs as a simple linear associator that is trained with supervision. In the model,any unit can be connected to any other unit.

Assume that there are two groups of N neurons, X and Y and every neuron in X projects to every neuron in Y. A synaptic strength, a_{ij}, connects the jth neuron in X with the ith neuron in Y. Let f_1 be a vector that represents X and g_1 be a vector that represents Y. An association can be made between f_1 and g_1 so that the presentation of f1 alone will give rise to g_1. This association is developed by

$A_1*f_1=g_1$ where A_1 is the connectivity matrix.

A_1 can be computed by

$$A_1=g_1*f_1^T.$$

When there are k sets of neurons, namely (f_1,g_1), $(f_2,g_2),\ldots,(f_k,g_k)$, there exists a single connectivity

matrix to associate the set of activity patterns. Each pair of neurons can generate a connectivity matrix, A_i. Then the overall connectivity matrix A can be computed by

$$A = \sum A_i .$$

It has been mathematically proven that, if the input vectors are mutually orthogonal, the system can perform the association perfectly (Anderson, Silverstein, Ritz, and Jones, 1977).

Hogg and Huberman Models

Hogg and Huberman models, which are flow forward and synchronous networks, were developed in 1984 and 1985. They will be referred to as Model H-H1 and Model H-H2, respectively. The models are able to map many inputs to a few final states. The final states are called fixed point attractors. The set of inputs that map into a given output defines the basin of the attractor for that output.

The architectures of Model H-H1 and Model H-H2 are exactly the same and each can be represented by rectangular matrices (memory matrices) that consist of M rows and N columns of identical processors, each of which is locally connected to its neighbors. Each element has a value stored in it. Additionally, each processor has an adjustable internal state, or memory, which allows it to adapt to its

local environment. The overall input and output take place at the edges of the matrix, with the upper edge of the matrix (the first row) for input and the lower edge (the last row) for output. For each element, it receives two integer inputs from its neighbors along its diagonals in the preceding row and produces an integer output which in turn becomes an input to the nodes along the diagonals in the following row (see Figure 4).

The hard limiter non-linearity function is used so that the output values are constrained to lie within a range, namely [Smin, Smax]. The memory values are limited within [Mmin, Mmax] range. Those output values that are equal to the extremes of these ranges are said to be saturated.

Let $IL_{ij}(k)$ and $IR_{ij}(k)$ be the inputs to the element in the ith row and the jth column after the kth time step and let $O_{ij}(k)$ be the output value for the ith row, jth column element. The connections between elements are defined by these relations.

$$IL_{ij}(k) = O_{i-1,j-1}(k),$$
$$IR_{ij}(k) = O_{i-1,j+1}(k).$$

where $1 \leq i \leq M$ and $i \leq j \leq N$

The boundaries of the matrix, for the top, bottom, and sides edges are specified respectively as

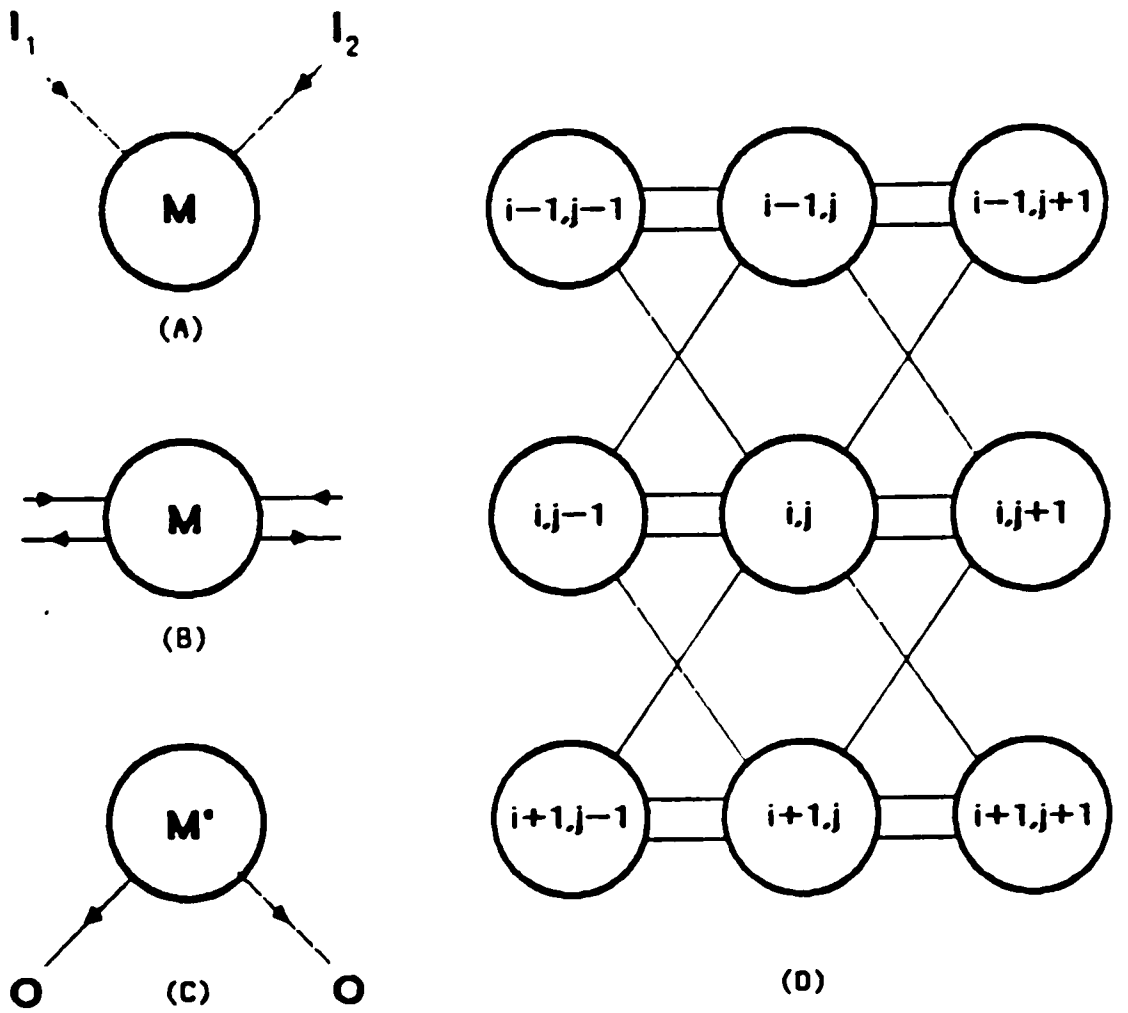


Figure 4: The Hogg and Huberman Model

$$O_{0j}(k) = S_j(k),$$

$$O_{mj}(k) = R_j(k),$$

$$O_{i0}(k) = O_{in}(k),$$

$$O_{i,n+1}(k) = O_{i1}(k).$$

where S(k) is the external input signal to the matrix at step k, and R(k) is the resulting output vector.

The two models employ different output functions and learning rules which are described next.

(1) Model H-H1:

(A) The output from each element for the k+1 step is

$$O_{ij}(k+1)=\max\{Smin,\min(Smax,M_{ij}(k)[IL_{ij}(k)-IR_{ij}(k)]\}.$$

This rule enhances the differences of the inputs by multiplying them by the memory value. The saturation process keeps the values within the specified interval.

(B) The memory values are updated by the following rule:

if $O_{ij}(k) > O_{i,j-1}(k)$ and $O_{ij}(k) > O_{i,j+1}(k)$ then

$M_{ij}(k) = \max\{Mmin,\min(Mmax,M_{ij}(k-1)+1)\}$

else

if $O_{ij}(k) < O_{i,j-1}(k)$ and $O_{ij}(k) < O_{i,j+1}(k)$ then

$M_{ij}(k) = \max\{Mmin,\min(Mmax,M_{ij}(k-1)-1)\}$

else

$M_{ij}(k) = M_{ij}(k-1).$

(2) Model H-H2:

(A) The output function for the second model is

$$O_{ij}(k+1)=\max(Smin,\min(Smax,S(IL_{ij}(k),IR_{ij}(k))*(|IR_{ij}(k)|+|IL_{ij}(k)|)+M_{ij}(k)))$$

where for even rows, if $IL_{ij}(k)$ is zero, then $S(IL_{ij}(k),IR_{ij}(k))$ is the sign of $IR_{ij}(k)$, otherwise the sign of $IL_{ij}(k)$; and for odd rows the roles of $IL_{ij}(k)$ and $IR_{ij}(k)$ are reversed.

(B) Learning rules for the second model are:

(a) The coalescence Rule (contracting rule) is capable of dynamically associating the basins of two or more attractors to produce the same attractor. Figure 5 pictorically demonstrates the basins of two attractors before and after the process according to the rule listed below.

if at least one of $O_{ij}(k-1)$ and $O_{ij}(k)$ is not saturated AND $O_{ij}(k)*O_{ij}(k-1) < 0$ then change M_{ij} by 1, with the sign of the change given by the sign of the output with largest magnitude,

else

M_{ij} is unchanged.

(b) The dissociation rule (expanding rule) is used to separate the inputs which initially map into

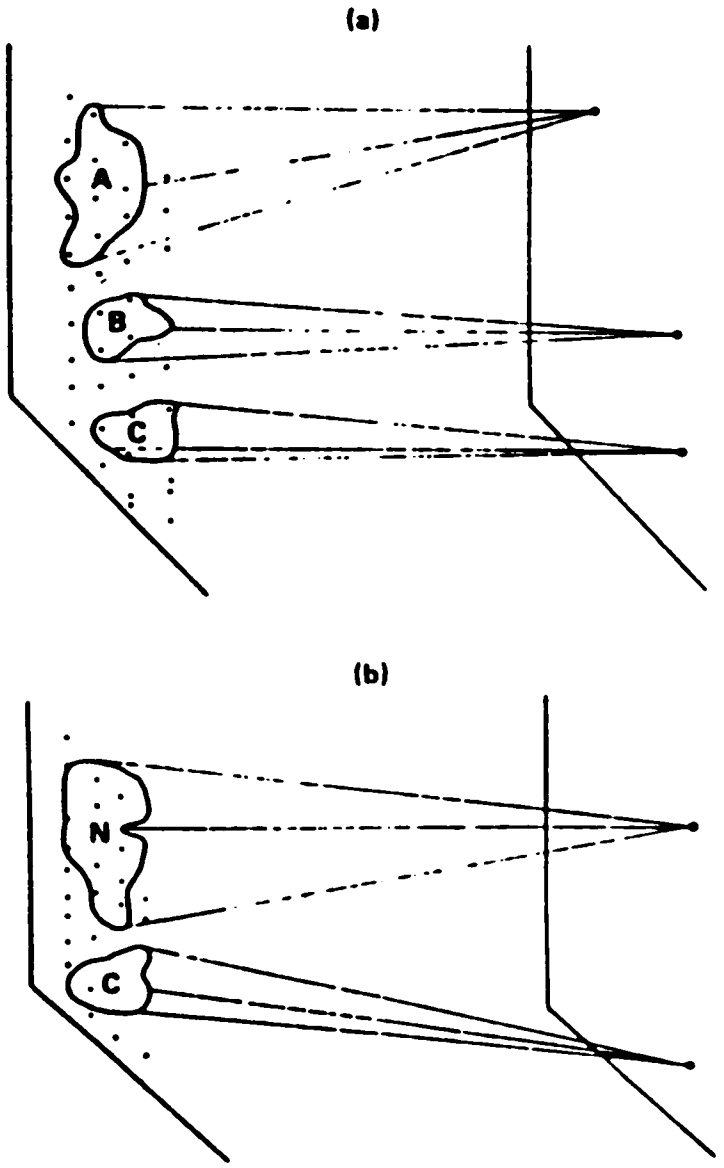


Figure 5: Basins of Attractor Change
After the Coalescence Process

the same output. The expanding rule operates opposite to the contracting rule.

if at least one of $O_{ij}(k-1)$ and $O_{ij}(k)$ is not saturated AND $O_{ij}(k)*O_{ij}(k-1) > 0$ then change M_{ij} by 1, with the sign of the change opposite of that of either output

else

M_{ij} is unchanged.

In the training phase, a set of training patterns are submitted to the models, periodically, as a pipeline. Then the outputs and the memory values are updated according to the various rules. The model is said to be stabilized if the output values for the input patterns do not change. Once trained, the memory states will be fixed. The official outputs can be obtained by running the inputs through the model one more time. This has the effect of fixing the output patterns after the memory is locked and bars against matrix changes to the output patterns due to one or more memory element changing in the later training phase. In the recognition phase, the recognition set is sent to the model. By comparing the outputs with the official outputs, the model is capable of determining whether the input is one of the trained patterns.

Self-organizing, self-repairing and conditional learning exist in the Hogg and Huberman models. Self-

organizing means that the model is capable of converging to fixed point attractors. Self-repairing means that small fluctuations in either data or memory values won't cause the model to relax to another attractor. Conditional learning means that a set of input patterns can be learned faster if they are close to previously learned ones.

Summary

The capability of dynamically changing the basins of attractors opens a new research area. As Hogg and Huberman (1985) suggested, the coalescence and dissociation processes provide a flexible way to transform desired groupings of inputs into specific outputs. These processes are particular useful in font recognition since the sets of inputs generated by an encoding scheme for the same letters in different fonts can be identified as an equivalent class.

Little is known about the dynamics of attractor states and their behavior under general circumstance. In fact, a whole new area of research has been initiated in this area called "chaos theory" in the last 10 years (Schuster, 1988). Issues such as the performance stability of the models when they are subjected to parameter changes, can not be answered with ease, even in the simplest nontrivial cases. Thus more research work is needed to gain insight and an understanding of the behavior of these models.

CHAPTER III

METHOD AND APPROACH

This study was conducted to understand, elucidate, and evaluate the two models developed by Hogg and Huberman (H-H1 and H-H2). These goals were accomplished by applying the two models to font recognition problem. Font recognition was achieved by following these steps: (1) Problem Formulation, (2) Software Development, (3) Encoding Scheme Development, (4) Application of Models Through Simulation, and (5) Model Comparisons.

Problem Formulation

Dynamic modification of the basins of attractors is very useful in recognition problems that require putting sets of inputs into the same equivalent classes. Thus font recognition was chosen to explore the models' recognition capabilities.

Each English letter has many font representations. Although fonts for a particular letter differ from one another, the overall images are recognized by humans as the same letter. In other words, letters in various fonts are treated as being in the equivalence class. When the models are applied to font recognition, it is desired that they

recognize letters independent of their font representations. For this study, six fonts were selected for the 26 capital letters of the English alphabet. The fonts, which included (1) Courier,(2) New York, (3) Chicago, (4) Geneva, (5) Times, and (6) Venice, were generated for each letter by a MacIntosh computer. Figure 6 lists the 156 target characters.

The goal of this study was to discover how well the models can learn and recognize the letters in various fonts. The learning phase was accomplished by submitting the training set to train the model. The training set, in this case, are the vector representations of 78 characters (3 fonts for 26 capital letters). The recognition phase was accomplished by submitting the vector representations of 156 characters (6 fonts for 26 capital letters) to determine whether the models can recognize the letters correctly.

Software Development

In order to explore the behavior of the models, programs were developed in PL/I and were executed on a VAX 8650. The programs were implemented using top-down design and step-wise refinement schemes. The program to simulate Model H-H1 contains 685 statements and is provided in Appendix A; the program to simulate Model H-H2 contains 917 statements and is provided in Appendix B. The inputs to the models, one-dimensional character vectors that represent the

Courier:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

New York:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Chicago:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Geneva:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Times:

ABCDEFGHIJKLMNOPQRSTUVWXYZ

Venice

ASDFGHJKLQWERTYUIOPZXCVBNM

Figure 6: The 156 Target Characters

target characters, will be discussed in detail in following sections.

Encoding Scheme Development

The principal difficulty encountered in the pattern recognition problem is finding a way to present the patterns in a formalized manner. In many successful character recognition systems, a character is first normalized (e.g., aligned in position), then preprocessed (e.g., by feature extraction), and then classified. In order to produce satisfactory results, a good preprocessing algorithm (encoding scheme) is needed. A good encoding scheme must be able to express the objects under consideration in a compact way, without losing any information. The steps to encode the input characters are (1) Normalizing fonts and Generating character matrices; (2) Selecting character properties and Generating property matrices; (3) Computing the Filter Matrix; and (4) Extracting Properties and Constructing Character Vectors. These steps are discussed in the following paragraphs.

Normalizing Fonts and Generating Character Matrices

A total of 156 capitalized English characters were translated into 18x18 character matrices. In the character matrices, 0s represent the background and 1s represent the

character image. The matrices are the inputs to the next process. An Example of the character 'A' in Courier font and its matrix representation are shown in Figure 7.

Selecting Character Properties and Generating Property Matrices

The inputs to models H-H1 and H-H2 are one-dimensional vector. Therefore, the character matrices (18x18) have to be transferred to vector representations. To accomplish this, 14 selected properties (Fujii and Morita, 1971) that constitute the basic building blocks of a character were extracted from each character matrix. Each property was represented by a 3x3 matrix. The selected character properties and their corresponding property matrix are shown in Figure 8. The 14 properties are extracted from each character to build a 14-tuple (the character vector C) in which c_i is associated with the ith property.

A combined property matrix (X) is a 14x9 matrix which represents the 14 selected character properties. Each row of X is a 1x9 vector (x_i) that represents a character property (see Figure 8). The nine elements in x_i are obtained by chaining the three rows of the 3x3, ith property matrix into a 1x9 vector. For example:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \Rightarrow \quad x_i = [1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1].$$

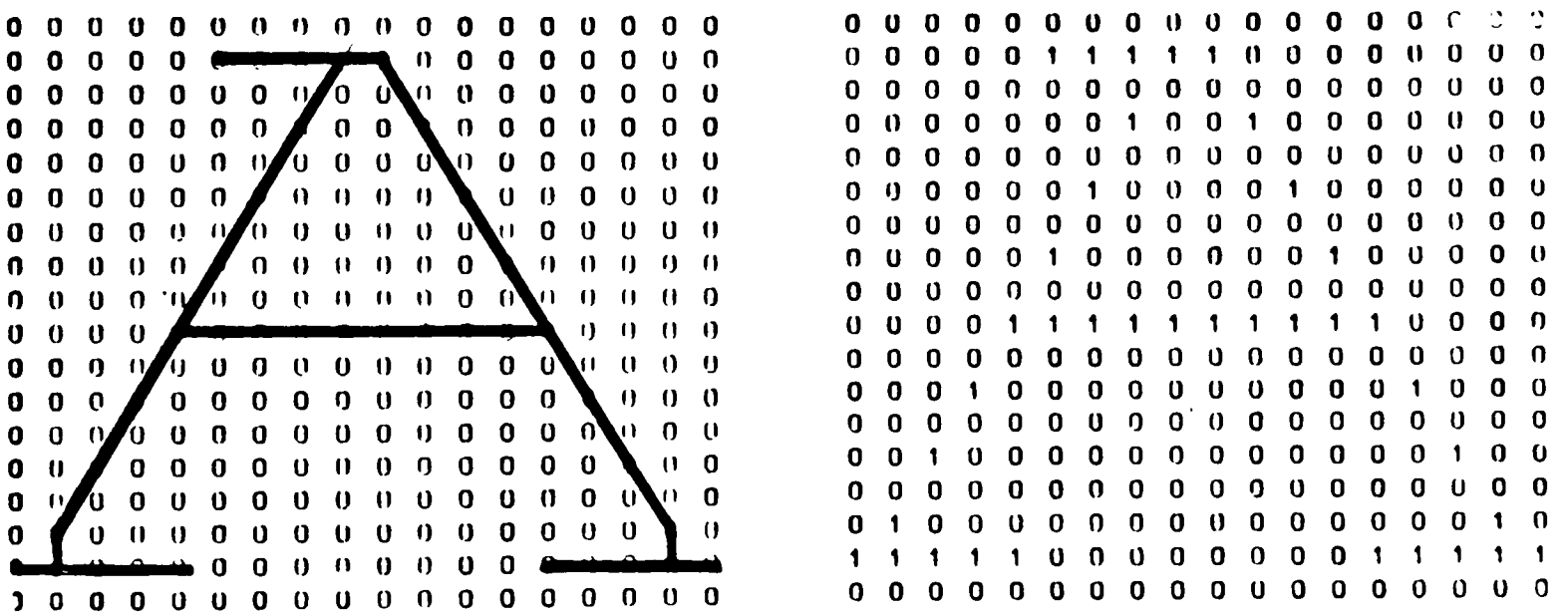
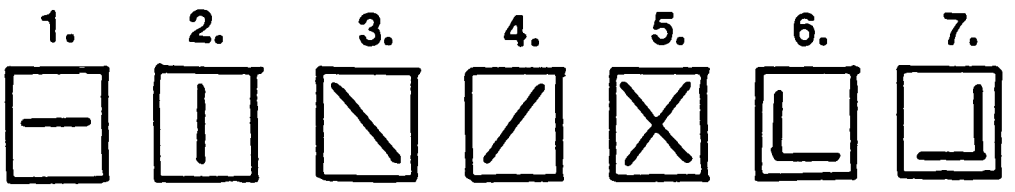
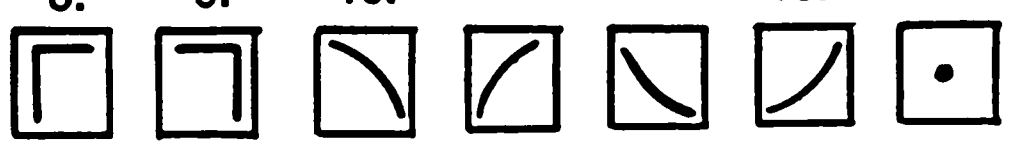


Figure 7: The Courier 'A' Font and Its Matrix Representation



1.	2.	3.	4.	5.	6.	7.
0 0 0	0 1 0	1 0 0	0 0 1	1 0 1	1 0 0	0 0 1
1 1 1	0 1 0	0 1 0	0 1 0	0 1 0	1 0 0	0 0 1
0 0 0	0 1 0	0 0 1	1 0 0	1 0 1	1 1 1	1 1 1



8.	9.	10.	11.	12.	13.	14.
1 1 1	1 1 1	1 1 0	0 1 1	1 0 0	0 0 1	0 0 0
1 0 0	0 0 1	0 0 1	1 0 0	1 0 0	0 0 1	0 1 0
1 0 0	0 0 1	0 0 0	0 0 0	0 1 0	0 1 0	0 0 0

Figure 8: The Selected Character Properties and Their Matrix Representations

To be specific, X is constructed by putting the vector corresponding to the first property (x_1) in the first row of X, the vector corresponding to the second property (x_2) in the second row of X, and so on. That is,

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \vdots \\ x_{13} \\ x_{14} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ & & & \vdots & & & & & \\ & & & \vdots & & & & & \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} .$$

Computing Filter Matrix

In this section, the concepts of the property recognition matrix (Y) and the filter matrix (W) will be introduced and their functions will be explained.

The property recognition matrix (Y) is a 14x9 matrix which represents a simplification of the property matrix. Matrix Y is an arbitrarily chosen matrix that is constructed as simple as possible. For example:

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ \vdots \\ y_{13} \\ y_{14} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & & \vdots & & & & & \\ & & & \vdots & & & & & \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} .$$

The filter matrix (W) is a 9x9 matrix which maps X (the combined property matrix) to Y. Their relationship is

$$Y = XW.$$

Given X and Y, W is equal to $X^{-1}Y$, if X and Y are square and non-singular matrices. In this case, X is a 14x9 rectangular matrix (there are more equations than unknown). Therefore, W is over-determined and no unique solution exists. In this project, the minimum squared error (MSE) technique was adopted (Duda and Hart, 1973) to approximate W. The MSE procedure minimizes the squared error between Y and XW. Using this procedure, the pseudoinverse of X (X^{+}) is computed by

$$X^{+} = (X^{t}X)^{-1}X^{t}.$$

where X^{t} is the transpose of X and $(X^{t}X)^{-1}$ is the inverse of $X^{t}X$, thus

$$W = X^{+}Y.$$

After W is computed by minimizing $(Y-XW)^{2}$, the exact Y is recomputed as the product of X and W.

Extracting Properties and Construct Character Vectors

Recall that each character is represented by a 18x18 character matrix. The purpose of this step is to extract the selected properties from the character matrix and to construct the character vector C. To extract character properties, let A be the search area (character matrix) and

let w be a 3x3 window matrix as shown in Figure 9. The window starts moving from the left upper corner of the search area all the way across the first three rows (rows 1, 2, and 3). It then moves down one row (rows 2, 3, and 4) and scans all the way across, and so on. Basically, the movement is made from left to right and from top to bottom. The search is stopped when the lower right corner of the window coincides with the lower right corner of the character matrix. Let z' be a 1x9 vector whose elements are obtained by chaining the 3-row elements in w. For example:

$$w = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad => \quad z' = [1\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1]$$

Then z' is recognized as one of the selected properties if it maps to y' ($y' = z'W$) which is one of the rows in Y.

If z' is recognized, say $y'=y_k$, then the kth element (c_k) in the character vector is incremented by a weighting factor. The weighting factor is a function of property as well as the property location. For those properties that occur rarely, such as property cross, their location in the character matrices were identified by assigning different weights. The values of the weights are arbitrarily assigned and have no meaning other than for distinguishing purposes. To determine the weighting factor, the character matrix was divided into nine 6x6 portions with the left upper portion assigned score 1, 2 to its right and so forth (see Figure

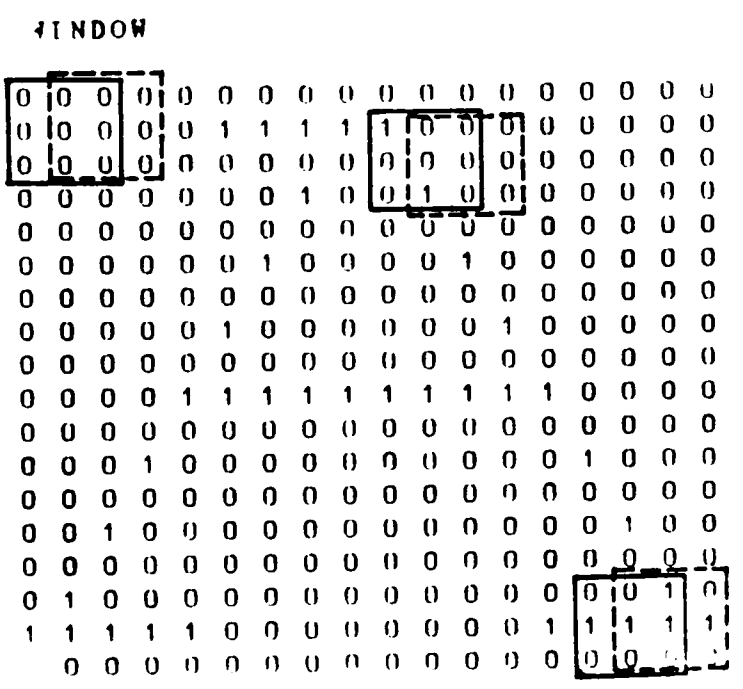


Figure 9: The Search Matrix and the Windows

10). Table 1 lists the weighting factors. On the other hand, for those character properties that occur frequently, their location were ignored. One was assigned to every occurrence of the property no matter where it is. The character vector C is a 1x14 vector ($C=[c_1, c_2, \ldots, c_{14}]$) and each element in C is an accumulated score for its corresponding property. The character vector for each character matrix is extracted according to the following algorithm:

do i=1 to 16 by 1

  do j=1 to 16 by 1

    $z'=[A_{i,j}\ A_{i,j+1}\ A_{i,j+2}\ A_{i+1,j}\ A_{i+1,j+1}\ A_{i+1,j+2}$
    $A_{i+2,j}\ A_{i+2,j+1}\ A_{i+2,j+2}]$

    $y'=z'W$

    if y' exactly matches one of the rows in Y, say $y'=y_k$,
    then C_k is incremented by the weighting factor of the
    kth property

  end j

end i.

The encoding scheme is affected by the relative position of the properties. Also, the thickness of the line of the character is restricted to be one unit.

Only the sign distribution of the inputs affects the output of Model H-H2. To increase the variabilities of Model H-H2 output, the character vector (all of its elements

location 1	location 2	location 3
location 4	location 5	location 6
location 7	location 8	location 9

Figure 10: Location Assignment in the Character Matrix

Table 1

Assignment of Weighting Factor

location	1	2	3	4	5	6	7	8	9
property 1	1	1	1	1	1	1	1	1	1
property 2	1	1	1	1	1	1	1	1	1
property 3	1	1	1	1	1	1	1	1	1
property 4	1	1	1	1	1	1	1	1	1
property 5	1	2	3	4	5	6	7	8	9
property 6	1	2	3	4	5	6	7	8	9
property 7	1	2	3	4	5	6	7	8	9
property 8	1	2	3	4	5	6	7	8	9
property 9	1	2	3	4	5	6	7	8	9
property 10	1	1	1	1	1	1	1	1	1
property 11	1	1	1	1	1	1	1	1	1
property 12	1	1	1	1	1	1	1	1	1
property 13	1	1	1	1	1	1	1	1	1
property 14	1	1	1	1	1	1	1	1	1

are positive before the process) is modified according to the following rules:

```
do i=1 to 14 by 1
     if threshold[i]=0 then
        if C[i]=0 then
           C[i]=-4
        else
           C[i]=C[i]
     else
       C[i]=C[i]-threshold[i]
end i.
```

where threshold = [7 7 4 4 0 0 0 0 0 0 0 0 0 4].

The values of the threshold vector were chosen according to the frequency occurrence of the selected character properties for the 156 characters. The codes that go through the above processes are the inputs to the models.

Application of Models Through Simulation

During these simulations, the training set was composed of the vector representations of 78 characters: three out of the six fonts for each letter of the alphabet were randomly selected. All of the 156 character were the candidates for the recognition set.

For Model H-H1, the training set was fed to the model, repeatedly, during the training phase. The model computed the outputs and adjusted the memory matrix values, according

to its output function and the learning rule until there was convergence. After the model stabilized, the memory matrix values were fixed and the official outputs were generated by running the model one more time, using the training set. The model recorded the outputs as well as their associated letters. Thus the inputs were divided into several categories. In general, several letters fell into the same categories. This happened when more than two letters created the same output. Figure 11 shows an example of the decision tree. In this example, character E and character F produced the same output. The learning process was started again to create a child model using the vector representations of E's and F's as inputs. Another memory matrix was created to distinguish E's from F's. This process continued recursively, to build the decision tree until either each output was associated with only one letter or the depth of the tree reached 7. The depth of the decision tree was limited to 6 to avoid having the learning process run endlessly.

For Model H-H2, three characters (three fonts for the same letter) of the training set were submitted to the model at one time. The model computed the outputs and adjusted the memory matrix values based on the output function and coalescence rule described in Chapter II. After Model H-H2 converged, three vectors that represented another letter were sent to the model and the previous process was started

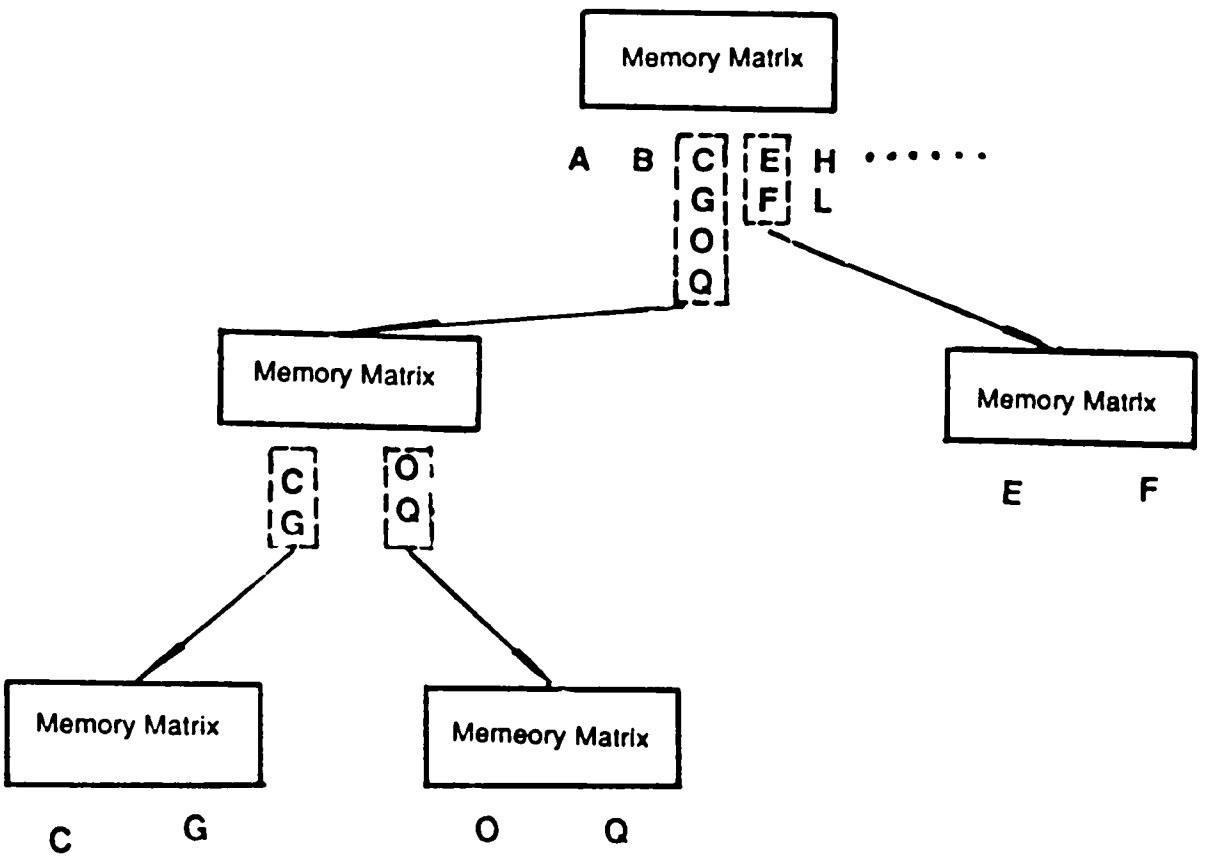


Figure 11: An Example of the Decision Tree

over again. After all of the 78 characters were learned, the values of the memory matrix were fixed. The training set was submitted to the model to obtain the official outputs. The model recorded the outputs as well as their associated letters. Normally, the model divided the inputs into several categories with those that produced the same output in one category. If two or more letters fell into the same category, the vector representations for those letters were again submitted to the model. The child model was created to discriminate the different letters that fell into one category by adopting the dissociation rule. These processes were run recursively to build the decision tree until either each output was associated with only one letter or the depth of the tree reached 7. When building the decision tree, the coalescence rule and the dissociation rule were used alternatively, with the odd depth for the coalescence rule and the even depth for the dissociation rule. Again, the depth of the decision tree was limited to 6 to prevent the process from running forever.

In the recognition process, all 156 codes were input to the models. The recognition set was submitted to the memory matrix in the root of the decision tree. If the output matched one of the outputs previously recorded and the output was only associated with one letter, then the model indicated that the input was the recorded character. On the other hand, if the output matched one of the outputs but

more than two letters were associated with it, the input code was sent to its child memory matrix for further process. The searching started from the root of the tree, and it stopped either when the memory matrix was a leaf node or the output did not match any of the outputs.

As a result, the simulation programs generated the following information:

1. overall recognition rate,
2. number of rejected characters,
3. number of correctly recognized characters,
4. recognition rate of the trained characters,
5. rejection rate of the trained characters,
6. recognition rate of the untrained characters,
7. rejection rate of the untrained characters,
8. depth of the decision tree,
9. number of the memory matrix used in the decision tree,
10. learning speed (in terms of number of iterations),
11. recognition speed (in terms of number of times),
12. number of correctly recognized characters for each alphabet,
13. number of rejected characters for each alphabet,
14. number of correctly recognized characters for each font, and
15. number of rejected characters for each font.

Model Comparisons

The parameters, such as memory matrix size, Smin_Smax, Mmin_Mmax, were manipulated to determine the models' performance. The memory matrix sizes were set to be 4x14, 6x14, 8x14, and 10x14. Smin_Smax were assigned to be ±12, ±15, and ±18. For H-H1, Mmax was set to be 4, 6, and 8 with Mmin fixed to 1; for Model H-H2, Mmin_Mmax was set to be ±8, ±10, and ±12. Thus 36 observations were obtained for each model. These levels were determined by expanding the levels used by Hogg and Huberman (1985). Table 2 presents the levels of parameters.

The performances of these models were determined using the following criteria: accuracy, required memory space, learning speed, and recognition speed. These four criteria are defined below

(1) The accuracy is scored as the fraction of correctly recognized characters with respect to the 156 characters.

(2) The required memory space is defined by the number of memory matrices used to build the decision tree. The less discriminative the model is, the more memory matrices are needed to build the tree. Generally speaking, the number of memory matrices required depends on the shape of the tree.

(3) The learning speed is determined by the number of iterations required to build the decision tree. The fewer

Table 2

Levels of Parameters

Model	Parameter	Levels 1	2	3	4
H-H1	Memory Matrix Size	4x14	6x14	8x14	10x14
	Smin_Smax	±12	±15	±18	
	Mmin_Mmax	1_4	1_6	1_8	
H-H2	Memory Matrix Size	4x14	6x14	8x14	10x14
	Smin_Smax	±12	±15	±18	
	Mmin_Mmax	±8	±10	±12	

iterations the model needs in the learning phase, the faster the learning speed is.

(4) The recognition speed is determined by the number of times the input data have to be submitted to the memory matrices in the decision tree in order to recognize 156 characters. The fewer the number of times the input data need to be submitted, the faster the recognition speed is. The recognition speed depends on the shape of the decision tree. Normally, the shallower the tree is, the faster the recognition speed is.

CHAPTER IV

SIMULATION RESULTS

To analyze the performances of the models, Statistics Analysis System, a computer system of software products for data analysis, was used. Basically, descriptive statistics (mean and standard deviation: STD) were obtained for performance data for each model. The effects of model parameters on model performance were determined statistically using an ANalysis Of VAriance (ANOVA) procedure. The performance comparison was conducted by Pairwised T Test.

In the first section, the extracted properties of the input characters are summarized. The principal component analysis of the character vectors is presented in the second section. The behavior of models as functions of the size of memory array, Smin_Smax, and Mmin_Mmax are presented in the third and fourth sections, respectively. Finally, model comparisons are discussed in the last section.

Characteristics of Collected Characters

Recall that 156 machine-printed characters were created for determinating the recognition performances of the models and fourteen selected properties were extracted from each

character. Figure 12 presents the frequency occurrence of the properties of the 156 letters. The property occurrence ranges from 7 for "cross," which is 0.16% of the total number of occurrence, to 1754 for "vertical line", which is over 40.44% of the total number of occurrence.

Principal Component Analysis

The dimensionality of the data can be reduced by removing or combining highly correlated data (Duda and Hart, 1973). Principal component analysis was used to reduce the dimensionality by forming linear combinations of character vectors features. SAS was used for this purpose. Table 3 illustrates the statistical result.

The eigenvalues indicate that nine components, which account for 92.55% of the standardized variance, provide a good summary of the data. The eigenvectors provide the principals as the linear combinations of the fourteen elements in the character vectors. The results of the computer simulations show that the recognition rate is lower when using the nine principals as the inputs than when using the 1x14 vector as the input code. The results also show that it requires more memory space and that the learning and recognition speeds are much slower. As a result, only 1x14 vectors were used as the input data to the models in this study. The performance data are presented in Table 4.

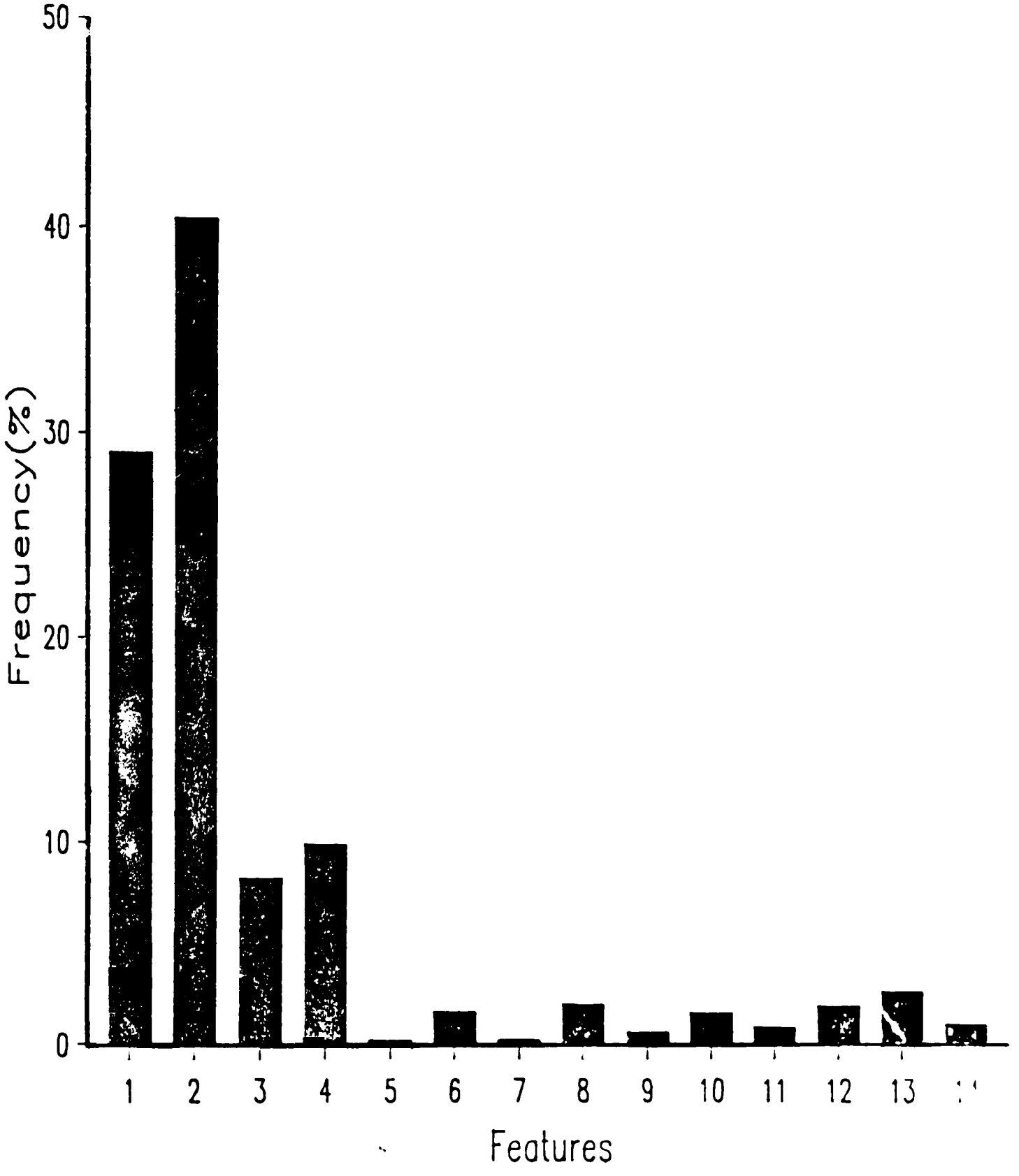


Figure 12: Frequency Occurrence of the Extracted Properties

Table 3

Principal Component Analysis of the Character Vectors

	EIGENVALUE	PROPORTION	CUMULATIVE
PRIN1	3.25390	0.232421	0.23242
PRIN2	2.24902	0.160644	0.39307
PRIN3	1.95603	0.139716	0.53278
PRIN4	1.47279	0.105199	0.63798
PRIN5	1.23766	0.088404	0.72639
PRIN6	1.03572	0.073980	0.80037
PRIN7	0.76504	0.054646	0.85501
PRIN8	0.52029	0.037164	0.89217
PRIN9	0.46661	0.033329	0.92550
PRIN10	0.34913	0.024938	0.95044
PRIN11	0.28687	0.020491	0.97093
PRIN12	0.20617	0.014726	0.98566
PRIN13	0.13671	0.009765	0.99542
PRIN14	0.06406	0.004576	1.00000

EIGENVECTORS

	PRIN1	PRIN2	PRIN3	PRIN4	PRIN5	PRIN6	PRIN7
C1	0.297	0.457	0.145	-.069	0.096	0.112	-.293
C2	0.227	-.347	0.170	0.214	-.128	-.409	0.349
C3	-.351	0.250	-.042	0.301	-.205	0.071	0.403
C4	-.343	0.191	-.226	0.139	-.151	-.035	-.496
C5	-.260	0.150	-.230	0.257	-.390	0.293	0.179
C6	0.386	0.281	0.149	0.036	-.268	0.036	-.052
C7	0.223	-.185	-.236	0.488	0.245	0.204	-.118
C8	0.392	0.273	0.092	0.133	-.184	0.200	0.024
C9	0.264	-.187	-.256	0.426	0.211	0.280	0.004
C10	-.049	0.407	0.334	0.199	0.308	-.034	0.387
C11	-.250	0.215	0.067	0.181	0.613	-.166	-.065
C12	-.228	-.283	0.463	0.101	0.032	0.315	-.226
C13	-.116	-.179	0.582	0.144	-.134	0.306	-.103
C14	0.003	-.048	-.152	-.481	0.240	0.587	0.346

Table 3

(Continued)

EIGENVECTORS

	PRIN8	PRIN9	PRIN10	PRIN11	PRIN12	PRIN13	PRIN14
C1	0.091	-.159	-.266	0.100	0.012	0.662	-.131
C2	0.335	-.046	-.136	0.526	0.005	0.185	0.060
C3	-.175	0.283	0.383	0.055	0.197	0.416	-.200
C4	0.282	0.436	-.166	0.399	0.050	-.135	0.172
C5	0.190	-.619	-.267	-.023	-.139	-.089	0.058
C6	0.382	-.077	0.379	-.111	0.529	-.286	0.059
C7	0.393	0.126	0.303	-.260	-.389	0.158	0.031
C8	-.416	0.064	0.185	0.426	-.453	-.260	0.081
C9	-.330	0.054	-.352	0.093	0.518	-.083	-.060
C10	0.116	0.250	-.352	-.237	-.067	-.163	0.384
C11	0.045	-.392	0.262	0.328	0.054	-.218	-.254
C12	-.129	-.180	0.211	0.107	0.136	0.203	0.572
C13	0.146	0.161	-.179	-.034	-.084	-.179	-.594
C14	0.311	0.128	0.044	0.323	0.041	-.007	0.012

EIGENVALUES: the eigenvalues of the correlation or covariance matrix.
PROPORTION: the proportion of variance explained by each eigenvalue.
CUMULATIVE: the cumulative of variance explained

Table 4

Performance Comparison of 1x9 and 1x14 Input Code

	Recognition Rate		Memory Space		Learning Speed		Recognition Speed	
	H-H1	H-H2	H-H1	H-H2	H-H1	H-H2	H-H1	H-H2
1X14	84%	86%	6	34	492	1211	186	355
1X9	56%	80%	104	26	3689	1052	553	359

Behavior of Model H-H1

When observing the output patterns of Model H-H1, it can be seen that each element of the output vector is affected by half of the elements in the input vector. For example, the output element O_5 is a function of the input elements I_1, I_3, I_5, I_7, and I_9. Figure 13 illustrates this behavior.

For the purpose of illustration, an example of the program output from Model H-H1, where the size of the memory matrix was 8x14, Smin was -15, Smax was 15, Mmin was 1, and Mmax was 4, is presented in Table 5. In this example, 78 letters (three fonts for each alphabetic character) were submitted to the model as the training set and all 156 letters were used as the testing data. As Table 5 indicates, the overall recognition rate was 84%. The recognition rate for the 78 trained letters was 100%, and 68% for the untrained letters. The depth of the recognition tree was 2. There were 6, 8x14 memory matrices in the tree. The learning speed was 492 iterations and the recognition speed was 186. The output also shows the number of correctly recognized letters and the number of rejected letters for each letter and for each font.

A total of 36 sets of computer output were generated from Model H-H1, using the following model parameters: (1) memory matrix sizes used were 4x14, 6x14, 8x14, and 10x14; (2) Smin_Smax values of $\pm$12, $\pm$15, and $\pm$18 were used; and (3)

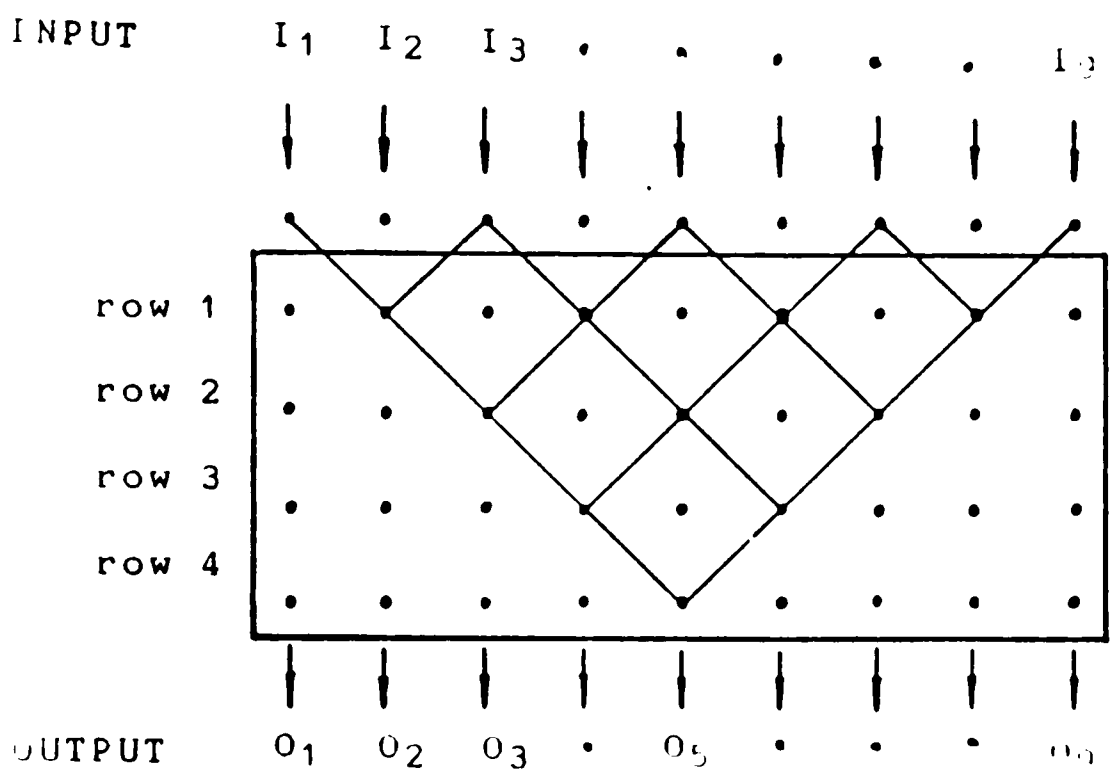


Figure 13: Input and Output Relationships of Model H-H1

Table 5

An Example Output (Model H-H1)

	Correct Letters	Recognition Rate	Rejected Letters	Rejection Rate
Trained	78	100%	0	0
Untrained	53	68%	22	28%
Total	131	84%	22	14%

Depth of the decision tree = 2

Number of memory matrices used = 6

Learning speed = 492

Recognition speed = 186

Alphabet	A	B	C	D	E	F	G	H	I	J	K	L	M
Correct	5	3	3	6	6	6	3	6	6	5	6	6	5
Rejected	1	3	3	0	0	0	2	0	0	1	0	0	1

Alphabet	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Correct	6	5	6	4	4	3	6	6	5	5	4	5	6
Rejected	0	1	0	2	2	3	0	0	1	0	2	0	0

	Courier	New York	Chicago	Geneva	Times	Venice
# of trained	17	12	16	12	7	14
# of correct	24	24	23	20	22	18
# of rejected	2	2	2	6	4	6

the Mmax values were 4, 6, and 8 and Mmin was fixed at 1. Table 6 lists the resulting recognition rates, the number of memory matrices used, the learning speeds, and the recognition speeds.

Across all variables, the mean recognition rate was 82.15% (STD=2.31%); it ranged from 77.56% to 87.82%. On the average, 12.86 memory matrices were needed in the learning phase; the number ranged from 2 to 23. The mean learning speed was 707.14 (STD=215.79) iterations with the highest speed at 258 iterations and the speed at 1143 iterations. As for recognition speed, the mean was 216 (STD=26.15) and the range was 161 to 263. Overall, the program averaged 5.385 seconds of CPU time on the VAX 8650.

The ANOVA procedures were used to analyze the effects of varying the parameter levels on the performances of the models. Table 7 illustrates the ANOVA results. In this analysis, the dependent variables are recognition rate, memory space used, learning speed, and recognition speed. The independent variables were memory matrix size, Smin_Smax, and Mmin_Mmax.

Most of the variation in recognition rate was explained by the variable Mmin_Mmax (Table 7 (a)). The classifications of memory size and Smin_Smax made little contribution. A total of 32.68% of the variations were explained by these three variables. The Least Significant Difference (LSD) test showed no statistical difference in

Table 6

Model Performance (Model H-H1; Trained 3 Fonts)

Memory-Size	Smin_Smax	Mmax	Recognition Rate	Memory Space	Learning Speed	Recognition Speed
4x14	12	4	82.05	4	306	178
4x14	12	6	82.69	9	525	214
4x14	12	8	83.97	8	510	210
4x14	15	4	77.56	3	369	166
4x14	15	6	82.05	6	594	184
4x14	15	8	83.97	6	522	185
4x14	18	4	81.41	2	258	161
4x14	18	6	81.41	7	543	194
4x14	18	8	80.13	6	750	195
6x14	12	4	81.41	7	401	194
6x14	12	6	84.61	9	609	205
6x14	12	8	83.97	14	739	241
6x14	15	4	78.85	10	566	209
6x14	15	6	78.20	9	693	204
6x14	15	8	83.97	14	697	229
6x14	18	4	78.20	14	568	222
6x14	18	6	80.77	10	638	217
6x14	18	8	85.90	11	733	214
8x14	12	4	80.13	21	834	231
8x14	12	6	83.33	17	857	213
8x14	12	8	83.97	14	752	233
8x14	15	4	83.97	6	492	186
8x14	15	6	81.41	14	728	198
8x14	15	8	83.97	23	965	235
8x14	18	4	87.82	16	751	211
8x14	18	6	81.41	23	1085	251
8x14	18	8	81.41	23	1097	249

Table 6

(Continued)

Memory-Size	Smin_ Smax	Mmax	Recognition Rate	Memory Space	Learning Speed	Recognition Speed
10x14	12	4	80.77	21	852	220
10x14	12	6	84.61	7	891	213
10x14	12	8	84.61	14	816	233
10x14	15	4	80.13	15	771	245
10x14	15	6	81.41	12	672	209
10x14	15	8	82.69	19	945	251
10x14	18	4	82.05	16	798	263
10x14	18	6	78.85	20	987	254
10x14	18	8	83.93	23	1143	259

Unit: Recognition rate= % of recognition set
Memory space= no. of memory matrix
Learning speed= no. of iterations
Recognition speed= no. of times

Table 7

Analysis of Variance of Performance Data of Model H-H1

(a) Recognition Rate

SOURCE	DF	ANOVA SS	F VALUE	PR > F
Memory Matrix Size	3	10.4569	0.78	0.5172
Smin_Smax	2	14.2208	1.58	0.2232
Mmax	2	36.3734	4.05	0.0285

(b) Memory Space

SOURCE	DF	ANOVA SS	F VALUE	PR > F
Memory Matrix Size	3	878.9722	23.83	0.0001
Smin_Smax	2	48.2222	1.96	0.1595
Mmax	2	66.8889	2.72	0.0832

(c) Learning Speed

SOURCE	DF	ANOVA SS	F VALUE	PR > F
Memory Matrix Size	3	909166.5278	27.53	0.0001
Smin_Smax	2	93853.7222	4.26	0.0242
Mmax	2	318565.3889	14.47	0.0001

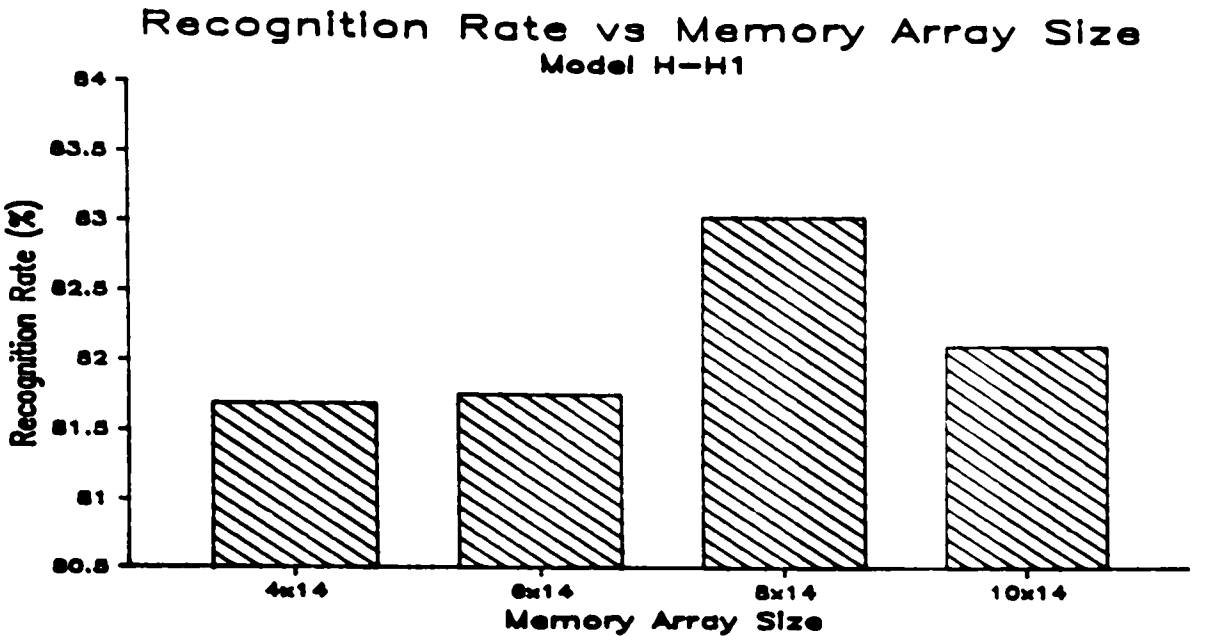
(d) Recognition Speed

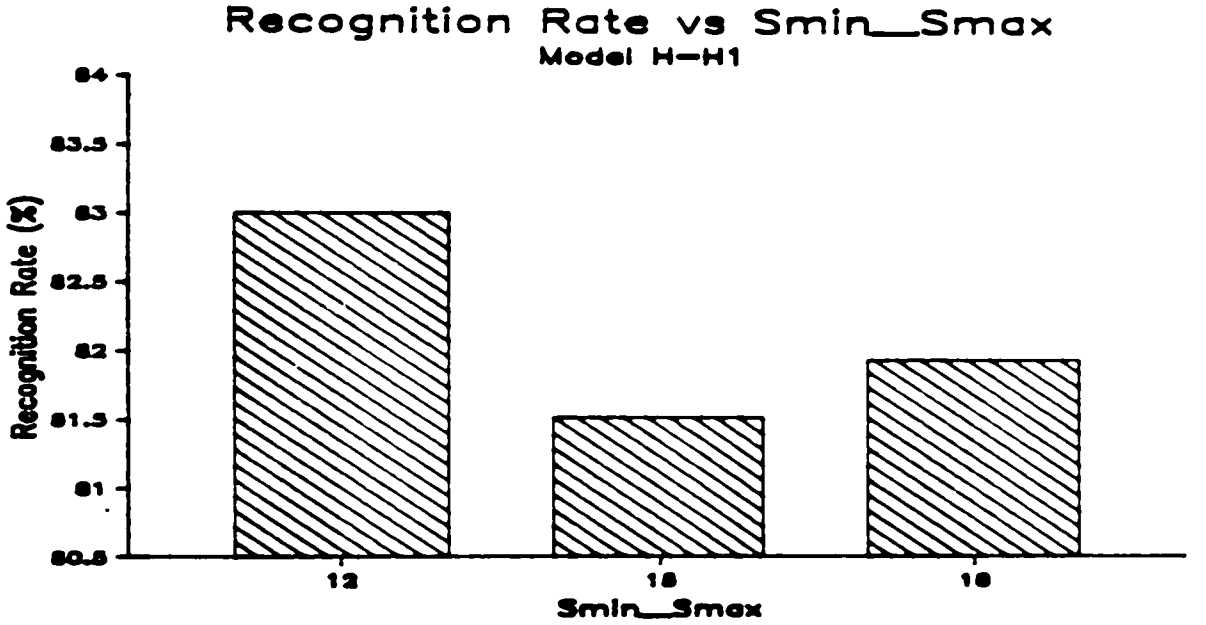
SOURCE	DF	ANOVA SS	F VALUE	PR > F
Memory Matrix Size	3	12367.5556	15.72	0.0001
Smin_Smax	2	1494.5000	2.85	0.0748
Mmax	2	2724.6667	5.19	0.0121

DF: degree of freedom
ANOVA SS: the sum of squares
F VALUE: F statistic
PR: the probability value associated with the F VALUE.

recognition rate as a result of using the four memory matrix sizes and the three Smin_Smax values. The third level of Mmax, which was 8 (mean=83.54%), has a statistically higher recognition rate when compared to Mmax equal to 4 (mean=81.20%) and 6 (mean=81.72%). This behavior suggests that the value of Mmax may have a significant impact on the recognition rate. Figure 14 illustrates the recognition rate as a function of memory matrix size, Smin_Smax, and Mmin_Mmax, respectively.

Table 7 (b) shows that memory matrix sizes, Smin_Smax, and Mmax can account for a 74.28% (R-Square) variation in memory space used by this model. Most of the variation was explained by the variable memory size; Smin_Smax and Mmax had little influence on the number of memory matrices needed. The LSD test showed that as the memory matrix size increased, the model needed more memory matrices to learn the letters. In other words, the larger the memory matrix was, the less discriminating the model was. For the 8x14 and the 10x14 memory matrix sizes, the mean number of memory matrices needed was 17.44; for the 6x18 memory size, the mean was mean was 10.89; for the 4x14 memory size, only 5.67 memory matrices were needed. The different values of Smin_Smax did not make a statistically significant difference in the number of memory matrices needed. The larger the value of Mmax was, the more memory matrices were needed. This suggests that the larger the value of Mmax,





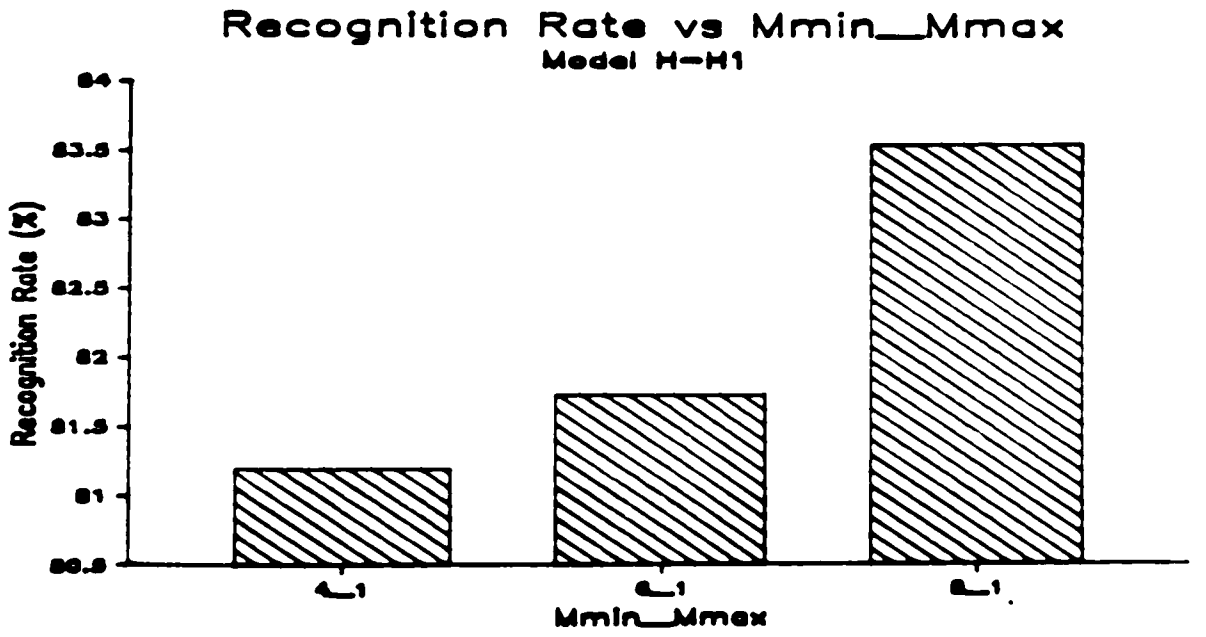
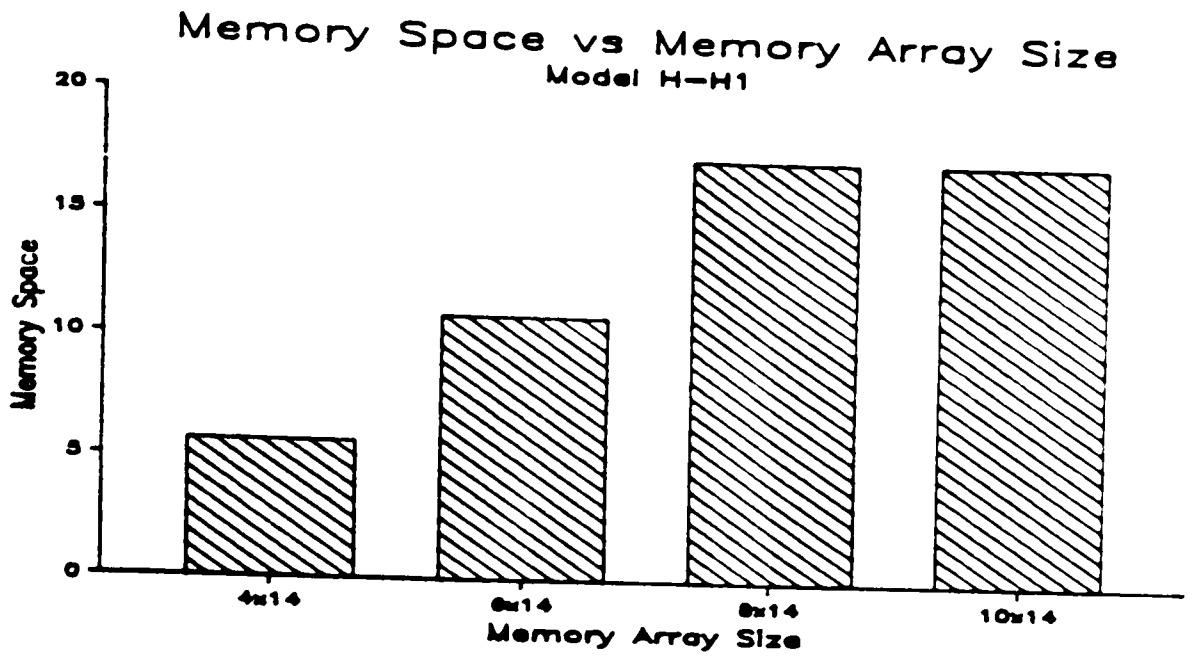


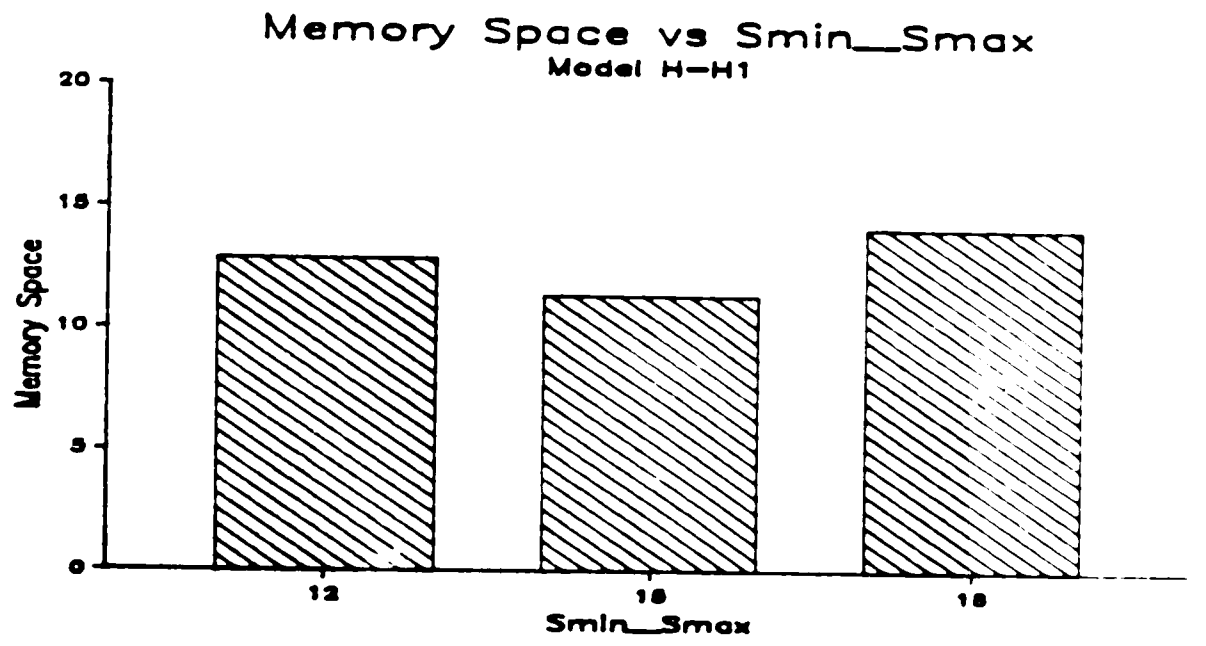
Figure 14: Recognition Rate as a Function of Memory Matrix Size, Smin_Smax, and Mmin_Mmax (Model H-H1)

the less discriminating the model was. Figure 15 shows the memory space as a function of memory matrix size, Smin_Smax, and Mmin_Mmax, respectively.

The statistical results show that 81.09% of the variation in learning speed was accounted for by the variations in memory matrix size, Smin_Smax, and Mmax (Table 7 (c)). Memory matrix size and the value of Mmax explained most of the variations in learning speed. As the memory matrix size increased, more iterations were needed in the learning phase. In other words, it took more time to build the decision tree. Also, the larger the magnitude of Smin_Smax, the slower the learning speed was. This was due to the fact that the larger the magnitude of Smin_Smax, the larger the range to be adjusted. By the same token, the larger the value of Mmax was, the slower the learning speed was. Figure 16 illustrates the learning speed as a function of memory matrix size, Smin_Smax, and Mmin_Mmax, respectively.

For the recognition speed, 69.31% of the variation was accounted for by the size of the memory matrix, Smin_Smax, and Mmax value (Table 7 (d)). Of these three independent variables, the size of the memory matrix accounted for most of the variation. The LSD test showed that the larger the memory matrix size, the more times the input data needed to be submitted to the memory matrices during the recognition phase. This resulted in a slower recognition speed. It





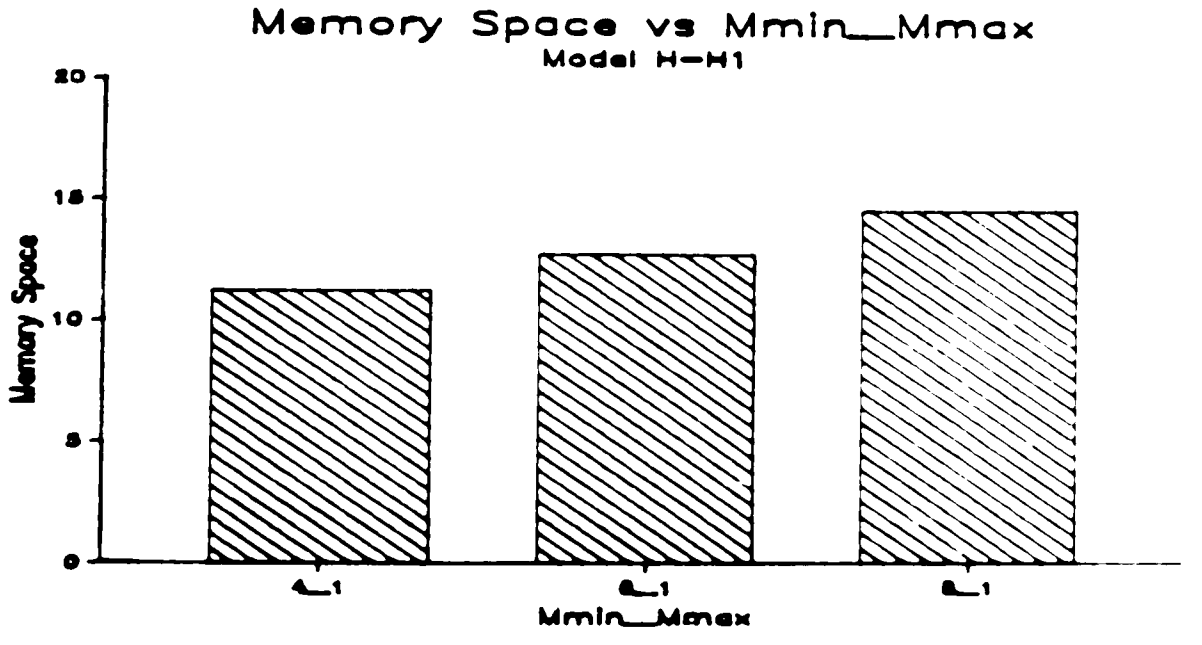
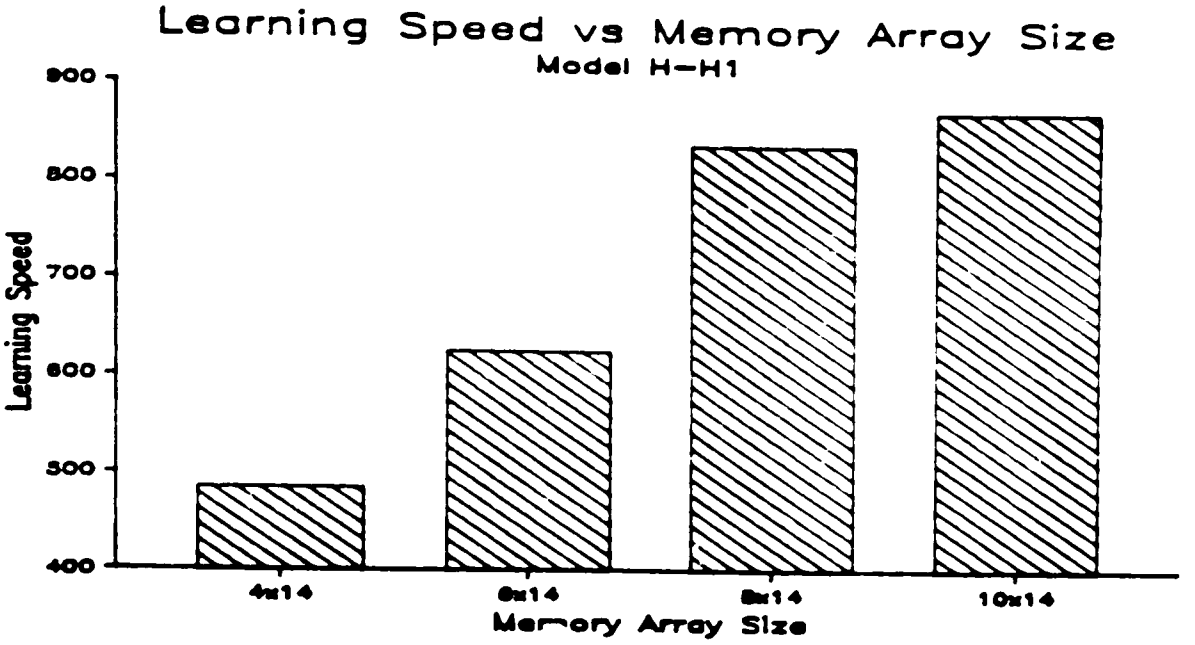
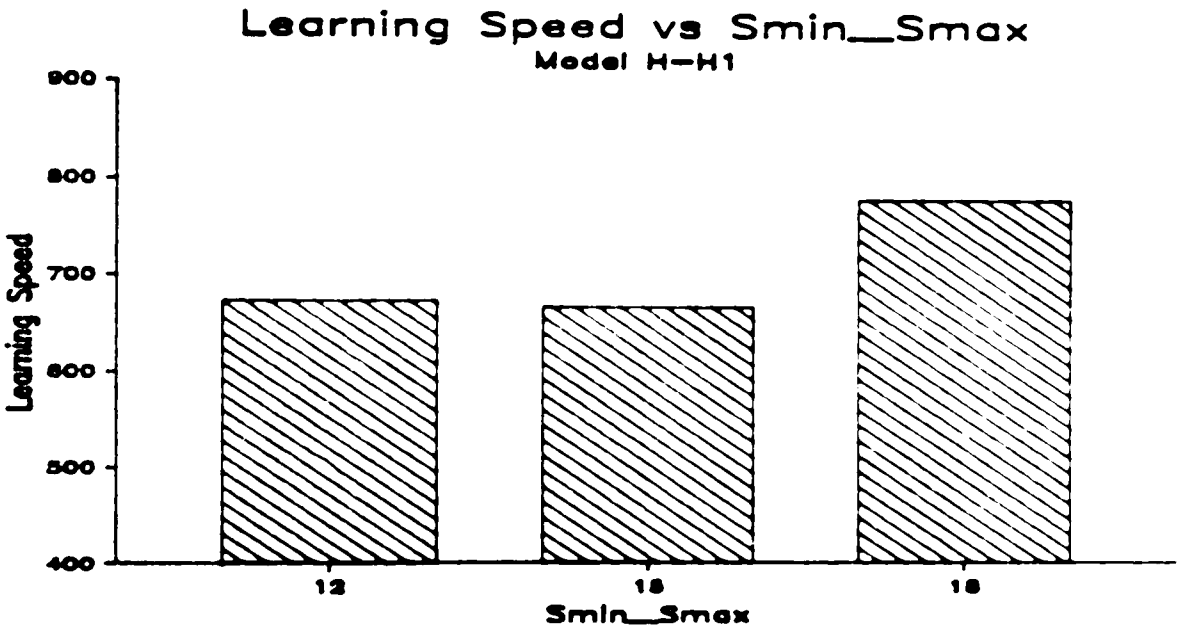


Figure 15: Memory Space as a Function of Memory Matrix Size, Smin_Smax, and Mmin_Mmax (Model H-H1)





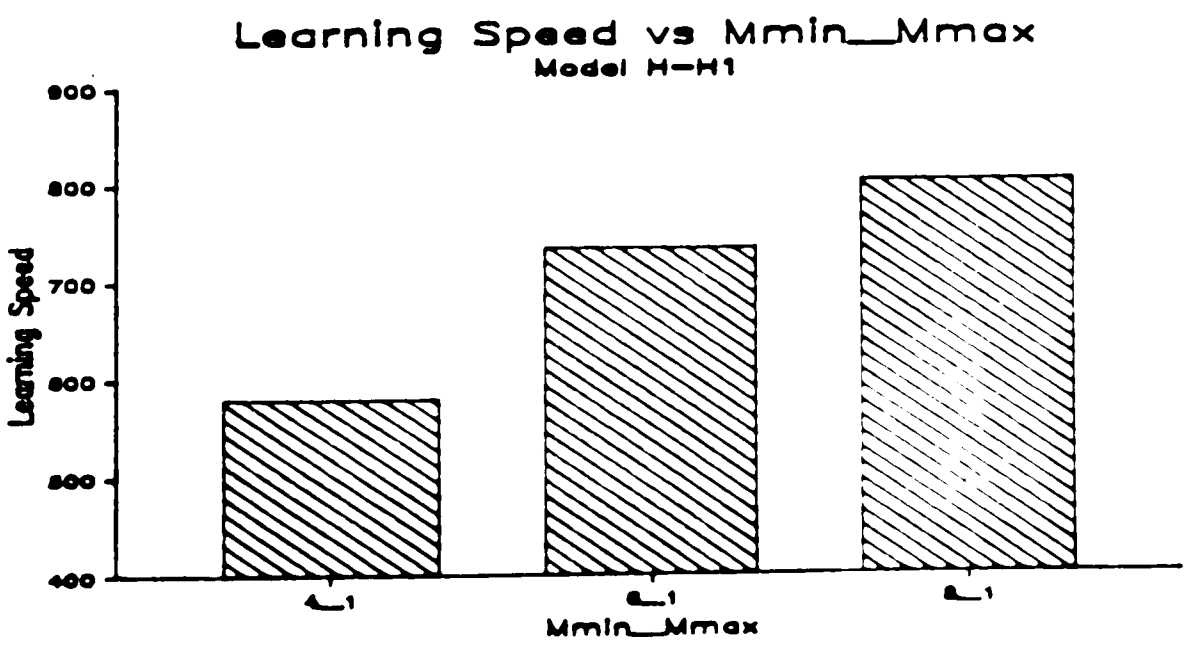
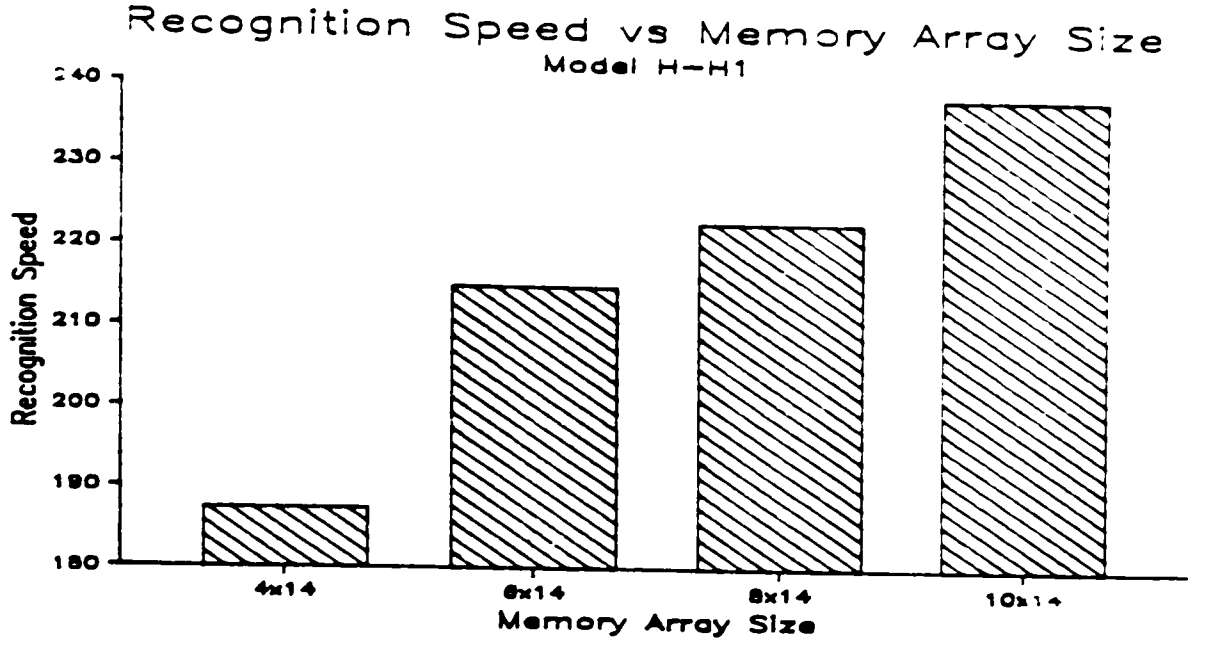


Figure 16: Learning Speed as a Function of Memory Matrix Size, Smin_Smax, and Mmin_Mmax (Model H-H1)

also showed that the larger the Smin_Smax, the slower the recognition speed. Figure 17 illustrates the recognition speed as a function of memory matrix size, Smin_Smax and Mmin_Mmax, separately.

For comparison purposes, other computer simulations were conducted using all of the letters (156) as the training and the recognition set. Table 8 presents the 36 observations of Model H-H1 for recognition rate, memory space used, learning speed, and recognition speed. The mean recognition rate (STD= 2.59%) was 96.07% with a maximum of 100% and minimum of 89.1%. For number of memory matrices used, the mean was 37.17 (STD=25.75) and the range was from 8 to 104. The mean for learning speed was 2937 iterations (STD=1794.03) with the highest speed at 852 iterations and the lowest speed at 8148 iterations. The mean recognition speed was 255.86 (STD=40.11) times; the range was from 194 to 340 times.

Analysis of variance was used to analyze the data again. The result showed that the smallest memory matrix size (4x14) provided the highest recognition rate, the least number of memory matrices, the fastest learning speed, and the fastest recognition speed. This behavior was totally different from that when only three fonts were used in the training set.



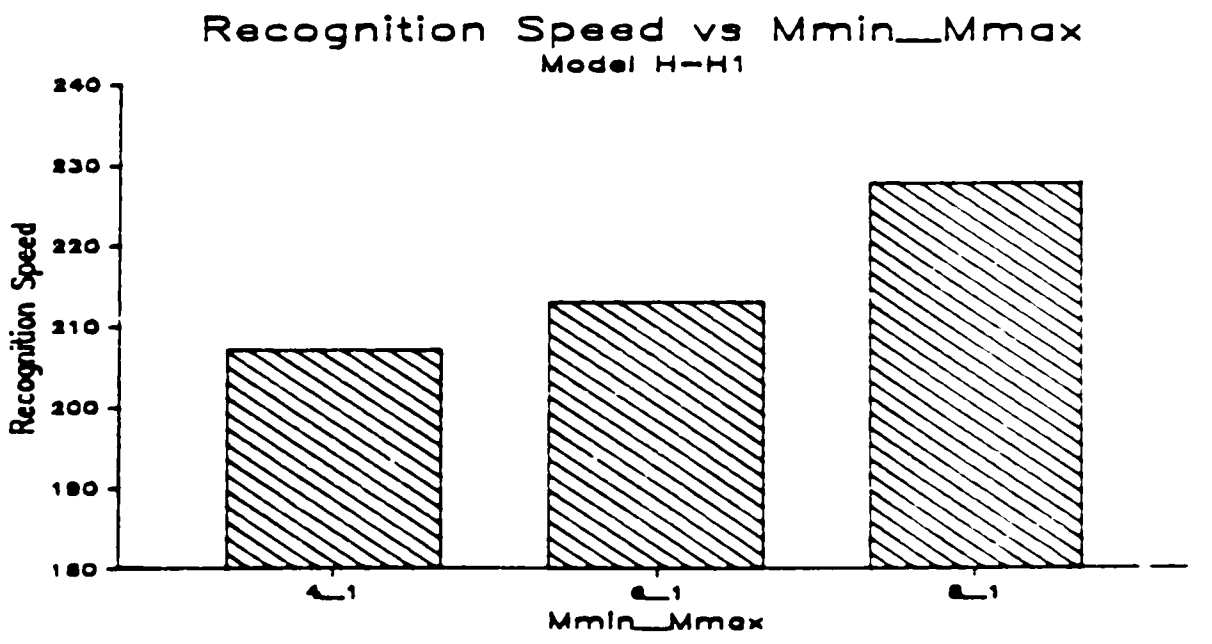


Figure 17: Recognition Speed as a Function of Memory Matrix Size, Smin_Smax, and Mmin_Mmax (Model H-H1)

Table 8

Model Performance (Model H-H1; Trained 6 Fonts)

Memory-Size	Smin_Smax	Mmax	Recognition Rate	Memory Space	Learning Speed	Recognition Speed
4x14	12	4	98.72	19	1461	205
4x14	12	6	97.43	18	1410	224
4x14	12	8	98.72	14	1458	201
4x14	15	4	99.36	17	1299	208
4x14	15	6	97.73	48	2940	243
4x14	15	8	98.72	15	1587	205
4x14	18	4	100.00	8	852	194
4x14	18	6	98.08	20	1680	228
4x14	18	8	98.72	16	1494	214
6x14	12	4	96.15	33	2163	272
6x14	12	6	95.51	28	2042	280
6x14	12	8	97.43	16	1736	241
6x14	15	4	97.43	27	2157	252
6x14	15	6	95.51	29	2089	276
6x14	15	8	97.43	17	1831	245
6x14	18	4	98.72	16	1424	223
6x14	18	6	96.79	27	2103	269
6x14	18	8	98.72	14	1594	217
8x14	12	4	98.74	96	7158	338
8x14	12	6	92.95	60	4392	302
8x14	12	8	94.23	21	2085	260
8x14	15	4	90.38	80	5630	332
8x14	15	6	92.95	61	4411	310
8x14	15	8	94.87	14	1760	239
8x14	18	4	89.10	60	4350	340
8x14	18	6	94.23	98	7304	284
8x14	18	8	96.79	23	2513	240
10x14	12	4	91.67	104	8148	305
10x14	12	6	94.87	45	3303	243
10x14	12	8	96.79	48	3312	239
10x14	15	4	94.23	56	4008	301
10x14	15	6	94.87	45	3651	256
10x14	15	8	96.79	45	3513	236
10x14	18	4	93.59	30	2490	302
10x14	18	6	95.51	49	3825	253
10x14	18	8	94.87	22	2574	234

Behavior of Model H-H2

Model H-H2 behaves in such a way that when the memory values are fixed to zeroes and there are no zero valued elements in the input vector, the output can be easily predicted without going through the computation. The behavior is demonstrated in Figure 18. Since the output values are saturated very fast (it takes $\log_2$(Smax) rows to saturate), the magnitudes of the output elements are the same as Smax. Their signs are determined by the size of the memory matrix and the signs of the input elements. To be specific, when the memory matrix contains an even number of rows, each output element has the same sign as its corresponding input element and a magnitude equal to Smax; when the matrix has an odd number rows, the signs of the output elements are the result of the right-shift of the signs of their corresponding input elements.

The output from program for Model H-H2 provided the same information as the output from Model H-H1. Table 9 presents an example of the program output, when the size of the memory matrix was 8x14, Smin was -15, Smax was 15, Mmin was -10, and Mmax was 10. In this example, the overall recognition rate was 86% of the recognition set. The recognition rate for the 78 training letters was 100%, and the recognition rate for the other letters was 72%. The depth of the decision tree was 6 and there were 34, 8x14 memory matrices in the tree. The learning speed was 1211

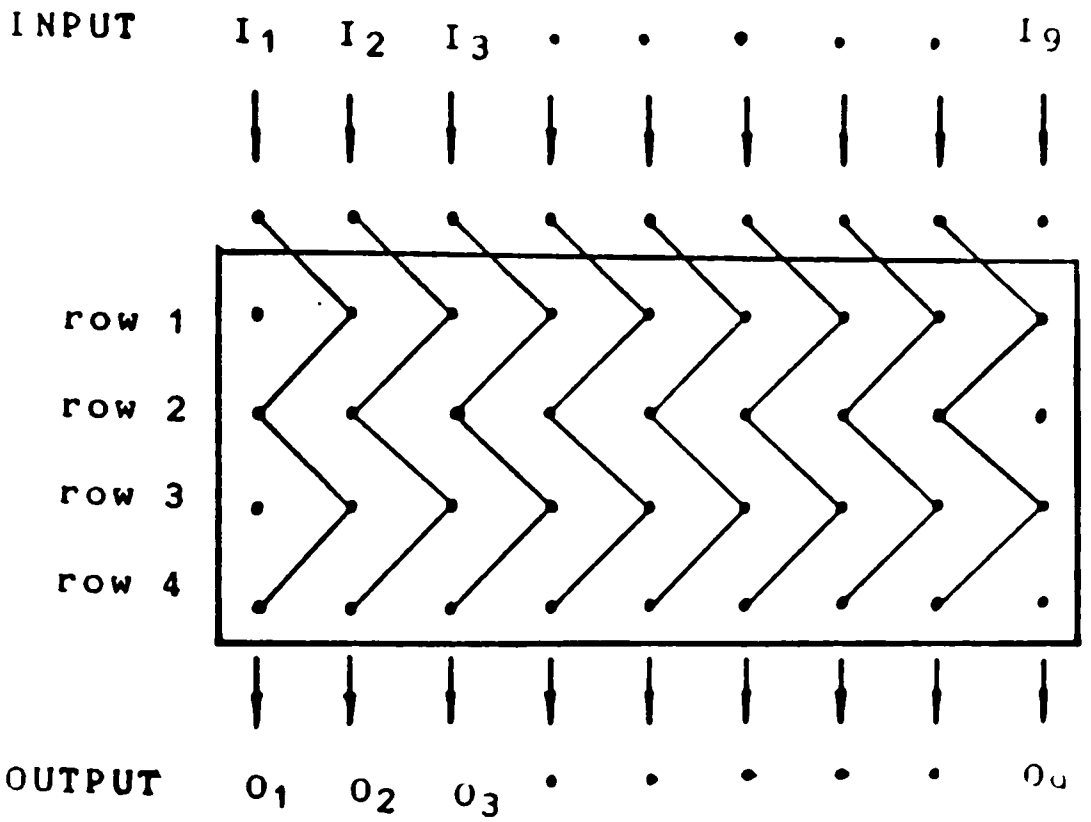


Figure 18: Input and Output Relationships of Model H-H2

Table 9

An Example Output (Model H-H2)

	Correct Characters	Recognition Rate	Rejected Characters	Rejection Rate
Trained	78	100%	0	0
Untrained	56	72%	22	28%
Total	134	86%	22	14%

Depth of the decision tree = 6

Number of memory matrices used = 34

Learning speed = 1211

Recognition speed = 355

Alphabet	A	B	C	D	E	F	G	H	I	J	K	L	M
Correct	5	5	5	6	6	6	4	6	5	5	4	6	5
Rejected	1	1	1	0	0	0	2	0	1	1	2	0	1

Alphabet	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Correct	6	5	6	4	5	4	6	4	4	3	6	5	6
Rejected	0	1	0	2	1	2	0	1	1	3	0	1	0

	Courier	New York	Chicago	Geneva	Times	Venice
# of trained	17	12	16	12	7	14
# of correct	23	22	24	24	23	18
# of rejected	3	4	2	2	3	8

iterations and recognition speed was 355. The Table also shows the number of correctly recognized characters and rejected characters for each letter and for each font.

Thirty-six sets of computer output were generated using Model H-H2, using the following values for the three variables. The sizes of the memory matrix were 4x14, 6x14, 8x14, and 10x14; Smin_Smax took on values of ±12, ±15, and ±18; and Mmin_Mmax was set equal to ±8, ±10, and ±12. Again the performance of the model was measured by recognition rate, memory space needed, learning speed, and recognition speed. Table 10 lists the results.

Overall, the mean recognition rate was 83.67% (STD=4.17%) of the recognition set. It ranged from 72.43% to 89.10%. On average, 13.17 memory matrices were needed in the learning phase; this ranged from 4 to 29. The mean learning speed was 654.81 (STD=255.75) iterations with the highest speed at 316 iterations and the lowest speed at 1418 iterations. For recognition speed, the mean was 283.67 times (STD=38.46); it ranged from 194 to 346. The average CPU time was 6.47 seconds for Model H-H2 to complete the recognition process on the VAX 8650.

Table 11 illustrates the results of an analysis of variance of recognition rate, memory space used, learning speed, and recognition speed, respectively. According to the statistical analysis, 66.23% of the variations of recognition rate were accounted for by the three independent

Table 10

Model Performance (Model H-H2; Trained 3 Fonts)

Memory-Size	Smin_ Smax	Mmin_ Mmax	Recognition Rate	Memory Space	Learning Speed	Recognition Speed
4x14	12	8	76.92	10	431	273
4x14	12	10	80.13	6	327	238
4x14	12	12	75.00	9	427	270
4x14	15	8	78.20	5	392	194
4x14	15	10	76.92	11	356	224
4x14	15	12	83.97	5	316	204
4x14	18	8	78.85	7	397	203
4x14	18	10	79.49	7	492	273
4x14	18	12	81.41	4	369	249
6x14	12	8	85.26	14	541	302
6x14	12	10	83.97	17	687	306
6x14	12	12	72.43	4	365	287
6x14	15	8	85.26	15	643	270
6x14	15	10	84.61	6	552	289
6x14	15	12	86.54	16	805	346
6x14	18	8	82.69	13	624	280
6x14	18	10	82.05	12	577	331
6x14	18	12	81.41	11	621	303
8x14	12	8	87.82	19	751	297
8x14	12	10	85.26	27	964	325
8x14	12	12	85.90	26	943	315
8x14	15	8	87.18	23	785	327
8x14	15	10	89.10	21	967	312
8x14	15	12	83.33	6	504	295
8x14	18	8	86.54	12	625	288
8x14	18	10	86.54	18	805	299
8x14	18	12	88.46	26	1227	324

Table 10

(Continued)

Memory-Size	Smin_ Smax	Mmin_ Mmax	Recognition Rate	Memory Space	Learning Speed	Recognition Speed
10x14	12	8	88.46	9	604	248
10x14	12	10	85.26	13	782	319
10x14	12	12	83.97	6	648	296
10x14	15	8	89.10	9	525	240
10x14	15	10	86.54	8	594	299
10x14	15	12	85.90	22	1033	313
10x14	18	8	87.82	11	600	250
10x14	18	10	87.18	17	876	320
10x14	18	12	82.69	29	1418	303

Unit: Recognition rate= % of recognition set
Memory space= no. of memory matrix
Learning speed= no. of iterations
Recognition speed= no. of times

Table 11

Analysis of Variance of Performance
Data of Model H-H2

(a) Recognition Rate

SOURCE	DF	ANOVA SS	F VALUE	PR > F
Memory Matrix Size	3	350.948	15.93	0.0001
Smin_Smax	2	28.899	1.97	0.1586
Mmin_Mmax	2	23.337	1.59	0.2219

(b) Memory Space

SOURCE	DF	ANOVA SS	F VALUE	PR > F
Memory Matrix Size	3	739.0000	6.74	0.0015
Smin_Smax	2	17.1667	0.23	0.7923
Mmin_Mmax	2	15.1667	0.21	0.8139

(c) Learning Speed

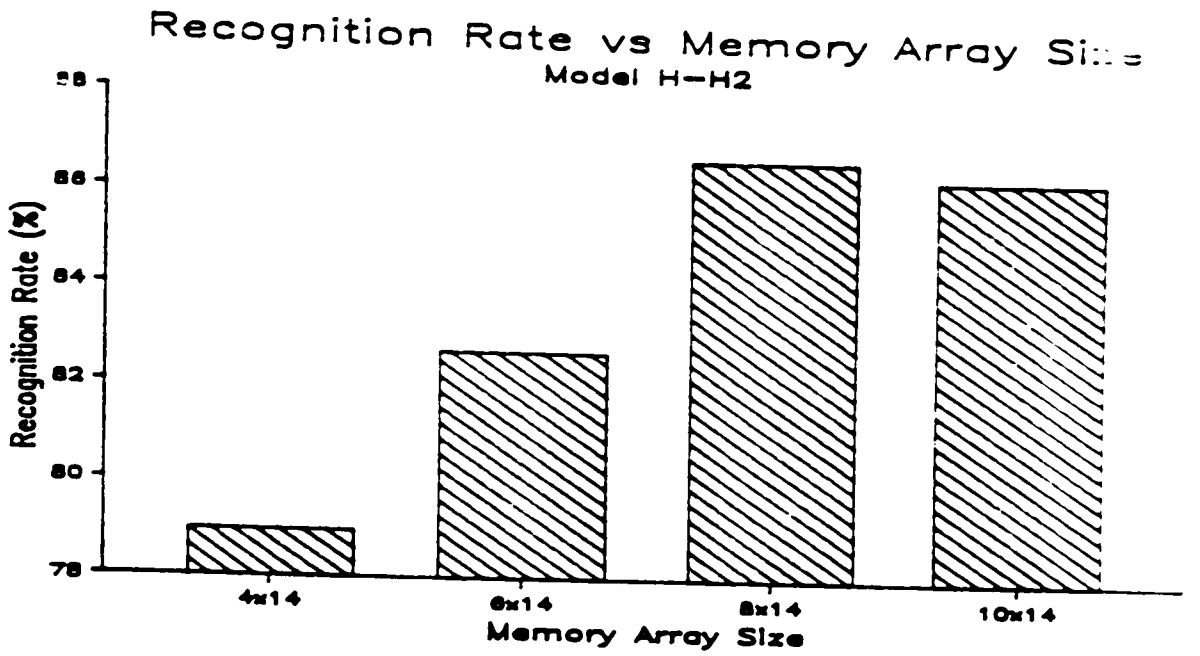
SOURCE	DF	ANOVA SS	F VALUE	PR > F
Memory Matrix Size	3	1127348.0833	11.00	0.0001
Smin_Smax	2	74755.7222	1.09	0.3487
Mmin_Mmax	2	130613.7222	1.91	0.1667

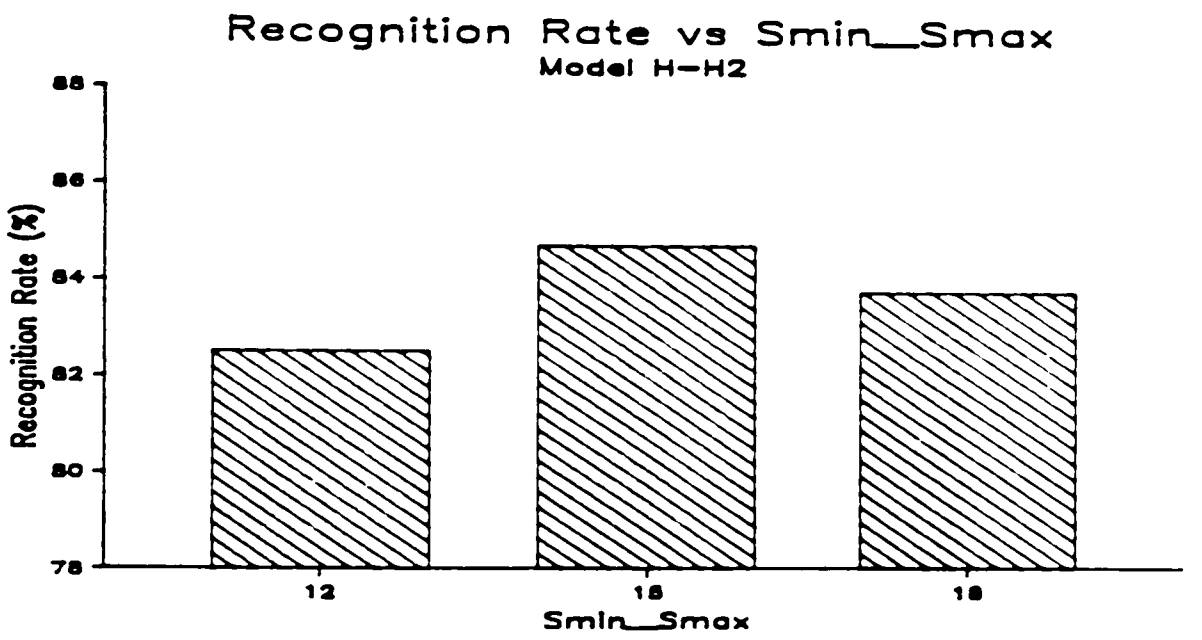
(d) Recognition Speed

SOURCE	DF	ANOVA SS	F VALUE	PR > F
Memory Matrix Size	3	28912.4444	18.04	0.0001
Smin_Smax	2	1152.1667	1.08	0.3539
Mmin_Mmax	2	6765.5000	6.33	0.0054

variables (Table 11 (a)). Most of the variation in recognition rate was explained by memory matrix size. The classifications of Smin_Smax and Mmin_Mmax made little contribution. The LSD test indicated that the recognition rate was (86.68%) higher when the memory size was 8x14 or 10x14 compared to 82.69% and 78.99% with memory matrix sizes of 6x14 and 4x14. On the other hand, the LSD test showed no statistical difference among the three levels of Smin_Smax values and the three levels of Mmin_Mmax values in terms of recognition rate. Figure 19 illustrates the recognition rate as a function of memory matrix size, Smin_Smin, and Mmin_Mmax, respectively.

Table 11 (b) shows that memory matrix sizes, Smin_Smax, and Mmax can account for only 42.97% (R-Sguare) of the variation in memory space used in this model. Most of the variation was explained by the memory matrix size. Smin_Smax and Mmax had little influence on the number of memory matrices needed. The LSD test showed that as the memory matrix size increased, the number of memory matrices used in the learning phase increased with statistical significance. With a memory size of 4x14, the mean number of memory matrices used was only 7.11; with a 10x14 memory matrix, the mean number of memory matrices used increased to 19.78. In other words, the larger the memory matrix was, the less discriminating the model was. The different values of Smin_Smax and Mmin_Mmax made no statistically significant





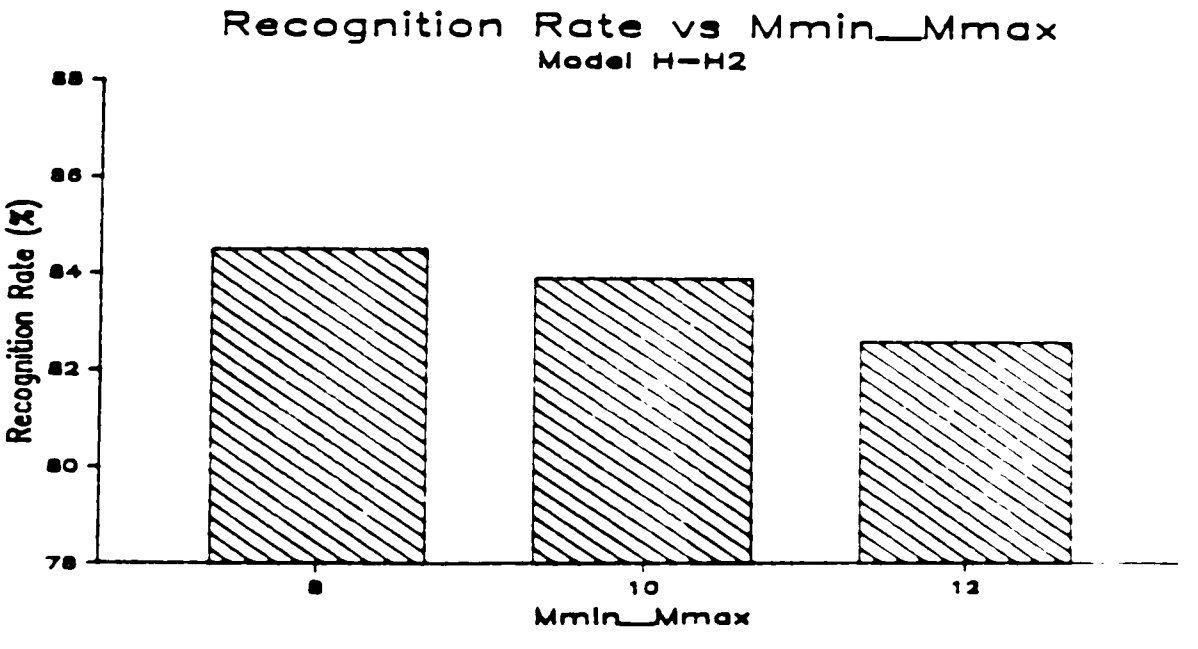
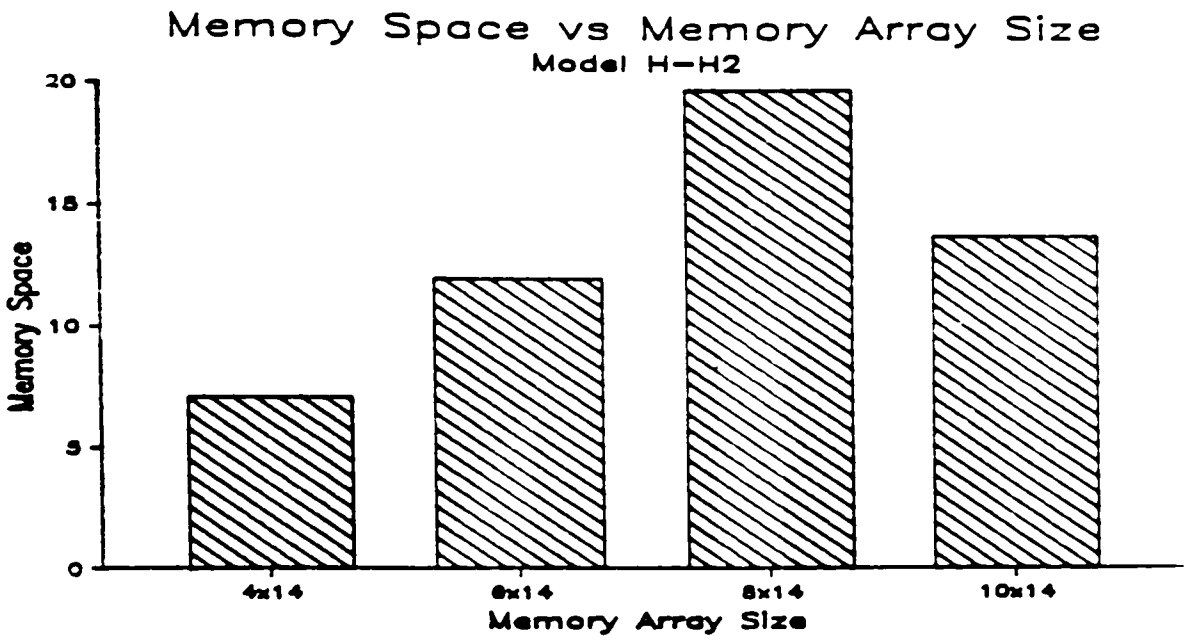


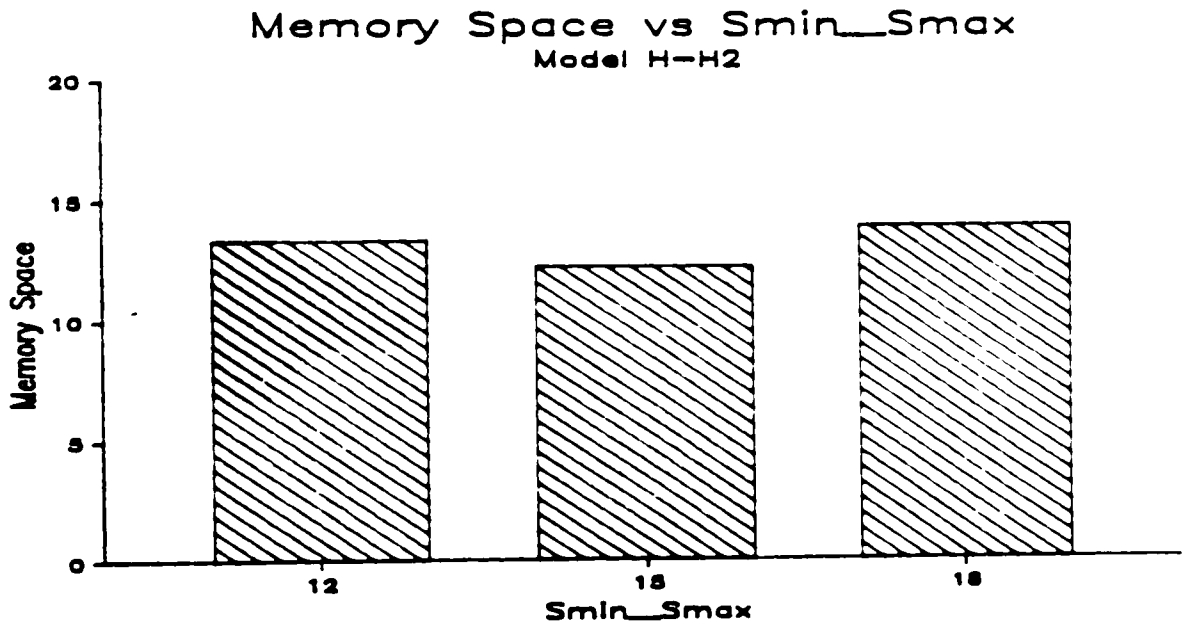
Figure 19: Recognition Rate as a Function of Memory Matrix Size, Smin_Smax, and Mmin_Mmax (Model H-H2)

difference in terms of number of memory matrix needed. Figure 20 illustrates the memory space as a function of memory matrix size, Smin_Smax, and Mmin_Mmax, separately.

The statistical analysis showed that 58.21% of the variation in learning speed could be accounted for by memory matrix size, Smin_Smax, and Mmin_Mmax (Table 11 (c)). Memory matrix size explained most of the variation in the learning speed; the different values of Smin_Smax and Mmin_Mmax made small contributions. The result of the LSD test showed that as the memory matrix size increased more iterations were needed. It took more time to build the decision tree. The levels of Smin_Smax and Mmin_Mmax make no statistically significant difference in the learning speed. Figure 21 illustrates the learning speed as a function of memory matrix size, Smin_Smax and Mmin_Mmax, respectively.

As far as recognition speed was concerned, the greater the number of times needed to recognize the letters, the slower the recognition speed was. The result indicated that 71.11% of the variations in recognition speed were explained by the various sizes of the memory matrix, Smin_Smax, and Mmin_Mmax values (Table 11 (d)). Of these three independent variables, the size of the memory matrix contributed most to the explanation of the variation. The LSD test showed that the larger the memory matrix size, the more times were needed. For the 4x14 memory matrix size, it only took





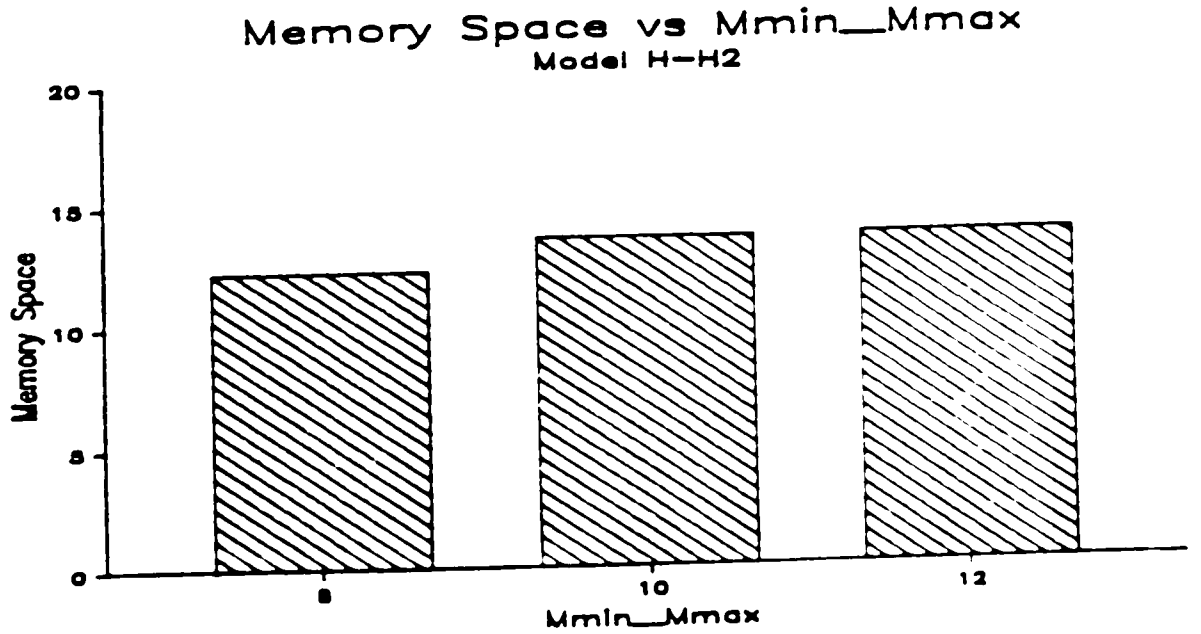
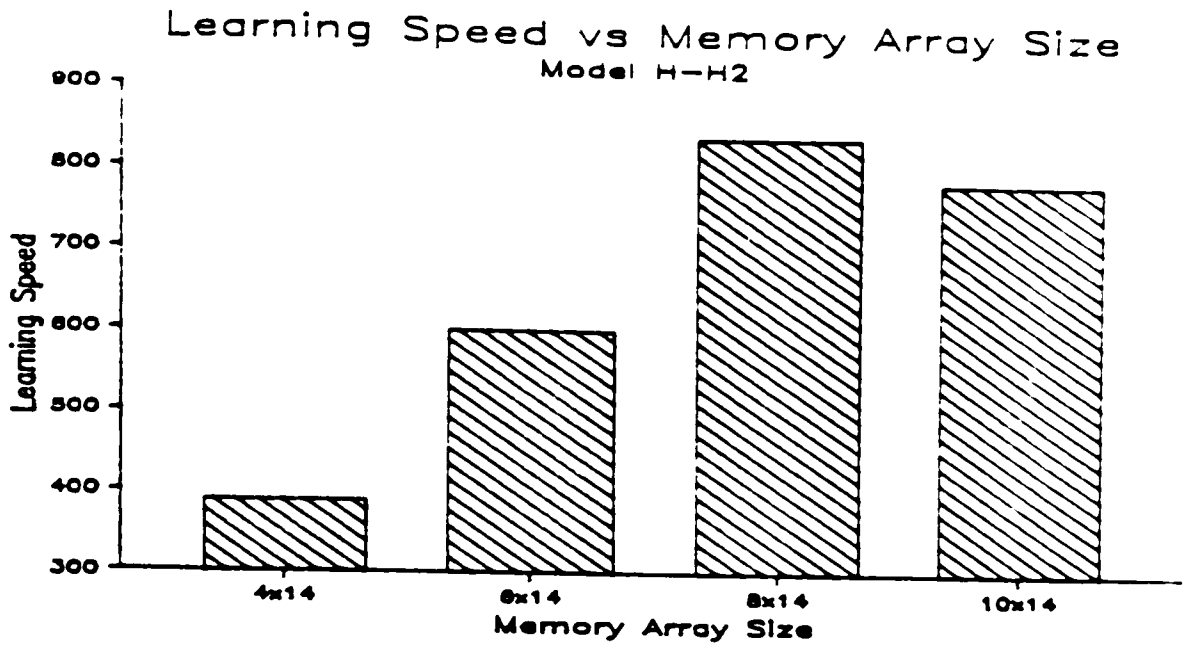
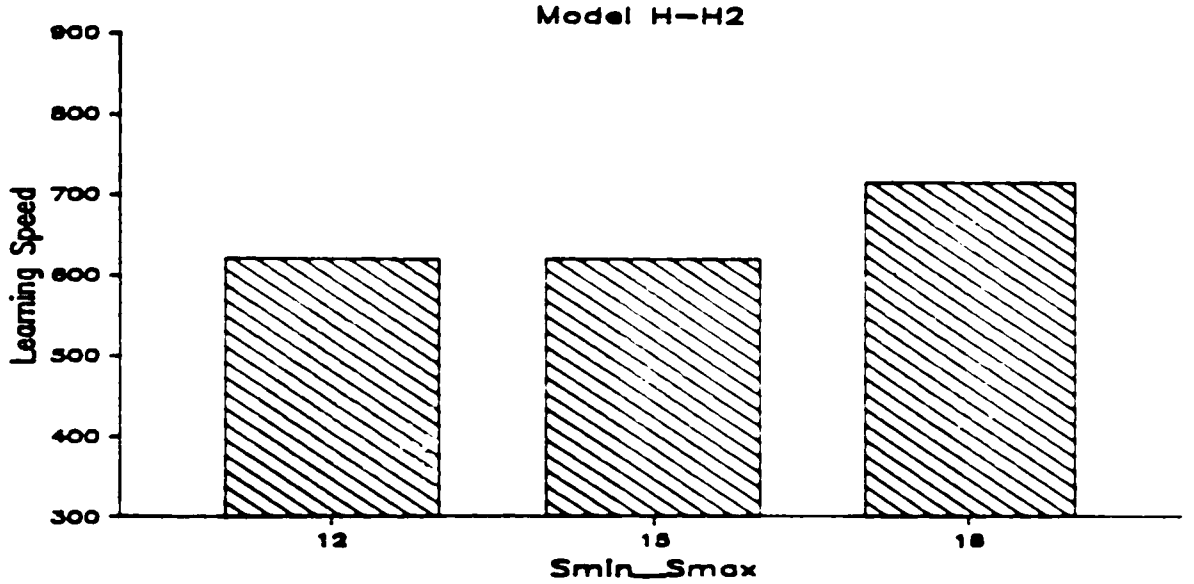


Figure 20: Memory Space as a Function of Memory Matrix Size, Smin_Smax, and Mmin_Mmax (Model H-H2)





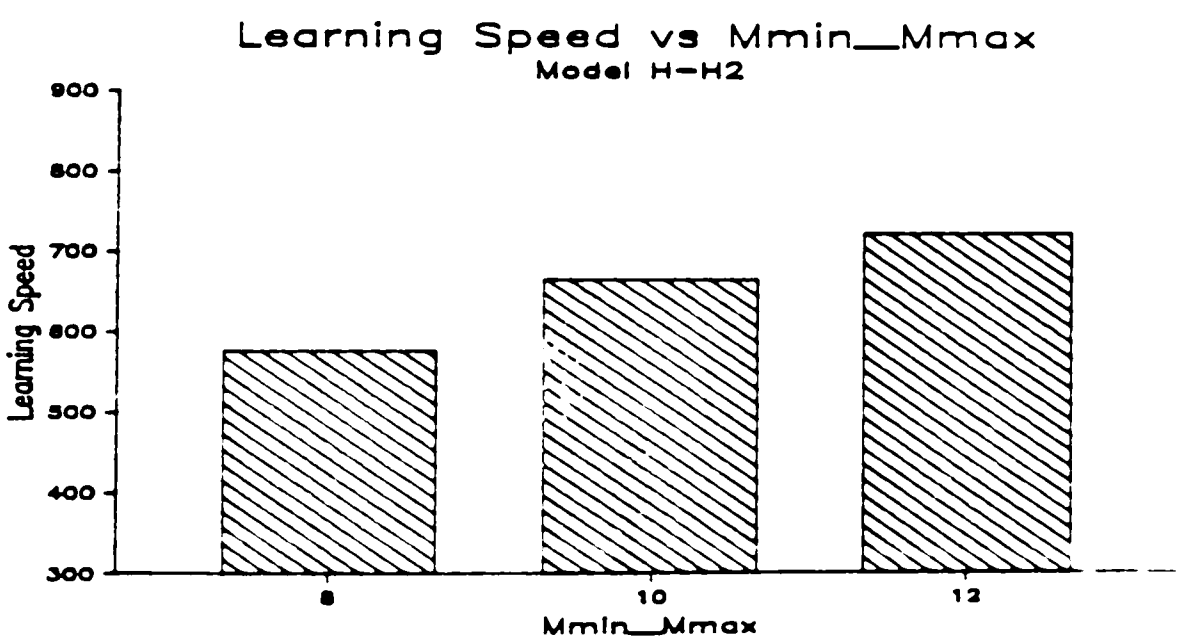
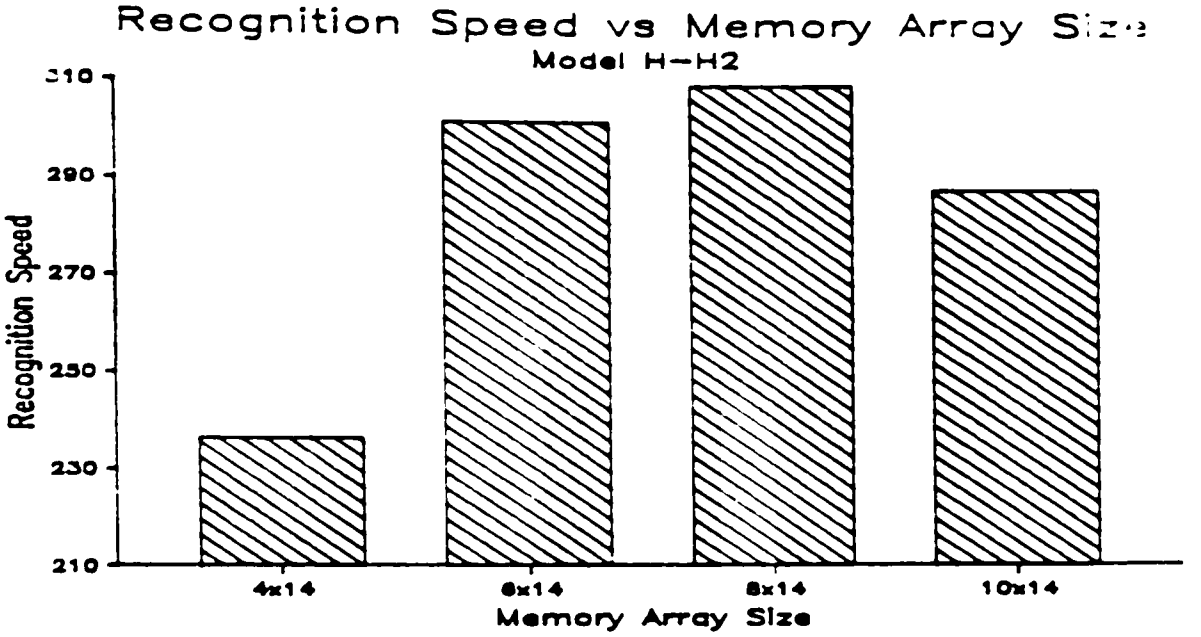


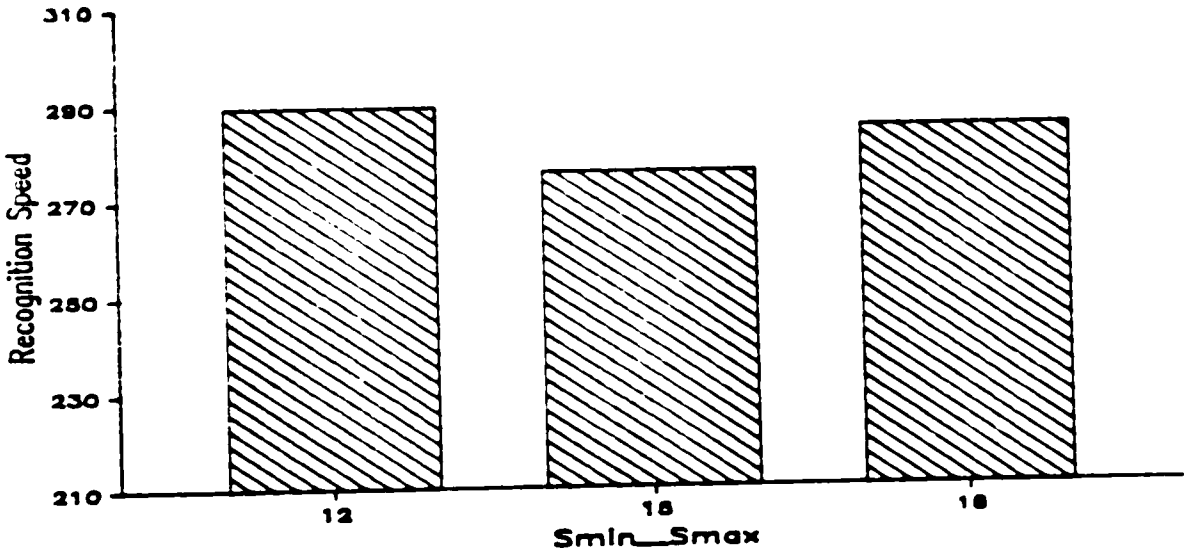
Figure 21: Learning Speed as a Function of Memory Matrix Size, Smin_Smax, and Mmin_Mmax (Model H-H2)

236.44 (on average) times to recognize 156 letters; whereas with 8x14 memory matrix size, it took 309.11 times. It also showed that the larger the magnitude value of Mmin_Mmax was, the slower the recognition speed was. Figure 22 illustrates the recognition speed as a function of memory matrix size, Smin_Smax and Mmin_Mmax, separately.

Table 12 presents the 36 observations of Model H-H2 for recognition rate, memory space used, learning speed, and recognition speed, when all 156 characters were submitted as both the training set and the recognition set. The mean recognition rate was 99.41% (STD=7.24%) with the highest at 100% and lowest at 96.15%. This suggested that learning was perfect only for some parameter combinations. For number of memory matrices used, the mean was 19.11 (STD=8.61); it ranged from 5 to 38. The mean for learning speed was 1285.78 iterations (STD=424.38) with the highest speed at 715 iterations and the lowest speed at 2090 iterations. The mean recognition speed was 309.11 (STD=42.96) times and ranged from 178 to 382.

Analysis of variance was used to analyze the data again. Among all memory matrix sizes, the smallest memory matrix size (4x14) provided the highest recognition rate, the least number of memory matrices needed, and the fastest learning and recognition speeds. This behavior is totally different from the behavior when only three fonts were used as the training set.





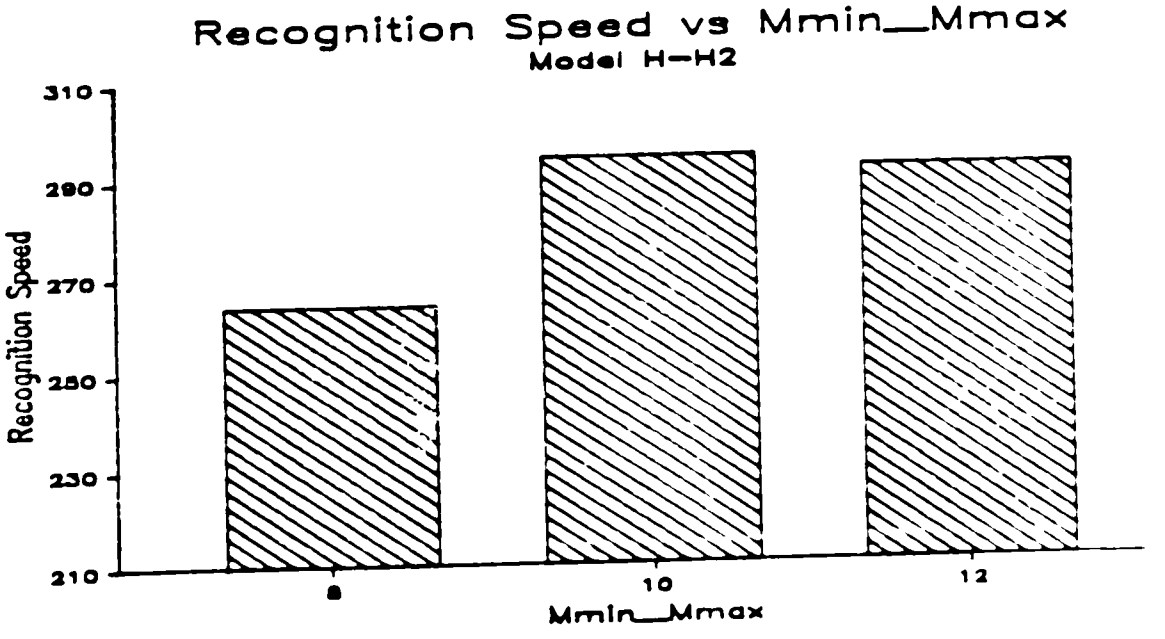


Figure 22: Recognition Speed as a Function of Memory Matrix Size, Smin_Smax, and Mmin_Mmax (Model H-H2)

Table 12

Model Performance (Model H-H2; Trained 6 Fonts)

Memory-Size	Smin_ Smax	Mmin_ Mmax	Recognition Rate	Memory Space	Learning Speed	Recognition Speed
4x14	12	8	100.00	16	824	247
4x14	12	10	100.00	6	715	293
4x14	12	12	100.00	13	718	241
4x14	15	8	100.00	16	746	246
4x14	15	10	100.00	6	794	283
4x14	15	12	99.36	11	820	300
4x14	18	8	100.00	5	746	178
4x14	18	10	100.00	9	732	261
4x14	18	12	100.00	8	717	284
6x14	12	8	100.00	9	865	292
6x14	12	10	99.36	18	1341	325
6x14	12	12	99.36	20	1311	282
6x14	15	8	99.36	17	1235	275
6x14	15	10	99.36	16	1112	309
6x14	15	12	96.15	26	1612	354
6x14	18	8	100.00	20	1155	295
6x14	18	10	100.00	11	852	314
6x14	18	12	99.36	15	1073	330
8x14	12	8	99.36	26	1580	309
8x14	12	10	99.36	30	1839	368
8x14	12	12	99.36	34	2010	374
8x14	15	8	98.72	18	1318	305
8x14	15	10	98.08	38	2090	349
8x14	15	12	99.36	25	1684	376
8x14	18	8	100.00	18	1351	308
8x14	18	10	99.36	26	1603	341
8x14	18	12	99.36	25	1634	336
10x14	12	8	99.36	19	1464	321
10x14	12	10	99.36	14	1379	309
10x14	12	12	99.36	28	2057	361
10x14	15	8	98.72	38	1927	340
10x14	15	10	99.36	27	1691	382
10x14	15	12	98.72	17	1228	315
10x14	18	8	100.00	24	1392	282
10x14	18	10	99.36	24	1436	330
10x14	18	12	99.36	15	1237	313

Model Comparisons

Model H-H1 versus Model H-H2

To distinguish the performance differences between Model H-H1 and Model H-H2, the Pairwised Comparison Test (Pairwised T Test) was conducted. Table 13 lists the performance differences between Model H-H1 and Model H-H2 when using three out of six fonts as the training set. Table 14 presents the test results.

The Pairwised T Test indicated that only the recognition speed had a statistically significant difference when models H-H1 and H-H2 were subjected to the experimental variables in the study. The sign of the mean score indicated that Model H-H2 required more time during the recognition phase. The test results failed to justify statistically that there was any difference in recognition rate, memory space, and learning speed between the two models. In spite of that, Model H-H1 had a lower mean recognition rate and a slower mean learning speed. Model H-H1 required less mean memory space and provided a faster mean recognition speed.

On the other hand, the two models behaved significantly different under when six fonts were used as both the training set and the recognition set. Table 15 lists these performance differences.

The Pairwised T Test (Table 16) illustrated that the performance data of the two models had a statistically

Table 13

Performance Differences of Model H-H1 and Model H-H2 (Trained 3 fonts)

OBS	DIFF1(%)	DIFF2	DIFF3	DIFF4
1	5.13	-6	-125	-95
2	2.56	3	198	-24
3	8.97	-1	83	-60
4	-0.64	-2	-23	-28
5	5.13	-5	238	-40
6	0.00	1	206	-19
7	2.56	-5	-139	-42
8	1.92	0	51	-79
9	-1.28	2	381	-54
10	-3.85	-7	-140	-108
11	0.64	-8	-78	-101
12	1.54	10	374	-46
13	-6.41	-5	-77	-61
14	-6.41	3	141	-85
15	-2.57	-2	-108	-117
16	-4.49	1	-56	-58
17	-1.28	-2	61	-114
18	4.49	0	112	-89
19	-7.69	2	83	-66
20	-1.93	-10	-107	-112
21	-1.93	-12	-191	-82
22	-3.21	-17	-293	-141
23	-7.69	-7	-239	-114
24	0.64	17	461	-60
25	1.28	4	126	-77
26	-5.13	5	280	-48
27	-7.05	-3	-130	-75
28	-7.69	12	248	-28
29	-0.65	4	109	-106
30	0.64	8	168	-63

Table 13

(Continued)

31	-8.97	6	246	5
32	-5.13	4	78	-90
33	-3.21	-3	-88	-62
34	-5.77	5	198	13
35	-8.33	3	111	-66
36	1.28	-6	-275	-44

OBS: observation number
DIFF1: difference of recognition rate (%)
DIFF2: difference of memory space required
DIFF3: difference of learning speed (iterations)
DIFF4: difference of recognition speed (times)

Table 14

Pairwised T Test of Performance of
Model H-H1 and Model H-H2
(Trained 3 fonts)

VARIABLE	MEAN	STD ERROR	T	PR>|T|
DIFF1	-1.5147	81.8412	-1.85	0.0727
DIFF2	-0.3056	1.1400	-0.27	0.7903
DIFF3	52.3333	31.8334	1.64	0.1091
DIFF4	-67.6667	5.8626	-11.54	0.0001

MEAN: average
STD ERROR: standard error of mean
T: T test score
PR: the probability of a greater absolute value of T

Table 15

Performance Differences of Model H-H1 and Model H-H2 (Trained 6 fonts)

OBS	DIFF1(%)	DIFF2	DIFF3	DIFF4
1	-1.28	3	637	-42
2	-2.57	12	695	-69
3	-1.28	1	740	-40
4	-0.64	1	553	-38
5	-2.27	42	2146	-40
6	-0.64	4	767	-95
7	0.00	3	106	16
8	-1.92	11	948	-33
9	-1.28	8	777	-70
10	-3.85	24	1298	-20
11	-3.85	10	701	-45
12	-1.93	-4	425	-41
13	-1.93	10	922	-23
14	-3.85	13	977	-33
15	1.28	-9	219	-109
16	-1.28	-4	269	-72
17	-3.21	16	1251	-45
18	-0.64	-1	521	-113
19	-0.62	70	5578	29
20	-6.41	30	2553	-66
21	-5.13	-13	75	-114
22	-8.34	62	4312	27
23	-5.13	23	2321	-39
24	-4.49	-11	76	-137
25	-10.90	42	2999	32
26	-5.13	72	5701	-57
27	-2.57	-2	879	-96
28	-7.69	85	6684	-16
29	-4.49	31	1924	-66
30	-2.57	20	1255	-122
31	-4.49	18	2081	-39
32	-4.49	18	1960	-126
33	-1.93	28	2285	-79
34	-6.41	6	1098	20
35	-3.85	25	2389	-77
36	-4.49	7	1337	-79

Table 16

Pairwised T-test of Performance of
Model H-H1 and Model H-H2
(Trained 6 fonts)

VARIABLE	MEAN	STD ERROR	T	PR>|T|
DIFF1	-3.3408	0.4243	-7.87	0.0001
DIFF2	18.0833	3.9656	4.56	0.0001
DIFF3	1651.6389	270.6564	6.10	0.0001
DIFF4	-53.2500	7.4720	-7.13	0.0001

significant difference. The recognition rate of Model H-H2 was significantly higher (3.34% higher) than that of Model H-H1. For the number of memory matrices used, Model H-H1 required many more matrices than Model H-H2. As Table 16 shows, on the average, Model H-H1 required 37.17 memory matrices which was 18.08 more than Model H-H2. Also, the learning speed of Model H-H1 was much slower than that of Model H-H2. Model H-H1 required 2937 iterations which was 1651.63 more than Model H-H2 in the training phase. As far as the recognition speed was concerned, Model H-H1 performed much faster than Model H-H2 did. On average, the input data had to be submitted 309.11 times in Model H-H1 which was 53.25 more times than in Model H-H2 in the recognition phase. In conclusion, Model H-H2 outperformed Model H-H1 in terms of recognition rate, memory space used, and learning speed when 156 characters were used as the training set.

Hogg and Huberman versus Cash and Hatamian

The recognition rates of models H-H1 and H-H2 were compared to the result of Cash and Hatamian's study (1987). In this study, both H-H1 and H-H2 performed better, in terms of recognition rate, than the optical character recognition using the method of moments (Cash and Hatamian, 1987). In their study, the training and recognition sets are both six machine-printed fonts in alphanumeric characters. The six

fonts used in their study were Courier, Elite, Pica, Helvetica, Memphis Medium, and Times Bold Italic. In the training phase, the training documents were isolated by contour tracing, and then the 2D moments of each character were computed and stored in a library of feature vectors. In the recognition phase, the document to be recognized was scanned, and the 2D moments of its characters were compared with those in the library for classification. They claimed that recognition rates between 98.5% and 99.7% have been achieved for all fonts test. However, The Hogg and Huberman models (Model H-H1 and Model H-H2) have 100% recognition rate for some parameter combinations (e.g., 4x14 memory matrix size, Smin_Smax=±18, Mmin=1, and Mmax=4 for Model H-H1; 4x14 memory matrix size, Smin_Smax=±18, and Mmin_Mmax=±8 for Model H-H2). Clearly, the performance of models H-H1 and H-H2 are significantly better. This comparison was made when all of the recognition set was used as the training set, as was the case in Cash and Hatamian's study.

Hogg and Huberman versus Fujii and Morita

When comparing the performance of Model H-H1 and Model H-H2 to the performance of the recognition system using a similar encoding scheme, the Hogg and Huberman models outperformed the Fujii and Morita's study (1971). Fujii and Morita used a simulation of the visual nervous system as

their recognition system. In their study, a cascade connection of the lateral inhibition structure and the Adaline learning system were applied to the handwritten recognition problem. Both the training and the recognition sets were handwritten numerals (0-9). Eleven properties were extracted from each of the handwritten letters. Then the adaptive classifier was trained with supervision. In the training phase, the training set (400 numerals) were submitted to the classifier and the connection weights in the classifier were adjusted accordingly. One hundred suitably selected characters from the training set were submitted to the classifier in the recognition phase. Fujii and Morita claimed that the 100 suitably selected characters were identified 100% correctly. On the other hand, models H-H1 and H-H2 have achieved 100% accurate recognition for all the training data. This indicates that the Hogg and Huberman models are more appropriate than the system that was based on the simulation of the visual nervous system for the font recognition application.

CHAPTER V

CONCLUSIONS AND RECOMMENDATIONS FOR FUTURE RESEARCH

Conclusions

Two neural network models have been successfully applied to the font recognition problem. The models used were developed by Hogg and Huberman (1984, 1985) and were used in this research to recognize the 26 capitalized English letters, each with six font representations. Recognition rate, memory space used, learning speed, and recognition speed were used to measure models' performances. Model parameters such as memory array size, Smin_Smax, and Mmin_Mmax were varied to elucidate the models' behaviors. To interpret the results of this study, it is important to consider the validity and sensitivity of these models. The conclusion of this study are as follows:

(1) The Hogg and Huberman models (Model H-H1 and Model H-H2) are useful in font recognition. Both models achieved a 100% recognition rate for particular parameter combinations (e.g., 4x14 memory matrix size, Smin_Smax =$\pm$18, Mmin=1, and Mmax=4 for Model H-H1; 4x14 memory matrix size, Smin_Smax =$\pm$18, and Mmin_Mmax =$\pm$8 for

Model H-H2) when all six fonts were submitted as both the training and the recognition set.

(2) Recognition is better, when using these models, than any previous models for this problem, the comparisons are: (a) The recognition performance of these two models are considerably better than the character recognition method using moments (Cash and Hatamian, 1987). Model H-H1 and Model H-H2 achieved a perfectly accurate recognition performance; while Cash and Hatamian's research achieved only a 98.5% to 99.7% recognition rate. (b) Models H-H1 and H-H2 outperformed the recognition system of Fujii and Morita (1971). The encoding scheme used in this study was similar to the encoding scheme used in their study. Although a 100% recognition rate was obtained in their study, their testing data consisted of only 1/4 of the training data and only numerals were recognized in their research. In this study, when there were twice as many fonts in the recognition set as they were in the training set, the highest recognition rate was 87.82% for Model H-H1 and 89.10% for Model H-H2. When all the training sets (six fonts) were submitted as the recognition set, the highest recognition rate was 100.00% for both models. This indicates that the Hogg and Huberman models are better recognition algorithms than the simulated visual nervous recognition system used by Fujii and Morita.

(3) Model H-H2 significantly outperformed Model H-H1 in terms of recognition rate, use of memory space, and learning speed when the models trained all six fonts for the 26 English letters. This was supported by the results of the Pairwised T Test However, the test results showed that Model H-H2 required more time to recognize the recognition set than Model H-H1 did.

(4) The recognition rate for fonts that were not in the training set was also acceptable. Other models did not test fonts that are not trained. When three out of six fonts were using for training, Model H-H1 achieved a maximum recognition rate 87.82% and Model H-H2 achieved a maximum recognition rate 89.10%. This indicates that the models were capable of recognizing some of the untrained characters. This behavior shows that the basins of attractor states existed for the letters in most of the various font presentations.

(5) The Hogg and Huberman models can be useful in other pattern recognition problems. The fonts used in this study certainly represent a pattern class. Since Model H-H1 and Model H-H2 can achieve reliable font recognition, it seems that they should be able to exhibit highly accurate recognition behavior in other pattern recognition problems.

Recommendations for Future Research

This study applied the Hogg and Huberman models to the font recognition problem and it has prepared a foundation for further research in these areas:

(1) This work only investigated the line characters (i.e., characters that are only with one unit wide). Most fonts have a width larger than 1 for various parts of the letters. More work needs to be done to see if the excellent results achieved here can be extended to more realistic fonts. The lower case English letters, numerals, and handwritten letters also should be investigated.

(2) Further improvement of model performance for the untrained fonts should be attempted be achieved by increasing the number of character properties and/or by selecting different character properties.

(3) Further research on how the performance is affected by the introduction of degraded or corrupted letters should also be investigated prior to committing to a system design.

(4) A hardware implementation of the models appears feasible since a reliable recognition performance (100% recognition rate) has been achieved. According to the study done by McClain, Rogers, and Oldham in 1988, the inherent parallelism operations in the models gives them a definite speed advantage over conventional sequential computers. In their study, they claimed that the performance of the Hogg and Huberman model (Model H-H1) can gain in excess of 400

times using two GAPP chips versus the VAX (11/780). The CPU time for running Model H-H1 to train 156 characters and to recognize 156 characters was 4.5 seconds, when the memory matrix size was 4x6, Smin_Smax=±18, Mmin=1, and Mmax=4. Using the results of McClain, et al., it should take only 11.2 millisecond to train and recognize 156 characters if Model H-H1 is implemented in hardware. Because the learning speed versus recognition speed was 852:194, it should take less than 2.08 milliseconds to recognize 156 characters. Assume that an envelope has 100 characters on it (all English capital letters), it would cost only 1.33 milliseconds to recognize the characters on the envelope. This suggests that the hardware implementation of the Hogg and Huberman models can provide satisfactory speed and can be used as a character recognition device in post offices.

BIBLIOGRAPHY

Anderson, J. A., Silverstein, J. W., Rits, S. A., and Jones, R. S., "Distinctive Features, Categorical Perception, and Probability Learning: Som Applications of a Neural Model," Psychological Review, Vol. 84, member 5, Sept. 1977 pp. 413-443

Bongard, M., Pattern Recognition, Spartan Books, 1970

Carpenter, G. A. and Grossberg, S., "A Massively Parallel Architecture for a Self-Organizating Neural Pattern Recognition Machine," Computer Vision, Graphics, and Image Processing, 37, 1987, pp. 54-115

Carpenter, G. A. and Grossberg, S., "The Art of Adaptive Pattern Recognition by a Self-organizing Neural Network," IEEE, March 1988, pp. 77-88

Cash, G. L. and Hatamian, M., "Optical Character Recognition by the Methods of Moments," Computer Vision, Graphics, and Image Processing, 39, 1987, pp. 291-310

Duda, R.O. and Hart P. E., "Linear Discriminant Functions," Pattern Classification and Scene Anaysis, 1973, pp. 151-152

Fahlman, S. E. and Hinton, G. E., "Connectionist Architectures for Artificial Intelligence," IEEE, Jan. 1988, pp. 100-109

Fujii, K. and Morita, T., "Recognition System for Handwritten Letters Simulating Visual Nervous System," Pattern Recognition and Machine Learning, 1971, pp. 56-69

Hecht-Nielsen, R., Internal Report, TRW Rancho Carmel Artificial Intelligence Center, One Rancho Carmel, San Diego, CA 92128, 1987

Hogg, T. and Huberman, B. A., "Understanding Biological Computation: Reliable Learning and Recognition," Proc. Nat. Acad. Sci. U.S.A.-Neurobiology, Vol. 81, Sep. 1984, pp. 6871-6875

Hogg, T. and Huberman, B. A., "Parallel Computing Structures Capable of Flexible Associations and Recognition of Fuzzy Inputs," J. of Statistical Physics, Vol.41, Oct., 1985, pp. 115-123

Hogg, T. and Huberman, B. A., "Attractors on Finite Sets: The Dissipative Dynamics of Computing Structures," Physical Review, Vol. 32, No. 4, Oct. 1985, pp. 2338-2346

Hopfield, J. J., "Neural networks and Physical Systems with Emergent Collective Computational Abilities," Proc. Natl. Acad. Sci. U.S.A. - Biophysics, Vol. 79, April, 1982, pp. 2551-2558

Hopfield, J. J. and Tank, D. W., "Neural Computation of Decisions in Optimization Problems," Biological Cybernetics, Vol. 52, No. 3, 1985, pp.141-152

Huberman, B. A. and Hogg, T., "Adaptation and Self-Repair in Parallel Computing Structures, Physical Review Letters, Vol. 52, No. 12, March, 1984, pp. 1048-1051

d'Humieres, D. and Huberman, B. A., "Dynamics of Self-Organization in Complex Adaptive Networks," J. of Statistical Physics, Vol. 34, Nos. 3/4, 1984, pp. 361-379

Lippmann, R., "An Introduction to Computing with Neural Nets," IEEE ASSP Magazine, April, 1987, pp.4-22

McClelland, J. L., Rumelhart, D. E., and Hinton, G. E., "The Appeal of Parallel Distributed Processing," Parallel Distributed Processing, pp.4-76, The Massachusetts Institute of Technology," 1986

Oldham, W. J. B., "Modeling Neural Networks," G3854.1401.01c, 1986

Schuster, H., "Deterministic Chaos," 1988

APPENDIX A

MODEL H-H1

```
/*============================================================*/
/*                    MODEL1: MAIN PROGRAM                    */
/*============================================================*/
HH1: PROCEDURE OPTIONS(MAIN);
 DECLARE
  (OUTFILE,IN1,IN2,THRES)                    FILE STREAM EXTERNAL,
  INFILE                                       FILE VARIABLE,
  (SMIN,SMAX,MMIN,MMAX,THRESHOLD(1:14),OUTP(1:80,1:25),
   TOTAL_LETTERS,B(1:25,1:25),OUTTEMP(1:80,1:25),
   OUT(0:25,0:26))                          FIXED DECIMAL(2,0),
  (NO_REJECTED,NO_RECOGNITION,NO_ITERATION,DEEPEST,
   MEMORY_NO,NUM_RUN,RUN,CORRECT,I,J,L,IN_PAT,REC_PAT,
   TOTAL_PAT,ROW_SIZE,COL_SIZE,TOTAL_FONT_ACCEPTED(1:6),
   TOTAL_FONT_REJECTED(1:6),ACCEPTED(1:26),REJECTED(1:26))
                                            FIXED BINARY(15,0),

  1 TRAINED,
    3 REJECTED                              FIXED BINARY(15,0),
    3 ACCEPTED                              FIXED BINARY(15,0),
    3 NO OF FONT(1:6)                       FIXED BINARY(15,0),
  1 UNTRAINED,
    3 REJECTED                              FIXED BINARY(15,0),
    3 ACCEPTED                              FIXED BINARY(15,0),
  1 INP(1:78),
    3 CODE(1:25)                            FIXED DECIMAL(2,0),
    3 ALPHA                                       CHARACTER(1),
  1 RECP(1:78),
    3 CODE(1:25)                            FIXED DECIMAL(2,0),
    3 ALPHA                                       CHARACTER(1),
  1 LEARNED                                         BASED(LOC),
    3 MEMORY(1:25,1:25)                     FIXED DECIMAL(2,0),
    3 OUT_HEAD                                          POINTER,
  1 LIST                                            BASED(ADR),
    3 CODE(1:25)                          FIXED DECIMAL(2,0),
    3 LETTERS                           CHARACTER(20) VARYING,
    3 NEXTLIST                                      POINTER,
    3 NEXTMEM                                       POINTER,
  1 INPATTERN(1:78),
    3 FONT                                  FIXED DECIMAL(1),
    3 CODE(1:25)                            FIXED DECIMAL(2,0),
    3 ALPHA                                       CHARACTER(1),
  ALPHABET
        CHARACTER(26) INITIAL('ABCDEFGHIJKLMNOPQRSTUVWXYZ'),
                                                    BUILTIN,
  NULL                                              POINTER;
  (HEAD,LOC,ADR,PTR)
CALL INITIALIZE;
```

```
INFILE=IN1;
OPEN FILE(IN1) INPUT;
CALL INPUT_LETTER_CODE(INFILE,IN_PAT*TOTAL_PAT,INP,1);
CALL LEARNING;
CLOSE FILE(IN1);
DO I=1 TO 78 BY 1;
   J=INPATTERN.FONT(I);
   TRAINED.NO_OF_FONT(J)=TRAINED.NO_OF_FONT(J)+1;
END;
HEAD=NULL;
CALL OFFICIAL_OUTPUT(HEAD);
PTR=HEAD;
MEMORY_NO=1;
DEEPEST=1;
PTR=HEAD->LEARNED.OUT_HEAD;
CALL DISTINCT(PTR,2);
INFILE=IN2;
OPEN FILE(IN2) INPUT;
REC_PAT=6;
PTR=HEAD->LEARNED.OUT_HEAD;
DO I=1 TO 26 BY 1;
   CALL INPUT_LETTER_CODE(INFILE,REC_PAT,RECP,1);
   CALL RECOGNITION(I);
END;
OPEN FILE(OUTFILE) OUTPUT;
PUT FILE(OUTFILE) SKIP(2) EDIT('THE CORRECTLY RECOGNIZED
 PATTERN =',CORRECT,'THE RECOGNITION RATE = ',CORRECT/156)
  (COL(1),A,F(3),X(10),A,F(5,3));
PUT FILE(OUTFILE) SKIP(2) EDIT('NUMBER OF REJECTION = ',
 NO_REJECTED)(COL(1),A,F(6));
PUT FILE(OUTFILE) SKIP(2) EDIT('# OF CORRECT RECONGNITION
  RECOGNITION RATE    # OF REJECTION      REJECTION RATE')
  (COL(20),A);
PUT FILE(OUTFILE) SKIP(0) EDIT((100)'-')(COL(17),A);
PUT FILE(OUTFILE) EDIT('TRAINED',TRAINED.ACCEPTED,
 TRAINED.ACCEPTED/78,TRAINED.REJECTED,TRAINED.REJECTED/78)
 (COL(1),A,COL(30),F(4),COL(50),F(6,4),COL(70),F(4),COL(90),
 F(6,4));
PUT FILE(OUTFILE) EDIT('UNTRAINED',UNTRAINED.ACCEPTED,
 UNTRAINED.ACCEPTED/78,UNTRAINED.REJECTED,
 UNTRAINED.REJECTED/78)(COL(1),A,COL(30),F(4),COL(50),
 F(6,4),COL(70),F(4),COL(90),F(6,4));
PUT FILE(OUTFILE) SKIP(2) EDIT('DEPTH = ',DEEPEST,'NUMBER
 OF MEMORY MODELS USED = ',MEMORY_NO)(COL(1),A,F(3),X(10),
 A,F(3));
PUT FILE(OUTFILE) SKIP(2) EDIT('NUMBER OF ITERATION = ',
 NO_ITERATION)(COL(2),A,F(7));
PUT FILE(OUTFILE) SKIP(2) EDIT('NUMBER OF RECOGNITION = ',
 NO_RECOGNITION)(COL(2),A,F(7));
DO L=1 TO 26 BY 1;
   PUT FILE(OUTFILE) EDIT(SUBSTR(ALPHABET,L,1))
```

```
    (COL(12+L*5),A);
END;
PUT FILE(OUTFILE) SKIP(2) EDIT('ACCEPTED =')(COL(1),A);
DO L=1 TO 26 BY 1;
   PUT FILE(OUTFILE) EDIT(ACCEPTED(L))(COL(10+L*5),F(3));
END;
PUT FILE(OUTFILE) SKIP(2) EDIT('REJECTED =')(COL(1),A);
DO L=1 TO 26 BY 1;
   PUT FILE(OUTFILE) EDIT(REJECTED(L))(COL(10+L*5),F(3));
END;
PUT FILE(OUTFILE) SKIP(2) EDIT('NUMBER OF TRAINED FONT =')
 (COL(1),A);
DO L=1 TO 6 BY 1;
   PUT FILE(OUTFILE) EDIT(TRAINED.NO_OF_FONT(L))
    (COL(30+L*6),F(3));
END;
PUT FILE(OUTFILE) SKIP(2) EDIT('CORRECT =')(COL(1),A);
DO L=1 TO 6 BY 1;
   PUT FILE(OUTFILE) EDIT(TOTAL_FONT_ACCEPTED(L))
    (COL(30+L*6),F(3));
END;
PUT FILE(OUTFILE) SKIP(2) EDIT('REJECTED =')(COL(1),A);
DO L=1 TO 6 BY 1;
   PUT FILE(OUTFILE) EDIT(TOTAL_FONT_REJECTED(L))
    (COL(30+L*6),F(3));
END;
    (COL(30+L*6),F(3));
END;
 CLOSE FILE(SYSPRINT);
 CLOSE FILE(OUTFILE);
/*============================================================*/
/*                    DISTINCT PROCEDURE                      */
/*============================================================*/
DISTINCT: PROCEDURE(PTR,DEPTH) RECURSIVE ;
  DECLARE
   (PTR,ADDR)                                        POINTER,
   (LEN, J,L,K,M,DEPTH)                  FIXED BINARY(15,0);

DO WHILE(PTR^=NULL & DEPTH<7);
  LEN=LENGTH(PTR->LIST.LETTERS);
  IF LEN>1 THEN
    DO;
      MEMORY_NO=MEMORY_NO+1;
      IF DEPTH>DEEPEST THEN
        DEEPEST=DEPTH;
      ELSE;
      TOTAL_PAT=LEN;
      L=0;
      DO J=1 TO 78 BY 3;
        IF INDEX(PTR->LIST.LETTERS,INPATTERN(J).ALPHA)^=0
           THEN
          DO;
            DO M=1 TO 3 BY 1;
```

```
            L=L+1;
            DO K=1 TO COL_SIZE BY 1;
              INP(L).CODE(K)=INPATTERN(J+M-1).CODE(K);
            END;
            INP(L).ALPHA=INPATTERN(J).ALPHA;
          END;
        END;
      ELSE;
    END; /* J */
    CALL RESET_MEMORY_ARRAY;
    CALL RESET_OUT_ARRAY;
    CALL LEARNING;
    CALL OFFICIAL_OUTPUT(PTR);
    ADDR=PTR->LIST.NEXTMEM->LEARNED.OUT_HEAD;
    CALL DISTINCT(ADDR,DEPTH+1);
  END;
 ELSE;
 PTR=PTR->LIST.NEXTLIST;
END; /* WHILE */
END DISTINCT;
/*=========================================================*/
/*                OFFICIAL_OUTPUT PROCEDURE                */
/*=========================================================*/
OFFICIAL_OUTPUT: PROCEDURE(LIST_HEAD);
  DECLARE
   (PTR,LIST_HEAD,TRAIL)                          POINTER,
   FOUND                                          BIT (1),
   INDEX                                          BUILTIN,
   (K, I, J, L)                        FIXED BINARY(15,0);
ALLOCATE LEARNED;
LOC->LEARNED.OUT_HEAD=NULL;
DO I=1 TO ROW_SIZE BY 1;
   DO J=1 TO COL_SIZE BY 1;
      LOC->LEARNED.MEMORY(I,J)=B(I,J);
   END;
END;
IF LIST_HEAD=NULL THEN
   HEAD=LOC;
ELSE
   LIST_HEAD->LIST.NEXTMEM=LOC;
DO I=1 TO TOTAL_PAT BY 1;
   DO K=1 TO 3 BY 1;
      L=(I-1)*3+K;
      DO J=1 TO COL_SIZE BY 1;
        OUT(0,J)=INP(L).CODE(J);
      END;
      OUT(0,COL_SIZE+1)=OUT(0,1);
      OUT(0,0)=OUT(0,COL_SIZE);
      CALL RESET_OUT_ARRAY;
      CALL COMPUTE_OUTPUT(1,ROW_SIZE,1,OUT,B);
      PTR=LOC->LEARNED.OUT_HEAD;
```

```
FOUND='0'B;
TRAIL=PTR;
DO WHILE(^FOUND & PTR^=NULL);
   TRAIL=PTR;
   L=1;
   FOUND='1'B;
   DO WHILE(FOUND & L<=COL_SIZE);
      IF PTR->LIST.CODE(L)=OUT(ROW_SIZE,L) THEN
         L=L+1;
      ELSE
        FOUND='0'B;
   END;
   IF FOUND THEN
      IF INDEX(PTR->LIST.LETTERS,
                        INP((I-1)*3+K).ALPHA)=0 THEN
        PTR->LIST.LETTERS=
            PTR->LIST.LETTERS||INP((I-1)*3+K).ALPHA;
      ELSE;
   ELSE
     PTR=PTR->LIST.NEXTLIST;
END;
IF ^FOUND THEN
   DO;
     ALLOCATE LIST;
     ADR->LIST.NEXTLIST=NULL;
     ADR->LIST.NEXTMEM=NULL;
     DO L=1 TO COL_SIZE BY 1;
        ADR->LIST.CODE(L)=OUT(ROW_SIZE,L);
     END;
     ADR->LIST.LETTERS=INP((I-1)*3+K).ALPHA;
     IF HEAD=LOC THEN
        IF LIST_HEAD->LEARNED.OUT_HEAD=NULL THEN
           LIST_HEAD->LEARNED.OUT_HEAD=ADR;
        ELSE
          TRAIL->LIST.NEXTLIST=ADR;
     ELSE
     IF LIST_HEAD->LIST.NEXTMEM->LEARNED.OUT_HEAD=
                                          NULL THEN
        LOC->LEARNED.OUT_HEAD=ADR;
     ELSE
       TRAIL->LIST.NEXTLIST=ADR;
   END;
  END;
END OFFICIAL_OUTPUT;
/*=======================================================*/
/*              INPUT_LETTER_CODE PROCEDURE              */
/*=======================================================*/
INPUT_LETTER_CODE: PROCEDURE(INFILE,PAT,ARRAY,PAT_NO);
  DECLARE
    INFILE                                  FILE VARIABLE,
```

```
     (DUMMY)                                          FIXED DECIMAL(2,0),
     1 ARRAY(1:78),
       3 CODE(1:25)                                   FIXED DECIMAL(2,0),
       3 ALPHA                                              CHARACTER(1),
     (I,J,K,P,PAT_NO,PAT)                             FIXED BINARY(15,0);
DO P=1 TO PAT BY 1;
    GET FILE(INFILE) LIST(INPATTERN(P).FONT);
    DO I=1 TO COL_SIZE BY 1;
       GET FILE(INFILE) LIST (DUMMY);
       IF THRESHOLD(I)=0 THEN
          IF DUMMY>=1 THEN
             ARRAY(P).CODE(I)=DUMMY;
          ELSE
             ARRAY(P).CODE(I)=-4;
       ELSE
         ARRAY(P).CODE(I)=DUMMY-THRESHOLD(I);
       INPATTERN(P).CODE(I)=ARRAY(P).CODE(I);
    END;
    GET FILE(INFILE) LIST(ARRAY(P).ALPHA);
    INPATTERN(P).ALPHA=ARRAY(P).ALPHA;
    GET FILE(INFILE) SKIP;
END;
END INPUT_LETTER_CODE;
/*=============================================================*/
/*                  COMPUTE_OUTPUT PROCEDURE                   */
/*=============================================================*/
COMPUTE_OUTPUT:PROCEDURE(START,END,STEP,OUT,B);
  DECLARE
    (B(1:25,1:25),OUT(0:25,0:26))                  FIXED DECIMAL(2,0),
    (DIF,IR,IL)                                    FIXED DECIMAL(3,0),
    (START,END,STEP,ROW,PRE_ROW,COLUMN)            FIXED BINARY(15,0);
DO ROW=START TO END BY STEP;
    PRE_ROW=ROW-1;
    DO COLUMN=1 TO COL_SIZE BY 1;
       IL=OUT(PRE_ROW, COLUMN-1);
       IR=OUT(PRE_ROW, COLUMN+1);
       DIF=IL-IR;
       OUT(ROW,COLUMN)=MAX(SMIN,MIN(SMAX,DIF*B(ROW,COLUMN)));
    END; /* END OF COLUMN */
    OUT(ROW,COL_SIZE+1)=OUT(ROW,1);     /* WRAP AROUND */
    OUT(ROW,0)=OUT(ROW,COL_SIZE);
END;
END COMPUTE_OUTPUT;
/*=============================================================*/
/*                 ADJUST_WEIGHTS PROCEDURE                    */
/*=============================================================*/
ADJUST_WEIGHT: PROCEDURE(CUR_ROW);
  DECLARE
    (CUR_ROW,COLUMN,ROW)                           FIXED BINARY(15,0);
DO ROW=1 TO CUR_ROW BY 1;
    DO COLUMN=1 TO COL_SIZE BY 1;
```

```
      IF ( OUT(ROW, COLUMN) > OUT(ROW, COLUMN-1) &
           OUT(ROW, COLUMN) > OUT(ROW, COLUMN+1)) THEN
         B(ROW,COLUMN)=B(ROW,COLUMN)+1;
      ELSE
         IF ( OUT(ROW, COLUMN) < OUT(ROW, COLUMN-1) &
              OUT(ROW, COLUMN) < OUT(ROW, COLUMN+1)) THEN
            B(ROW,COLUMN)=B(ROW,COLUMN)-1;
      B(ROW,COLUMN)=MAX(MMIN,MIN(MMAX, B(ROW,COLUMN)));
   END;
END;
END ADJUST_WEIGHT;
/*=========================================================*/
/*                  LEARNING PROCEDURE                     */
/*=========================================================*/
LEARNING:PROCEDURE;
  DECLARE
   (CONVERGE,DIFFERENT)                          BIT (1),
   (TOTAL_LETTERS,ROW1,I,K,L,ROW)       FIXED BINARY(15,0);
CONVERGE='0'B;
K=1;
TOTAL_LETTERS=TOTAL_PAT*3;
DO WHILE (^CONVERGE & K<10000);
   NO_ITERATION=NO_ITERATION+1;
   ROW1=MOD(K-1,TOTAL_LETTERS)+1;
   IF K>ROW_SIZE THEN
      L=ROW_SIZE;
ELSE
   L=K;
DO I=1 TO COL_SIZE BY 1;
   OUT(0,I)=INP(ROW1).CODE(I);
END;
OUT(0,COL_SIZE+1)=OUT(0,1);
OUT(0,0)=OUT(0,COL_SIZE);
CALL COMPUTE_OUTPUT(L,1,-1,OUT,B);
CALL ADJUST_WEIGHT(L);
DIFFERENT='1'B;
IF (K>=ROW_SIZE) THEN
   DO;
     ROW=MOD((K-ROW_SIZE),TOTAL_LETTERS)+1;
     DO I=1 TO COL_SIZE BY 1;
        OUTTEMP(ROW,I)=OUT(ROW_SIZE,I);
     END;
     IF (ROW=TOTAL_LETTERS) THEN
        CALL COMPARE_PATTERN(DIFFERENT);
     ELSE;
   END;
 ELSE;
 IF ^DIFFERENT THEN
    CONVERGE='1'B;
 ELSE
     K=K+1;
```

```
  END; /* END OF WHILE LOOP */
END LEARNING;
/*=============================================================*/
/*                COMPARE_PATTERN PROCEDURE                    */
/*=============================================================*/
COMPARE_PATTERN: PROCEDURE(DIFF);
  DECLARE
    DIFF                                                BIT(1),
    (ROW1,I,TOTAL_LETTERS)                 FIXED BINARY(15,0);

TOTAL_LETTERS=TOTAL_PAT*3;
ROW1=1;
DIFF='0'B;
DO WHILE(ROW1<=TOTAL_LETTERS & ^DIFF);
   I=1;
   DO WHILE(I<=COL_SIZE & ^DIFF);
      IF (OUTP(ROW1,I)=OUTTEMP(ROW1,I)) THEN
         I=I+1;
      ELSE
        DIFF='1'B;
   END;
   ROW1=ROW1+1;
END;
IF DIFF THEN
   DO ROW1=1 TO TOTAL_LETTERS BY 1;
      DO I=1 TO COL_SIZE BY 1;
         OUTP(ROW1,I)=OUTTEMP(ROW1,I);
      END;
   END;
ELSE;
END COMPARE_PATTERN;
/*=============================================================*/
/*                      INITIALIZE PROCEDURE                   */
/*=============================================================*/
INITIALIZE:PROCEDURE;
 DECLARE
  (I, J)                              FIXED BINARY(15,0);
OPEN FILE(THRES) INPUT;
GET FILE(THRES) LIST (ROW_SIZE, COL_SIZE);
PUT FILE(OUTFILE) SKIP(2) EDIT('NUMBER OF ROW = ',ROW_SIZE,
 ' NUMBER OF COLUMN = ',COL_SIZE)(COL(1),A,F(4),A,F(4));
PUT FILE(OUTFILE) EDIT ('THE THRESHOLDS ARE:')(COL(1),A);
PUT FILE(OUTFILE) SKIP(2);
DO I=1 TO COL_SIZE BY 1;
   GET FILE(THRES) LIST (THRESHOLD(I));
   PUT FILE(OUTFILE) EDIT (THRESHOLD(I))(COL(4*I),F(3));
END;
GET FILE(THRES) LIST (NUM_RUN,IN_PAT,TOTAL_PAT);
PUT FILE(OUTFILE) SKIP(2) EDIT ('NUMBER OF RUNS = ',NUM_RUN,
 'IN_PAT=',IN_PAT,'NUMBER OF PATTERNS = ',TOTAL_PAT)(COL(1),
 A,F(4),COL(30),A,F(4));
```

```
GET FILE(THRES) LIST (SMIN, SMAX,MMIN, MMAX);
PUT FILE(OUTFILE) SKIP(2) EDIT('SMIN= ',SMIN,' SMAX= ',SMAX,
 ' MMIN= ',MMIN,' BMAX = ',MMAX)(COL(1),A,F(4),
 A,F(4),A,F(4),A,F(4));
CALL RESET_MEMORY_ARRAY;
CALL RESET_OUT_ARRAY;
CALL RESET_OUTP;
END INITIALIZE;
/*=============================================================*/
/*                    RESET_OUTP PROCEDURE                     */
/*=============================================================*/
RESET_OUTP:PROCEDURE;
  DECLARE
     (I,J)              FIXED BINARY(15,0);
DO I=1 TO TOTAL_LETTERS BY 1;
   DO J=1 TO COL_SIZE BY 1;
      OUTP(I,J)=0;
   END;
END;
END RESET_OUTP;
/*=============================================================*/
/*                      RESET_MEMORY_ARRAY                     */
/*=============================================================*/
RESET_MEMORY_ARRAY: PROCEDURE;
 DECLARE
    (I,J)                 FIXED BINARY(15,0);
DO I=1 TO ROW_SIZE BY 1;
   DO J= 1 TO COL_SIZE BY 1;
      B(I,J)=1;
   END;
END;
END RESET_MEMORY_ARRAY;
/*=============================================================*/
/*                        RESET_OUT_ARRAY                      */
/*=============================================================*/
RESET_OUT_ARRAY: PROCEDURE;
 DECLARE
    (I,J,SIZE)                    FIXED BINARY(15,0);
SIZE=COL_SIZE+1;
DO I=1 TO ROW_SIZE BY 1;
   DO J= 0 TO SIZE BY 1;
      OUT(I,J)=0;
   END;
END;
END RESET_OUT_ARRAY;
/*=============================================================*/
/*                     RECOGNITION PROCEDURE                   */
/*=============================================================*/
RECOGNITION: PROCEDURE(L);
 DECLARE
  (P,I,J,K,L,ROW,ROW1,DEPTH,COLUMN,PRE_ROW,SIZE)
```

```
                                                    FIXED BINARY(15,0),
  (SUM,IL,IR)                                       FIXED DECIMAL(2,0),
  PTR                                                     POINTER,
  (CONTINUE,GO_ON,DIFFERENT)                               BIT(1);
DO P=1 TO REC_PAT BY 1;
   CONTINUE='1'B;
   PTR=HEAD;
   DEPTH=1;
   DO WHILE(CONTINUE & PTR^=NULL);
      DO I=1 TO COL_SIZE BY 1;
         OUT(0,I)=RECP(P).CODE(I);
      END;
      OUT(0,COL_SIZE+1)=OUT(0,1);
      OUT(0,0)=OUT(0,COL_SIZE);
      CALL RESET_OUT_ARRAY;
      NO_RECOGNITION=NO_RECOGNITION+1;
      CALL COMPUTE_OUTPUT(1,ROW_SIZE,1,OUT,
                          PTR->LEARNED.MEMORY);
      PTR=PTR->LEARNED.OUT_HEAD;
      GO_ON='1'B;
      DO WHILE(GO_ON & PTR^=NULL);
         DIFFERENT='0'B;
         COLUMN=1;
         DO WHILE(COLUMN<=COL_SIZE & ^DIFFERENT);
            IF (OUT(ROW_SIZE,COLUMN)=PTR->LIST.CODE(COLUMN))
               THEN
               COLUMN=COLUMN+1;
            ELSE
              DIFFERENT='1'B;
        END;
        IF DIFFERENT THEN
           PTR=PTR->LIST.NEXTLIST;
        ELSE
          DO;
            GO_ON='0'B;
            IF LENGTH(PTR->LIST.LETTERS)>1 &
                      PTR->LIST.NEXTMEM^=NULL THEN
               DO;
                  PTR=PTR->LIST.NEXTMEM;
                  DEPTH=DEPTH+1;
               END;
            ELSE
               CONTINUE='0'B;
          END;
     END;
END;
I=INDEX(ALPHABET,RECP(P).ALPHA);
J=INPATTERN(P).FONT;
IF PTR=NULL THEN
   DO;
     REJECTED(I)=REJECTED(I)+1;
```

```
        TOTAL_FONT_REJECTED(J)=TOTAL_FONT_REJECTED(J)+1;
        NO_REJECTED=NO_REJECTED+1;
        IF L<14 THEN
           TRAINED.REJECTED=TRAINED.REJECTED+1;
        ELSE
            UNTRAINED.REJECTED=UNTRAINED.REJECTED+1;
      END;
    ELSE
      DO;
        IF SUBSTR(PTR->LIST.LETTERS,1,1)=RECP(P).ALPHA THEN
           DO;
             CORRECT=CORRECT+1;
             ACCEPTED(I)=ACCEPTED(I)+1;
             TOTAL_FONT_ACCEPTED(J)=
                     TOTAL_FONT_ACCEPTED(J)+1;
             IF L<14 THEN
                TRAINED.ACCEPTED=TRAINED.ACCEPTED+1;
             ELSE
                UNTRAINED.ACCEPTED=UNTRAINED.ACCEPTED+1;
           END;
         ELSE;
      END;
  END; /* P */
END RECOGNITION;
END HH1;
```

APPENDIX B

Model H-H2

```
/*=========================================================*/
/*                  MODEL2: MAIN PROGRAM                   */
/*=========================================================*/
HH2: PROCEDURE OPTIONS(MAIN);
 DECLARE
  (OUTFILE,IN1,IN2,THRES)                FILE STREAM EXTERNAL,
  INFILE                                        FILE VARIABLE,
  (THRESHOLD(1:14),B(1:25,1:25),PREV_MEMORY(1:25,1:25),
   IN(1:25,1:25),PREV_OUT(0:25,0:26),OUT(0:25,0:26),
   OUT_ARRAY(1:25,1:25),SMIN,SMAX,MMIN,MMAX)
                                          FIXED DECIMAL(2,0),
  (NO_REJECTED,NO_ITERATION,NO_RECOGNITION,MEMORY_NO,L,K,J,
   DEEPEST,NUM_RUN,RUN,CORRECT,P,I,IN_PAT,REC_PAT,ROW_SIZE,
   COL_SIZE,TOTAL_PAT,TOTAL_FONT_ACCEPTED(1:6),
   TOTAL_FONT_REJECTED(1:6),ACCEPTED(1:26),REJECTED(1:26))
                                          FIXED BINARY(15,0),
  1 INP(1:78),
    3 CODE(1:25)                          FIXED DECIMAL(2,0),
    3 ALPHA                                     CHARACTER(1),
  1 RECP(1:78),
    3 CODE(1:25)                          FIXED DECIMAL(2,0),
    3 ALPHA                                     CHARACTER(1),
  1 INPATTERN(1:78),
    3 FONT                                  FIXED DECIMAL(1),
    3 CODE(1:25)                          FIXED DECIMAL(2,0),
    3 ALPHA                                     CHARACTER(1),
  1 TRAINED,
    3 REJECTED                            FIXED BINARY(15,0),
    3 ACCEPTED                            FIXED BINARY(15,0),
    3 NO_OF_FONT(1:6)                     FIXED BINARY(15,0),
  1 UNTRAINED,
    3 REJECTED                            FIXED BINARY(15,0),
    3 ACCEPTED                            FIXED BINARY(15,0),
  ALPHABET                                      CHARACTER(26)
                     INITIAL('ABCDEFGHIJKLMNOPQRSTUVWXYZ'),
  1 LEARNED                                       BASED(LOC),
    3 MEMORY(1:25,1:25)                   FIXED DECIMAL(2,0),
    3 OUT_HEAD                                       POINTER,
  1 LIST                                          BASED(ADR),
    3 CODE(1:25)                          FIXED DECIMAL(2,0),
    3 LETTERS                           CHARACTER(20) VARYING,
    3 NEXTLIST                                       POINTER,
    3 NEXTMEM                                        POINTER,
  NULL                                               BUILTIN,
  (HEAD,LOC,ADR,PTR)                                 POINTER;
```

```
OPEN FILE(OUTFILE) OUTPUT;
CALL INITIALIZE;
INFILE=IN1;
DO RUN=1 TO NUM_RUN BY 1;
   OPEN FILE(IN1) INPUT;
   DO L=1 TO TOTAL_PAT BY 1;
      DO I=1 TO ROW_SIZE BY 1;
         DO J= 0 TO COL_SIZE BY 1;
            OUT(I,J)=0;
         END;
      END;
      CALL INPUT_LETTER_CODE(INFILE,IN_PAT,INP,IN,L);
      CALL COALES;
      DO J=1 TO IN_PAT BY 1;
         I=INPATTERN.FONT((L-1)*IN_PAT+J);
         TRAINED.NO_OF_FONT(I) =TRAINED.NO_OF_FONT(I)+1;
      END;
   END;
   CLOSE FILE(IN1);
END;
HEAD=NULL;
CALL OFFICIAL_OUTPUT(HEAD);
MEMORY_NO=1;
DEEPEST=1;
PTR=HEAD;
PTR=HEAD->LEARNED.OUT_HEAD;
CALL DISTINCT(PTR,2);
INFILE=IN2;
OPEN FILE(IN2) INPUT;
REC_PAT=6;
PTR=HEAD->LEARNED.OUT_HEAD;
DO L=1 TO 26 BY 1;
   CALL INPUT_LETTER_CODE(INFILE,REC_PAT,RECP,IN,1);
   CALL RECOGNITION(L);
END;
PUT FILE(OUTFILE) SKIP(2) EDIT('THE CORRECTLY RECOGNIZED
 PATTERN =',CORRECT,'THE RECOGNITION RATE = ',CORRECT/156)
 (COL(1),A,F(3),X(10),A,F(5,3));
PUT FILE(OUTFILE) EDIT('NUMBER OF REJECTION = ',
 NO_REJECTED)(COL(1),A,F(6));
PUT FILE(OUTFILE) SKIP(2) EDIT('# OF CORRECT RECONGNITION
 RECOGNITION RATE   # OF REJECTION      REJECTION RATE')
 (COL(20),A);
PUT FILE(OUTFILE) SKIP(0) EDIT((100)'-')(COL(17),A);
PUT FILE(OUTFILE) EDIT('TRAINED',TRAINED.ACCEPTED,
 TRAINED.ACCEPTED/78,TRAINED.REJECTED,
 TRAINED.REJECTED/78)(COL(1),A,COL(30),F(4),COL(50),
 F(6,4),COL(70),F(4),COL(90),F(6,4));
PUT FILE(OUTFILE) EDIT('UNTRAINED',UNTRAINED.ACCEPTED,
 UNTRAINED.ACCEPTED/78,UNTRAINED.REJECTED,
 UNTRAINED.REJECTED/78)(COL(1),A,COL(30),F(4),COL(50),
```

```
 F(6,4),COL(70),F(4),COL(90),F(6,4));
PUT FILE(OUTFILE) SKIP(2) EDIT('DEPTH = ',DEEPEST,'NUMBER
 OF MEMORY ARRAYS USED = ',MEMORY_NO)(COL(1),A,F(3),X(10),
 A,F(3));
PUT FILE(OUTFILE) SKIP(2) EDIT('NUMBER OF ITERATION = ',
 NO_ITERATION)(COL(2),A,F(7));
PUT FILE(OUTFILE) SKIP(2) EDIT('NUMBER OF RECOGNITION = ',
 NO_RECOGNITION)(COL(2),A,F(7));
DO L=1 TO 26 BY 1;
   PUT FILE(OUTFILE) EDIT(SUBSTR(ALPHABET,L,1))
    (COL(12+L*4),A);
END;
PUT FILE(OUTFILE) SKIP(2) EDIT('ACCEPTED =')(COL(1),A);
DO L=1 TO 26 BY 1;
   PUT FILE(OUTFILE) EDIT(ACCEPTED(L))(COL(10+L*4),F(3));
END;
PUT FILE(OUTFILE) SKIP(2) EDIT('REJECTED =')(COL(1),A);
DO L=1 TO 26 BY 1;
   PUT FILE(OUTFILE) EDIT(REJECTED(L))(COL(10+L*4),F(3));
END;
PUT FILE(OUTFILE) SKIP(2) EDIT('NUMBER OF LETTER TRAINED =')
 (COL(1),A);
DO L=1 TO 6 BY 1;
   PUT FILE(OUTFILE) EDIT(TRAINED.NO_OF_FONT(L))
    (COL(30+L*6),F(6));
END;
PUT FILE(OUTFILE) SKIP(2) EDIT('CORRECT =')(COL(1),A);
DO L=1 TO 6 BY 1;
   PUT FILE(OUTFILE) EDIT(TOTAL_FONT_ACCEPTED(L))
    (COL(30+L*6),F(6));
END;
PUT FILE(OUTFILE) SKIP(2) EDIT('REJECTED =')(COL(1),A);
DO L=1 TO 6 BY 1;
   PUT FILE(OUTFILE) EDIT(TOTAL_FONT_REJECTED(L))
    (COL(30+L*6),F(6));
END;
CLOSE FILE(OUTFILE);
/*=========================================================*/
/*                    DISTINCT PROCEDURE                   */
/*=========================================================*/
DISTINCT: PROCEDURE(PTR,DEPTH) RECURSIVE ;
  DECLARE
   (PTR,ADDR,TEMPPTR)                          POINTER,
   (LEN,I,J,L,K,M,DEPTH)              FIXED BINARY(15,0);
DO WHILE(PTR^=NULL & DEPTH<7);
   LEN=LENGTH(PTR->LIST.LETTERS);
   IF LEN>1 THEN
      DO;
         MEMORY_NO=MEMORY_NO+1;
         IF DEPTH>DEEPEST THEN
            DEEPEST=DEPTH;
```

```
ELSE;
TOTAL_PAT=LEN;
L=0;
DO J=1 TO 78 BY IN_PAT;
   IF INDEX(PTR->LIST.LETTERS,
                  INPATTERN(J).ALPHA)^=0 THEN
      DO;
         DO M=1 TO IN_PAT BY 1;
            L=L+1;
            DO K=1 TO COL_SIZE BY 1;
               INP(L).CODE(K)=
                     INPATTERN(J+M-1).CODE(K);
               IN(M,K)=INP(L).CODE(K);
            END;
            INP(L).ALPHA=INPATTERN(J).ALPHA;
         END;
         IF MOD(DEPTH,2)=1 THEN
            DO;
              CALL RESET_OUT_ARRAY;
              CALL RESET_MEMORY_ARRAY;
              CALL COALES;
            END;
         ELSE;
      END;
   ELSE;
         END; /* J */
         IF MOD(DEPTH,2)=0 THEN
            DO;
               DO I=1 TO LEN BY 1;
                  DO K=1 TO COL_SIZE BY 1;
                     IN(I,K)=INP((I-1)*IN_PAT+1).CODE(K);
                  END;
               END;
               CALL RESET_OUT_ARRAY;
               CALL DISSOCIATION;
            END;
         ELSE;
         CALL OFFICIAL_OUTPUT(PTR);
         ADDR=PTR->LIST.NEXTMEM->LEARNED.OUT_HEAD;
         CALL DISTINCT(ADDR,DEPTH+1);
         IF DEPTH=6 THEN
            DO;
               TEMPPTR=ADDR->LIST.NEXTMEM;
               TEMPPTR->LEARNED.OUT_HEAD=NULL;
               ADDR->LIST.NEXTMEM=NULL;
            END;
         ELSE;
      END;
   ELSE;
   PTR=PTR->LIST.NEXTLIST;
END; /* WHILE */
```

```
END DISTINCT;
/*=============================================================*/
/*                  OFFICIAL_OUTPUT PROCEDURE                  */
/*=============================================================*/
OFFICIAL_OUTPUT: PROCEDURE(LIST_HEAD);
  DECLARE
   (TEMPPTR,PTR,LIST_HEAD,ADDRESS,TRAIL)          POINTER,
   (FOUND,SAME,DIFFERENT)                         BIT (1),
   (SUM,IR,IL)                        FIXED DECIMAL(2,0),
   INDEX                                          BUILTIN,
   (K,I,J,L,ROW,ROW1,PRE_ROW,COLUMN,SIZE2)
                                      FIXED BINARY(15,0);
ALLOCATE LEARNED;
LOC->LEARNED.OUT_HEAD=NULL;
DO I=1 TO ROW_SIZE BY 1;
   DO J=1 TO COL_SIZE BY 1;
      LOC->LEARNED.MEMORY(I,J)=B(I,J);
   END;
END;
IF LIST_HEAD=NULL THEN
   HEAD=LOC;
ELSE
   LIST_HEAD->LIST.NEXTMEM=LOC;
DO I=1 TO TOTAL_PAT BY 1;
DO K=1 TO IN_PAT BY 1;
   L=(I-1)*IN_PAT+K;
   DO J=1 TO COL_SIZE BY 1;
     OUT(0,J)=INP(L).CODE(J);
   END;
   OUT(0,COL_SIZE+1)=OUT(0,1);
   OUT(0,0)=OUT(0,COL_SIZE);
   CALL RESET_OUT_ARRAY;
   CALL COMPUTE_OUTPUT(1,ROW_SIZE,1,OUT,B);
   PTR=LOC->LEARNED.OUT_HEAD;
   FOUND='0'B;
   TRAIL=PTR;
   DO WHILE(^FOUND & PTR^=NULL);
      TRAIL=PTR;
      L=1;
      FOUND='1'B;
      DO WHILE(FOUND & L<=COL_SIZE);
         IF PTR->LIST.CODE(L)=OUT(ROW_SIZE,L) THEN
            L=L+1;
         ELSE
           FOUND='0'B;
      END;
      IF FOUND THEN
         IF INDEX(PTR->LIST.LETTERS,
                        INP((I-1)*IN_PAT+K).ALPHA)=0 THEN
           PTR->LIST.LETTERS=PTR->LIST.LETTERS||
                                INP((I-1)*IN_PAT+K).ALPHA;
```

```
        ELSE;
      ELSE
        PTR=PTR->LIST.NEXTLIST;
END;
IF ^FOUND THEN
      DO;
        ALLOCATE LIST;
        ADR->LIST.NEXTLIST=NULL;
        ADR->LIST.NEXTMEM=NULL;
        DO L=1 TO COL_SIZE BY 1;
           ADR->LIST.CODE(L)=OUT(ROW_SIZE,L);
        END;
        ADR->LIST.LETTERS=INP((I-1)*IN_PAT+K).ALPHA;
        IF HEAD=LOC THEN
           IF LIST_HEAD->LEARNED.OUT_HEAD=NULL THEN
              LIST_HEAD->LEARNED.OUT_HEAD=ADR;
           ELSE
              TRAIL->LIST.NEXTLIST=ADR;
        ELSE
        IF LIST_HEAD->LIST.NEXTMEM->LEARNED.OUT_HEAD
                                                =NULL THEN
           LOC->LEARNED.OUT_HEAD=ADR;
        ELSE
          TRAIL->LIST.NEXTLIST=ADR;
    END;
      END;
  END;
END OFFICIAL_OUTPUT;
/*=============================================================*/
/*                    INPUT_LETTER_CODE PROCEDURE              */
/*=============================================================*/
INPUT_LETTER_CODE: PROCEDURE(INFILE,PAT,ARRAY,IN,PAT_NO);
  DECLARE
    INFILE                                    FILE VARIABLE,
    (IN(1:25,1:25),DUMMY)                FIXED DECIMAL(2,0),
    1 ARRAY(1:78),
      3 CODE(1:25)                       FIXED DECIMAL(2,0),
      3 ALPHA                                 CHARACTER(1),
    (I,J,K,P,PAT_NO,PAT)                 FIXED BINARY(15,0);
DO P=1 TO PAT BY 1;
   J=(PAT_NO-1)*PAT+P;
   GET FILE(INFILE) LIST(INPATTERN(J).FONT);
   DO I=1 TO COL_SIZE BY 1;
      GET FILE(INFILE) LIST (DUMMY);
      IF THRESHOLD(I)=0 THEN
         IF DUMMY>=1 THEN
            ARRAY(J).CODE(I)=DUMMY;
         ELSE
           ARRAY(J).CODE(I)=-4;
      ELSE
      ARRAY(J).CODE(I)=DUMMY-THRESHOLD(I);
```

```
      IN(P,I)=ARRAY(J).CODE(I);
      INPATTERN(J).CODE(I)=ARRAY(J).CODE(I);
   END;
   GET FILE(INFILE) LIST(ARRAY(J).ALPHA);
   INPATTERN(J).ALPHA=ARRAY(J).ALPHA;
   GET FILE(INFILE) SKIP;
END;
END INPUT_LETTER_CODE;
/*=========================================================*/
/*                  COMPUTE_OUTPUT PROCEDURE               */
/*=========================================================*/
COMPUTE_OUTPUT:PROCEDURE(START,END,STEP,OUT,B);
  DECLARE
   (OUT(0:25,0:26),B(1:25,1:25),SUM,IR,IL)
                                          FIXED DECIMAL(2,0),
   (START,END,STEP,I,J,L,ROW,ROW1,PRE_ROW,COLUMN,SIZE2)
                                          FIXED BINARY(15,0);
DO ROW=START TO END BY STEP;
   PRE_ROW=ROW-1;
   DO COLUMN=1 TO COL_SIZE BY 1;
      IL=OUT(PRE_ROW, COLUMN-1);
      IR=OUT(PRE_ROW, COLUMN+1);
      SUM=ABS(IL)+ABS(IR);
      IF MOD(PRE_ROW,2)=0 THEN /* PRE_ROW IS THE EVEN ROW */
         IF IL=0 THEN
            IF IR<0 THEN
               SUM=-SUM;
            ELSE;
          ELSE
          IF IL<0 THEN
             SUM=-SUM;
          ELSE;
       ELSE         /* PRE_ROW IS THE ODD ROW */
         IF IR=0 THEN
            IF IL<0 THEN
               SUM=-SUM;
            ELSE;
         ELSE
           IF IR<0 THEN
              SUM=-SUM;
           ELSE;
           OUT(ROW,COLUMN)=MAX(SMIN,MIN(SMAX,SUM+
                                        B(ROW,COLUMN)));
   END; /* END OF COLUMN */
   OUT(ROW,COL_SIZE+1)=OUT(ROW,1);     /* WRAP AROUND */
   OUT(ROW,0)=OUT(ROW,COL_SIZE);
END;
END COMPUTE_OUTPUT;
/*=========================================================*/
/*                    INITIALIZE PROCEDURE                 */
/*=========================================================*/
```

```
INITIALIZE:PROCEDURE;
 DECLARE
  (I,J,K,SIZE)                          FIXED BINARY(15,0);

OPEN FILE(THRES) INPUT;
GET FILE(THRES) LIST (ROW_SIZE, COL_SIZE);
PUT FILE(OUTFILE) SKIP(2) EDIT('NUMBER OF ROW = ',ROW_SIZE,
 ' NUMBER OF COLUMN = ',COL_SIZE)(COL(1),A,F(4),A,F(4));
PUT FILE(OUTFILE) EDIT ('THE THRESHOLDS ARE:')(COL(1),A);
PUT FILE(OUTFILE) SKIP(2);
DO I=1 TO COL_SIZE BY 1;
   GET FILE(THRES) LIST (THRESHOLD(I));
   PUT FILE(OUTFILE) EDIT (THRESHOLD(I))(COL(4*I),F(3));
END;
GET FILE(THRES) LIST (NUM_RUN,IN_PAT,TOTAL_PAT);
PUT FILE(OUTFILE) SKIP(2) EDIT ('NUMBER OF RUNS = ',NUM_RUN,
 'IN_PAT=',IN_PAT,'NUMBER OF PATTERNS = ',TOTAL_PAT)
 (COL(1),A,F(4),COL(30),A,F(4));
GET FILE(THRES) LIST (SMAX, MMAX);
PUT FILE(OUTFILE) SKIP(2) EDIT ('SMAX= ',SMAX,' MMAX = ',
 MMAX)(COL(1),A,F(4),A,F(4));
SMIN=-SMAX;
MMIN=-MMAX;
CALL RESET_MEMORY_ARRAY;
CALL RESET_OUT_ARRAY;
END INITIALIZE;
/*============================================================*/
/*          RESET_MEMORY_ARRAY PROCEDURE                      */
/*============================================================*/
RESET_MEMORY_ARRAY: PROCEDURE;
 DECLARE
    (I,J)                                FIXED BINARY(15,0);
DO I=1 TO ROW_SIZE BY 1;
   DO J= 1 TO COL_SIZE BY 1;
      B(I,J)=0;
   END;
END;
END RESET_MEMORY_ARRAY;
/*============================================================*/
/*              RESET_OUT_ARRAY PROCEDURE                     */
/*============================================================*/
RESET_OUT_ARRAY: PROCEDURE;
 DECLARE
    (I,J,SIZE)                           FIXED BINARY(15,0);
SIZE=COL_SIZE+1;
DO I=1 TO ROW_SIZE BY 1;
   DO J= 0 TO SIZE BY 1;
      OUT(I,J)=0;
   END;
END;
END RESET_OUT_ARRAY;
```

```
/*=============================================================*/
/*                    RECOGNITION PROCEDURE                    */
/*=============================================================*/
RECOGNITION: PROCEDURE(L);
 DECLARE
  (P,I,J,K,L,ROW,ROW1,DEPTH,COLUMN,PRE_ROW,SIZE)
                                        FIXED BINARY(15,0),
  (SUM,IL,IR)                           FIXED DECIMAL(2,0),
  PTR                                         POINTER,
  (CONTINUE,GO_ON,DIFFERENT)                   BIT(1);
DO P=1 TO REC_PAT BY 1;
   CONTINUE='1'B;
   PTR=HEAD;
   DEPTH=1;
   DO WHILE(CONTINUE & PTR^=NULL);
      DO I=1 TO COL_SIZE BY 1;
         OUT(0,I)=RECP(P).CODE(I);
      END;
      OUT(0,COL_SIZE+1)=OUT(0,1);
      OUT(0,0)=OUT(0,COL_SIZE);
      NO_RECOGNITION=NO_RECOGNITION+1;
      CALL RESET_OUT_ARRAY;
      CALL COMPUTE_OUTPUT(1,ROW_SIZE,1,OUT,
                                    PTR->LEARNED.MEMORY);
      PTR=PTR->LEARNED.OUT_HEAD;
      GO_ON='1'B;
      DO WHILE(GO_ON & PTR^=NULL);
         DIFFERENT='0'B;
         COLUMN=1;
         DO WHILE(COLUMN<=COL_SIZE & ^DIFFERENT);
            IF (OUT(ROW_SIZE,COLUMN)=PTR->LIST.CODE(COLUMN))
               THEN COLUMN=COLUMN+1;
            ELSE
              DIFFERENT='1'B;
         END;
         IF DIFFERENT THEN
            PTR=PTR->LIST.NEXTLIST;
         ELSE
            DO;
               GO_ON='0'B;
               IF LENGTH(PTR->LIST.LETTERS) > 1 &
                              PTR->LIST.NEXTMEM^=NULL THEN
                  DO;
                    PTR=PTR->LIST.NEXTMEM;
                    DEPTH=DEPTH+1;
                  END;
               ELSE
                 CONTINUE='0'B;
            END;
      END;
END;
```

```
I=INDEX(ALPHABET,RECP(P).ALPHA);
J=INPATTERN(P).FONT;
IF PTR=NULL THEN
   DO;
     REJECTED(I)=REJECTED(I)+1;
     TOTAL_FONT_REJECTED(J)=TOTAL_FONT_REJECTED(J)+1;
     NO_REJECTED=NO_REJECTED+1;
     IF L<14 THEN
        TRAINED.REJECTED=TRAINED.REJECTED+1;
     ELSE
        UNTRAINED.REJECTED=UNTRAINED.REJECTED+1;
   END;
ELSE
  DO;
    IF SUBSTR(PTR->LIST.LETTERS,1,1)=RECP(P).ALPHA
       THEN DO;
         CORRECT=CORRECT+1;
         ACCEPTED(I)=ACCEPTED(I)+1;
         TOTAL_FONT_ACCEPTED(J)=TOTAL_FONT_ACCEPTED(J)
              +1;
         IF L<14 THEN
            TRAINED.ACCEPTED=TRAINED.ACCEPTED+1;
         ELSE
            UNTRAINED.ACCEPTED=UNTRAINED.ACCEPTED+1;
       END;
          ELSE;
        END;
  END; /* P */
END RECOGNITION;
/*=============================================================*/
/*                      COALES PROCEDURE                       */
/*=============================================================*/
COALES:PROCEDURE;
  DECLARE
   (SAME,DIFFERENT)                                    BIT (1),
   (PREV_OUT(1:25,1:25),SUM,IR,IL)        FIXED DECIMAL(2,0),
   (K,I,J,L,ROW,ROW1,PRE_ROW,COLUMN)      FIXED BINARY(15,0);
SAME='0'B;
DIFFERENT='1'B;
K=1;
DO WHILE (^SAME & K<100);
   NO_ITERATION=NO_ITERATION+1;
   ROW1=MOD(K-1,IN_PAT)+1;
   IF K>ROW_SIZE THEN
      L=ROW_SIZE;
   ELSE
      L=K;
   DO I=1 TO COL_SIZE BY 1;
      OUT(0,I)=IN(ROW1,I);
   END;
   OUT(0,COL_SIZE+1)=OUT(0,1);
```

```
OUT(0,0)=OUT(0,COL_SIZE);
CALL COMPUTE_OUTPUT(L,1,-1,OUT,B);
DO ROW=1 TO L BY 1;
   DO COLUMN=1 TO COL_SIZE BY 1;
      IF K^=ROW THEN
         DO;
           IF ((ABS(OUT(ROW,COLUMN))<15 | ABS(PREV_OUT
           (ROW,COLUMN))<15) & OUT(ROW,COLUMN)*
           PREV_OUT(ROW,COLUMN)<=0) THEN
           IF ABS(OUT(ROW,COLUMN))<
                      ABS(PREV_OUT(ROW,COLUMN)) THEN
              IF PREV_OUT(ROW,COLUMN)<0 THEN
                 B(ROW,COLUMN)=B(ROW,COLUMN)-1;
              ELSE
                B(ROW,COLUMN)=B(ROW,COLUMN)+1;
           ELSE
             IF ABS(OUT(ROW,COLUMN))>
                      ABS(PREV_OUT(ROW,COLUMN)) THEN
               IF OUT(ROW,COLUMN)<0 THEN
                  B(ROW,COLUMN)=B(ROW,COLUMN)-1;
               ELSE
                 B(ROW,COLUMN)=B(ROW,COLUMN)+1;
            ELSE
              IF PREV_OUT(ROW,COLUMN)<0 THEN
                 B(ROW,COLUMN)=B(ROW,COLUMN)-1;
                 ELSE
                   B(ROW,COLUMN)=B(ROW,COLUMN)+1;
              ELSE;
           END;
           B(ROW,COLUMN)=MAX(MMIN,MIN(MMAX,B(ROW,COLUMN)));
           PREV_OUT(ROW,COLUMN)=OUT(ROW,COLUMN);
        END;
END; /* ROW */
IF (K>=ROW_SIZE) THEN
   DO;
     ROW1=MOD(K-ROW_SIZE,IN_PAT)+1;
     IF (ROW1=IN_PAT & K>ROW_SIZE) THEN
        DO;
          DIFFERENT='0'B;
          I=1;
          DO WHILE (I<=ROW_SIZE & ^DIFFERENT);
             J=1;
             DO WHILE(J<=COL_SIZE & ^DIFFERENT);
                IF PREV_MEMORY(I,J)=B(I,J) THEN
                   J=J+1;
                ELSE
                  DIFFERENT='1'B;
             END;
             I=I+1;
          END;
          IF DIFFERENT THEN
```

```
                    DO I=1 TO ROW_SIZE BY 1;
                       DO J=1 TO COL_SIZE BY 1;
                          PREV_MEMORY(I,J)=B(I,J);
                       END;
                    END;
                 ELSE;
              END;
           ELSE;
        END;
     ELSE;
     IF ^DIFFERENT THEN
        SAME='1'B;
     ELSE
        K=K+1;
END;
END COALES;
/*=============================================================*/
/*                  DISSOCIATION PROCEDURE                     */
/*=============================================================*/
DISSOCIATION:PROCEDURE;
  DECLARE
   (CONTINUE,DIFFERENT)                              BIT (1),
   SUM                                    FIXED DECIMAL(4,0),
   (PREV_OUT(1:25,1:25),OUTP(1:25,1:25),IR,IL)
                                          FIXED DECIMAL(2,0),
   (I,K,J,L,ROW,ROW1,PRE_ROW,COLUMN)      FIXED BINARY(15,0);
CONTINUE='1'B;
DIFFERENT='0'B;
K=1;
DO WHILE (CONTINUE & K<300);
   NO_ITERATION=NO_ITERATION+1;
   ROW1=MOD(K-1,TOTAL_PAT)+1;
   IF K>ROW_SIZE THEN
      L=ROW_SIZE;
   ELSE
      L=K;
   DO I=1 TO COL_SIZE BY 1;
      OUT(0,I)=IN(ROW1,I);
   END;
   OUT(0,COL_SIZE+1)=OUT(0,1);
   OUT(0,0)=OUT(0,COL_SIZE);
   CALL COMPUTE_OUTPUT(K,1,-1,OUT,B);
   DO ROW=1 TO L BY 1;
      DO COLUMN=1 TO COL_SIZE BY 1;
         IF ((ABS(OUT(ROW, COLUMN))<15 | ABS(PREV_OUT(ROW,
            COLUMN))<15) & OUT(ROW,COLUMN)*PREV_OUT(ROW,
            COLUMN)>0 ) THEN
            IF OUT(ROW,COLUMN)<0 THEN
               B(ROW,COLUMN)=B(ROW,COLUMN)+1;
            ELSE
               B(ROW,COLUMN)=B(ROW,COLUMN)-1;
```

```
        ELSE;
        B(ROW,COLUMN)=MAX(MMIN,MIN(MMAX, B(ROW,COLUMN)));
        PREV_OUT(ROW,COLUMN)=OUT(ROW,COLUMN);
    END;
END; /* ROW */
IF (K>=ROW_SIZE) THEN
    DO;
        ROW1=MOD(K-ROW_SIZE,TOTAL_PAT)+1;
        DO I=1 TO COL_SIZE BY 1;
            OUTP(ROW1,I)=OUT(ROW_SIZE,I);
        END;
        IF (ROW1=TOTAL_PAT & K>ROW_SIZE) THEN
            DO;
                DIFFERENT='0'B;
                ROW1=2;
                DO WHILE (ROW1<=TOTAL_PAT & ^DIFFERENT);
                    I=1;
                    DO WHILE(I<=COL_SIZE & ^DIFFERENT);
                        IF OUTP(1,I)=OUTP(ROW1,I) THEN
                            I=I+1;
                        ELSE
                            DIFFERENT='1'B;
                    END;
                    ROW1=ROW1+1;
                END;
            ELSE;
        END;
    ELSE;
    IF DIFFERENT THEN
        CONTINUE='0'B;
    ELSE
        K=K+1;
END; /* END OF WHILE LOOP */
END DISSOCIATION;
END HH2;
```

PERMISSION TO COPY

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Texas Tech University, I agree that the Library and my major department shall make it freely available for research purposes. Permission to copy this thesis for scholarly purposes may be granted by the Director of the Library or my major professor. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my further written permission and that any user may be liable for copyright infringement.

Disagree (Permission not granted)	Agree (Permission granted)
	Ming-Chih Lee
Student's signature	Student's signature
	Nov. 1, 88
Date	Date