Portland State University

PDXScholar

Dissertations and Theses Dissertations and Theses

11-27-1989

An Approach to Pattern Recognition of Multifont Printed Alphabet Using Conceptual Graph Theory and Neural Networks

Ihab A. Harb
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Part of the Electrical and Computer Engineering Commons

Let us know how access to this document benefits you.

Recommended Citation
Harb, Ihab A., "An Approach to Pattern Recognition of Multifont Printed Alphabet Using Conceptual Graph Theory and Neural Networks" (1989). *Dissertations and Theses.* Paper 3923.
https://doi.org/10.15760/etd.5807

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

AN ABSTRACT OF THE THESIS OF Ihab A. Harb for the Master of Science in Electrical and Computer Engineering presented November 27, 1989.

Title: An Approach to Pattern Recognition of Multifont Printed Alphabet Using Conceptual Graph Theory and Neural Networks

APPROVED BY MEMBERS OF THE THESIS COMMITTEE:

[redacted]

George G. Lendaris, Chairman

[redacted]

Faris Badi'i

[redacted]

Faryar Etesami

This thesis describes an approach for accomplishing a pattern recognition task using conceptual graph theory and neural networks (NNs). The set of patterns to be recognized are the capital letters of six different fonts of the English alphabet, plus two shifted and six rotated versions of each. The letters are represented to the neural network on a 16x16 input grid (256 "sensor lines"). A standard classification encoding for such patterns is to use a 26-bit vector (26 lines at the NN's output), one bit corresponding to each letter. Experiments with such an encoding yielded results with poor generalization capability. A new encoding scheme was developed, based on the conceptual graph formalism. This entailed designing a set of concepts and a set of associated relations appropriate to the upper case letters of the English alphabet. From these, the following were developed: a conceptual graph representation for each letter, a connection matrix for each, and finally, a C-vector and an R-vector representation for each. The latter were used as the output encoding (21 bits) of the NN pattern recognizer. A feed-forward neural network with 256 inputs, 21 outputs, and 2 hidden layers was trained using the back-propagation-of-error algorithm. Results were significantly better than with the more standard encoding. Generalization on fonts improved from 74% to 96%, generalization on rotations improved from 83% to 94%, and finally, generalization on shifts improved from 2% to 93%.

AN APPROACH TO PATTERN RECOGNITION OF MULTIFONT PRINTED ALPHABET USING CONCEPTUAL GRAPH THEORY AND NEURAL NETWORKS

by

IHAB A. HARB

A thesis submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE
in
ELECTRICAL AND COMPUTER ENGINEERING

Portland State University
1989

TO THE OFFICE OF GRADUATE STUDIES:

The members of the Committee approve the thesis of Ihab A. Harb presented November 27, 1989.

[redacted]

George G. Lendaris, Chairman

[redacted]

Faris Badi'i

[redacted]

Faryar Etesami

APPROVED:

[redacted]

Rolf Schaumann, Chairman, Department of Electrical Engineering

[redacted]

C. William Savery, Vice Provost for Graduate Studies and Research

ACKNOWLEDGEMENTS

This research project could not have been made possible without the tremendous amount of help of Dr. George G. Lendaris. In particular, his careful assistance in the development of thoughts and experiments has made the production of this thesis run much more smoothly than I have ever hoped; I owe him my deepest respect and a particular debt of gratitude. I appreciate as well the contribution of Dr. Faris Badi'i on data preprocessing methods for pattern recognition applications. Again, I wish to thank my friends Mohammad Assaf and Shihab Hanayneh for the enthusiasm they showed, and the comments they gave.

Finally, I would like to thank my family for their support, understanding, and encouragement throughout my years of study, helping give in the process a very special meaning to the creation of this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The underlying objective of the research reported herein has been to develop an approach to performing certain pattern recognition tasks using neural networks. This situation may be characterized as shown in Figure 1, wherein an input pattern is presented to a "black box" containing a neural network (NN), and the classification of the input pattern is given by the NN on its output terminals. An important consideration is how to represent the pattern appropriately for NN processing, and in turn, how the NN is to represent the classification to the outside world.

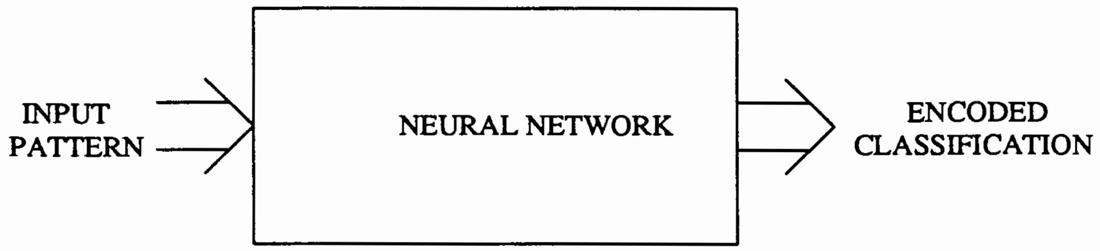


Figure 1. Pattern recognition approach via neural networks.

PATTERN RECOGNITION

Definition

Pattern recognition (by machine) is typically considered as the categorization of input patterns into their respective classes via feature extraction, wherein an individual pattern is characterized by the relations among its constituent features, rather than by the original measurements via which it was acquired [Wiener, 1986]. The following is a typical definition of the pattern recognition problem: there exists a set of N objects divided into M nonintersecting subsets, referred to as object classes (for the problem of this thesis, alphabetic characters); to each object, there corresponds a particular description U; it is required to implement a system which, on the basis of such a description, indicates the class to which a particular object belongs [Tou, 1984].

Available Approaches

The design of pattern recognition systems typically falls into one of two principal approaches: (1) decision theoretic and (2) syntactic, or more generally, structural. The former is based on using decision functions for classifying the patterns. As an example of this concept, Figure 2 shows the result of plotting the values of measurements (features) X and Y for exemplars from two pattern classes, W_1 and W_2. It is easily noted that for this case, a line may be drawn that separates the two classes. Such a line is called a decision boundary, and denoted here by D(X,Y).

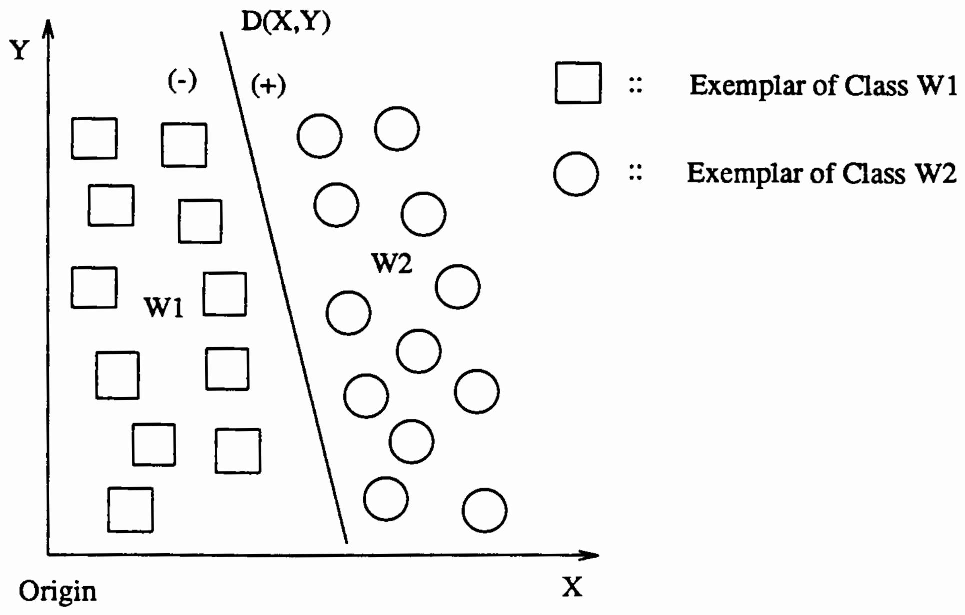


Figure 2. A simple decision function for the separation of classes W_1 and W_2.

In general for this approach, a multi-dimensional feature space is defined for the given problem context, and one or more decision surfaces are derived that define distinct regions in the space, each corresponding to a different one of the classification categories for the problem context. The feature values are measured (or, computed from measurements) for a given exemplar, and the classification given to it is determined by which region in the feature space it falls into. For the present example, the decision surface D(X,Y) is linear, but this is not a requirement.

The syntactic approach, on the other hand, is based on the decomposition of patterns into primitives, encoding each primitive by a symbol, and then representing the pattern as a string of these symbols. Subsequent processing of the patterns is performed on the corresponding string of symbols. Figure 3(b) shows the decomposition of two chromosome structures in terms of the primitives defined in Figure 3(a).

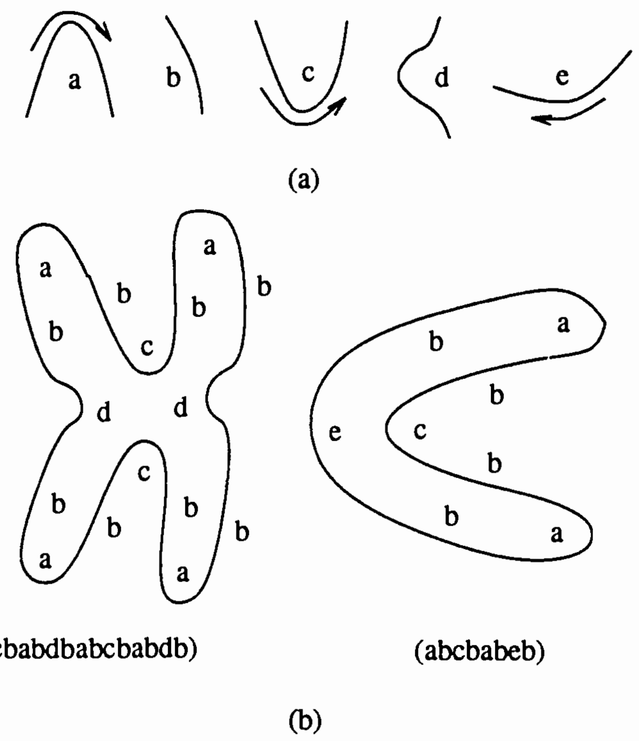


Figure 3. A syntactic description of patterns: (a) primitives; (b) coded chromosomes.

By tracking each chromosome boundary, it is possible to detect and then encode these primitives; the chromosome on the left in the figure may thus be represented by the string "abcbabdbabcbabdb", and the one on the right by the string "abcbabeb".

A NEW APPROACH

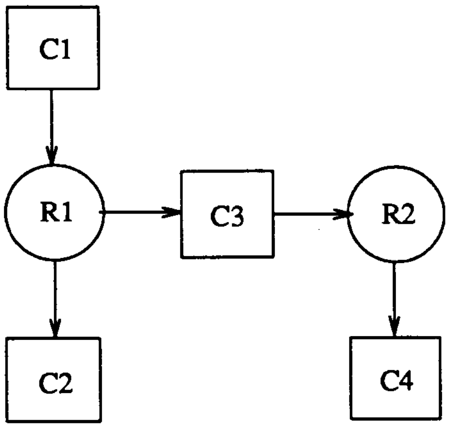
Breaking with the tradition of the syntactic approach, which as indicated above, normally uses sequences of symbols to represent a pattern, this thesis explores a different structural approach, based on a recently developed knowledge representation formalism called conceptual graph theory [Sowa, 1984]. The conceptual graph (CG) methodology suggests itself in the present context because, though not exploited yet, a CG may be used to represent a pattern in a manner that is amenable to parallel processing [Lendaris, 1988a], in contrast to the symbol-string representation cited above, which is typically processed sequentially.

Conceptual Graph Overview

A conceptual graph consists of two kinds of nodes (concept nodes and relation nodes) and of directed arcs. An arc connects a concept node to a relation node, or a relation node to a concept node; connections are not allowed between nodes of the same type [Lendaris, 1988a]. Figure 4 shows the display form of a generic conceptual graph. Here circles are used to denote conceptual relations, and rectangles to denote concepts. An alternative linear form, with parentheses around relations, brackets around concept labels, and arrows to indicate connections is used to represent a conceptual graph (CG) for easier input and output operations in digital computers.

The Connection Matrix

Once conceptual graph representations are realized over a whole problem context, if neural networks (NNs) are to be used to process the CGs, attention must be directed



C1	::	Concept1
C2	::	Concept2
C3	::	Concept3
C4	::	Concept4
R1	::	Relation1
R2	::	Relation2

Figure 4. A generic conceptual graph representation.

towards developing a way of coding the CGs so they may be used as input to the neural networks. A means suggested in [Lendaris, 1988a] is to use a matrix form representation, called R-C Connection Matrix (R-C CM). The R-C CM is defined to contain one row for each relation, and one column for each concept in the given conceptual graph. An entry of 1 is placed in the (row_i, $column_j$) slot of the matrix if there exists a connection from relation $node_i$ to concept $node_j$ in the conceptual graph. A zero entry is made where there is no corresponding connection. A -1 is entered to show a connection in the opposite direction from concept $node_j$ to relation $node_i$ [Lendaris, 1988a]. Using the conceptual graph of Figure 4, a 2x4 R-C Connection Matrix is constructed as shown in Table I.

TABLE I

A 2x4 CONNECTION MATRIX FOR THE CONCEPTUAL GRAPH OF FIGURE 4

Connection Matrix				
	Concept1	Concept2	Concept3	Concept4
Relation1	-1	1	1	0
Relation2	0	0	-1	1

The Vector Form

In general, there will be a large catalog of concepts associated with a given problem context, with say numC entries, and similarly, a large catalog of relations, with numR entries. A template R-C Connection Matrix (R-C CM) consisting of numR rows and numC columns is used to encode each conceptual graph. Clearly, only a small fraction of the slots in the template R-C CM will be filled in for any specific conceptual graph. There are a variety of ways to encode such sparse matrices to make storage more efficient. A method is given in [Lendaris, 1988a] which stores only those rows of the matrix which contain non-zero entries, and along with this, an R-vector to keep track of where the rows came from. The R-vector and C-vectors are defined via the R-C CM: the C-vector has numC slots, and a slot contains a 1 if there are any non-zero entries anywhere in the column of the R-C CM matrix corresponding to the position of the given slot; the R-vector has numR slots, and a slot contains a 1 if there are any non-zero entries anywhere in the row of the R-C CM corresponding to the position of the given slot. If there are no entries in a column (row) of the R-C CM, then the corresponding slot in the C-vector (R-vector) contains a zero. Table II shows an example of a C-vector and of an R-vector, as derived from the example of Figure 4.

TABLE II

C- AND R-VECTORS AS DERIVED FROM TABLE I

C- and R-vectors				
C-vector	1	1	1	1
R-vector	1	1	0	0

In this case, the slots in both vectors will contain 1's in as many rows and columns as the number of relations and concepts used in the CG; thus, all rows and columns have at least one non-zero entry. This will not be true in general.

Neural Networks and Generalization

A neural network (NN) consists of basic computational elements called neurodes (approximations of the neurons in biological brains) and connections which provide paths for the outputs of certain neurodes to serve as inputs to other neurodes. The connection paths have weights associated with them, and these are modified during a process called "training" so the NN ends up performing the desired task properly. The training process begins with a selected architecture (number of neurodes, their general arrangement, and the connections among them) and a selected "training algorithm." A pattern is presented to the NN, the NN generates an output classification according to the current setting of its weights, and if the output classification is wrong, adjustments are made to selected weights (decided by the training algorithm) in such a way that the NN will likely give a different output the next time the same input is presented. A different exemplar from the set of patterns used for training (training set) is presented to the NN, and the the above is repeated. This process is repeated a large number of times until the NN performs the desired classification correctly (if this be possible for the selected architecture).

The specific pattern recognition task chosen for the present research was recognition of upper-case letters of the English alphabet, using a 16x16 pixel, black/white representation for the input. The starting data set consisted of 10 fonts selected from those available on the Macintosh computer, originally encoded into a 12x12 grid, justified to the upper left corner. This data was modified so the letters were each centered in a 16x16 grid, and additional data were generated to represent various shifts and rotations of each of the letters.

A typical approach in neural network pattern recognition research is to train the NN on a representative set of data, and then test the NN on a different set of data. If the NN does well on the new set of data, the NN is said to generalize well; if not, it is said to generalize poorly. In the present context, there are two types of generalization sought:

1) train the NN on some of the fonts, and test it on the remaining fonts -- a generalization on fonts;

2) train the NN on fonts with a couple examples of a transformation (e.g., rotation) for each, and then test the NN on other examples of the transformation (rotation) -- a generalization on the transformation.

Good generalization is an important quality to strive for in developing any pattern recognition system. The conceptual-graph formalism mentioned in the previous section is applied here specifically to assist in developing a neural network implementation that yields better generalizations than has been found otherwise.

The "straight forward" way of applying a neural network to the given pattern recognition task would be as shown in Figure 1, with the input consisting of 16x16 = 256 inputs (one for each pixel), and 26 outputs (one for each letter of the English alphabet). Such an experiment was run, and the NN learned all the fonts in the training set, but was unable to generalize well enough on any of the transformations of the fonts used in the experiment.

The New Approach

An approach for providing better generalization is presented here which consists of two-stages. This approach depends upon developing a conceptual graph representation for the patterns of interest (in this case, capital letters of the English alphabet), with the design requirement that the representation be independent of the rotation, translation, or size of the pattern (letter) in the scanning window (the CG representation developed is also independent of some other distortions, but these were not tested in the present experiments). Once such a conceptual graph representation is designed, a NN is trained with the 16x16 = 256 pixel values as its inputs, and outputs which represent the (previously described) R-vector and C-vector for the given input pattern (letter). As a separate development, another NN is trained with the R-vector and C-vector for a given pattern (capital letter of the English alphabet) at the input, and the NN is to provide the correct answer on one of the output terminals (one for each letter of the English alphabet). After the two NNs are trained to perform the two separate tasks, they can be put in series, with the output of the first providing input to the second. In this way, the tandem combination of the two NNs provides a two-stage, composite, NN to solve the desired pattern recognition problem. See Figure 5.

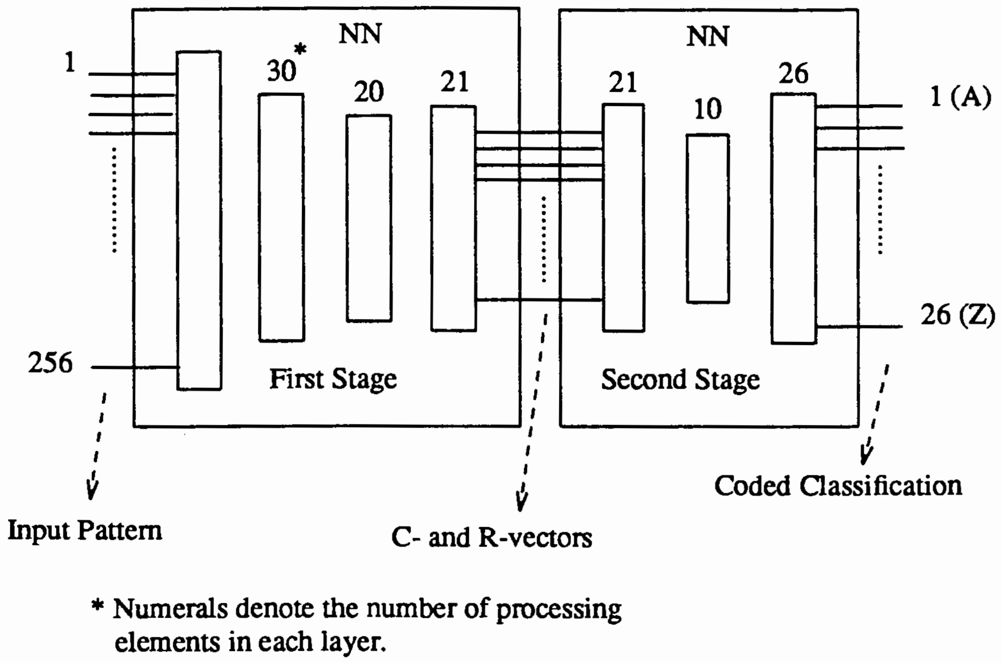


Figure 5. A tandem combination of two neural networks

The remainder of this thesis is organized as follows: Chapter II develops a conceptual graph representation meeting the above design criteria; Chapter III introduces some of the underlying principles of neural networks, and describes the architecture and training algorithm selected for the present research; Chapter IV describes the experiments that were run, and gives the results accomplished; and finally, Chapter V gives some concluding remarks, and suggestions for future research.

CHAPTER II

CONCEPTUAL GRAPH REPRESENTATION OF INPUT PATTERN SET

As mentioned in Chapter I, one of the tasks before us is (if possible) to develop a onceptual graph representation of our input pattern set (capital letters of the English lphabet) that has the properties of rotation, scale, and translation invariance. This hapter describes the concepts and relations that were developed to serve as a basis for uch a representation. A conceptual graph representation for each of the letters was eveloped, and corresponding to each of these, an R-C Connection Matrix and the orresponding R-vector and C-vector were developed. A catalog of the matrix and vector epresentations for each of the 26 letters is given in Appendix D.

KNOWLEDGE SYSTEM DEVELOPMENT

ine Segment Representation

One of the early steps in typical pattern analysis methods is to break up a pattern nto its constituent parts; this process is often called "segmentation." For the case of the etters of the English alphabet, this process would yield a set of straight line segments to epresent the exemplar pattern being processed. One such method is presented in [Badi'i, 983]. For the purposes of the present thesis research, such a procedure is presumed to xist and be available. The concepts and relations defined herein presume that a list of ine segments (described by their endpoints) is available for the exemplar pattern being nalyzed.

Concept Types and Definitions

Concepts are the product of the human mind. They are implemented in modeling the real world, selecting abstract features, and ignoring details and complexities inherent in the real continuous world. Hence, concepts cannot be perfect representations of the world, given that the world is a medium of continuity, and that concepts are discrete in nature; they can only be an applicable approximation [Jaensch, 1930].

Given that the exemplar pattern could be provided as a set of straight line segments, it was decided to experiment with the following list of concept types as building blocks for representing the capital letters of the English alphabet:

1) Long line (LL).

2) Short line (SL).

3) Shorter line [or serif] (SE).

4) Curve (CU).

To simplify the definition of these concepts, it is assumed that the scan window is variably scaled to fit the pattern of interest during input. Once this window is defined, the "long," "short," and "shorter" of concept types 1-3 are defined relative to the greatest side length of the scan window. One possible way to implement such a variable size scan window is as follows:

- During a first phase, determine the horizontal line that passes through the upper-most point on the pattern and define this as the top of the scan window. Similarly, the horizontal line going through the lower-most point on the pattern is used to define the bottom of the scan window.
- During a second phase, determine the vertical line that passes through the left-most point on the input pattern and define this as the left edge of the scan window, with the vertical line passing through the right-most point defining the right edge

of the scan window.

The four concept types mentioned above are defined as follows:

Long Line (LL): A "long line" is defined as a line whose length is 80% or more of the length of the longest side of the scan window.

Short Line (SL): A "short line" is a line whose length is in the range of 20-80% of the length of the longest side of the scan window.

Shorter Line (SE): A "shorter line" (or serif) is a line whose length is less than 20% of the length of the longest side of the scan window.

Curve (CU): A "curve" is a connected sequence of three short lines.

[Note: It was decided here to assume a segmentation algorithm of the type defined in [Badi'i, 1983], which provides the list of line segments in a sequential format. The present definition for "curve" depends on such a representation being available.]

The following is a segment of an algorithm to test for "curve":

(a) Given a short line L_i, test if the next two connected lines are short. If so, the three line segments $L_i, L_{(i+1)}$, and $L_{(i+2)}$ constitute a curve, C_i.

(b) After a curve C_i is defined, let the last line of the curve, $L_{(i+2)}$, be the start of a possible next curve $C_{(i+1)}$, and use (a) to test this possibility.

(c) If the test for $C_{(i+1)}$ fails, then move pointer to the line that would have been $L_{(i+3)}$ relative to the starting point in (a).

Relation Types and Definitions

Concepts by themselves are not sufficient to define a pattern -- the relations among the concepts are also important. The following list of relation types were developed along with the previous list of concept types for characterizing the capital letters of the English alphabet:

1) Touch (T).

2) Abut (A).

3) Intersect (I).

4) Not Available (N/A).

A Boolean function is defined for each of the above, which takes on a value of 1 if the relation holds, and a value of 0 if the relation does not hold. For example, $\mathit{Touch}\ (L_i, L_{(i+1)}) = 1$ if, and only if, L_i touches $L_{(i+1)}$. As a general observation, the Touch and Intersect relation types are commutative, e.g.,

$$\mathit{Touch}\ (L_i, L_{(i+1)}) = \mathit{Touch}\ (L_{(i+1)}, L_i), \tag{2.1}$$
$$\mathit{Intersect}\ (L_i, L_{(i+1)}) = \mathit{Intersect}\ (L_{(i+1)}, L_i). \tag{2.2}$$

The Abut relation, however is not commutative. That is,

$$\mathit{Abut}\ (L_i, L_{(i+1)}) \neq \mathit{Abut}\ (L_{(i+1)}, L_i). \tag{2.3}$$

The four relation types listed above are defined as follows:

[Notation: $L_i\ (m_1, n_1, m_2, n_2)$ is a line with start coordinates (m_1, n_1) and end coordinates (m_2, n_2).]

Touch:

1) Given a line segment $L_i\ (m_1^{(i)}, n_1^{(i)}, m_2^{(i)}, n_2^{(i)})$, there exists another line $L_{(i+1)}\ (m_1^{(i+1)}, n_1^{(i+1)}, m_2^{(i+1)}, n_2^{(i+1)})$, such that (1) the end-point $(m_2^{(i)}, n_2^{(i)})$ coincides with the starting point $(m_1^{(i+1)}, n_1^{(i+1)})$, or (2) the starting point $(m_1^{(i)}, n_1^{(i)})$ coincides with the end-point $(m_2^{(i+1)}, n_2^{(i+1)})$. Similarly, the Touch relation also applies when (3) the end-point $(m_2^{(i)}, n_2^{(i)})$ coincides with the end-point $(m_2^{(i+1)}, n_2^{(i+1)})$, or (4) the starting point $(m_1^{(i)}, n_1^{(i)})$ coincides with the starting point $(m_1^{(i+1)}, n_1^{(i+1)})$. Figure 6(a) shows possibility (1).

2) If one of the lines extends beyond the point of intersection the equivalent amount of a shorter line (serif), this is still considered a Touch. Figure 6(b) shows the situation.

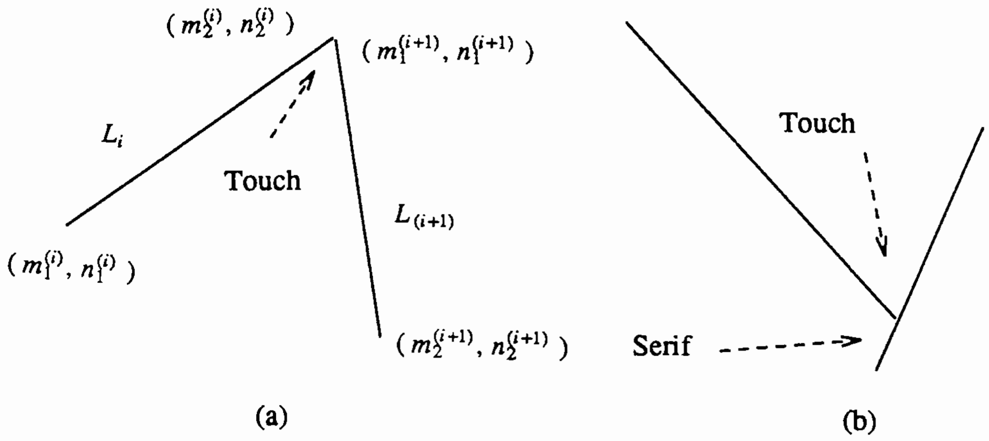


Figure 6. (a) A Example of a Serif-Free Touch Relation, (b) A Touch Relation Allowing Serifs.

Abut:

1) Given a line segment L_i $(m_1^{(i)}, n_1^{(i)}, m_2^{(i)}, n_2^{(i)})$, there exists another line $L_{(i+1)}$ $(m_1^{(i+1)}, n_1^{(i+1)}, m_2^{(i+1)}, n_2^{(i+1)})$ such that the end point $(m_2^{(i)}, n_2^{(i)})$ lies on the line $L_{(i+1)}$, anywhere between $(m_1^{(i+1)}, n_1^{(i+1)})$ and $(m_2^{(i+1)}, n_2^{(i+1)})$. In this case the line L_i is said to *Abut* line $L_{(i+1)}$, with $(m_2^{(i)}, n_2^{(i)})$ at the common point. Figure 7 shows an example of the Abut relation.

2) In this relation, the abutting line (or Agent) does not extend beyond the point of intersection. The abutted line (or Object), however, extends past that same point for a length of type "short." The Abut relation is distinguished from a Touch relation with serifs by means of the length of the extending line: when the line is of type "short," the corresponding relation is an Abut; when the line is of type "shorter," the corresponding relation is a Touch.

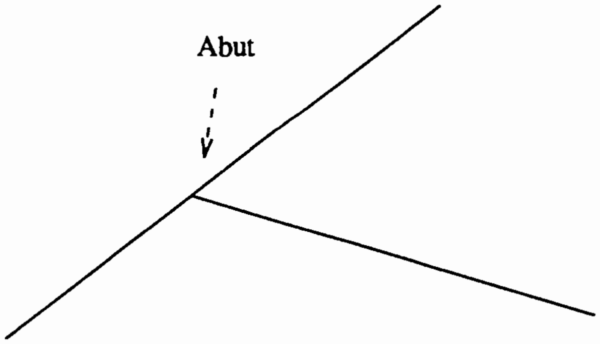


Figure 7. An Example of the Abut Relation.

Intersect:

Given a line segment $L_i\ (m_1^{(i)}, n_1^{(i)}, m_2^{(i)}, n_2^{(i)})$, there exists another line $L_{(i+1)}\ (m_1^{(i+1)}, n_1^{(i+1)}, m_2^{(i+1)}, n_2^{(i+1)})$ sharing a common point with L_i, but the common point of the two (intersecting) lines does not coincide with either the starting nor the ending points of either line. An example of this relation is defined in Figure 8.

Not Available:

This relation is used to denote a line segment that is independent of or not connected to any of the other line segments that comprise a particular input pattern. One example where this would occur, is the letter *I*, which is represented by a single line segment of type "long."

Conceptual Graph Encoding of the Letters

A unique conceptual graph has been developed to represent each of the 26 capital letters of the English alphabet using the above four concept types and four relation types. These CGs were created with the intention that they be independent of specific font characteristics such as serifs or aspect ratios. As desired, these CGs are inherently independent of rotation and a number of elongation distortions of the underlying letters

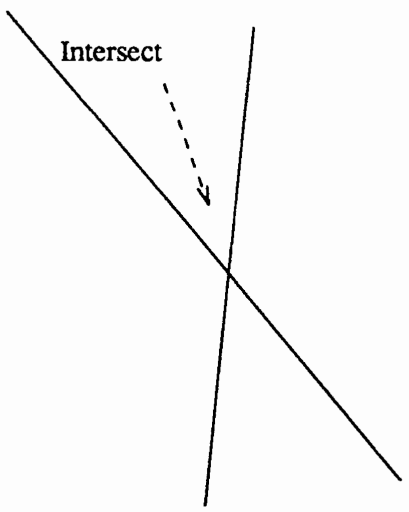


Figure 8. An Example of the Intersection Relation.

they represent. The CGs are also independent of translation and of scale of the letters, but in the process described so far (non neural net), these attributes are already taken care of via the dynamic window scaling. A few example of the conceptual graphs using the Display Form are given in Figure 9.

It was determined that a maximum of 5 Touch relations are involved in the representation of any of the 26 letters, a maximum of 2 Abut relations, and 1 each of the remaining two relations. Similarly, the following limits were determined for the concept types: 4 long lines, 3 short lines and 5 curves. A template R-C Connection Matrix consisting of 9 rows and 12 columns was designed to allow for any of the possible combinations satisfying the above constraints. Appendix D contains Tables D.1-D.26 which show the R-C Connection Matrix and the corresponding C-vector and R-vector for each of the 26 letters, encoded via the specific set of concept types and relation types defined in this chapter. It should be noted that if different concept types and/or relation types had been defined, the conceptual graphs could look significantly different. Another item to point out is that since the representations have been defined to be rotation invariant, the CG for the letter M is identical to the CG for the letter W. Before despairing of this, the reader

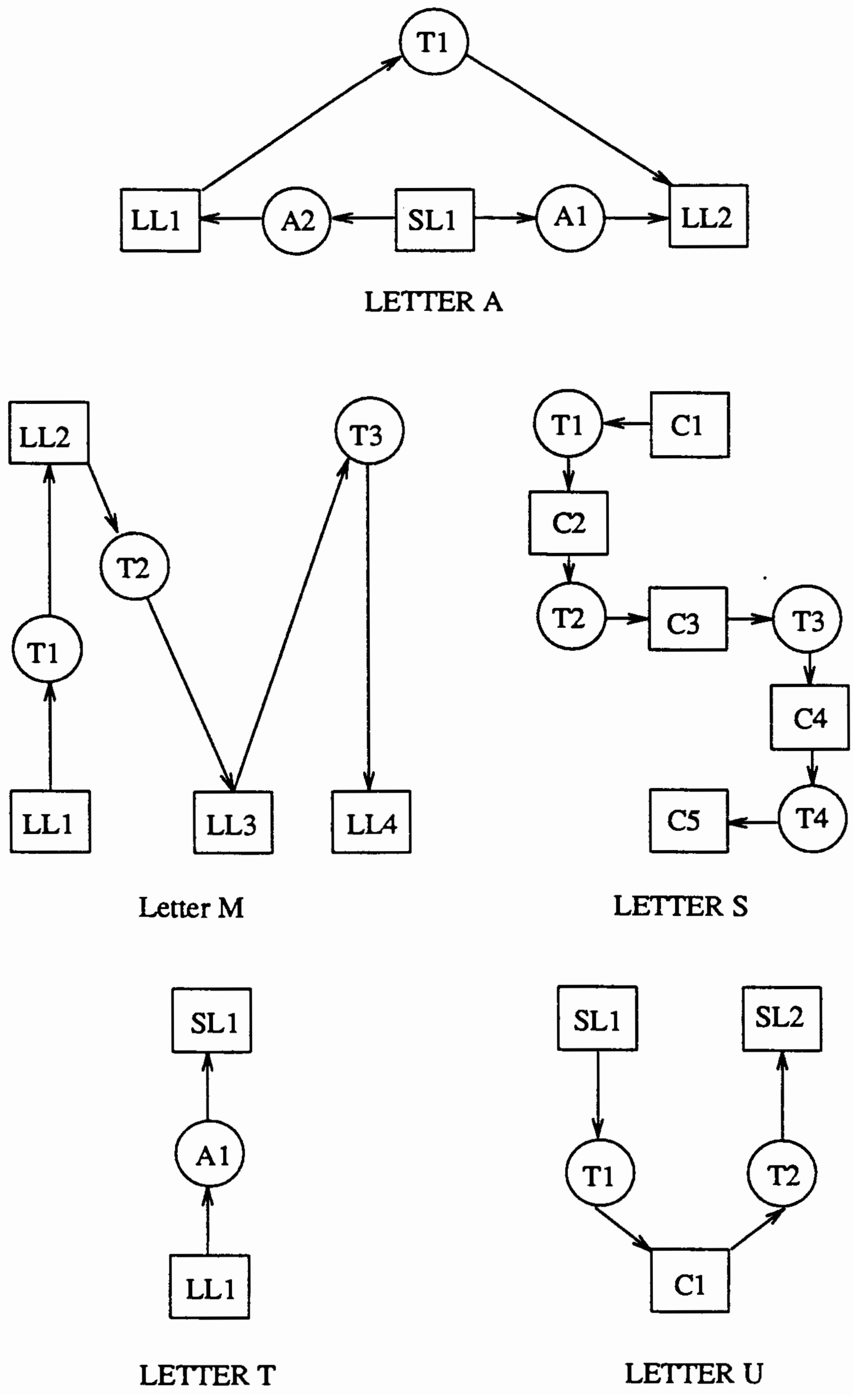


Figure 9. Conceptual Graphs Representing The Letters A, M, S, T, and U respectively.

should note that even a human observer can't tell the difference between a generic W and a generic M unless some collateral information is provided concerning which direction is up. It was decided, therefore, to allow this ambiguity for the experiments undertaken here.

COMMENT

This chapter so far has alluded to a pattern recognition process wherein a segmentation phase would take place which would yield an ordered list of line segments. Each of these line segments was then to be characterized as long, short, or shorter, and following this, the list was to be scanned for the existence of curves. In the process, the four defined relations were to be associated with pairs of lines and/or curves, as appropriate. From this information, a conceptual graph (CG) version of the input pattern would be generated. The rest of the process has not yet been described, but in principle, the next step would be to compare (in some appropriate way) the CG for the input pattern against a set of reference CGs and determine which one the input CG most closely resembles. The result of the latter step would be the desired classification of the input pattern. The above hypothetical process was used as a background context only for the purpose of guiding some of the choices made in defining the concept types, the relation types, and the mechanical processes for determining them. The result is a set of definitions that could in fact be used in a system of the type presupposed. For the neural network application underlying the present research, however, far fewer constraints would have sufficed. It would only have been necessary for a human pattern recognizer to come up with a set of concept types and relation types that could be manipulated using the human visual field, internal processing, and deductive/inductive capabilities to demonstrate that the representation does the job. Such a set of conceptual graph representations could have served the role required of them in the neural network experiments described and reported in the remainder of this thesis.

CHAPTER III

NEURAL NETWORKS

As mentioned in Chapter I, a neural network (NN) consists of basic computational elements and their interconnections. The elements of an NN carry out their computations essentially at the same time (in parallel), and since there are large numbers of elements with large numbers of connections, the adjectives "massively parallel" and "connectionist" are typically applied to neural networks. It should be noted, however, that there are basic distinctions between neural networks and other kinds of current-day parallel computers. In NNs, the computing element is a simple one, contrasted to the (Intel)-386, or -486, level processors in machines such as the Connection Machine, HyperCube, etc. Also, the connections play a fundamental information storage role in NNs, whereas in other parallel machines, they serve primarily a data communication and/or control communication role. In neural networks, the connection paths have weights associated with them, and these are modified during a process called "training" (as distinct from programming), which is intended to yield a network configuration that performs the desired task properly.

HISTORICAL PERSPECTIVE

The following historical comments are provided by my thesis advisor, Dr. George Lendaris:

> The basic ideas underlying the field now called neural networks have their roots in the two works [McCulloch & Pitts, 1943] and [Hebb, 1949]. McCulloch & Pitts developed a model which approximates the first-order properties of the neuron --

the basic computational element in biological brain -- and showed that when appropriately interconnected, collections of such model elements could perform a large variety of logical computations. Donald Hebb made an important observation concerning how the biological brain modifies the connections between neurons, and even today, virtually all training algorithms embody what is now called the Hebbian principle.

During the early 1950's, a seminal line of research was begun by Frank Rosenblatt. Armed with the McCulloch-Pitts model of the neuron, and the observation of Hebb about how connections are modified in biological brain, Rosenblatt set out to determine a methodology for "training" networks of McCulloch-Pitts neural elements to "learn" desired tasks -- in distinction to programming them to perform the task [Rosenblatt, 1962]. Rosenblatt named the system he developed the Perceptron. Fundamental to the Perceptron, and to any trainable system, was the algorithm used for effecting the training. It was proved mathematically that the Perceptron training algorithm converges to a solution (i.e., the NN is guaranteed to learn the given task), if a solution exists. A variety of research efforts were carried out during the 1960's in an attempt to extend the emerging methodology. A key item that constrained progress during that period, however, was the fact that Rosenblatt's guarantee of convergence applied only to single-layer, feed-forward networks. It was demonstrated in [Minskey & Pappert, 1969] that with such a limitation, neural networks could not be expected to perform most "interesting" tasks. For a variety of reasons, the Minsky/Pappert book being a substantial one, research in this field virtually dried up for some 15 years -- a notable exception was Stephen Grossberg, who continued his work throughout this entire period.

Finally, two theoretical developments emerged in the present decade that cleared the way for the large resurgence of research in neural networks currently underway all over the world. One of these was the work of John Hopfield, a physicist at Cal Tech, who developed a training algorithm, complete with proof of convergence, for a fully interconnected network [Hopfield, 1984]. This was a major breakthrough, both for its substance, and, equally important, for the new way of thinking about the dynamics of such networks (energy landscapes, etc.) The second development was the work of David Rummelhart and his group at UC San Diego, who generalized the training rule for one-layer feed-forward networks of the 1960's to multiple-layer feed-forward networks [Rummelhart, et al, 1987]. (As a historical note, it should be noted that this generalization was simultaneously developed by a researcher at Stanford [Parker, 1987]. Last year, it was discovered that this result was contained in a Ph.D. dissertation at Harvard University, Statistics Department, from the 1970's [Werbos, 1974]!)

The theoretical developments cited above helped to raise the vision about the possibilities for new research in neural networks from the level set by the Minsky & Pappert book, which stifled neural-network research for so long. Further, the orders-of-magnitude advances that have occurred in computing hardware technology since the 1960's have made possible experiments now that could not have been undertaken then. In addition, significant advances have been made since the 1960's in related arenas such as knowledge representation and cognitive psychology. The confluence of all these advances have had a dramatic enabling effect, and much creativity is now being applied to this area of research.

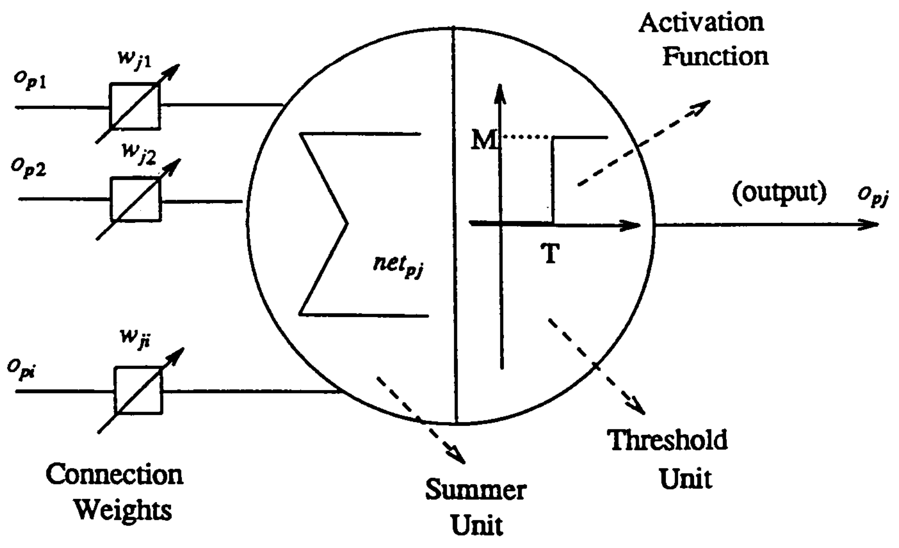
In 1987, a major conference on neural networks was sponsored by the IEEE, attracting some 1500 attendees from a broad range of disciplines; this was repeated in 1988. A professional society for researchers in neural networks (The International Neural Network Society) was inaugurated in 1988, and already has over 3,500 members. In 1989, the IEEE and the INNS held joint conferences (approximately 2000 attendees), and it is planned that these will be held twice a year, one on the east coast and one on the west coast. Research projects in this area have mainly been small efforts so far (with DARPA funding now entering the scene, this may change soon), however, the arena in which this work is being pursued has become very broad. To gain an appreciation for this breadth, the reader may refer to the Proceedings of these conferences [ICNN-87][ICNN-88][INNS-88] [IJCNN-89].

NEURAL NETWORK FUNDAMENTALS

The Computing Element

The basic makeup of the computing element used by neural network researchers follows from the one originally proposed in [McCulloch & Pitts, 1944]. These models go by a variety of names, including, "neurons" (the computer context being presumed), "neurodes" (makes the computer context more explicit), "threshold elements," "processing elements," etc. It will usually be convenient here to simply use the name 'elements.' The basic functions of these elements is described here with the aid of Figure 10. The element has a set of input signals, and a single output signal; each incoming signal is presumed to be the output of another element, so a similar notation is used for both. Each signal coming into an element is multiplied by a variable weight (w_{ij}) before entering the element. If the weight is positive, the input is said to be "excitatory," and if negative, is said to be "inhibitory." After the weighted signal enters the element, in the first half of the element, each of the weighted signals is summed to generate a total input (net_{pj}) as follows:

$$net_{pj} = \sum_i w_{ji} \, o_{pi} \qquad (3.1)$$



$$net_{pj} = \sum_i w_{ji}\, o_{pi}$$
$$o_{pj} = f_j\,(net_{pj})$$

o_{pj} = Current output state of j^{th} element, for a pattern p.
w_{ji} = Weight of the connection joining the i^{th} element to the j^{th} element.
net_{pj} = Weighted sum of input to the j^{th} element.
f (.) = Activation function (in this case, a step function at T).

Figure 10. A model for the neuron: A basic element in neural networks.

The second half of the element passes the net_{pj} signal through a transfer function which yields the element's output signal, o_{pj} . For the example shown in Figure 10, the transfer function is a simple threshold operation. This yields a value of $o_{pj} = 0$ for $net_{pj} < T$, and a value of $o_{pj} = M$ for $net_{pj} \geq T$

Virtually all elements used in NN research have the same left half (i.e., a summer); however, there are a number of variations on the transfer function used for the right half. In particular, the one used for the present research is the one known as the sigmoid function. Mathematically, this is represented as follows:

$$o_{ij} = \frac{1}{1 + \sum_i \exp\,(net_{pj} + T)} \qquad (3.2)$$

The Bias Element

In biological brain, the summed value of the inputs to an element typically must exceed some threshold value before the element generates an output [Hebb, 1949]. The notion of threshold was described earlier in the Computing Element sub-section. Also mentioned there was the mathematical equivalence of putting the negative of the threshold on the left side of the equation. In the network of Figure 13, a "bias" element is shown providing an input to all elements except those in the input layer. The bias element has a fixed output value of 1, but a modifiable weight on each path to the elements it feeds. The signal reaching each element plays the role of the negative of the threshold on the left side of the equation mentioned above. Using this construct, the transfer function of each element is made to be centered about a zero value instead of the threshold value. This method has the advantage that the threshold itself can be adjusted for each of the elements during training.

Geometric Interpretation

There is a useful geometric interpretation to the operation of generating the sum of the weighted input signals. To describe this, let us define a vector O_p $(o_{p1}, o_{p2}, \ldots\ldots, o_{pi})$ to represent the ensemble of input signals; also, define a vector W_j $(w_{j1}, w_{j2}, \ldots\ldots\ldots, w_{ji})$ to represent the corresponding ensemble of weights. It is easily seen that the dot product $O_p \cdot W_j$ carries out the same calculations as those in equation 3.1 above.

In analytic geometry, the dot product of two vectors is used to determine the projection of one vector onto the other. This projection becomes larger as the two vectors come closer into alignment, and is greatest when the two vectors are pointing in the same direction. From this perspective, it may be said that the element will "fire" only for those input signals O_p $(o_{p1}, o_{p2}, \ldots\ldots, o_{pi})$ whose projection on the weight vector W_j $(w_{j1}, w_{j2}, \ldots\ldots\ldots, w_{ji})$ has length larger than some (threshold) value T.

NEURAL NETWORK LEARNING MACHINE

To quote Herbert A. Simon, learning is defined as any change in a system that allows it to perform better the second time on repetition of the same task: this change should be more or less irreversible, in that the learning does not go away rapidly [Simon,1975].

In general, the process of learning in neural networks could entail the modification of any or all of the components comprising the NN that could have an effect on the response of the NN to inputs from its environment. Candidates for modification, therefore, would include the overall layout and connection pattern of the elements (architecture), the transfer function associated with each of the elements, and all of the connection weights. In research reported to date, the algorithm used for training is usually dependent upon the architecture, which once selected, is not change during training. The researcher must, however, take care in selecting the architecture to suit the problem context. Similarly, though a variety of transfer functions are being explored by different researchers, once a transfer function has been selected for a given experiment, it is not typically changed during the training process. This only leaves, then, the weights for modification during training. The main thrust of NN research has been to develop strategies (referred to as learning rules or training algorithms) for modifying the NN's connection weights in a meaningful way, i.e., so the NN will progress closer and closer to a "correct" solution (if one exists for this particular architecture and transfer function selection).

In the neural network context, learning is categorized as supervised or unsupervised. Supervised means that a role of "teacher" is built into the process, where the teacher observes (or provides) the input to the pupil, knows the correct response, monitors the output provided by the pupil, and takes corrective action accordingly. In most research as reported, corrective action is taken only if the pupil gives a wrong response (a "punish" action); if the pupil gives the correct response, nothing is done (i.e., no "reward" action). The training algorithms that derived from the Perceptron during the 1960's incorporated what was called the Delta Rule [Widrow, 1962]. This name referred to the correction increment (or, "delta") to be applied to a given connection weight. Rummelhart, et al [Rummelhart, 1985] adopted the name Generalized Delta Rule, when they modified the Delta Rule algorithm for applications in multi-layered feed-forward nets. The Generalized Delta Rule and its associated architecture were used for the present research. In the unsupervised case, an explicit teacher role is not present. Corrective information is deemed to be provided by the environment, and in this case, only a general right/wrong type of feedback is given, rather than feedback based on knowing exactly what output should have been given.

The Delta Rule

As mentioned above, the Delta Rule is of the supervised learning category. This is described in some detail here, as it (and its generalized version) form the basis of the experiments carried out for this thesis.

During the training process, an input is received by the pupil, and the pupil computes an output according to the settings of all its parameters. The teacher knows the desired (target) output for the given input, and compares the pupil's actual output with the desired output, as shown in Figure 11. The comparison generates an error quantity (notation) defined as:

$$\delta_{pj} = t_{pj} - o_{pj}, \tag{3.3}$$

where t_{pj} is the target output value for input pattern p, and o_{pj} is the output given by element j for input pattern p. The amount to correct the weights on the input lines to this element are given by the following rule:

$$\Delta_p w_{ji} = \eta \, (t_{pj} - o_{pj}) \, o_{pi} = \eta \, \delta_{pj} \, o_{pi}, \tag{3.4}$$

where $\Delta_p w_{ji}$ is the change to be made to the weight on the line going from the i^{th} element to the j^{th} element, η is the learning rate to be discussed later, t_{pj} and o_{pj} are as defined above, and o_{pi} is the value of the component of the input pattern presented to $element_i$.

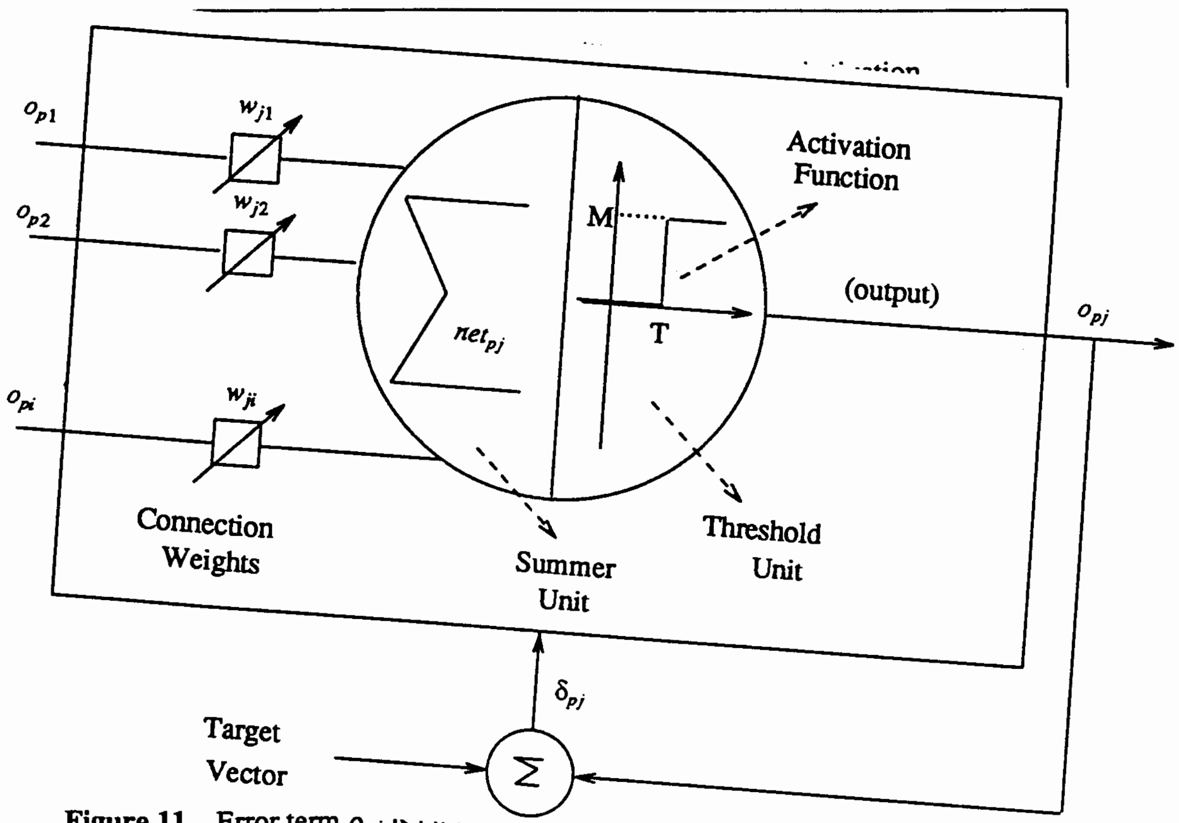


Figure 11. Error term o_{pj} is [illegible]

Mathematical Basis: Gradient Descent

Though the original Delta Rule was crafted based on the intuition provided by the Hebb principle (see historical comments, above), it was later demonstrated [Widrow, 1962] to be mathematically equivalent to a steepest descent procedure on a surface representing the squared value of the error given in Equation 3.3. Thus, training using the Delta Rule was shown to yield a solution which minimizes the squared error over the training samples. When modified properly, it yields the equivalent of the well known LMS (Least Mean Squared) procedure. With hindsight, then, the Delta Rule could have been mathematically derived.

For such a derivation, one must craft a mathematical expression to serve as the "error function" or "criterion function (CF)" which is to be minimized. To serve the needs of the hindsight provided above, the CF is defined in terms of the square of the error defined in Equation 3.2 as follows:

$$E_p = \frac{1}{2} \sum_j (t_{pj} - o_{pj})^2 \; . \tag{3.5}$$

To effect the gradient descent procedure, the magnitude and direction of the gradient vector at the operating point must be calculated. The only variable in Equation 3.5 is o_{pj} Taking the partial derivative of E_p with respect to the o_{pj} yields

$$\frac{\partial E_p}{\partial o_{pj}} = -(t_{pj} - o_{pj}) \equiv \delta_{pj}. \tag{3.6}$$

This equation is not yet useful, because the only parameters available for change in the NN are the connection weights. In the networks considered by Widrow and Hoff, the transfer function used for the neural element was a straight line, i.e., linear. For this case, the output is simply the value of net_{pj}, i.e.,

$$o_{pj} = \sum_i w_{ji} \, o_{pi}. \tag{3.7}$$

This expression can be differentiated with respect to w_{ji}, yielding

$$\frac{\partial o_{pj}}{\partial w_{ji}} = o_{pi}. \tag{3.8}$$

Putting equations (3.6) and (3.8) together, results in the following chain rule:

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}} = \delta_{pj} \, o_{pi}. \tag{3.9}$$

The above is equivalent to having defined an error surface in a "weight space," and to having derived the information needed to make incremental changes in the weights proportional to their contribution to the direction and magnitude of the gradient vector. The right hand side of this is the same as the Delta Rule.

Consider an example with two weights going into one output element. In this case, our "weight space" consists of two dimensions, one for each weight. We add a third dimension, and label it E_p . Assume that we are able to calculate the value of E_p for every possible set of values of (w_1, w_2) . Since E_p is a simple quadratic, the surface will be a quadratic with a unique minimum, as shown in Figure 12.

The minimum point represents the best value for the weights that can be achieved -- for the case where the NN can perform the desired function perfectly, this value will be zero. The network will start with some arbitrary values for (w_1, w_2), and thus will begin at some arbitrary point on the surface. [It is important to note that though the shape is assumed known for the purposes of this discussion, the teacher role does not have this (global) information during the training process. The only information the teacher has is the (local) gradient information.] Using the Delta Rule, the teacher is using the gradient information to guide the pupil, step by step, to the bottom of the surface.

All of the discussion surrounding Figure 12 generalizes to multiple weights for the single-layer, feed-forward case. One then talks about an m-dimensional weight space, where the CF defines a hyper surface, and the gradient is taken on this hyper surface.

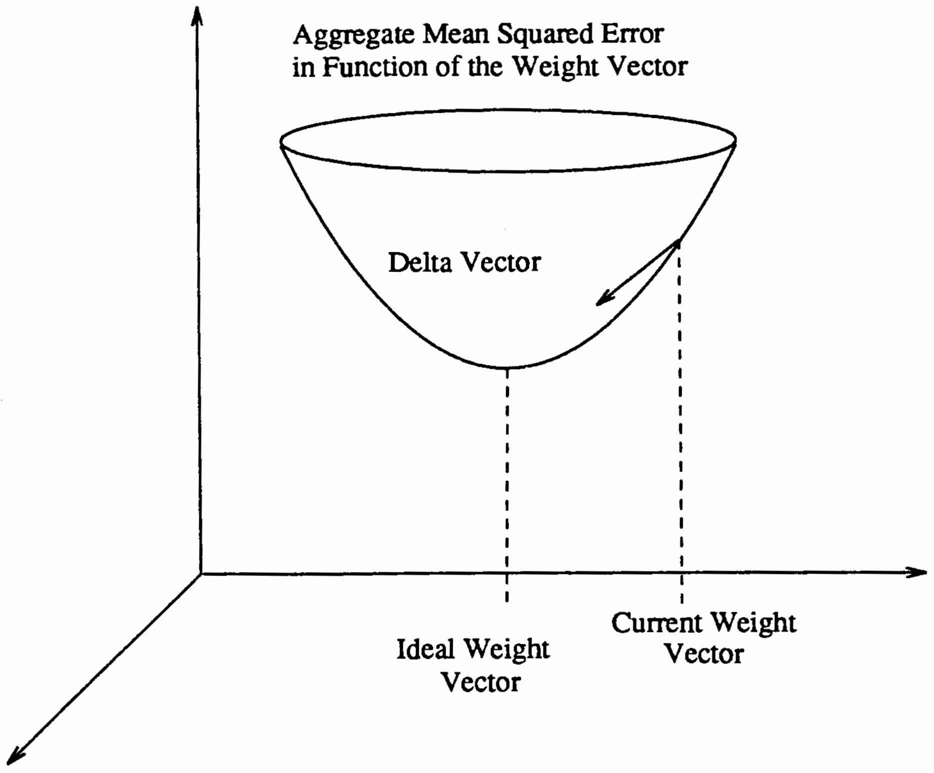

Aggregate Mean Squared Error
in Function of the Weight Vector
Delta Vector
Ideal Weight Vector
Current Weight Vector

Figure 12. Error minimization follows the steepest descent on a surface in weight space.

The Learning Rate: η

A term was included in Equation 3.3 called the learning rate, η . After calculating the gradient at the current operating point, the teacher must decide the size of the step along the path, and in which direction. The value of the multiplier η is used to make this choice. It is desired to go "downhill," therefore a minus sign is made an integral part of the Delta Rule, and thus the value of η must always be positive. From empirical studies, it has been found that a value for η in the interval (0,1) works best. Technically, the mathematics of the derivation requires a (very) small value for η; practically, however, using such small steps requires too many of these steps to reach the destination. On the other hand, if the step size is too big, oscillations could occur. To visualize this, refer back to Figure 12. Assume that the NN is currently very close to the minimum value of the CF surface. If the weights are changed too much (via a large value of the multiplier

η), the NN could end up on the other side of the minimum point; the next change would take the NN back to the other side of the minimum point; etc.

THE BACK-PROPAGATION-OF-ERROR METHOD

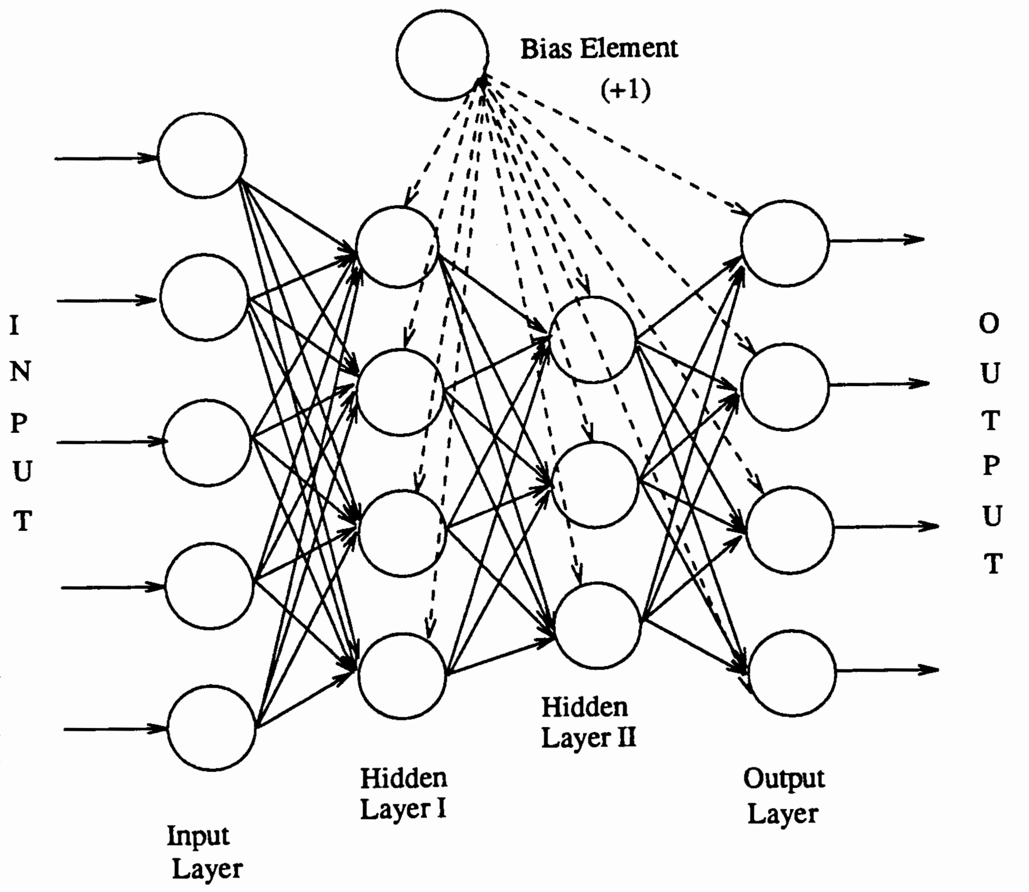
Introduction

As mentioned earlier, the single-layer, feed-forward NNs for which there existed a training algorithm with provable convergence properties (available from the 1960's), could not perform certain important classes of functions. It was shown in [Minskey & Pappert, 1969] that for these classes of functions, feed-forward networks with more than one layer are required. The difficulty encountered by a teacher of a multi-layer net is the so-called "credit assignment" problem, i.e., which weight(s) are to blame when the NN makes an error? The approach taken by Rummelhart, et all [see historical comments above], followed the constructive method used to give Equation 3.9, namely, the use of the chain rule for derivatives. The back-propagation-of-errors algorithm [cf. Training Procedure, pg. 33] is essentially an implementation of the chain rule, in a manner appropriate to the architecture of the NN being trained. The back-propagation-of-errors algorithm (often called, simply, Backprop) applies only to NNs with two or more modifiable layers of weights, with only feed-forward connections allowed. Because derivatives are required (backward, through the elements), the transfer function in the elements have to be differentiable. The threshold function used in the Perceptron does not qualify, as the derivative at the step does not exist. The linear function used for the earlier proof would be too constraining; a nonlinear transfer function was called for. Rummelhart, et al, decided on the sigmoid function (also called "squashing function" in the literature). The description of this was given earlier in Equation 3.2.

Feed-Forward, Multilayer NN Architecture

The architecture required for application of the Backprop algorithm is as shown in Figure 13. The input layer is simply a buffer, and serves to distribute the input signal from the NN's environment to the elements in the next layer. Each path from an element i in the input layer to an element j in the next layer contains a modifiable weight w_{ji}. The outputs of the elements in the second layer may only be fed "forward" to elements of the next layer [in contrast to being fed back around to the input of another element in its own layer, especially to itself]. This process continues for as many layers as are included in the architecture. The first layer has direct contact with the world "outside" the NN, as it receives the input from its environment. Similarly, the last layer has direct contact with the outside world, as this layer provides the outputs that go to the environment. All the in between layers have no direct contact with the outside world, so are said to be "hidden" from the environment. All such layers are called hidden layers. There are no constraints imposed by the Backprop algorithm on the number of elements in each layer, nor on the number of layers in the NN. The number of elements in the input and output layers is determined by the interface from-to the environment, i.e., the "dimensions" of the input vector and of the output vector. The number of hidden layers, and the number of elements in each layer, are determined by the researcher, and this has typically been based on experience and intuition. Research (as yet unpublished) is going on which should provide theoretically based assistance for these choices.

The only variables that are changed during training are the ensemble of modifiable weights on the various paths connecting the elements of one layer to the next. The information associated with the problem being learned is thus contained in these weights. There is no particular weight that contains a particular piece of information from the environment, rather, the information is distributed among all the weights of the NN. The NN is therefore said to have a distributed representation of the information



1) The Bias element is fully connected to both hidden layers, and output layer.

2) Before any training can occur, all the connection weights must be randomized.

Figure 13. The Back-Propagation network: Architecture and flow of information.

associated with the task it is performing, e.g., pattern recognition, associative memory recall, etc.

The Training Procedure

The Backprop training procedure is implemented in a cycle consisting of a forward pass and a backward pass. In the forward pass, the input is fed through the network and the output vector calculated. Once the output vector is available, it is compared with the desired (target) output vector, and an error value is computed. In the backward pass, the error at each output element is "propagated" backward through the various layers, using the equivalent of the chain rule for derivatives method to compute the changes for each weight w_{ji}. This process is implemented in four steps:

1) A pattern vector i is presented at the input buffer, and the computed activities are propagated through to the output layer, where an output value is computed:

$$o_{pj} = f_j\,(net_{pj}). \tag{3.10}$$

2) For each element in the output layer, the local error is computed as defined in Equation 3.11, as well as the corresponding change in weight using Equation 3.12.

$$\delta_{pj} = (t_{pj} - o_{pj})\, f_j'\,(net_{pj}) \tag{3.11}$$

$$\Delta_p\, w_{ji} = \eta\, \delta_{pj}\, o_{pi}, \tag{3.12}$$

where $f_j'\,(net_{pj})$ is the derivative of the element's transfer function (also called activation function).

3) For each succeeding layer (i.e., the hidden layers) -- recall, a specific target value for the output of the hidden layer elements is not known a priori -- the local error is given by Equation 3.13 [Rummelhart, et al., 1985, pg. 324], and the delta weight equation remains the same.

$$\delta_{pj} = f_j'\,(net_{pj}) \sum_k \delta_{pk}\, w_{kj}. \tag{3.13}$$

4) Finally, all the connection weights are updated by adding the delta weight values to the corresponding previous weight values, and another cycle begins. It should be mentioned that the training process must begin with randomizing all the weight values in the NN. If this were not done, say if the weights were all set to +1, the

delta for each weight would be the same, and the weights would each remain identical. This would not be very interesting.

The Momentum Term

The Back-Prop method, like all other gradient descent algorithms, follows a downward path on the weight surface defined by the gradient. A problem arises related to complex surface shapes (of the Criterion Function in the weight space) such as those typical for multi-layer NNs. If the surface has a long gentle slope, then one would like to take large steps to improve efficiency of the search; if there is a steep slope, the steps must be smaller, to avoid missing important landscape features and/or to avoid getting into oscillations. To assist in this process, researchers have developed a way of incorporating the equivalent of a little memory in the delta rule, so that if the slope is gentle, bigger steps are taken, and if steep, smaller steps are taken. The term added has been given the name "momentum term," because is has the equivalent effect of momentum for an object with mass moving along a terrain. The equation for Generalized Delta Rule including a momentum term is given as

$$\Delta w_{ji}(n+1) = \eta \, \delta_{pj} \, o_{pi} + \alpha \, \Delta w_{ji}(n). \tag{3.14}$$

We note that The momentum term consists simply of the previous weight change, multiplied by another coefficient. The latter coefficient must be selected by the researcher, based on empirically derived experience.

CHAPTER IV

EXPERIMENTS AND RESULTS

As mentioned in Chapter I, the pattern recognition task explored in this thesis was based on a set of 10 fonts of upper case letters of the English alphabet selected from those available on the Macintosh computer. The original data was provided to us on a 12x12 upper-left-corner justified grid representation of the individual letters. In order to provide more space in which to carry out the rotations and translations desired for the experiments, the grid was enlarged to 16x16, and the letters were moved to the center of this new grid. This 16x16 data set is shown in Appendix A.

In Chapter III, it was noted that a feed-forward, multi-layer neural network with the back-propagation-of-errors training algorithm was selected for the experimental work discussed here. The elements of such NNs have values on their output lines ranging from 0.0 to 1.0, but generally not reaching either limit due to the sigmoid transfer function. It is incumbent on the researcher to assign what the values on the lines on the output elements of the NN are to mean. In the present context, each line corresponds to a letter of the alphabet, so the value on a line is here assigned the meaning of "confidence that the present input pattern is the letter represented by the line." A given output line is said to cast a vote for its letter when the confidence is higher than some specified threshold value T. The kinds of answers the NN could give then are as follows:

i. none of the lines have values larger than the threshold, therefore no letter is voted for;

ii. exactly one line has a value larger than the threshold, and thus, exactly one letter is voted for;

iii. many lines have a value larger than the threshold, thus the NN is said to vote for more than one letter.

n case ii, if the vote is for the correct letter, all is well; if the vote is for the wrong letter, he NN is said to have an error of committing to the wrong letter (misclassification). In case i, the NN is said to commit to nothing (no vote). All the experiments yielded NNs that avoided the case iii situation. Later in this chapter, experimental results are given for various values of T, to show what sensitivity, if any, there is on the choice of the threshold.

Also mentioned in Chapter III was the fact that during the training process, the patterns from the training set are presented to the NN many times, until the desired mapping is learned. A count is kept of the number of presentations, and is used to label progress of the NN's learning performance.

PREPARATION OF TRAINING AND GENERALIZING DATA SETS

Partitioning of Alphabet Data Set

The 10 fonts in the original data set were studied and partitioned into subsets according to perceived similarities/dissimilarities of their general form. This partitioning is shown in Appendix A. Six of the fonts were grouped together based on their structural members being 1 pixel (picture element) wide, and the shapes of the letters being generally the same. This grouping is called Font Set I. Variations in this set arise from different sizes of the letters, and/or presence/absence of serifs on the letters. Four of the fonts in Set I were selected to be the Training Set, and called Font Set I-A. The remaining two comprise the font Generalize Set, and called Font Set I-B. Set I-B contains one of the three larger sized fonts among the six, and one of the smaller sized fonts.

The remaining 4 fonts comprise Font Set II, also divided into two subsets. Font Set II-A contains two fonts that are noticeably smaller than the fonts in Set I (particularly in the horizontal direction). Font Set II-B contains the two fonts that are significantly different from those in Set I, in particular, the structural members out of which the letters are constructed are all 2 pixels wide (vs. 1 pixel wide for the other fonts), and in addition, the shapes of the last font are also observed to be substantially different.

The experiments reported in this chapter are based on Font Set I.

Transformations of the Input Data Sets

Since a key issue in the performance of a pattern recognition device is how well it generalizes to patterns not seen during training, a number of transformed versions of the basic fonts were created. One set of generalization experiments are based on training with Set I-A and then testing on Set I-B; this tests for generalization on fonts. Beyond this, it was desired to test for generalization on translations and on rotations of the letters. To accomplish this, 8 transformed versions of the base fonts were generated. Two are for translation: +3 pixel (right) shift, and -3 pixel (left) shift. Six are for rotation: $\pm 45°$,$\pm 22.5°$, and $\pm 11.25°$. Examples of these transformations are shown in Appendix B. Generalization tests included 1) training on $\pm 45°$ and $\pm 22.5°$ and then testing for generalization with the $\pm 11.25°$ rotation and the original font (corresponding to a $0°$ rotation); and 2) training on the two shifted fonts, and then testing with the original font (corresponding to a 0-pixel shift).

GENERALIZATION EXPERIMENTS USING THE BASE NN CONFIGURATION

Selection of (First) Neural Network Configuration

The task to be learned by the NN is determined by the representation used for the inputs and the representation used for the outputs (cf. Figure 1). By altering these representations, the task is made easier or harder for the NN to learn. As mentioned above, a particular issue is that the NN be able to generalize well.

The starting (or, base) choice made for the present experiments was to let the NN have 256 input lines, each one receiving a 0/1 input from one of the 16x16 grid points used to represent the letters. The output of the NN was designed to consist of 26 lines, one for each of the letters of the English alphabet. Accordingly, the basic NN configuration consisted of 256 processing elements (PEs) at the input layer, 26 PEs at the output layer, and two arbitrarily chosen hidden layers: the first hidden layer having 30 PEs, the second having 20 PEs. This (base) configuration is used in all the following experiments.

The experiments described next are organized into three groupings:

1) generalization on fonts [experiment NN-1]

2) generalization on rotations [experiment NN-2]

3) generalization on shifts [experiment NN-3].

<u>Generalization on Fonts [Experiment NN-1].</u> As mentioned earlier, Font Set I was divided into the subsets I-A and I-B for this experimental objective. The training set for this part of the experiment consisted of Font Set I-A plus all of its 8 transformed versions (936 patterns), and the generalize set consisted of Font Set I-B plus all of its 8 transformed versions (468 patterns). This experiment required about 70,000 presentations to reach 100% accuracy on the training set. For all the experiments reported here, performance of NNs on the training set was typically monitored at intervals of 10,000 presentations, and when the NN reached its peak performance, the training was stopped. The training dynamics for experiment NN-1 are shown in Figure 14 [to maintain the flow of the text, all the figures and tables for this discussion are accumulated at the end of the chapter]. The percent correct classification of experiment NN-1 on the training set and on the generalizing set are tabulated in Table III for 4 different values of the threshold, T. The best performance normally shows up with T=0.6, so this value is used for all the

comparisons; the errors for NN-1's generalization test are listed in Table IV. The 74.5% correct generalization performance may at first seem rather good, but from a real-world applications point of view, it is not acceptable -- something in the high 90's is required.

Generalization on Rotations [Experiment NN-2]. The training set for this experiment consisted of the ±45° and ±22.5° rotations of Font Set I-A (416 patterns), and the generalize set consisted of the ±11.25° and 0° rotations of this font set (312 patterns). As for all experiments, an untrained version of the base neural network configuration, NN-2, was used for this experiment. The experiment was terminated after about 176,000 presentations of input patterns, with snap-shots of its performance taken every 10,000 presentations. In this case, the peak performance on the training set was at the 98.8% level, i.e., there were still errors on some of the patterns in the training set after the training was deemed complete; these errors are listed in Table VI. The training dynamics for this experiment are shown in Figure 15, and the percent correct classification on the training and generalization sets for various values of T are tabulated in Table V. The errors for the (T=.6) generalization test are listed in Table VII. The 83.2% correct generalization performance is better than in the case of experiment NN-1, but still not good enough from the point of view of real-world applications.

Generalization on Shifts [Experiment NN-3]. The training set for this experiment consisted of the +3 (right) and -3 (left) pixel shifts of Font Set I-A (208 patterns), and the generalize set consisted of the base (unshifted) Font Set I-A (104 patterns). Another untrained version of the base neural network configuration was used for this experiment. In the present case, some 20,000 presentations were needed to learn the task perfectly, so performance snap-shots were taken every 5,000 presentations. The training dynamics for this part of the experiment are shown in Figure 16, and the percent correct classification of on the train and generalize sets for various values of T are tabulated in Table VIII. The errors for the (T=.6) generalization test are listed in Table IX. The generalization performance is seen to be extremely poor, with only approximately 2% correct classifications. Requiring that the input pattern be accurately located to within 2-3 pixels will typically not be acceptable in real-world applications.

GENERALIZATION EXPERIMENTS USING AN NN CONFIGURATION INCORPORATING THE CONCEPTUAL GRAPH APPROACH

Selection of an NN Configuration for the New Approach

A new NN system structure was developed in an attempt to improve the poor generalization accomplished with a "standard" representation/encoding method. The new approach provides a different input/output representation, based on the conceptual graph ideas described in Chapter II. Referring back to Figure 5 (page 9), this new structure divides the task into two stages. In the first stage, the C- and R-vector representations derived in Chapter II are employed as the output, in place of the 26-bit output vector used in the original structure (used in experiments NN-1, NN-2, and NN-3). This first stage is trained to give on its output lines the C- and R-vector representations corresponding to the letter presented on its input lines. The structure of this first stage is the same as in the original structure, except that there are a different number of output elements (corresponding to the dimensions of the C- and R-vectors -- in this case, 21). The second stage of the NN system uses the C- and R-vector representation of a letter at its input lines, and has 26 output lines, one for each letter of the alphabet.

The basic idea for the two-stage NN system is to develop the second stage once and for all to translate the C- and R-vector representation into a 26-bit representation (one bit for each letter). Having done this, the experimental objective shifts to training the first stage to yield the C- and R-vector representation for the 16x16 pixel pattern on its 256 input lines. When this is accomplished, the two stages are connected, and the desired overall input/output mapping is achieved.

The experiments to be described next relate to training the first stage to give the C- and R-vector representations corresponding to the pattern given at its inputs. These experiments are organized into the same three groupings as in the previous section on experiments.

Generalization on Fonts [Experiment NN-A]. The training set for this part of the experiment consisted of Font Set I-A plus all of its 8 transformed versions (936 patterns), and the generalize set consisted of Font Set I-B plus all of its 8 transformed versions (468 patterns) [same as in experiment NN-1]. The first stage of the new neural network system in experiment NN-A, was trained for about 400,000 presentations, with snap-shots of its performance taken every 10,000 presentations. The training dynamics for this part of the experiment are shown in Figure 17, and the percent correct classification on the train and generalize sets for 4 values of T are tabulated in Table X. NOTE THE SIGNIFICANT IMPROVEMENT (21.7%) in the results from those yielded in experiment NN-1 (cf. Table III). The performance here in the high 90 percent category makes this a good beginning toward satisfying real-world requirements. A listing of the generalization test errors for this experiment are given in Table XI.

Generalization on Rotations [Experiment NN-B]. The training set for this experiment consisted of the ±45° and ±22.5° rotations of Font Set I-A (416 patterns), and the generalize set consisted of the ±11.25° and 0° rotations of this font set (312 patterns) [same as in experiment NN-2]. An untrained version of the first stage NN was used for this experiment. In this experiment, the Neural network was trained with 240,000 presentations, with snap-shots of its performance taken every 10,000 presentations. The training dynamics for this part of the experiment are shown in Figure 18, and the percent correct classification on the train and generalize sets for various values of T are tabulated in Table XII. The errors for the (T=.6) generalization test are listed in Table XIII. Again, there is a significant improvement (this time 10.4%) in the results from those yielded by experiment NN-2 (cf. Table V).

Generalization on Shifts [Experiment NN-C]. The training set for this experiment consisted of the +3 (right) and -3 (left) pixel shifts of Font Set I-A (208 patterns), and the generalize set consisted of the base (unshifted) Font Set I-A (104 patterns) [same as in experiment NN-3]. Another untrained version of the first stage NN was used for this experiment. In this experiment, the NN was trained with 170,000 presentations, with snap-shots of its performance taken every 10,000 presentations. The training dynamics for this part of the experiment are shown in Figure 19, and the percent correct classification on the train and generalize sets for various values of T are tabulated in Table XIV. The errors for the (T=.6) generalization test are listed in Table XV. This time, the improvement in the results are DRAMATIC: from 1.9% correct generalization to 93.3% correct generalization. An implementation that relieves the requirement for tight position control of the input pattern is significant.

TABLE III

PERCENT CORRECT CLASSIFICATION ON TRAINING AND GENERALIZATION FOR EXPERIMENT NN-1

Threshold	Training Results	Generalization Results
0.4	82.1%	25.4%
0.5	82.1%	27.1%
0.6	100%	74.5%
0.7	79.3%	0%
0.8	79.3%	0%
0.9	77.8%	0%

Note: The training set consists of font set I-A plus the ±45°, ±22.5°, ±11.25° rotations, and the ±3-pixel (left and right)shifts (936 patterns). The generalization set consists of font set I-B plus the same 8 transformations (468 patterns).

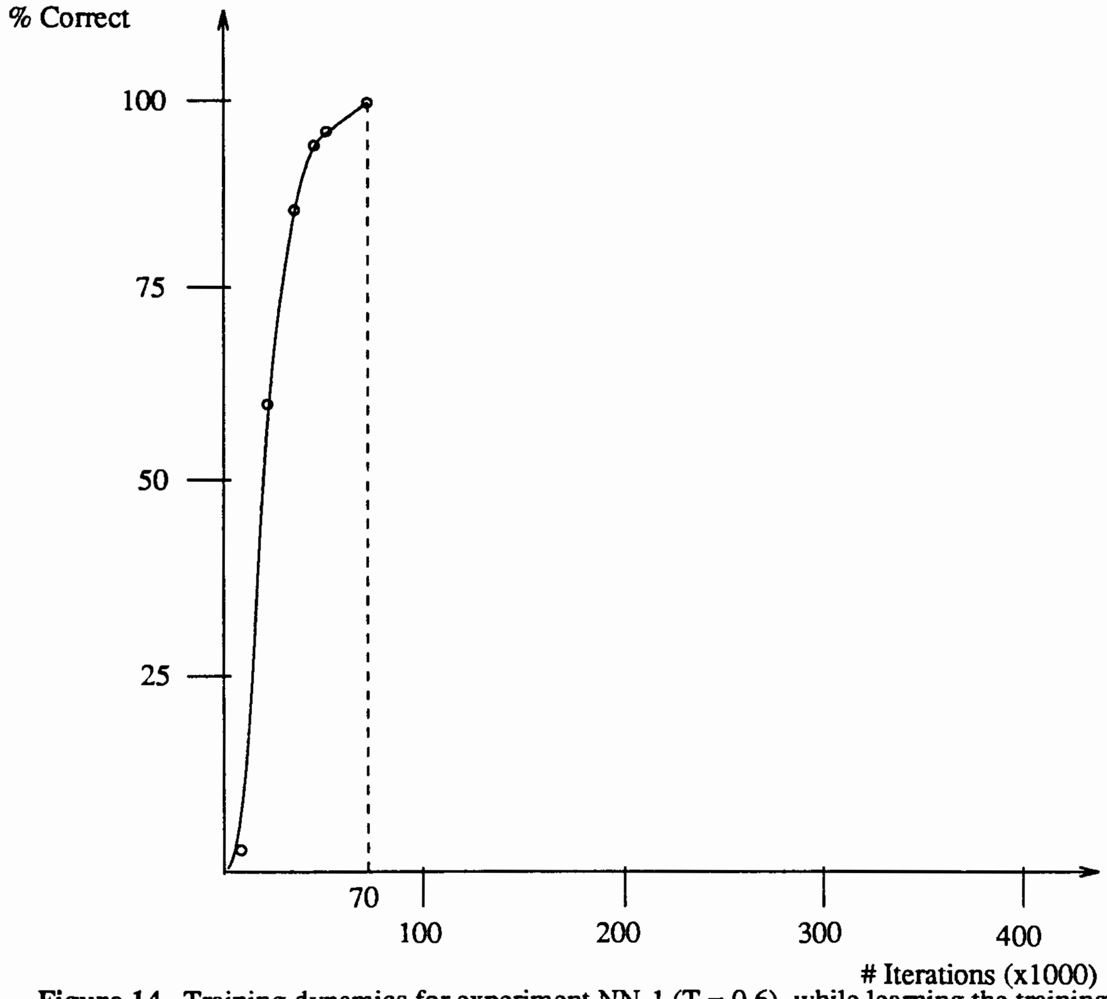


Figure 14. Training dynamics for experiment NN-1 (T = 0.6), while learning the training set consisting of font set I-A plus the ±45°, ±22.5°, ±11.25° rotations, and the ±3-pixel (left and right) shifts.

TABLE IV

ERRORS ON GENERALIZATION TEST
FOR EXPERIMENT NN-1

Errors (74.5% correct, T = 0.6)			
Letter	Error Type	Font	Transformation
A	?	Helvetica	+ 45 °
E (B)	#	Helvetica	+ 45 °
F (E)	#	Helvetica	+ 45 °
G (M)	#	Helvetica	+ 45 °
K	?	Helvetica	+ 45 °
O	?	Helvetica	+ 45 °
Q	?	Helvetica	+ 45 °
R (E)	#	Helvetica	+ 45 °
W	?	Helvetica	+ 45 °
X	?	Helvetica	+ 45 °
Y (F)	#	Helvetica	+ 45 °
B	?	New York	+ 45 °
C	?	New York	+ 45 °
D (G)	#	New York	+ 45 °
G	?	New York	+ 45 °
H	?	New York	+ 45 °
K	?	New York	+ 45 °
N	?	New York	+ 45 °
O (C)	#	New York	+ 45 °
P (R)	#	New York	+ 45 °
Q	?	New York	+ 45 °
R	?	New York	+ 45 °
U (Y)	#	New York	+ 45 °
X	?	New York	+ 45 °
Y	?	New York	+ 45 °
A (W)	#	Helvetica	+ 22.5 °
B (H)	#	Helvetica	+ 22.5 °
D (O)	#	Helvetica	+ 22.5 °
E	?	Helvetica	+ 22.5 °
O	?	Helvetica	+ 22.5 °
P	?	Helvetica	+ 22.5 °
Q	?	Helvetica	+ 22.5 °
R (S)	#	Helvetica	+ 22.5 °
W (N)	#	Helvetica	+ 22.5 °
Y	?	Helvetica	+ 22.5 °
B	?	New York	+ 22.5 °

TABLE IV

ERRORS ON GENERALIZATION TEST FOR EXPERIMENT NN-1 (Continued)

Errors (74.5% correct, T = 0.6)			
C (G)	#	New York	+ 22.5 °
D (U)	#	New York	+ 22.5 °
J	?	New York	+ 22.5 °
O (C)	#	New York	+ 22.5 °
P	?	New York	+ 22.5 °
Q	?	New York	+ 22.5 °
R (K)	#	New York	+ 22.5 °
U (W)	#	New York	+ 22.5 °
Y	?	New York	+ 22.5 °
A	?	Helvetica	+ 11.25 °
B (H)	#	Helvetica	+ 11.25 °
D (O)	#	Helvetica	+ 11.25 °
P	?	Helvetica	+ 11.25 °
Q (S)	#	Helvetica	+ 11.25 °
X (H)	#	Helvetica	+ 11.25 °
C	?	New York	+ 11.25 °
D (O)	#	New York	+ 11.25 °
J	?	New York	+ 11.25 °
O	?	New York	+ 11.25 °
P	?	New York	+ 11.25 °
S (R)	#	New York	+ 11.25 °
V	?	New York	+ 11.25 °
W	?	New York	+ 11.25 °
A	?	Helvetica	- 11.25 °
P	?	Helvetica	- 11.25 °
Q (O)	#	Helvetica	- 11.25 °
S	?	Helvetica	- 11.25 °
U	?	Helvetica	- 11.25 °
W (N)	#	Helvetica	- 11.25 °
C (O)	#	New York	- 11.25 °
D	?	New York	- 11.25 °
J	?	New York	- 11.25 °
O	?	New York	- 11.25 °
P (R)	#	New York	- 11.25 °
Q	?	New York	- 11.25 °
S	?	New York	- 11.25 °
V (C)	#	New York	- 11.25 °

TABLE IV

ERRORS ON GENERALIZATION TEST FOR EXPERIMENT NN-1 (Continued)

Errors (74.5% correct, T = 0.6)			
X	?	New York	- 11.25 °
A	?	Helvetica	- 22.5 °
H (R)	#	Helvetica	- 22.5 °
P	?	Helvetica	- 22.5 °
Q	?	Helvetica	- 22.5 °
R	?	Helvetica	- 22.5 °
W	?	Helvetica	- 22.5 °
C	?	New York	- 22.5 °
G (Q)	#	New York	- 22.5 °
J	?	New York	- 22.5 °
Q	?	New York	- 22.5 °
X	?	New York	- 22.5 °
Y	?	New York	- 22.5 °
B	?	Helvetica	- 45 °
H (M)	#	Helvetica	- 45 °
P	?	Helvetica	- 45 °
O	?	Helvetica	- 45 °
R	?	Helvetica	- 45 °
W	?	Helvetica	- 45 °
G (O)	#	New York	- 45 °
O	?	New York	- 45 °
P	?	New York	- 45 °
R	?	New York	- 45 °
U	?	New York	- 45 °
R	?	Helvetica	center
U	?	Helvetica	center
W	?	Helvetica	center
C	?	New York	center
G	?	New York	center
M (H)	#	New York	center
U	?	New York	center
X	?	New York	center
C	?	Helvetica	left
D (O)	#	Helvetica	left
G (Q)	#	Helvetica	left
Q	?	Helvetica	left
C	?	New York	left

TABLE IV

ERRORS ON GENERALIZATION TEST FOR EXPERIMENT NN-1 (Continued)

Errors (74.5% correct, T = 0.6)			
F	?	New York	left
H (X)	#	New York	left
J	?	New York	left
N	?	New York	left
U (C)	#	New York	left
Y (N)	#	New York	left
A	?	Helvetica	right
G	?	Helvetica	right
M	?	Helvetica	right
W	?	Helvetica	right
C	?	New York	right
G	?	New York	right
J	?	New York	right
M	?	New York	right
X	?	New York	right

Note: The generalization set consists of set I-B plus the ±45°, ±22.5°, ±11.25° rotations, and the ±3-pixel (left and right) shifts. A ? denotes no vote, and a # denotes misclassification. Letters in parenthesis denote the network's misclassification of the letter in question.

TABLE V

PERCENT CORRECT CLASSIFICATION ON TRAINING AND GENERALIZATION FOR EXPERIMENT NN-2

Threshold	Training Results	Generalization Results
0.4	87%	47.1%
0.5	89.1%	82.5%
0.6	98.1%	83.2%
0.7	97.9%	80.1%
0.8	97.9%	78.7%
0.9	97.9%	74.9%

Note: The training set consists of the ±45°, ±22.5° rotations of font set I-A (416 patterns). The generalization set consists of the ±11.25° rotation, and the 0° rotation of fonts set I-A (312 patterns).

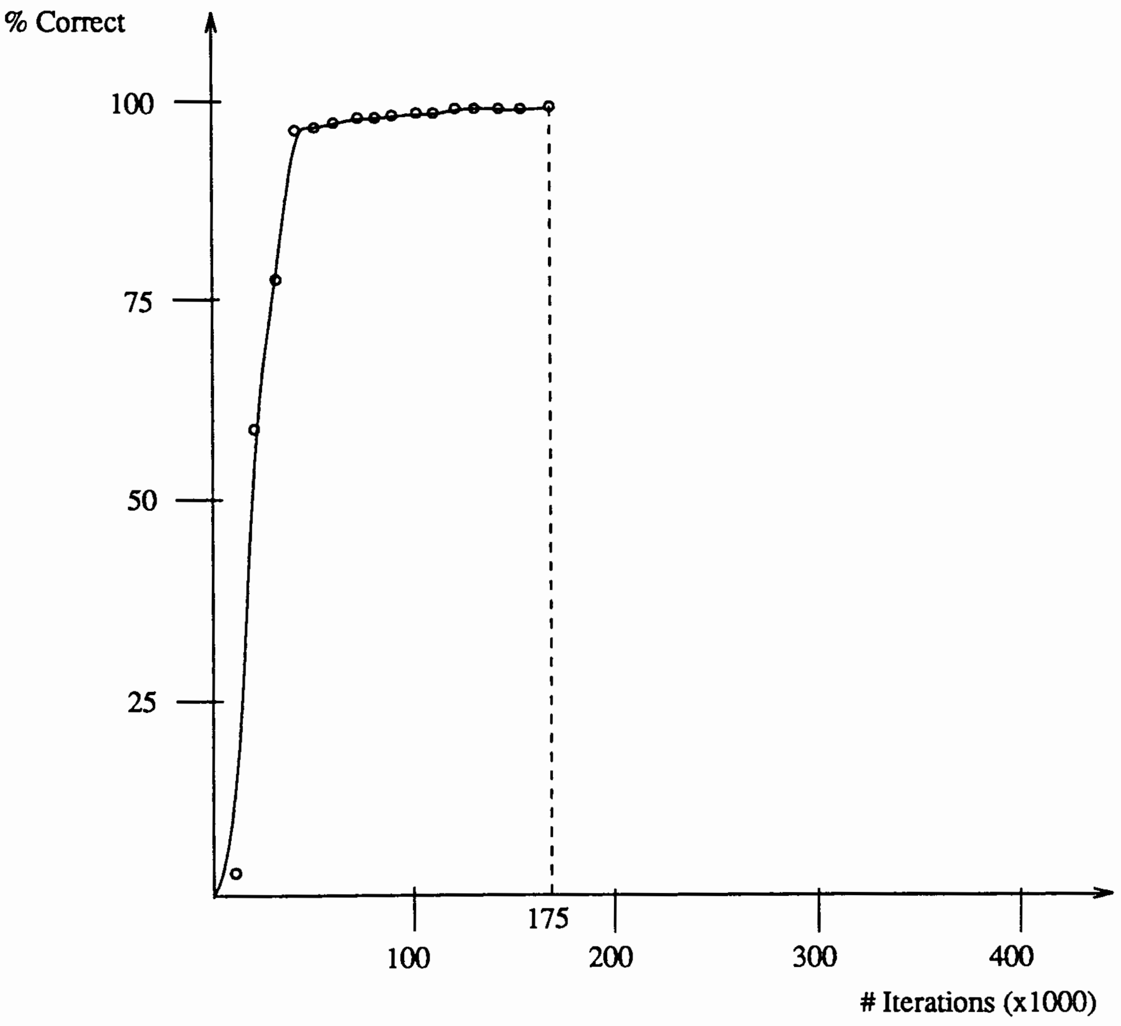


Figure 15. Training dynamics for experiment NN-2 (T = 0.6), while learning the ±45°, and the ±22.5° rotations of font set I-A.

TABLE VI

ERRORS ON TRAINING TEST FOR EXPERIMENT NN-2

Errors (98.1% correct, T = 0.6)			
Letter	Error Type	Font	Transformation
G	#	Palatino	+ 45 °
N	?	Geneva	+ 45 °
N	?	Palatino	+ 45 °
P	?	Avant Garde	+ 45 °
U	?	Geneva	+ 45 °
X	?	Times	+ 45 °
Z	?	Avant Garde	+ 45 °
Z	?	Palatino	+ 45 °

Note: The training set consists of the ±45°, and the ±22.5° rotations of font set I-A (416 patterns). A ? denotes no vote, and a # denotes misclassification: letter G in font Palatino (+45°) was misclassified as letter Q.

TABLE VII

ERRORS ON GENERALIZATION TEST FOR EXPERIMENT NN-2

Errors (83.2% correct, T = 0.6)			
Letter	Error Type	Font	Transformation
B	?	Avant Garde	- 11.25 °
C	#	Avant Garde	center
D	?	Avant Garde	- 11.25 °
D	?	Geneva	- 11.25 °
F	#	Geneva	- 11.25 °
F	?	Palatino	- 11.25 °
F	?	Times	- 11.25 °
G	?	Palatino	+ 11.25 °
G	?	Palatino	- 11.25 °
H	?	Times	- 11.25 °
J	?	Palatino	- 11.25 °
J	?	Times	- 11.25 °
K	?	Avant Garde	+ 11.25 °
K	?	Palatino	+ 11.25 °
K	?	Times	- 11.25 °
L	?	Avant Garde	+ 11.25 °
L	?	Times	- 11.25 °
M	?	Avant Garde	- 11.25 °
M	?	Geneva	- 45 °
N	?	Times	- 11.25 °
O	?	Avant Garde	+ 11.25 °
O	?	Avant Garde	- 11.25 °
O	?	Geneva	- 11.25 °
P	?	Geneva	+ 11.25 °
P	?	Times	- 11.25 °
Q	#	Palatino	+ 11.25 °
Q	?	Geneva	- 11.25 °
R	?	Geneva	+ 11.25 °
R	?	Geneva	- 11.25 °
S	?	Geneva	- 11.25 °
S	?	Times	- 11.25 °
T	?	Geneva	+ 11.25 °
T	?	Times	- 11.25 °
T	?	Geneva	- 11.25 °
U	#	Geneva	+ 11.25 °
U	?	Palatino	- 11.25 °
U	?	Times	- 11.25 °
V	?	Times	- 11.25 °
X	?	Times	- 11.25 °
X	?	Palatino	- 11.25 °

Note: The generalization set consists of set I-A (0° rotation) plus the ±11.25° rotations. A ? denotes no vote, and a # denotes misclassification: letter C in font Avant Garde (center) was misclassified as letter O; letter F in font Geneva (-11.25°) misclassified as letter C; letter Q in font Palatino (+11.25°) misclassified as letter G; letter U in font Geneva (+11.25°) misclassified as letter C.

TABLE VIII

PERCENT CORRECT CLASSIFICATION ON TRAINING AND GENERALIZATION FOR EXPERIMENT NN-3

Threshold	Training Results	Generalization Results
0.4	88%	52%
0.5	100%	65.6%
0.6	100%	1.9%
0.7	100%	0%
0.8	100%	0%
0.9	97.1%	0%

Note: The training set consists of the ±3-pixel (left and right) shifts of font set I-A (208 patterns). The generalization set consists of font set I-A in the original format (0-pixel shift) (104 patterns).

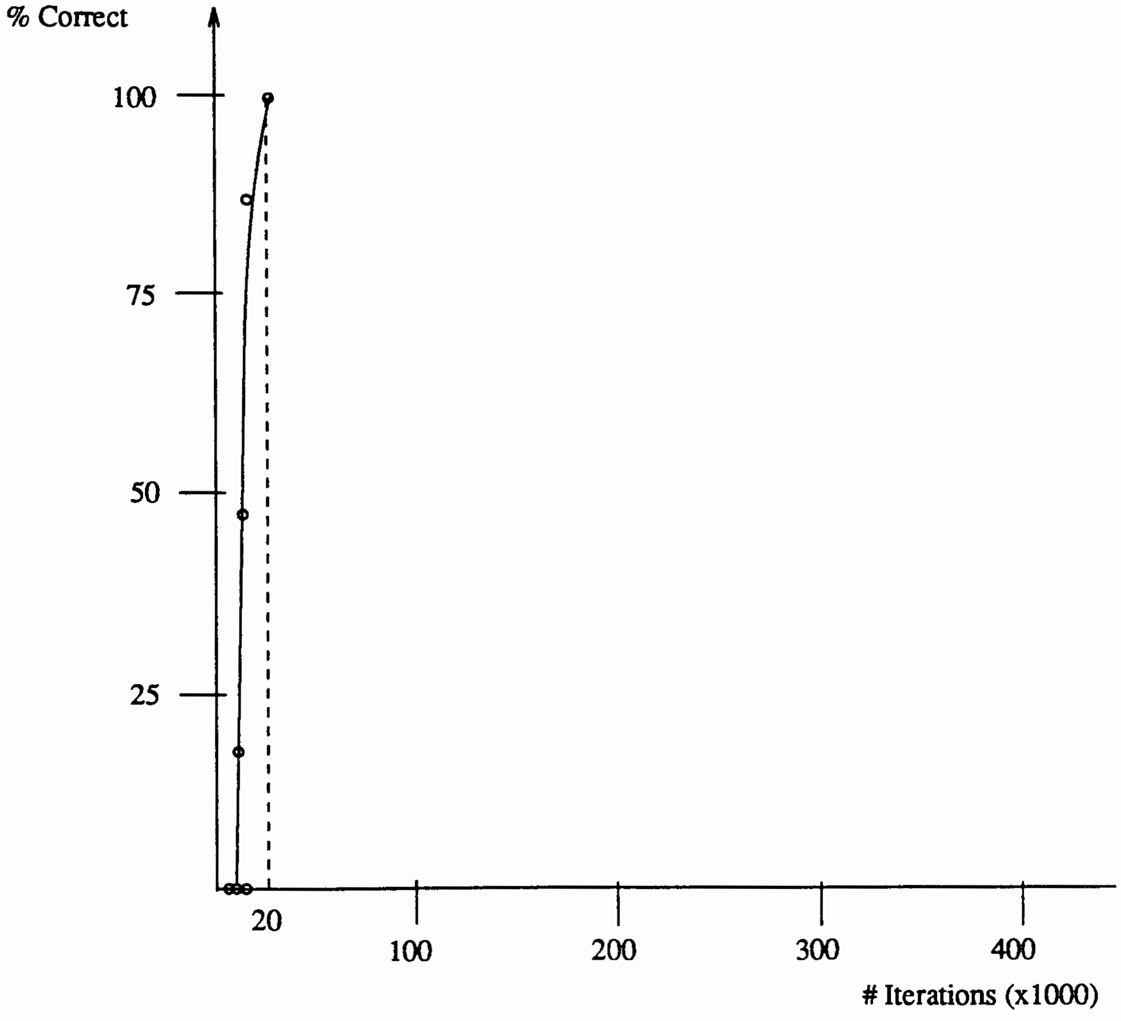


Figure 16. Training dynamics for experiment NN-3 (T = 0.6), while learning the ±3-pixel (left and right) shifts of font set I-A.

TABLE IX

ERRORS ON GENERALIZATION TEST FOR EXPERIMENT NN-3

Errors (1.9% correct, T = 0.6)			
Letter	Error Type	Font	Transformation
B (J)	#	Geneva	center
B (I)	#	Times	center
C (Z)	#	Avant Garde	center
C (I)	#	Geneva	center
C (B)	#	Palatino	center
D (I)	#	Geneva	center
E (I)	#	Geneva	center
E (I)	#	Palatino	center
G (R)	#	Palatino	center
G (Q)	#	Times	center
K (R)	#	Avant Garde	center
K (I)	#	Geneva	center
K (M)	#	Times	center
M (A)	#	Avant Garde	center
M (O)	#	Geneva	center
M (J)	#	Palatino	center
N (J)	#	Avant Garde	center
N (A)	#	Geneva	center
O (I)	#	Geneva	center
O (J)	#	Palatino	center
P (T)	#	Geneva	center
P (K)	#	Times	center
R (V)	#	Avant Garde	center
R (I)	#	Palatino	center
U (I)	#	Geneva	center
V (A)	#	Avant Garde	center
V (T)	#	Geneva	center
W (V)	#	Avant Garde	center
W (I)	#	Geneva	center
X (I)	#	Palatino	center
Y (B)	#	Avant Garde	center
Y (J)	#	Palatino	center
Z (A)	#	Geneva	center
Z (H)	#	Palatino	center

Note: The generalization set consists of set I-A in the original format (0-pixel shift). A ? denotes no vote, and a # denotes misclassification: letters in parentheses denote the network's misclassification of the letter in question. The network responded with a no vote for all other letters, except for the two it classified correctly (F and I).

TABLE X

PERCENT CORRECT CLASSIFICATION ON TRAINING AND GENERALIZATION FOR EXPERIMENT NN-A

Threshold	Training Results	Generalization Results
0.4	100%	80.5%
0.5	100%	85.3%
0.6	100%	96.2%
0.7	100%	93.1%
0.8	98.8%	89.7%
0.9	97.3%	85.3%

Note: The training set consists of font set I-A plus the ±45°, ±22.5°, ±11.25° rotations, and the ±3-pixel (left and right) shifts (936 patterns). The generalization set consists of font set I-B plus the same 8 transformations (468 patterns).

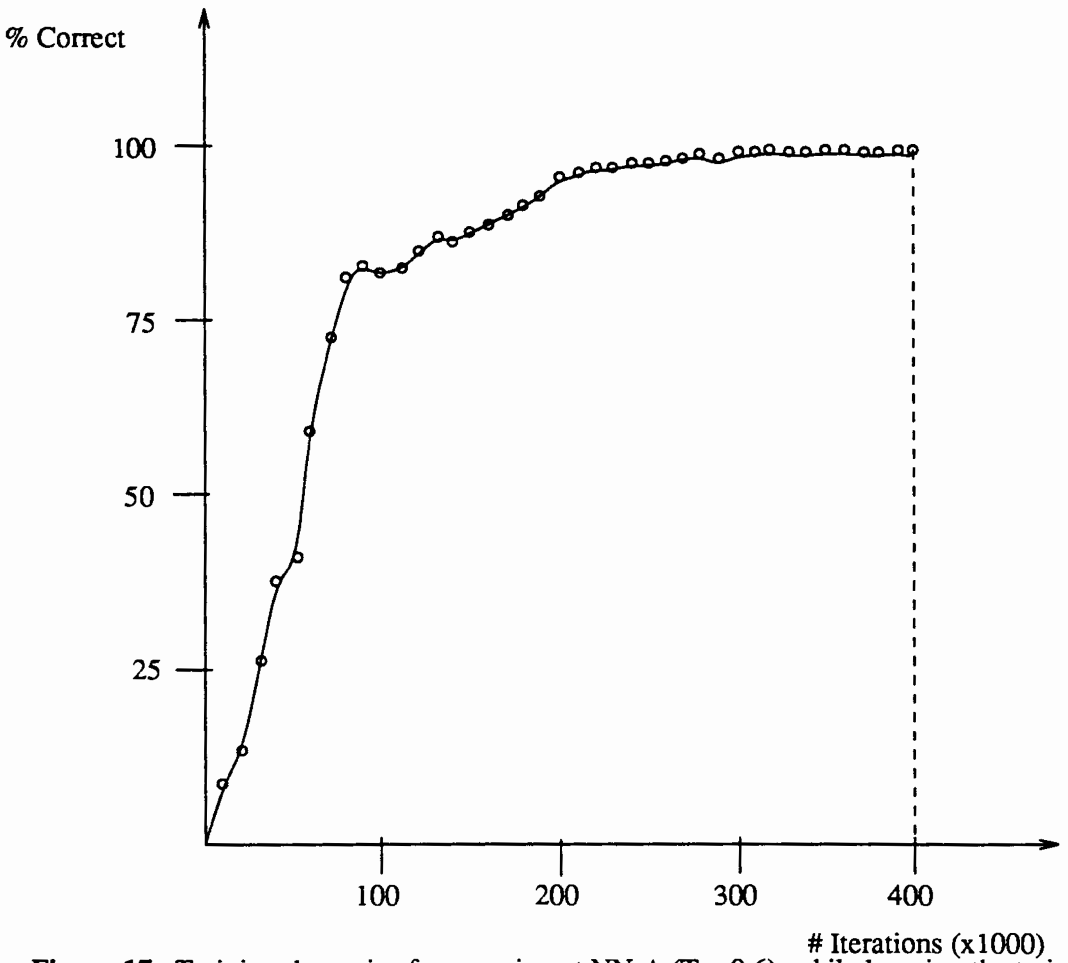


Figure 17. Training dynamics for experiment NN-A (T = 0.6), while learning the training set consisting of font set I-A plus the ±45°, ±22.5°, ±11.25° rotations, and the ±3-pixel (left and right) shifts.

TABLE XI

ERRORS ON GENERALIZATION TEST FOR EXPERIMENT NN-A

Errors (96.2% correct, T = 0.6)			
Letter	Error Type	Font	Transformation
B	?	Helvetica	center
B	?	Helvetica	right
G	?	New York	center
G	?	New York	-11.25°
I	?	Helvetica	+45°
I	?	Helvetica	-45°
J	#	New York	+45°
J	?	Helvetica	-11.25°
N	?	Helvetica	center
N	?	New York	-45°
P	?	New York	left
R	?	New York	-22.5°
S	?	Helvetica	center
S	?	New York	-11.25°
S	?	New York	-22.5°
V	?	Helvetica	-11.25°
Z	?	Helvetica	-11.25°
Z	?	New York	-45°

Note: The generalization set consists of set I-B plus the ±45°, ±22.5°, ±11.25° rotations, and the ±3-pixel (left and right) shifts. A ? denotes no vote, and a # denotes misclassification: letter J in font New York (+45°) was misclassified as letter I.

TABLE XII

PERCENT CORRECT CLASSIFICATION ON TRAINING AND GENERALIZATION FOR EXPERIMENT NN-B

Threshold	Training Results	Generalization Results
0.4	100%	77.1%
0.5	100%	87%
0.6	100%	93.6%
0.7	96.1%	84.2%
0.8	83.4%	78.1%
0.9	75.1%	69.9%

Note: The training set consists of the ±45° and ±22.5° rotation of set I-A (416 patterns). The generalization set consists of font set I-A plus the ±11.25° rotations (312 patterns).

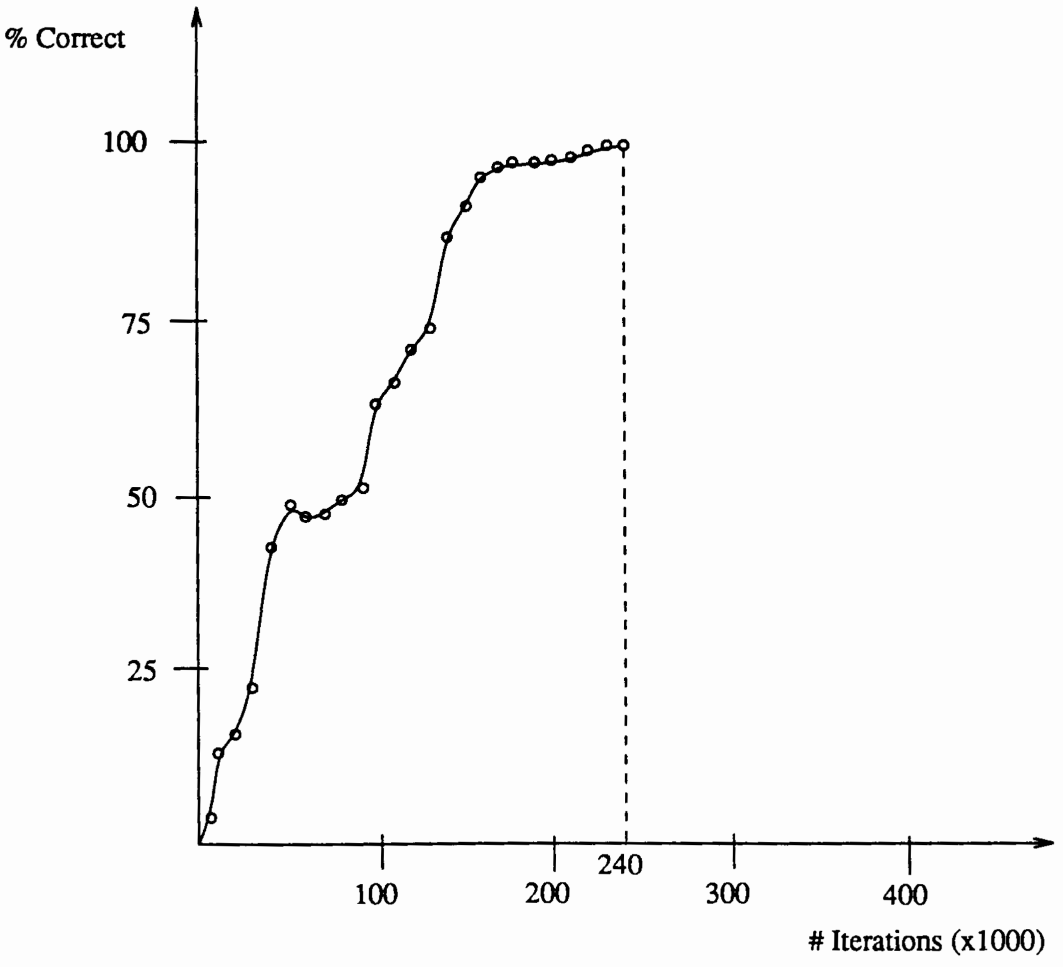


Figure 18. Training dynamics for experiment NN-B (T = 0.6), while learning the ±45° and ±22.5° rotations of font set I-A.

TABLE XIII

ERRORS ON GENERALIZATION TEST FOR EXPERIMENT NN-B

Errors (93.6% correct, T = 0.6)			
Letter	Error Type	Font	Transformation
B	?	Avant Garde	center
B	?	Times	+ 11.25 °
B	?	Avant Garde	- 11.25 °
C	?	Times	center
C	?	Times	- 11.25 °
G	?	Times	center
J	?	Palatino	center
J	#	Palatino	+ 11.25 °
J	?	Palatino	- 11.25 °
K	?	Geneva	center
K	?	Palatino	+ 11.25 °
K	?	Palatino	- 11.25 °
N	?	Geneva	center
Q	#	Geneva	center
Q	?	Avant Garde	+ 11.25 °
Q	?	Times	- 11.25 °
S	?	Avant Garde	center
S	?	Palatino	center
X	?	Times	- 11.25 °
Z	?	Palatino	- 11.25 °

Note: The generalization set consists of set I-A plus the ±11.25° rotations. A ? denotes no vote, a # denotes misclassification: letter J in font Palatino (+11.25°) was misclassified as I; letter Q in font Geneva (center) was misclassified as G.

TABLE XIV

PERCENT CORRECT CLASSIFICATION ON TRAINING AND GENERALIZATION FOR EXPERIMENT NN-C

Threshold	Training Results	Generalization Results
0.4	100%	73%
0.5	100%	82%
0.6	100%	93.3%
0.7	87.8%	83.9%
0.8	80.1%	72.1%
0.9	73.3%	63.8%

Note: The training set consists of the ±3-pixel (left and right) shifts of set I-A (208 patterns). The generalization set consists of font set I-A in the original format (0-pixel shift) (104 patterns).

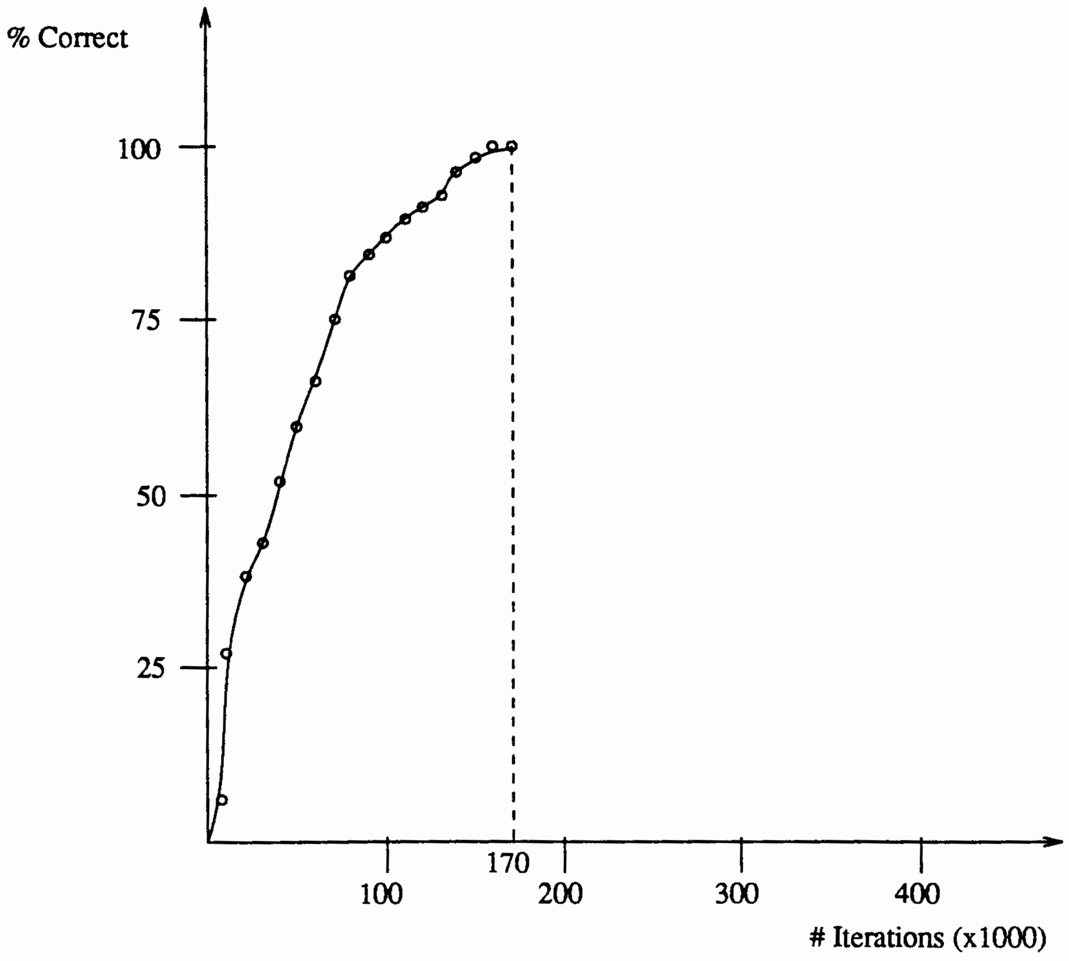


Figure 19. Training dynamics for experiment NN-C (T = 0.6), while learning the ±3-pixel (left and right) shifts of font set I-A.

TABLE XV

ERRORS ON GENERALIZATION TEST FOR EXPERIMENT NN-C

Errors (93.3% correct, T = 0.6)			
Letter	Error Type	Font	Transformation
C	?	Times	center
G	?	Times	center
K	?	Palatino	center
M	?	Geneva	center
Q	#	Times	center
W	?	Times	center
Z	?	Palatino	center

Note: The generalization set consists of set I-A in the original form (0-pixel shift). A ? denotes no vote, and a # denotes a misclassification: letter Q in font Times (center) was misclassified as O.

CHAPTER V

DISCUSSION OF RESULTS AND FUTURE WORK

DISCUSSION OF RESULTS

The experiments reported in the previous chapter demonstrate that the more complex encoding schema developed here for the NN's output -- based on the conceptual graph formalism (as suggested in [Lendaris, 1988a]) -- provides a significant improvement in the NN's capability to generalize well as compared to the case where the more straightforward output encoding consisting of 26 bits, one for each letter of the alphabet, is used. The improvements for the three different generalization experiments were as follows: from 75% to 96% correct generalization on fonts; from 83% to 94% correct generalization on rotations; and from 2% to 93% correct generalization on shifts. Performance in the 90+% category brings such NNs to a level that has possibilities (after more refinements) for real-world applications. Of course, such applications will no doubt require recognition of more than just the upper case letters of the English alphabet -- for example, the lower case letters, numerals, and other commonly used symbols, as a minimum. This will require augmenting the concept and relation lists introduced in Chapter II to allow proper description of these expanded pattern sets. More about this later. The present experiments lay the ground work for such future developments.

In addition to the above (main) conclusion about improved generalization results via the new output encoding developed here, there are two other observations that appear of interest. For ease in discussing these observations, certain of the data from the experiments of Chapter IV are tabulated in Table XVI.

TABLE XVI

COMPARISON OF RESULTS

Generalize on:	Number of Training Iterations to Peak Performance		Increase Factor in:	
	Basic Encoding Method	Conceptual Graph Encoding Method	Train Iterations	Generalization Results
Fonts	70,000	400,000	5.7	75% → 96%
Rotations	175,000	240,000	1.4	83% → 94%
Shifts	20,000	170,000	8.5	2% → 93%

Comparing columns 1 and 2, it is noted that the NN took significantly longer to learn the more complex output encoding. The reward for the extra effort, however, was improved generalization performance. Intuitively, one can surmise that more weight combinations (paths) in the NN are called into play to effect an output encoding with more than one element set equal to 1, as compared to the case where only one output element is set to 1. It is reasonable to further surmise that this extra amount of co-involvement of multiple paths would take more effort to learn -- hence, the larger number of training iterations. Regarding the improved generalization performance, the requirement for the NN to satisfy the additional constraints implied by the "extra co- involvement of multiple paths" results in a configuration of weights that somehow better captures the "structure" of the input patterns, and thus yields better generalization performance.

A further observation to be made here relates to the results given in Chapter IV regarding the classification performance of the NN, depending on the value chosen for the threshold T. For the purposes of this chapter, the data are presented in graphical form -- see Figures 20, 21 and 22. Note that in all cases except one, the best results occurred at T = 0.6 (this is the value used for all cited results). From an implementation point of view, a significant aspect to note is the dramatic drop in generalization performance as T is increased above 0.6 for experiments NN-1 and NN-3 as compared with experiments NN-A and NN-C. If the value given by an output element is thought of as a "vote" for a particular answer, then higher values of T could mean higher confidence votes. When the elements are successful in achieving correct performance during training on the more complex output encoding, they end up being able to cast higher confidence votes for patterns not seen during training. The resulting reduced sensitivity on the specific value for T can be important from an implementation point of view.

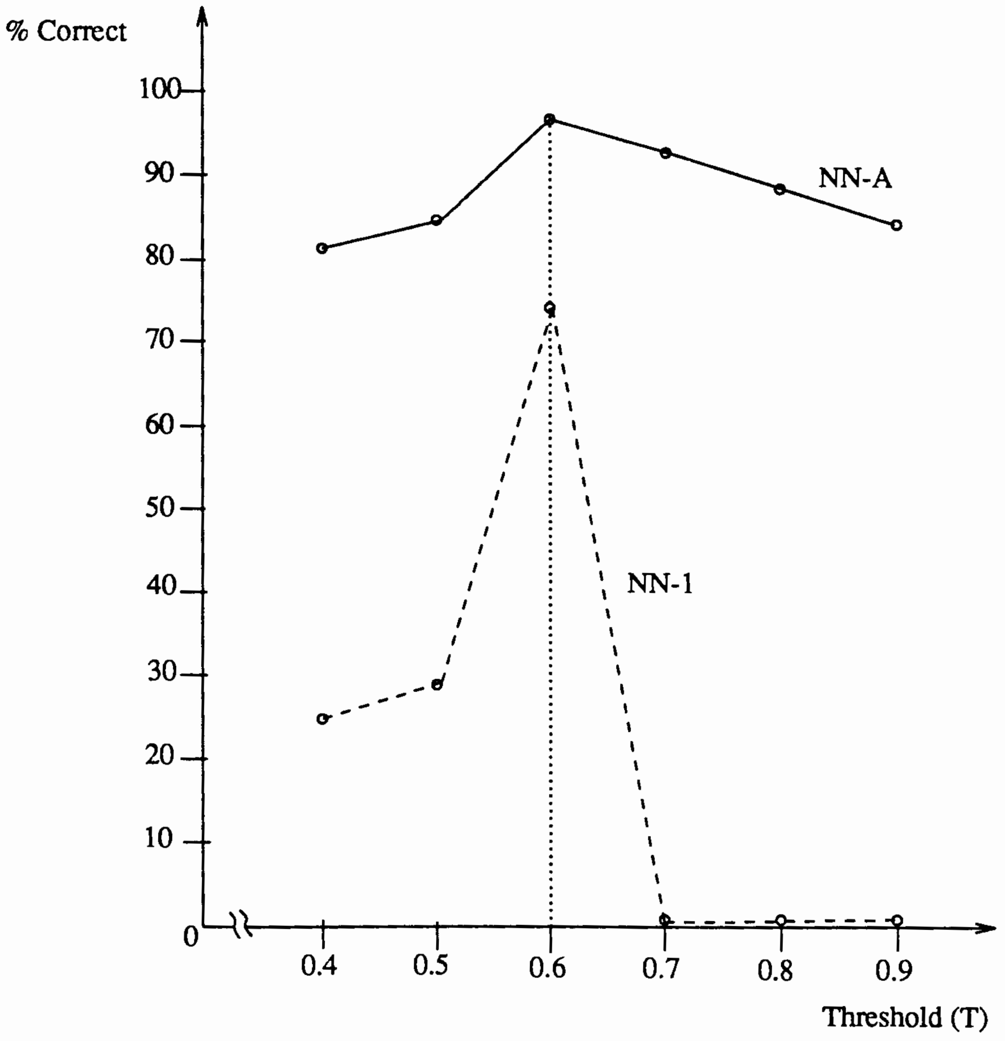


Figure 20. Classification performances for experiments NN-1 and NN-A, with respect to the various values of T.

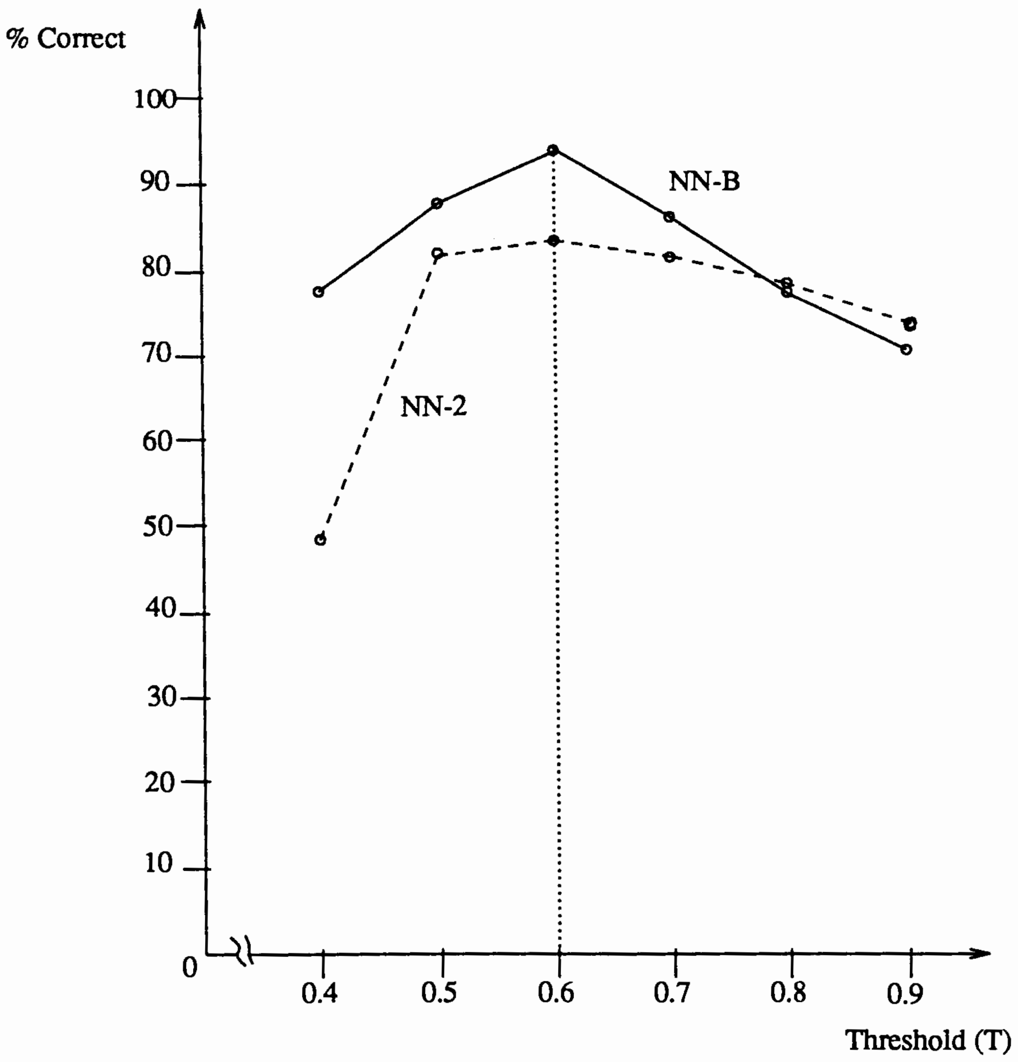


Figure 21. Classification performances for experiments NN-2 and NN-B, with respect to the various values of T.

FUTURE WORK

The upper case letters of the English alphabet served as a good starting point for the development of representation codes based on conceptual graphs, because a rather small set of concept types ("building blocks") are sufficient for constructing these letters. The representation codes developed in the present research were demonstrated to yield significantly better generalization than a more standard 26-bit (1 bit for each letter)

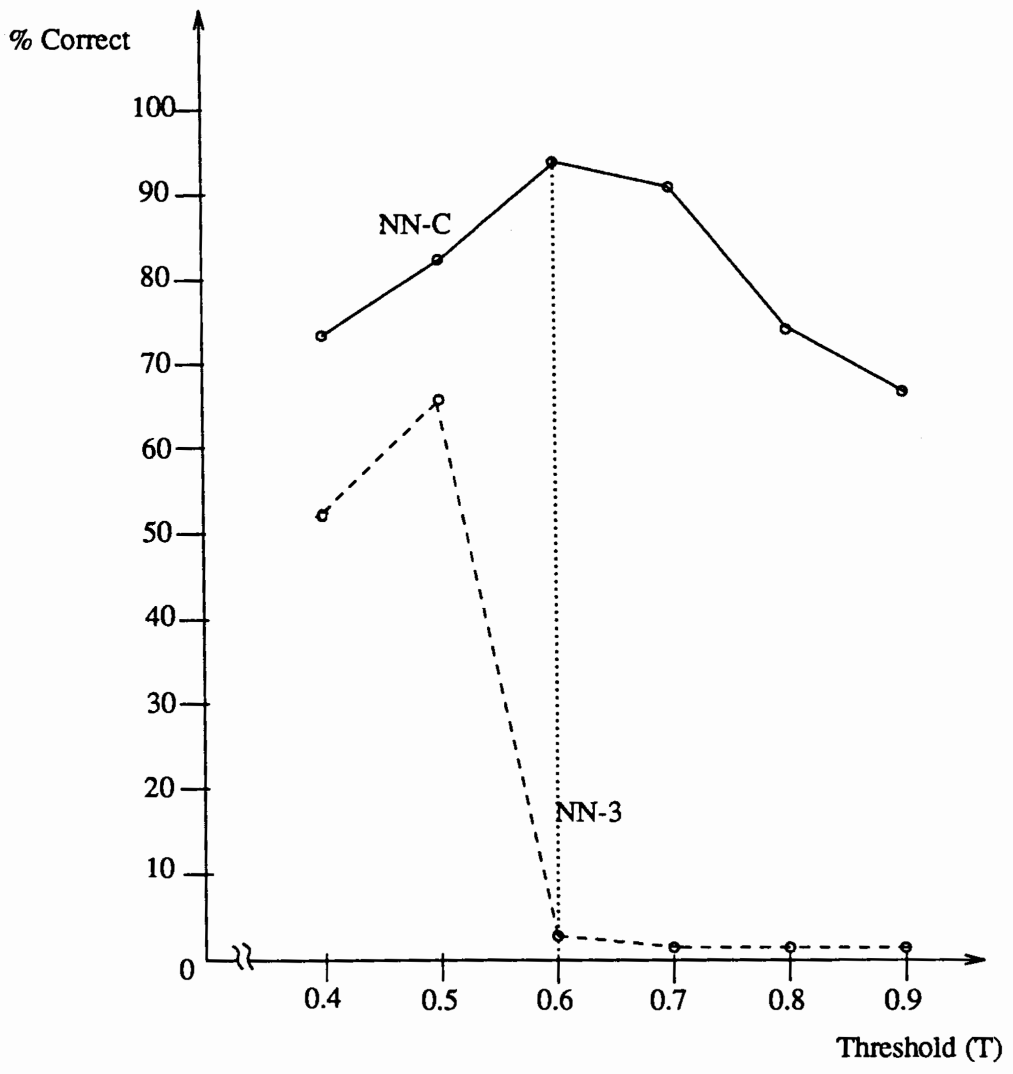


Figure 22. Classification performances for experiments NN-3 and NN-C, with respect to the various values of T.

encoding. The new encoding process entailed development of a list of concept types and relation types, and it only took 4 concept types and 4 relation types to construct the 26 upper case letters of the English alphabet.

An appropriate next step from the present research will be to consider the lower case letters of the English alphabet (in the same fonts used for the present research). Even cursory inspection of the lower case letters suggests that the present set of concept and relation types will not suffice to describe these patterns. Thus, an important issue to be dealt with in the next step, is that of augmenting the concept type list with appropriate new "building blocks" to allow construction of the new patterns. It may turn out that some new relation types will also have to be invented to go along with the new concept types. After this is accomplished, there will be a need to further increase the set of representable patterns to include the usual special symbols used in standard text -- perhaps the full printable ASCII set. If neural nets can be created to successfully classify the full ASCII set of characters for a large number of fonts similar to those used in the present experiments, with good generalization (in the 98+% region) on fonts, rotations, shifts, and possibly other distortions, then it could be considered a good candidate for real world applications.

There are fonts whose letter forms are significantly different from those in the 6 font sets used for the present research. Some of the differences include double-width component lines, different aspect (height/width) ratios, and modifications in basic structural form of some letters. The reader is invited to peruse Font Sets II-A & II-B in Appendix A for visual examples of such differences. As a demonstration of difficulties that are to be expected, the neural net trained in experiment NN-A was presented with the two fonts in Font Set II-A, and in each case the neural net only got about 20% correct classification. Even more difficulty was experienced with Font Set II-B: in this case, the neural net only got about 7% correct classification.

A preliminary effort was taken to develop a different representation code to account for the double-width lines of the letters in Font Set II-B. The resulting code consisted of 26 bits (rather than the 21 bits previously used), but the neural network did not even successfully complete the learning phase using this code. Further work on this aspect is beyond the scope of the present research project, but is an important direction for future research.

SELECTED BIBLIOGRAPHY

Books and Articles

Badi'i, F. & B. Peikari 1983. "Approximation of Multipath Planar Shapes in Pattern Analysis," International Journal of Computer and Information Sciences, Vol. 12, No. 2, pp. 99-110.

Hebb, D. O. 1949. The Organization of Behavior; a Neuropsycological Theory, Wiley, New York.

Hopfield, J. J. 1982. "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," Proceedings National Academy of Sciences USA, vol. 79, no. 8, pp. 2554-2558, National Academy of Sciences, Washington, D. C., April.

Jaensch, E. R. 1930. Eidetic Imagery, Keagan Paul, Trench, Trubner & Co., London.

Lendaris, G. G. 1988a. "Representing Conceptual Graphs for Parallel Processing," Proceedings AAAI Third Annual Workshop on Conceptual Graphs, AAAI-88, Maryan Kaufman, Los Altos, CA., August.

Lendaris, G. G. 1988b. "Neural Networks, Potential Assistants to Knowledge Engineers," Heuristics, Journal of the International Association of Knowledge Engineers, vol. 1, no. 2, December.

Minskey, M. & S. Pappert 1969. Perceptrons: An Introduction to Computational Geometry, MIT Press, Cambridge, MA.

McCulloch, W. S. & W. A. Pitts 1943. "Logical Calculus of the ideas Immanent in Nervous Activity," Bulletin of Mathematical Biophysics, vol. 5, pp. 115-133.

Parker D. B. 1985. "Learning Logic," Technical Report TR-47, Center for Computational Research in Economics and Management Science, MIT, Cambridge, MA., April.

Rosenblatt, F. 1962. Principles of Neurodynamics, Spartan Books, Washington D.C.

Rumelhart, D. E., J. L. McClelland & the PDP Research Group 1986. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1, The MIT Press, Cambridge, MA.

Simon H. A. 1975. Models of Discovery, Boston Studies in the Philosophy of Science, v. 54, p. 319, Reidel Publishing Co., Dordrecht, Holland.

Sowa, John F. 1984. Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley, Reading, Massachusetts.

Tou, J. T., & R. C. Gonzalez 1974. Pattern Recognition Principles, Addison-Wesley, Reading, Massachusetts.

Werbos, P. 1974. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral sciences, (Ph. D. Thesis), Cambridge, Mass.: Harvard University Committee on Applied Mathematics, November.

Widrow, B. 1962. "Generalization and Information Storage in Networks of Adaline Neurons," Self Organizing Systems-1962, M. C. Yovits, et al, eds, Spartan Books, Washington D.C., pp. 435-461.

Wiener, N. 1948. Cybernetics, Wiley, New York.

Proceedings
[For several hundred recent articles on neural network research and applications, consult the following Proceedings.]

Proceedings of IEEE International Conference on Neural Networks 1987 (ICNN-87), IEEE, New York.

Proceedings of IEEE International Conference on Neural Networks 1988 (ICNN-88), IEEE, New York.

Abstract of the First Annual INNS Meeting, Boston, 1988, in Neural Networks, Journal of the International Neural Network Society, vol.1, Supplement 1, Pergamon Press, New York.

Proceedings of International Joint Conference on Neural Networks 1989 (IJCNN-89), IEEE, New York.

APPENDIX A

ENGLISH ALPHABET FONT DATA USED FOR THE EXPERIMENTS REPORTED IN THIS THESIS

There are ten fonts*, organized into set I-A, set I-B, set II-A, and set II-B as shown (centered on a 16x16 grid). The names used for these fonts on the Macintosh computer are as follows (left to right):

Set I:

1) Geneva

2) Times

3) Avant Garde

4) Palatino

5) Helvetica

6) New York

Set II:

1) Cartoon

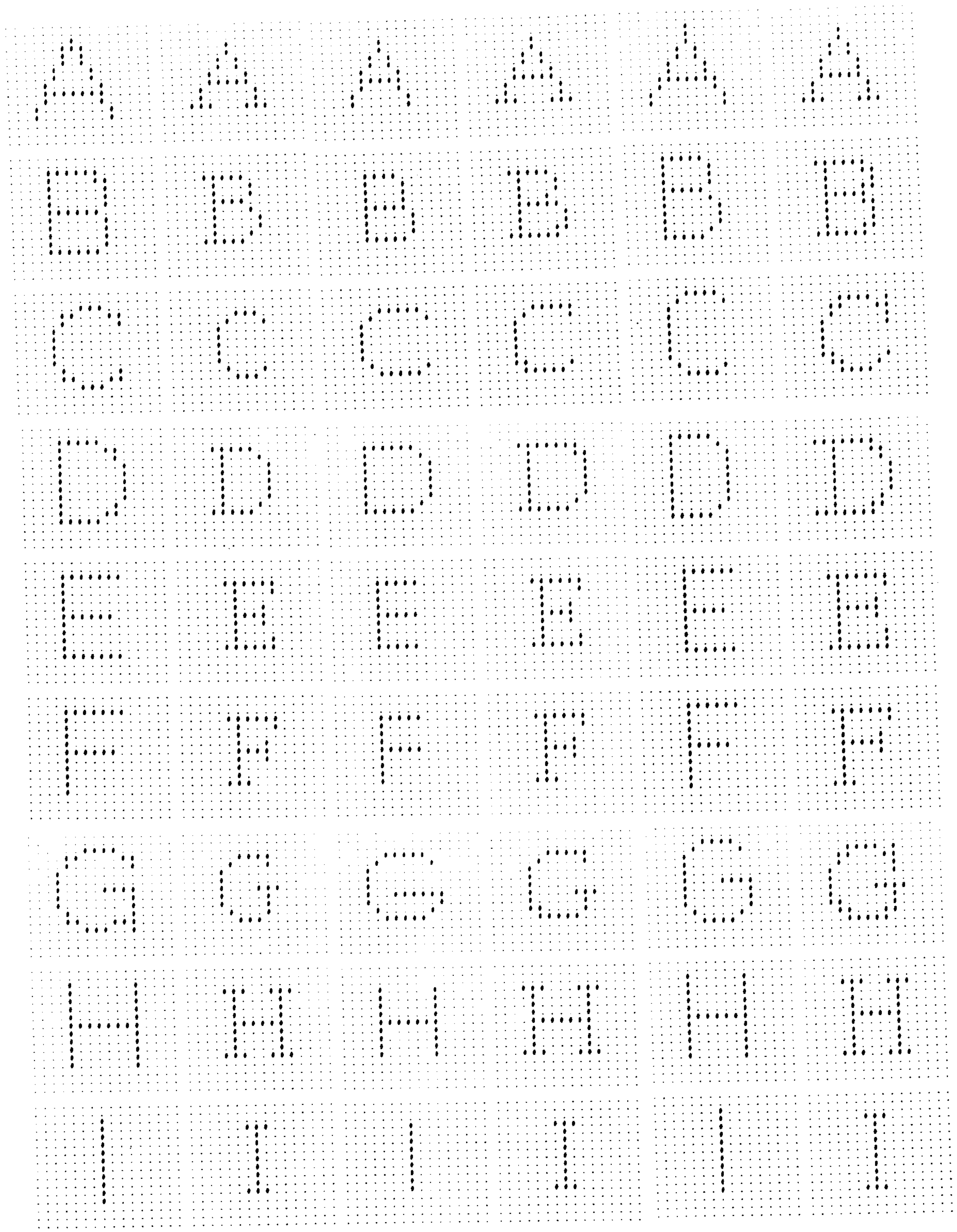
2) Chicago

3) Monaco

4) Ascham

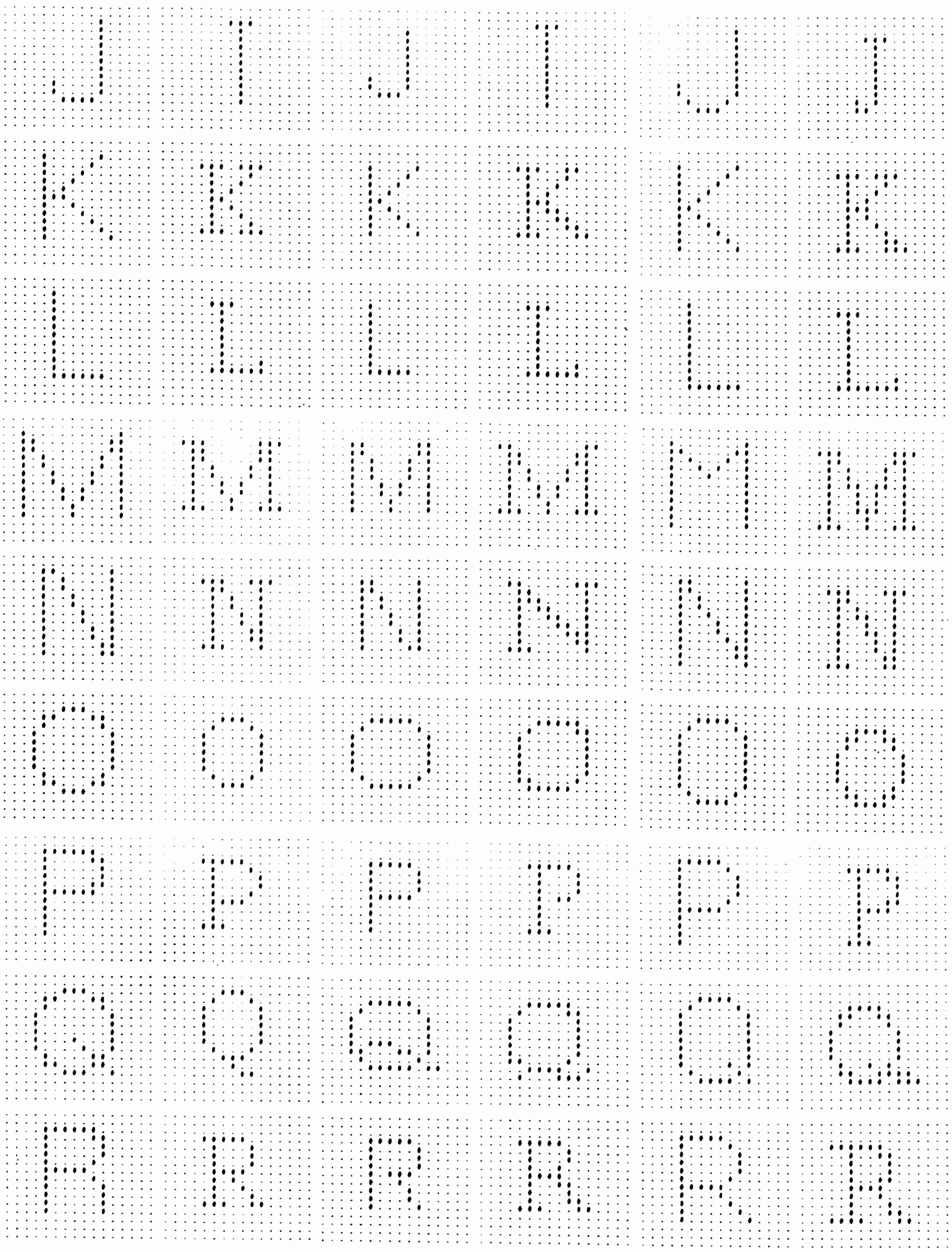
* The original data, with the letters upper-left-corner justified on a 12x12 grid, were provided to us by Krist D. Roginski, a graduate student at the Oregon Graduate Center; extracted (ca. 1988) from selected fonts available on the Macintosh Computer.

←----------------------------SetI-A----------------------------→|←-----------SetI-B-----------→|



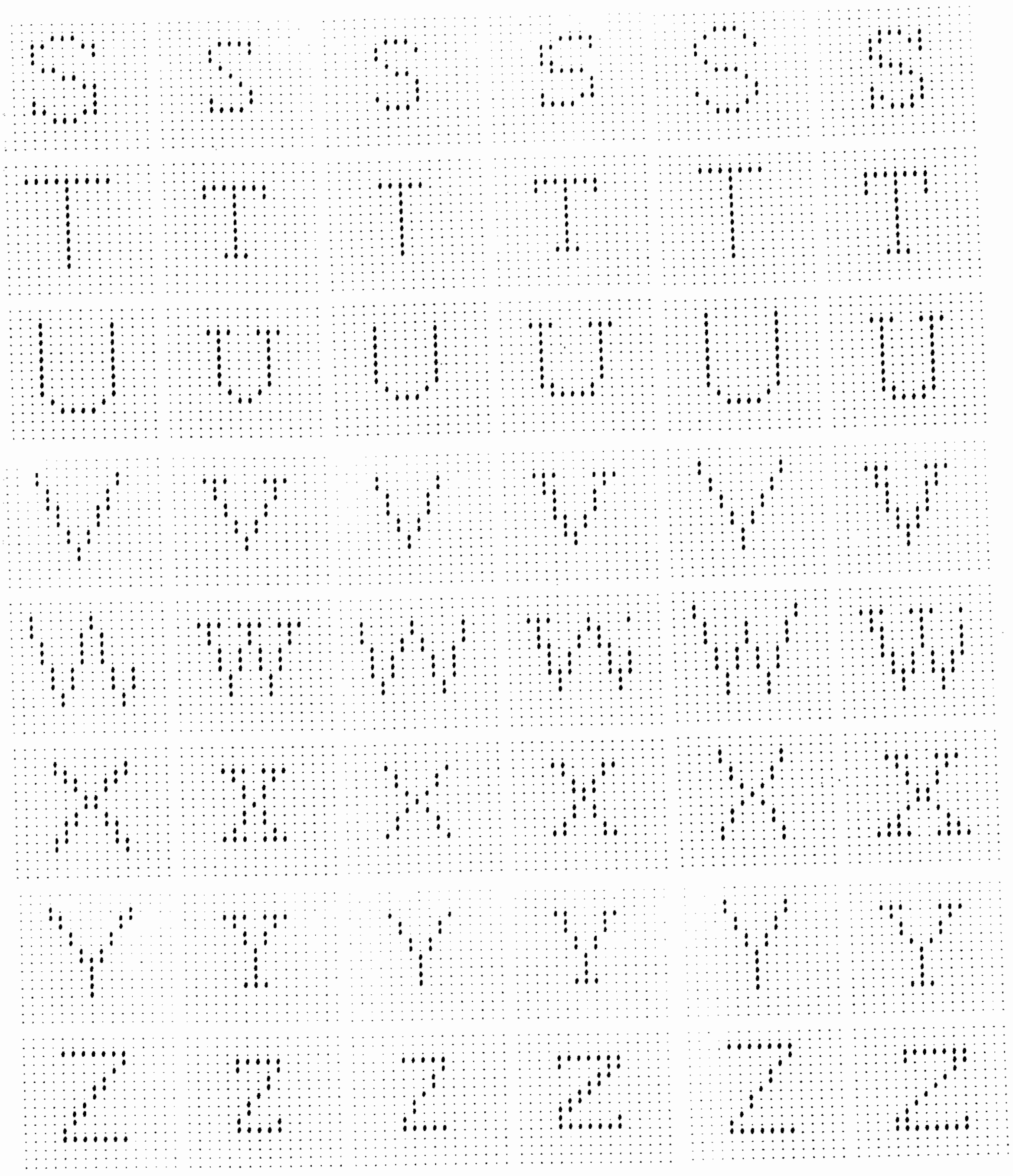
A.1: Font sets I-A and I-B.

←--------------------------SetI-A--------------------------→|←-----------SetI-B----------→|



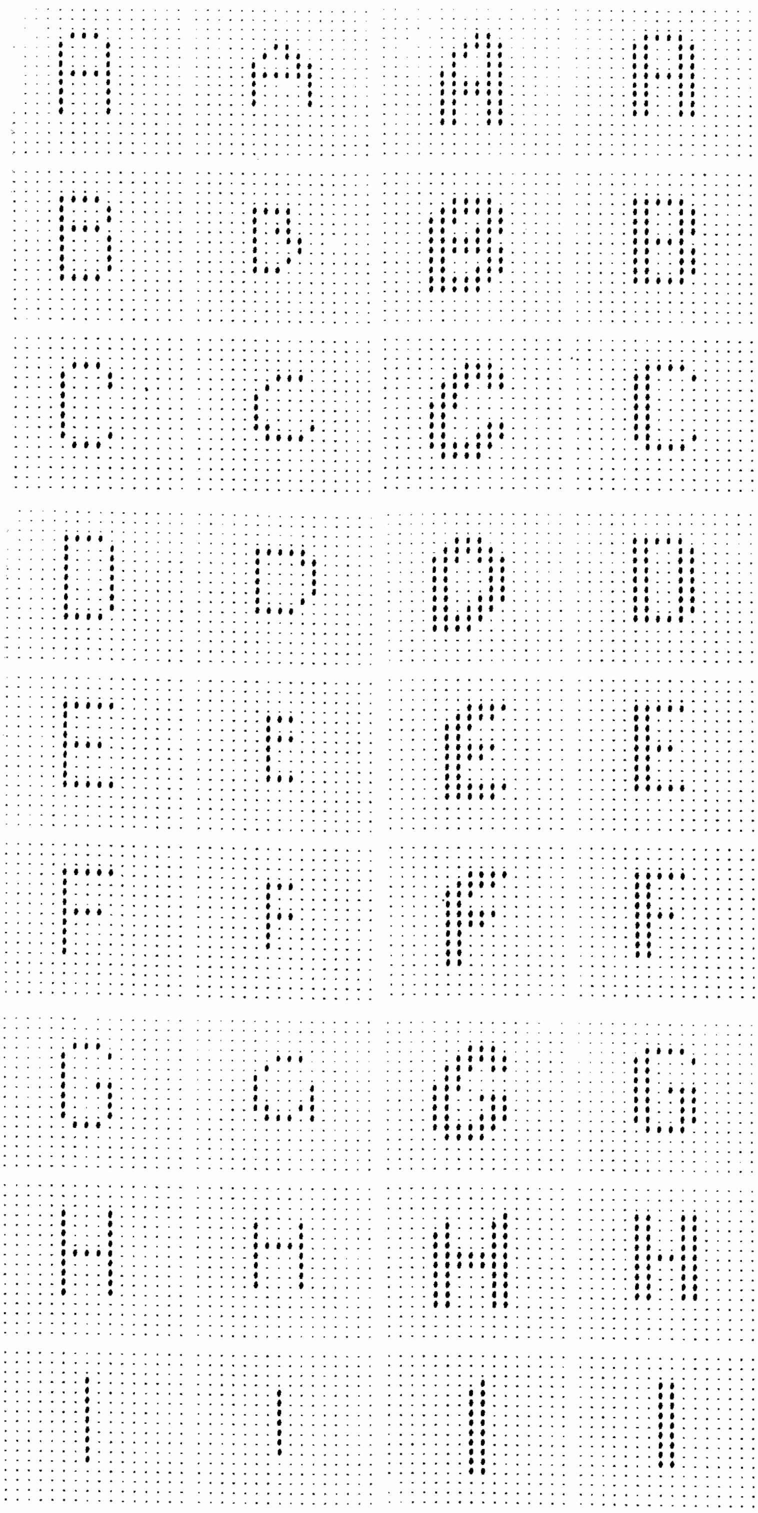
A.1 (continued): Font sets I-A and I-B.

---------------------------SetI-A---------------------------→|←-----------SetI-B-----------→|



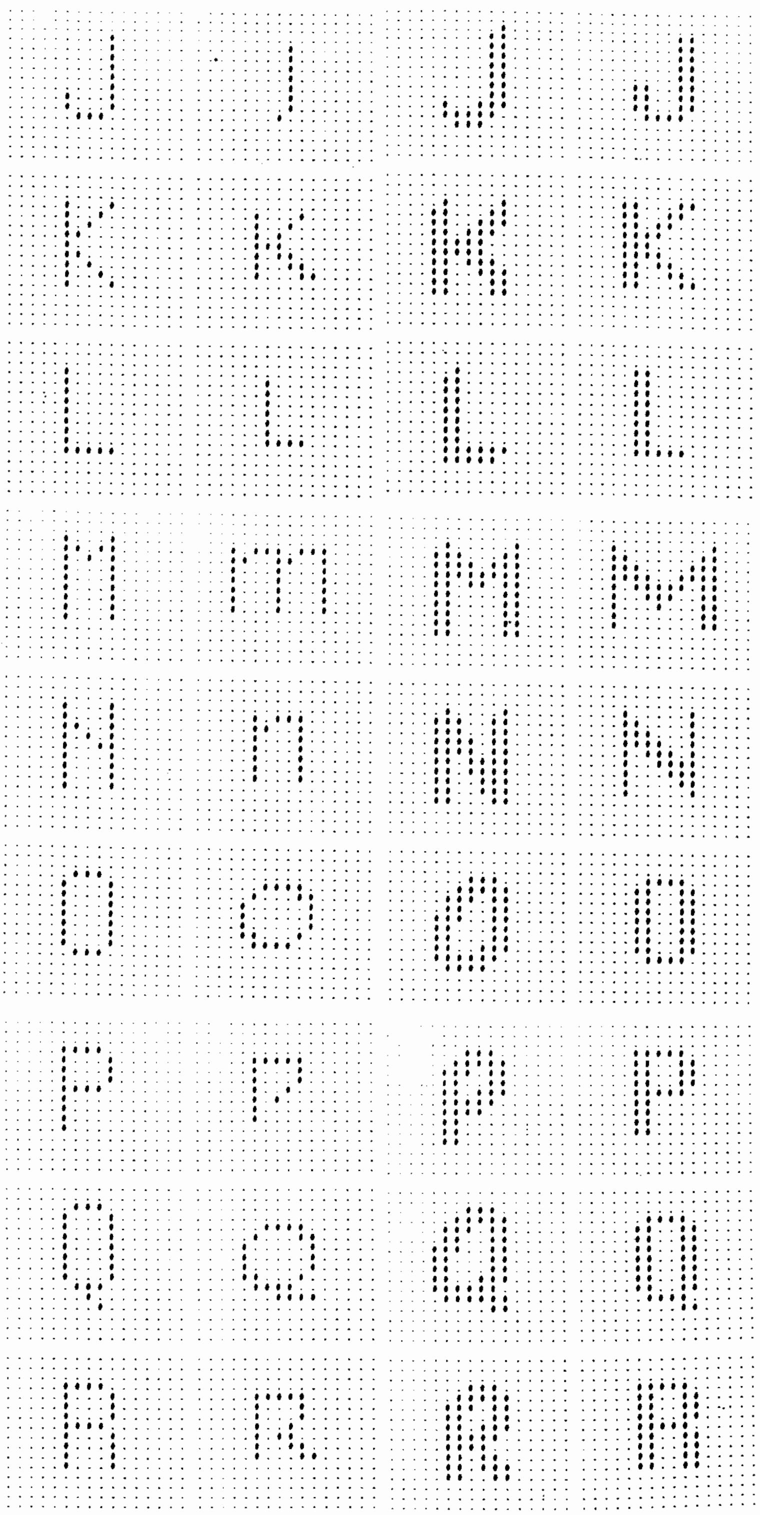
A.1 (continued): Font sets I-A and I-B.

←----------SetII-A---------→|←----------SetII-B---------→|



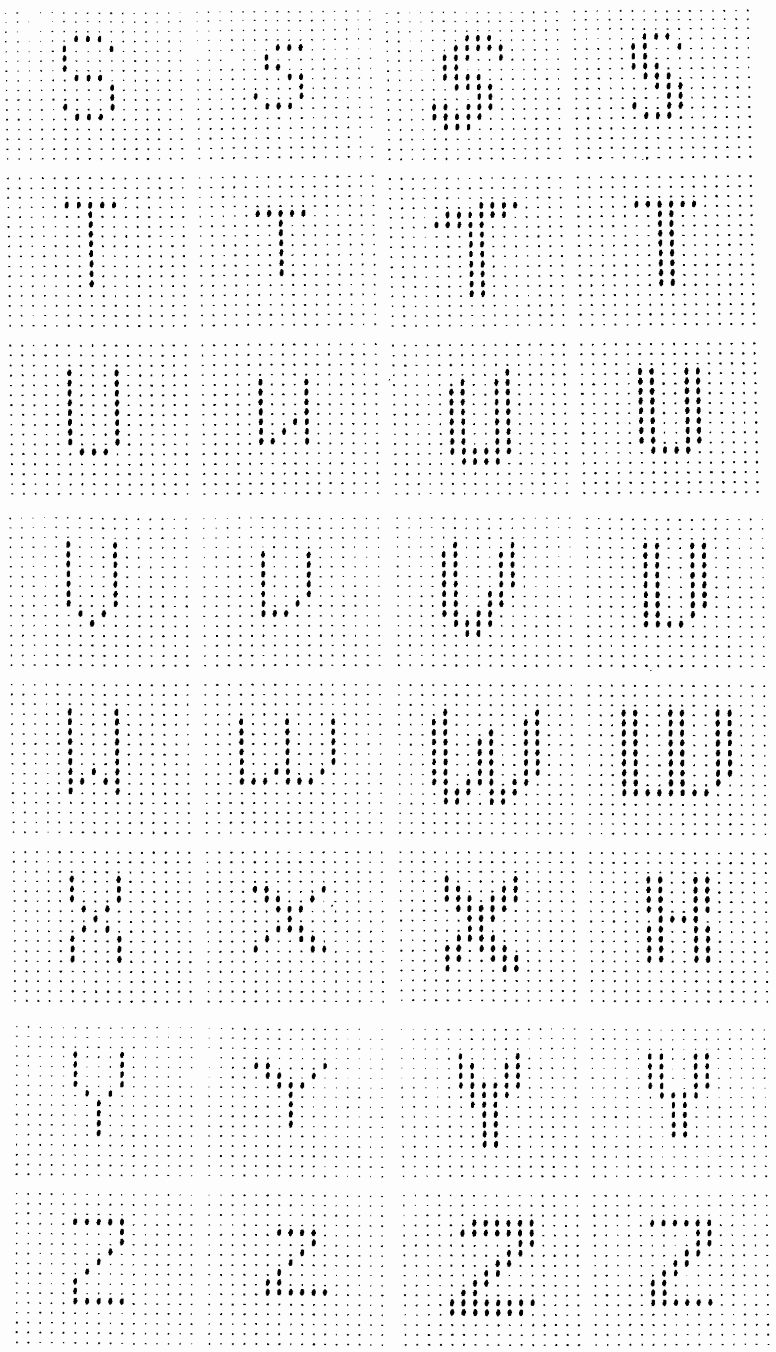
A.2: Font sets II-A and II-B.

|←----------SetII-A---------→|←-----------SetII-B----------→|



A.2 **(continued):** Font sets II-A and II-B.

|←----------SetII-A---------→|←----------SetII-B---------→|



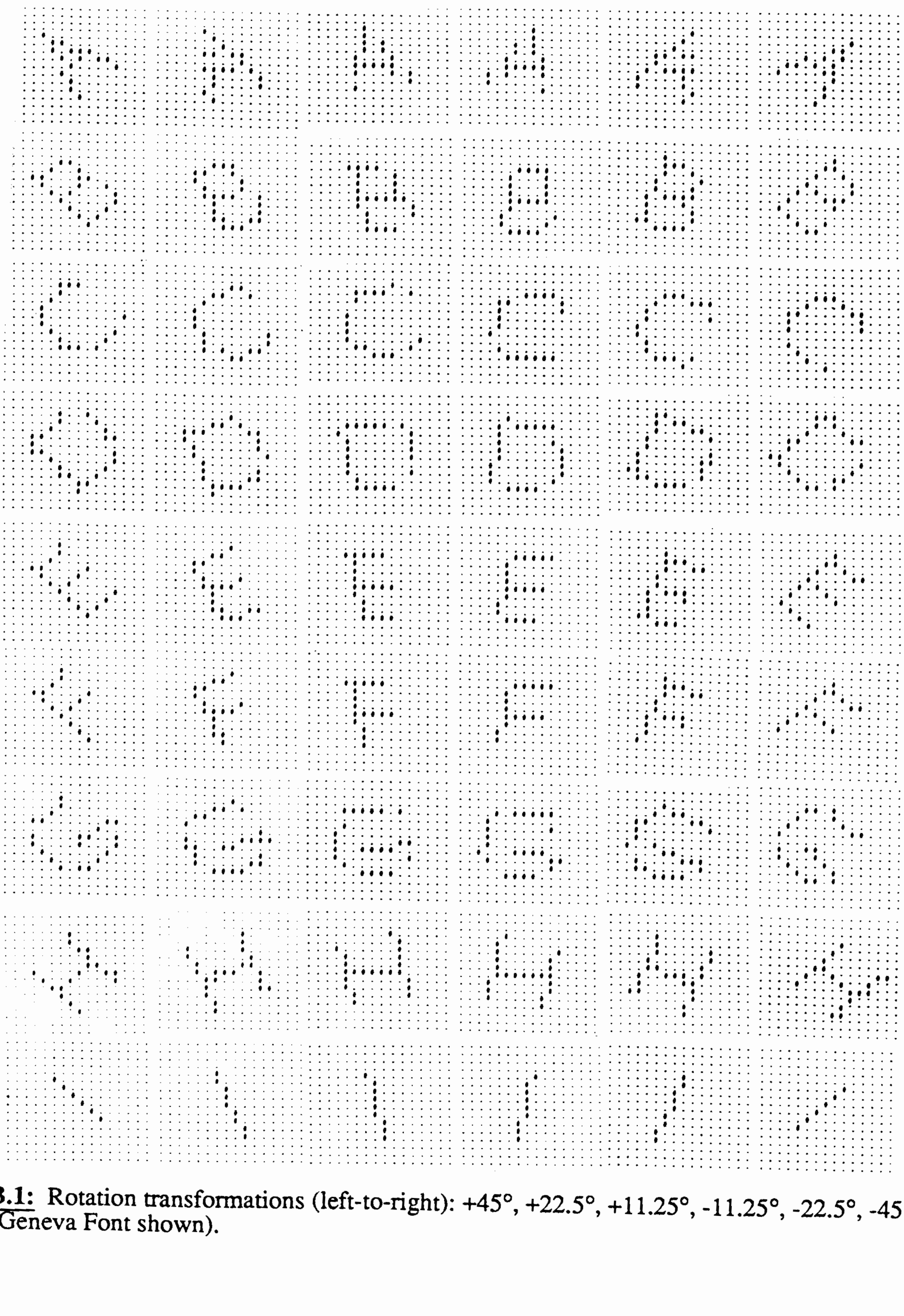
A.2 (continued): Font sets II-A and II-B.

APPENDIX B

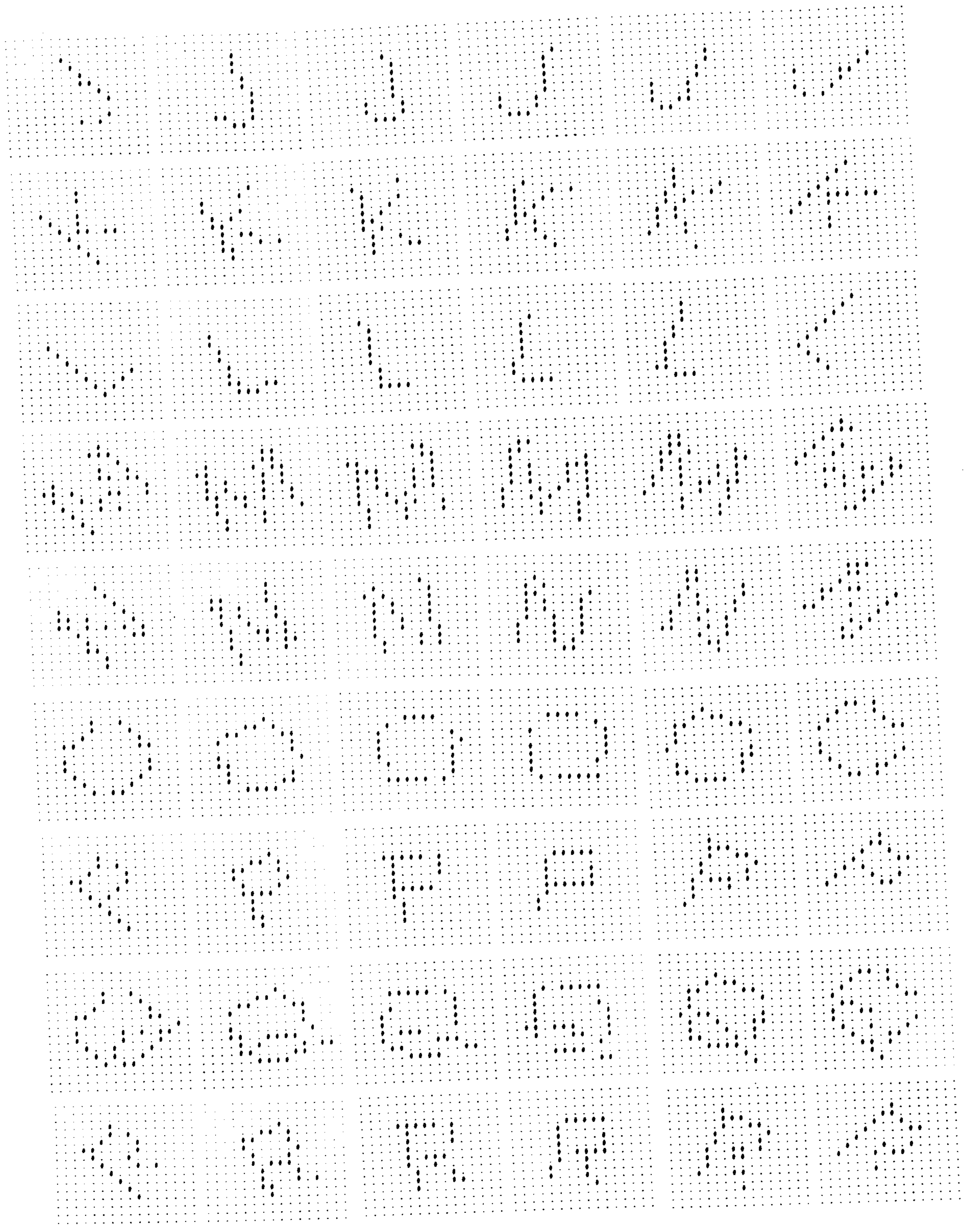
EXAMPLES OF TRANSFORMED ALPHABET DATA

The original data set, shown in Appendix A (centered on a 16x16 pixel black/white representation), was modified to represent the various rotations and translations needed for the research reported in this thesis. B.1 shows the set of rotations*, demonstrated here using the Geneva font (see Appendix A). B.2 shows the left and right shifts**, again demonstrated using the Geneva font.

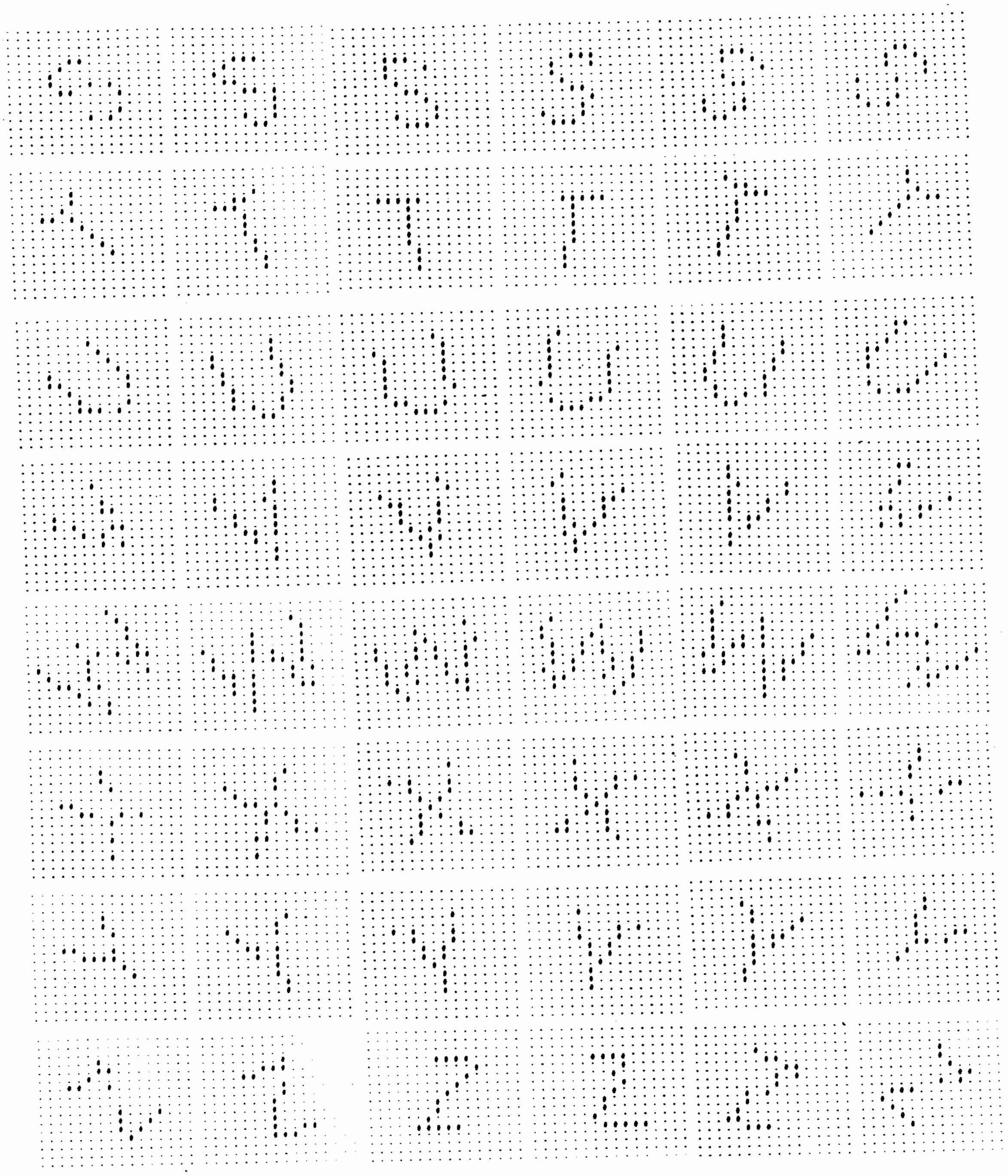
* These rotations were generated using two C modules, which were written by Daqiao Du, a graduate student at Portland State University, who is working on another pattern recognition project.
** The left and right shifts were implemented by the author.



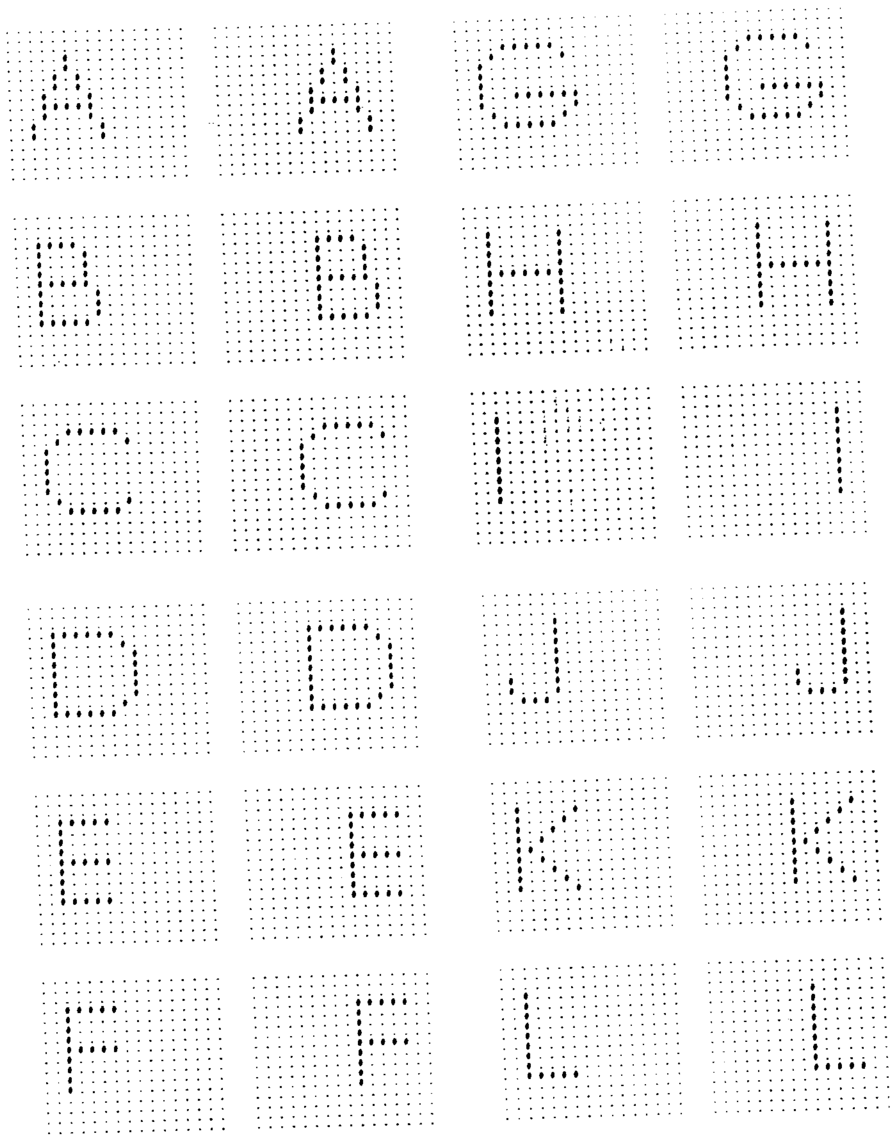
B.1: Rotation transformations (left-to-right): +45°, +22.5°, +11.25°, -11.25°, -22.5°, -45° (Geneva Font shown).



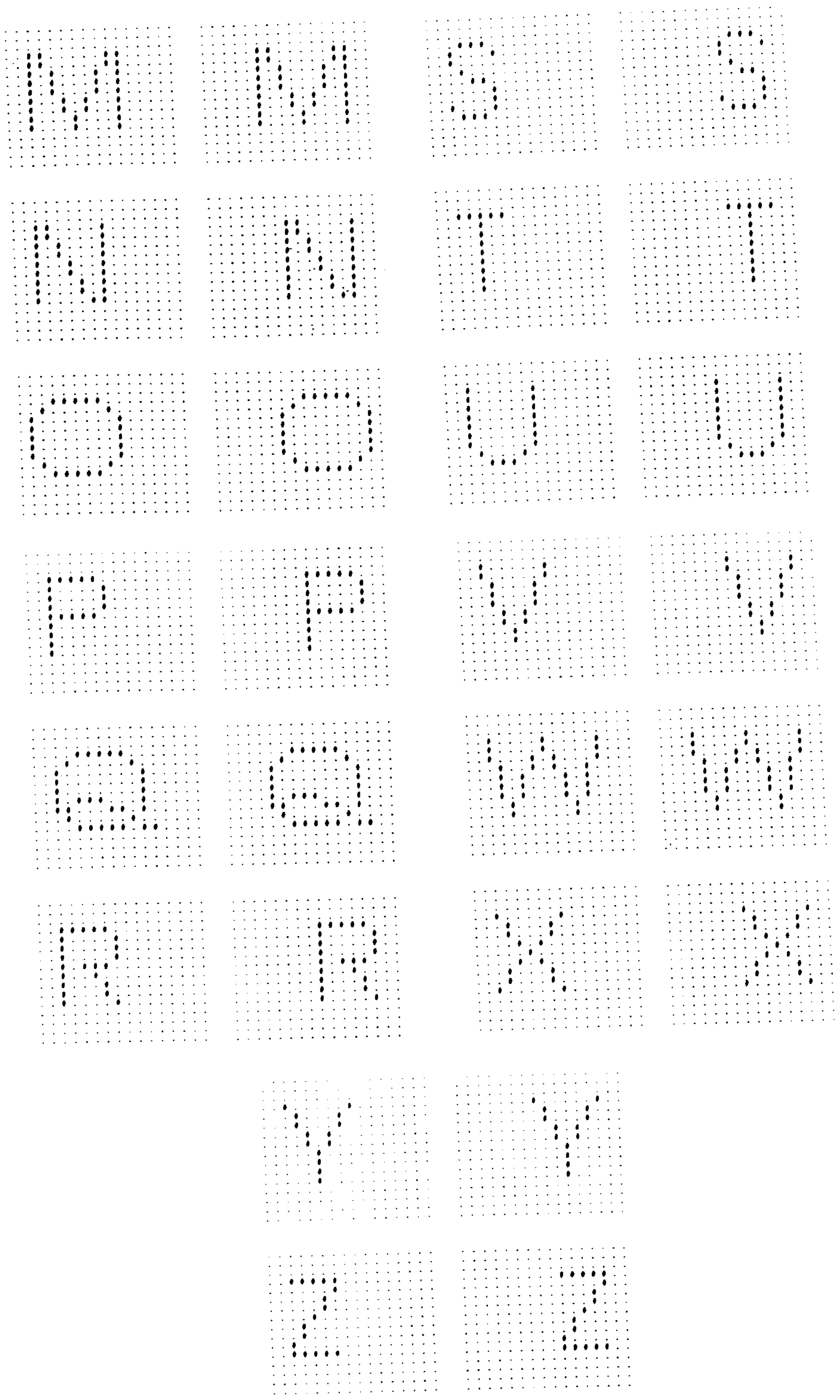
B.1 (continued): Rotation transformations (left-to-right): +45°, +22.5°, +11.25°, -11.25°, -22.5°, -45° (Geneva Font shown).



B.1 (continued): Rotation transformations (left-to-right): +45°, +22.5°, +11.25°, -11.25°, -22.5°, -45° (Geneva Font shown).



B.2: Translation transformations (left-to-right): -3-pixel shift, +3-pixel shift (Geneva Font shown).



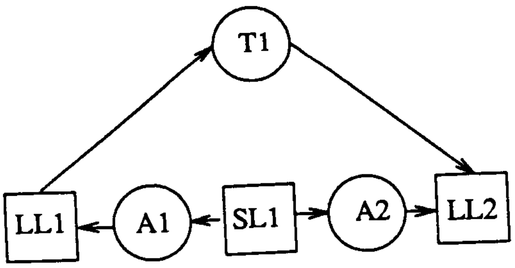
B.2: (continued): Translation transformations (left-to-right): -3-pixel shift, +3-pixel shift (Geneva Font shown).

APPENDIX C

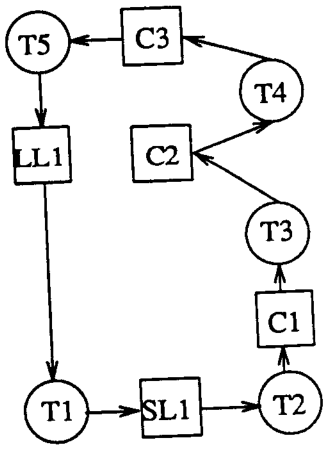
CONCEPTUAL GRAPHS OF INPUT PATTERN SET

This section depicts the conceptual graph representations, developed as part of the present research, of the capital letters of the English alphabet, following the approach presented in Chapter II.

Each of the pages that follow contains four CGs, representing four consecutive letters of the alphabet. An effort was made to construct these representations in a manner which visually resembles the structure of the letter in question, but always giving precedence to the laws and definitions of conceptual graph theory. For example, the first page contains the CGs of the letters A and B on top (left to right), and those of Letters C and D on bottom (left to right).

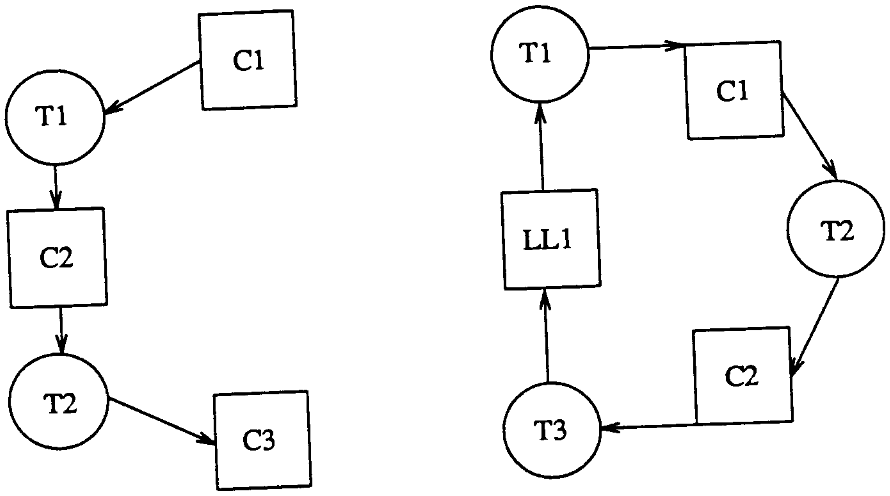


Letter A

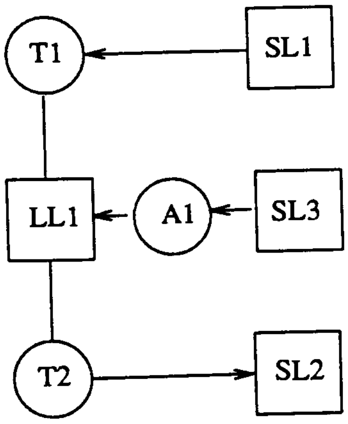


Letter B

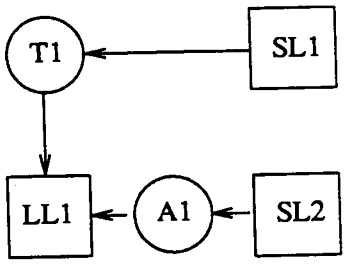
C.1: Conceptual graphs of Letters A and B.



C.2: Conceptual graphs of Letters C and D.

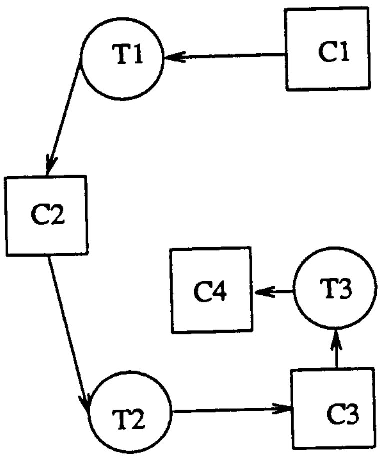


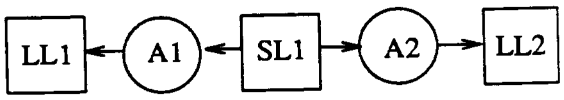
Letter E



LETTER F

C.3: Conceptual graphs of Letters E and F.

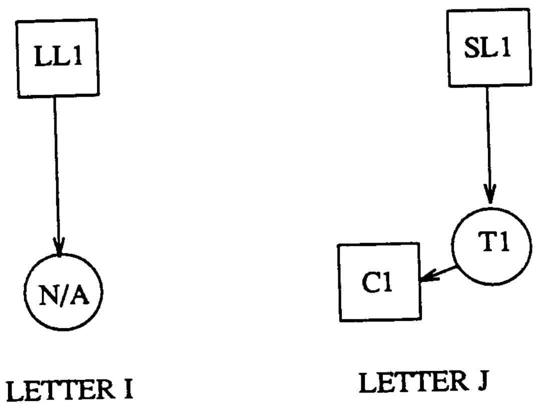




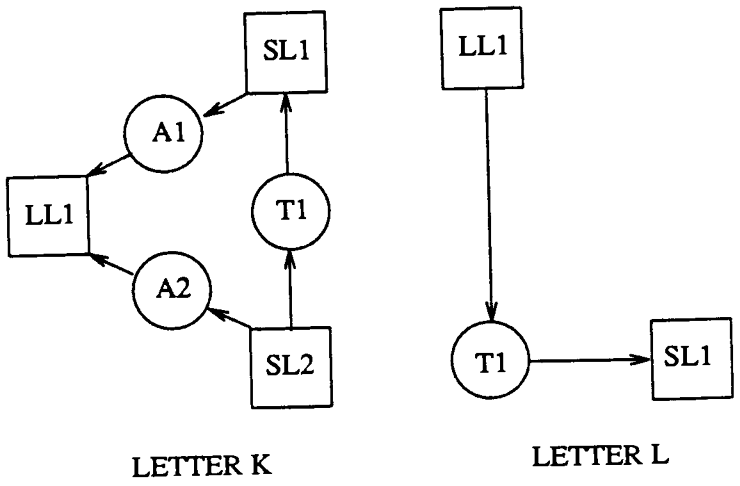
LETTER H

LETTER G

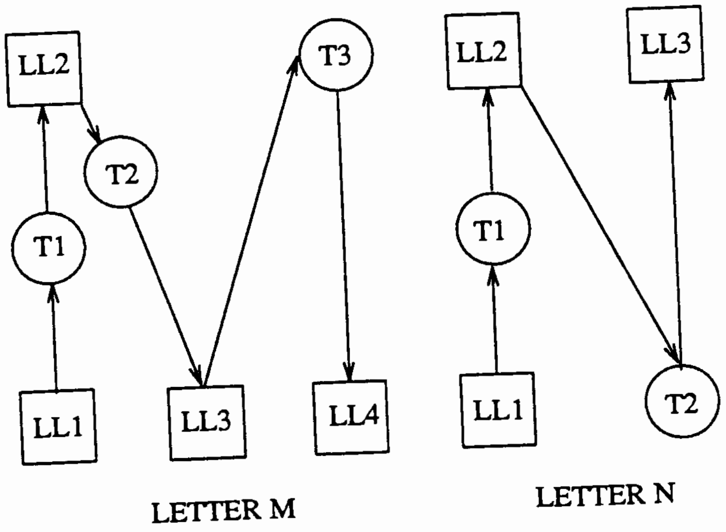
C.4: Conceptual graphs of Letters G and H.



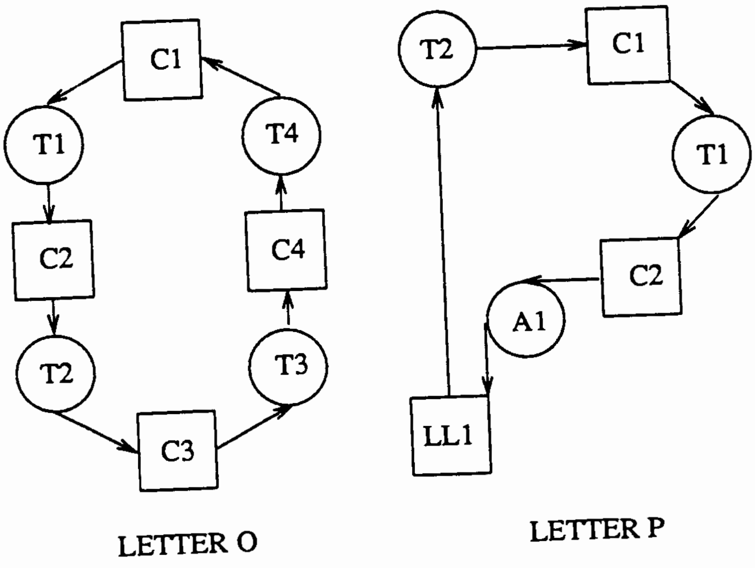
C.5: Conceptual graphs of Letters I and J.



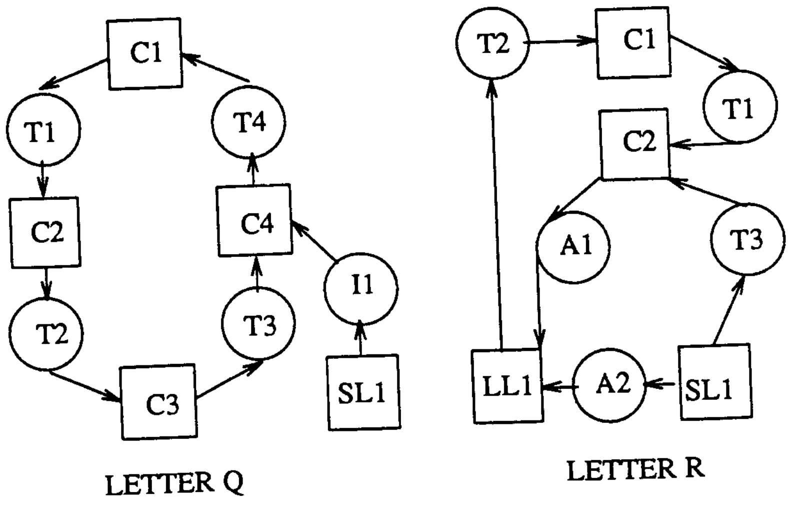
C.6: Conceptual graphs of Letters K and L.



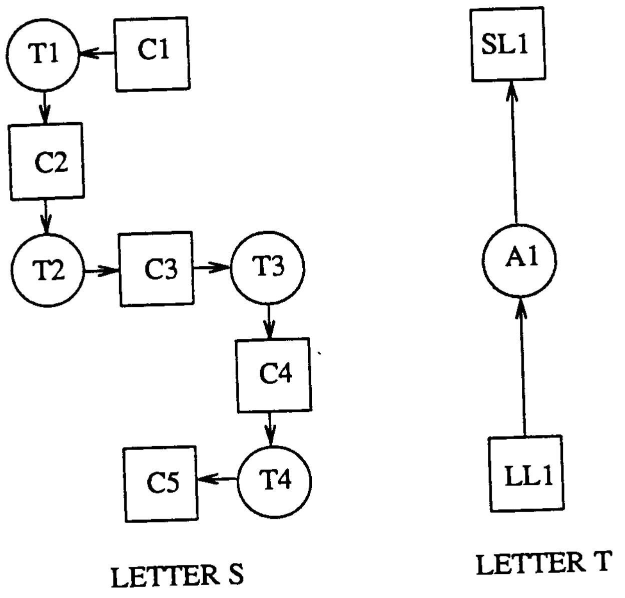
C.7: Conceptual graphs of Letters M and N.



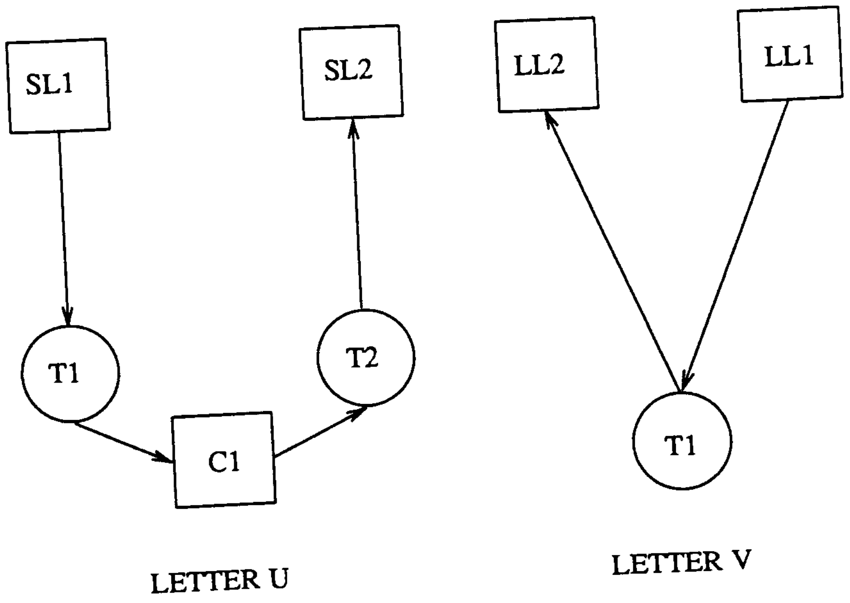
C.8: Conceptual graphs of Letters O and P.



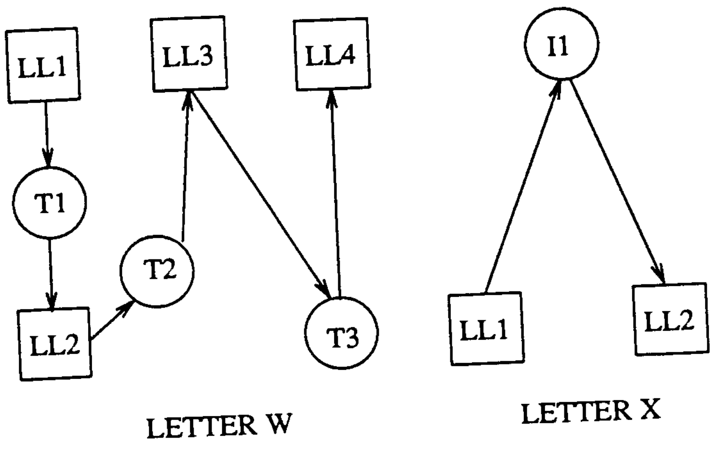
C.9: Conceptual graphs of Letters Q and R.



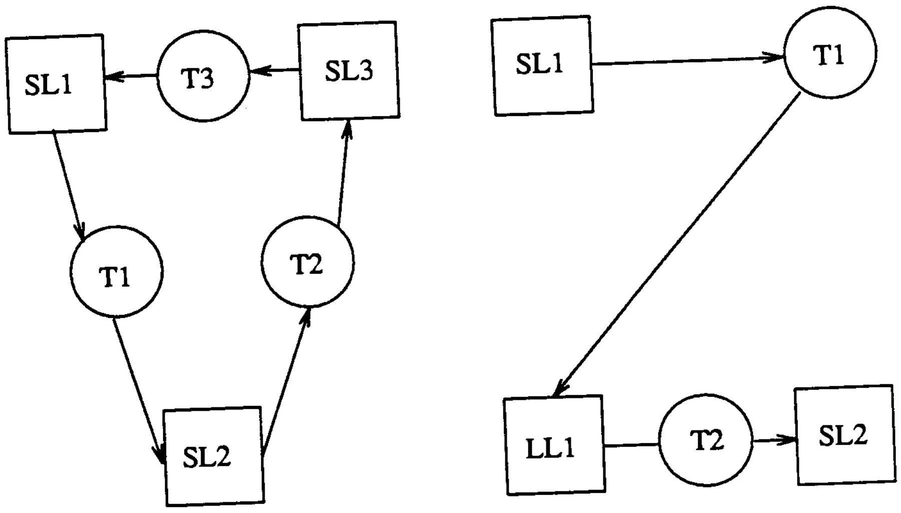
C.10: Conceptual graphs of Letters S and T.



C.11: Conceptual graphs of Letters U and V.



C.12: Conceptual graphs of Letters W and X.



LETTER Y

LETTER Z

C.13: Conceptual graphs of Letters Y and Z.

APPENDIX D

CONNECTION MATRICES AND VECTOR FORMS OF INPUT DATA SET

Each of the 26 tables in this appendix shows the Connection Matrix, and the C- nd R-vectors for one of the letters of the English alphabet. The connection matrices CMs), and the C- and R-vectors were constructed using the algorithm presented in Chapter II, and are based on the conceptual graph representations in Appendix C.

D.1: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER A

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	-1	1	0	0	0	0	0	0	0	0	0	0
T2	0	0	0	0	0	0	0	0	0	0	0	0
T3	0	0	0	0	0	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	1	0	0	0	-1	0	0	0	0	0	0	0
A2	0	1	0	0	-1	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	1	1	0	0	1	0	0	0	0	0	0	0
R-vector	1	0	0	0	0	1	1	0	0			

D.2: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER B

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	-1	0	0	0	1	0	0	0	0	0	0	0
T2	0	0	0	0	-1	0	0	1	0	0	0	0
T3	0	0	0	0	0	0	0	-1	1	0	0	0
T4	0	0	0	0	0	0	0	0	-1	1	0	0
T5	1	0	0	0	0	0	0	0	0	-1	0	0
A1	0	0	0	0	0	0	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	1	0	0	0	1	0	0	1	1	1	0	0
R-vector	1	1	1	1	1	0	0	0	0			

D.3: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER C

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	0	0	0	0	0	0	0	-1	1	0	0	0
T2	0	0	0	0	0	0	0	0	-1	1	0	0
T3	0	0	0	0	0	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	1	0	0	0	0	0	0	1	1	0	0	0
R-vector	1	1	1	0	0	0	0	0	0			

D.4: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER D

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	-1	0	0	0	0	0	0	1	0	0	0	0
T2	0	0	0	0	0	0	0	-1	1	0	0	0
T3	1	0	0	0	0	0	0	0	-1	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	1	0	0	0	0	0	0	1	1	0	0	0
R-vector	1	1	1	0	0	0	0	0	0			

D.5: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER E

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	1	0	0	0	-1	0	0	0	0	0	0	0
T2	-1	0	0	0	0	1	0	0	0	0	0	0
T3	0	0	0	0	0	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	1	0	0	0	0	0	-1	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	1	0	0	0	1	1	1	0	0	0	0	0
R-vector	1	1	0	0	0	1	0	0	0			

D.6: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER F

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	1	0	0	0	-1	0	0	0	0	0	0	0
T2	0	0	0	0	0	0	0	0	0	0	0	0
T3	0	0	0	0	0	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	1	0	0	0	0	-1	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	1	0	0	0	1	1	0	0	0	0	0	0
R-vector	1	0	0	0	0	1	0	0	0			

D.7: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER G

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	0	0	0	0	0	0	0	-1	1	0	0	0
T2	0	0	0	0	0	0	0	0	-1	1	0	0
T3	0	0	0	0	0	0	0	0	0	-1	1	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	0	0	0	0	0	0	0	1	1	1	1	0
R-vector	1	1	1	0	0	0	0	0	0			

D.8: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER H

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	0	0	0	0	0	0	0	0	0	0	0	0
T2	0	0	0	0	0	0	0	0	0	0	0	0
T3	0	0	0	0	0	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	-1	0	0	0	1	0	0	0	0	0	0	0
A2	0	1	0	0	-1	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	1	1	0	0	1	0	0	0	0	0	0	0
R-vector	0	0	0	0	0	1	1	0	0			

D.9: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER I

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	0	0	0	0	0	0	0	0	0	0	0	0
T2	0	0	0	0	0	0	0	0	0	0	0	0
T3	0	0	0	0	0	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	1	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	1	0	0	0	0	0	0	0	0	0	0	0
R-vector	0	0	0	0	0	0	0	0	1			

D.10: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER J

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	0	0	0	0	1	0	0	-1	0	0	0	0
T2	0	0	0	0	0	0	0	0	0	0	0	0
T3	0	0	0	0	0	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	0	0	0	0	1	0	0	1	0	0	0	0
R-vector	1	0	0	0	0	0	0	0	0			

D.11: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER K

Connection Matrix												
	L1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	0	0	0	0	1	-1	0	0	0	0	0	0
T2	0	0	0	0	0	0	0	0	0	0	0	0
T3	0	0	0	0	0	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	1	0	0	0	-1	0	0	0	0	0	0	0
A2	1	0	0	0	0	-1	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	1	0	0	0	1	1	0	0	0	0	0	0
R-vector	1	0	0	0	0	1	1	0	0			

D.12: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER L

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	1	0	0	0	-1	0	0	0	0	0	0	0
T2	0	0	0	0	0	0	0	0	0	0	0	0
T3	0	0	0	0	0	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	1	0	0	0	1	0	0	0	0	0	0	0
R-vector	1	0	0	0	0	0	0	0	0			

D.13: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER M

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	-1	1	0	0	0	0	0	0	0	0	0	0
T2	0	-1	1	0	0	0	0	0	0	0	0	0
T3	0	0	-1	1	0	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	1	1	1	1	0	0	0	0	0	0	0	0
R-vector	1	1	1	0	0	0	0	0	0			

D.14: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER N

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	-1	1	0	0	0	0	0	0	0	0	0	0
T2	0	-1	1	0	0	0	0	0	0	0	0	0
T3	0	0	0	0	0	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	1	1	1	0	0	0	0	0	0	0	0	0
R-vector	1	1	0	0	0	0	0	0	0			

D.15: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER O

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	0	0	0	0	0	0	0	-1	1	0	0	0
T2	0	0	0	0	0	0	0	0	-1	1	0	0
T3	0	0	0	0	0	0	0	0	0	-1	1	0
T4	0	0	0	0	0	0	0	1	0	0	-1	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	0	0	0	0	0	0	0	1	1	1	1	0
R-vector	1	1	1	1	0	0	0	0	0			

D.16: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER P

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	0	0	0	0	0	0	0	-1	1	0	0	0
T2	-1	0	0	0	0	0	0	1	0	0	0	0
T3	0	0	0	0	0	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	1	0	0	0	0	0	0	0	-1	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	1	0	0	0	0	0	0	1	1	0	0	0
R-vector	1	1	0	0	0	1	0	0	0			

D.17: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER Q

Connection Matrix											
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	0	0	0	0	0	0	0	-1	1	0	0	0
T2	0	0	0	0	0	0	0	0	-1	1	0	0
T3	0	0	0	0	0	0	0	0	0	-1	1	0
T4	0	0	0	0	0	0	0	1	0	0	-1	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	-1	0	0	0	0	0	1	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	0	0	0	0	1	0	0	1	1	1	1	0
R-vector	1	1	1	1	0	0	0	1	0			

D.18: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER R

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	0	0	0	0	0	0	0	-1	1	0	0	0
T2	-1	0	0	0	0	0	0	1	0	0	0	0
T3	0	0	0	0	-1	0	0	0	1	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	1	0	0	0	0	0	0	0	-1	0	0	0
A2	1	0	0	0	-1	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	1	0	0	0	1	0	0	1	1	0	0	0
R-vector	1	1	1	0	0	1	1	0	0			

D.19: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER S

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	0	0	0	0	0	0	0	-1	1	0	0	0
T2	0	0	0	0	0	0	0	0	-1	1	0	0
T3	0	0	0	0	0	0	0	0	0	-1	1	0
T4	0	0	0	0	0	0	0	0	0	0	-1	1
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	0	0	0	0	0	0	0	1	1	1	1	1
R-vector	1	1	1	1	0	0	0	0	0			

D.20: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER T

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	0	0	0	0	0	0	0	0	0	0	0	0
T2	0	0	0	0	0	0	0	0	0	0	0	0
T3	0	0	0	0	0	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	1	0	0	0	-1	0	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	1	0	0	0	1	0	0	0	0	0	0	0
R-vector	0	0	0	0	0	1	0	0	0			

D.21: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER U

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	0	0	0	0	-1	0	0	1	0	0	0	0
T2	0	0	0	0	0	1	0	-1	0	0	0	0
T3	0	0	0	0	0	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	0	0	0	0	1	1	0	1	0	0	0	0
R-vector	1	1	0	0	0	0	0	0	0			

D.22: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER V

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	-1	1	0	0	0	0	0	0	0	0	0	0
T2	0	0	0	0	0	0	0	0	0	0	0	0
T3	0	0	0	0	0	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	1	1	0	0	0	0	0	0	0	0	0	0
R-vector	1	0	0	0	0	0	0	0	0			

D.23: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER W

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	-1	1	0	0	0	0	0	0	0	0	0	0
T2	0	-1	1	0	0	0	0	0	0	0	0	0
T3	0	0	-1	1	0	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	1	1	1	1	0	0	0	0	0	0	0	0
R-vector	1	1	1	0	0	0	0	0	0			

D.24: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER X

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	0	0	0	0	0	0	0	0	0	0	0	0
T2	0	0	0	0	0	0	0	0	0	0	0	0
T3	0	0	0	0	0	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	-1	1	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	1	1	0	0	0	0	0	0	0	0	0	0
R-vector	0	0	0	0	0	0	0	1	0			

D.25: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER Y

Connection Matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	0	0	0	0	-1	1	0	0	0	0	0	0
T2	0	0	0	0	0	-1	1	0	0	0	0	0
T3	0	0	0	0	1	0	-1	0	0	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	0	0	0	0	1	1	1	0	0	0	0	0
R-vector	1	1	1	0	0	0	0	0	0			

D.26: CONNECTION MATRIX AND C- & R-VECTORS FOR THE LETTER Z

Connection matrix												
	LL1	LL2	LL3	LL4	SL1	SL2	SL3	C1	C2	C3	C4	C5
T1	1	0	0	0	-1	0	0	0	0	0	0	0
T2	-1	0	0	0	0	1	0	0	0	0	0	0
T3	0	0	0	0	0	0	0	0	0	0	0	0
T4	0	0	0	0	0	0	0	0	0	0	0	0
T5	0	0	0	0	0	0	0	0	0	0	0	0
A1	0	0	0	0	0	0	0	0	0	0	0	0
A2	0	0	0	0	0	0	0	0	0	0	0	0
I1	0	0	0	0	0	0	0	0	0	0	0	0
N/A	0	0	0	0	0	0	0	0	0	0	0	0

Vector Form												
C-vector	1	0	0	0	1	1	0	0	0	0	0	0
R-vector	1	1	0	0	0	0	0	0	0			