LEARNING TO READ ALOUD: A NEURAL NETWORK APPROACH USING SPARSE DISTRIBUTED MEMORY

Umesh Dwarkanath Joglekar

July 1989

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 89.27

NASA Cooperative Agreement Number NCC 2-387

RIACS

Research Institute for Advanced Computer Science

LEARNING TO READ ALOUD: A NEURAL NETWORK APPROACH USING SPARSE DISTRIBUTED MEMORY

Umesh Dwarkanath Joglekar

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 89.27
July 1989

The automation of the process of learning from examples has been of intense interest to AI researchers for a long time. This interest, together with recent breakthroughs in understanding the learning capabilities of "neural networks", or massively parallel distributed processing systems, have rekindled interest in neural network research. Additional interest stems from the possibility of constructing systems that learn in problem domains for which we have little understanding. Such systems therefore offer the additional attraction of enriching our understanding of a particular problem domain.

Reading aloud is among the problems that do not seem amenable to solution by use of standard algorithmic procedures. NETtalk (Sejnowski and Rosenberg, 1986) demonstrated that it is possible for a parallel network of computing units to be trained to form internal representations of the regularities in the training set. The NETtalk experiment opens the door to a host of questions such as what kind of network architecture is really suited to solving problems of this nature or what learning strategies could be used. In particular, we may ask whether it is possible to devise a system based on distributed representations that will be able to not only form abstractions of regularities in the training set but also translate these to other test data to show equally good generalization.

We attempt to solve the same text-to-phoneme mapping problem using Sparse Distributed Memory (Kanerva, 1984). We discuss an iterative supervised learning scheme that involves modification of thresholds of output units and changes in the data counters. (This is a modification of the generalized delta rule for the SDM case). A method is discussed to solve problems arising out of highly correlated real world data sets. The scheme is compared with related models. The network is trained using this scheme with examples drawn from informal speech. Performance of the trained network compares favorably with NETtalk. The trained network shows good generalization.

This research was supported in part by NASA co-operative agreement, NCC 2-408, with the Universities Space Research Association and was carried out at the Research Institute for Advanced Computer Science, NASA Ames Research, Moffett Field, CA 94035

UNIVERSITY OF CALIFORNIA
Santa Barbara

Learning to Read Aloud: A Neural Network Approach Using Sparse Distributed Memory

A Thesis submitted in partial satisfaction
of the requirements for the degree of
Master of Science
in
Electrical and Computer Engineering
with emphasis
on
Computer Science

by

Umesh Dwarkanath Joglekar

Committee in charge:

Professor Terence R. Smith, Chairman
Professor Omer Egecioglu
Professor Yuan-Fang Wang

December 1988

to my Mother,
and my Siblings,
and the loving memory
of my Father.

PAGE_____INTENTIONALLY [redacted]

Acknowledgements

I would like to thank my adviser, Prof . Terence Smith for all his help and support during my stay at UCSB. He made me aware of the field and its potential. He went through several drafts of this report and made many valuable suggestions. Dr. Egecioglu and Dr. Wang, the other two members of the thesis committee offered valuable suggestions on some aspects of the work.

This research was supported in part by NASA co-operative agreement NCC 2-408, with the Universities Space Research Association, and was carried out at Research Institute for Advanced Computer Science at NASA Ames Research Center in Moffett Field, Ca. I am grateful to Dr. Pentti Kanerva for offering me the opportunity to work on this problem. He was always willing to listen to my ideas and discuss their merits. He also showed enormous patience in going through various earlier drafts of the report.

I am deeply grateful to Dr. Joseph Cupin for providing the transcriptions which served as training and test material. Many of my earlier attempts to secure the data from other sources had proved unfruitful.

I am also grateful to Jonni Kanerva for his help in aligning the orthographic and phonetic streams of the data and cleaning up the transcriptions of errors and ambiguities.

The simulations reported here, were carried out on a simulator written in C, at first on a Sequent Balance Computer and later on SUN 3/50 machines. The

facilities management people at RIACS, notably Dave Gehrt, Dr. Bob Brown, and Phil Klimbal in particular, offered excellent facilities support. Thanks are also due to David Cohn. The summer before I joined RIACS, David Cohn wrote some well documented SDM simulators. I borrowed heavily from his work for my initial experiments to study the behavior of SDM. My later programs were, to some extent, influenced by his work.

At RIACS, I had the good fortune of interacting with Dr. James Keeler and Dr. Harrison Leong. They both made some valuable suggestions. Credit for the explanation of the behavior of the learning mechanism on the parity problem goes to Dr. Louis Jaeckel.

Gilbert Pitney introduced me to this field. I have benefitted from numerous discussions with him on various aspects of 'neural networks'. He was the poor guinea pig when I wanted to discuss any new ideas.

Professor Smith, Dr. Kanerva, Dr. Keeler, Dr. Surkan, Gilbert Pitney, and Dr. Leong were among the many people who contributed through their suggestions after reading earlier drafts of this report. I would like to thank them all for their suggestions.

Above all, I would like to thank members of my family. I cannot adequately acknowledge all of the ways in which they have contributed to my work. They have offered love, encouragement and enduring support in all my endeavors. I would like to dedicate this work to my mother, my siblings and the loving memory of my father.

Table of Contents

List of Figures

List of Displays


INTERNATIONALE

Introduction

The automation of the process of learning from examples has been of intense interest to AI researchers for a long time (see for example, Winston (1975), Michalski and Chilausky (1980), Mitchell (1982)). This interest, together with recent breakthroughs in understanding the learning capabilities of 'neural networks', or massively parallel distributed processing systems, have rekindled interest in neural network research. Additional interest stems from the possibility of constructing systems that learn in problem domains for which we have little understanding (see for example, Sejnowski and Rosenberg (1986), Tesauro and Sejnowski (1988b), Elman and Zipser (1988), Plaut and Hinton (1987)). Such systems therefore offer the additional attraction of enriching our understanding of a particular problem domain.

In the following report we describe an attempt to solve a problem of text-to-phoneme mapping, which does not appear amenable to solution by use of standard algorithmic procedures. We describe experiments based on a relatively novel model of distributed processing. We show that this model (Sparse Distributed Memory or SDM) can be used in an iterative supervised learning mode to solve our problem. We suggest additional improvements aimed at obtaining better performance. The title 'Learning to Read Aloud' has been used in a restricted sense to refer to pronouncing written text, i.e., mapping text to phonemes. No attempt at any 'graphemic recognition' is

included in this. Some other studies address this aspect of the problem (See, Reggia and Berndt, 1985).

This report is structured as follows: In the first section, we describe some of the problems associated with converting text to speech. Second section contains a brief description of parallel distributed processing with a description of NETtalk, while the third section describes the particular model of distributed processing that is used for solving the text-to-phoneme problem. Following this, in section four, we describe the main results obtained in the experiments using SDM. The learning scheme is described in detail. In section five, we describe the design decisions and contrast them with those of NETtalk. Then, in section six, we review some of the important related issues which should be raised, understood and addressed in further work. In Appendix A, we show how SDM can be viewed as a three-layered network and show how the learning rule is a modification of the generalized delta rule, as applied to the case of SDM. In Appendix B, we give a list of symbols used in the transcriptions. Finally, in Appendix C, we describe the performance of the learning scheme on the "parity problem ".

ONE

Text-to-speech

1.1 Introduction

Reading aloud is among the problems that cannot be easily solved by conventional computing methods. An automated procedure to convert unrestricted text to speech can lead to a host of exciting new applications. Possible applications include:

1. Reading machines for the blind. These are already commercially available (Telesensory Systems Inc.)
2. Transmitting information from data-bases via telephone lines for consumer applications (e.g., banks, airline reservations, and weather).
3. "Talking" books to teach reading.
4. "Talking" computer terminals and instrument panels.
5. Personal speech prostheses for use by nonvocal persons.

The task of developing an automated text-to-speech procedure is complex for various reasons. From the point of view of producing natural sounding speech, the simplest and the most effective way is to employ a dictionary of commonly used words. Dictionary lookup is successful for small vocabularies, but for any natural language, there is no such thing as a complete vocabulary, since words are continuously being added to the lexicon while

others are dropped. In a language such as English, using letter-to-sound rules to convert text to speech is unsatisfactory because the underlying linguistic structure is ignored. An approach using letter-to-sound rules also faces the problem that the most frequently occurring words in the language violate these rules. In order to attain high performance many systems have to rely upon complex linguistic analysis (Allen, . 1985) and a large variety of ad hoc rules. However, syntactic analysis is difficult since natural languages have context sensitive grammars. In speech, stress rhythm and inflexion help in providing a listener with valuable information. It is almost impossible to convey this information in speech that is automatically generated from unrestricted text.

Some of the difficulties in speech synthesis as well as speech recognition arise from the difficulties in processing the underlying natural languages. Natural languages contain a large number of contextual rules, as well as exceptions to these rules. Schemes using distributed representations and distributed processing are well suited to solving such problems since they are sensitive to context and exception. Many of the problems in language processing deal with the syntax. Distributed representations and distributed processing offer a promising approach to solving these. For interesting work in this area, see, Hanson and Kegl (1987), and Fanty (1985).

In later parts of this report, distributed representations and distributed processing are discussed in greater detail. Distributed representations are being increasingly used to solve speech related problems, notably speech recognition problems. For some of the work in this area, see Bourland and

Wellekens (1987), Cohen et al. (1987), Elman and Zipser (1988), Tank and Hopfield (1987), and Waibel et al. (1987).

Although the work discussed in this report addresses few of the problems that have been discussed so far, it offers a new approach to solving the text-to-speech problem. Clearly much work remains to be done. Much further research is needed in this difficult area in order to arrive at a method that can overcome the difficulties mentioned.

TWO

Parallel Distributed Processing

Computers are better than humans at certain kinds of tasks, for example, performing complex numerical computations or manipulating long strings of symbols. Living organisms, however, are far superior to computers in certain areas of perception and cognition. Humans can recognize a familiar person in different clothes or in a crowd or with a different hair style. Conventional computers cannot match human beings in such tasks. Distributed representations and distributed processing offer a way to mimic some of these human abilities to a certain extent.

Hubert Dreyfus and Stuart Dreyfus (1986) discuss a hierarchy of human skills with the novice at the bottom and expert at the top. In their model, problem-solving at the lowest skill level is characterized by application of basic rules to attain a desired goal. At the highest skill level goal attainment is sought through recall of abstractions of similar past situations and the memories of related past actions.

There seems to be a growing consensus among researchers that networks of distributed processing units, i.e., artificial neural nets, can be used for storage and retrieval of patterns to mimic the human abilities of formulating abstractions and recalling them when needed. NETtalk (Sejnowski and Rosenberg, 1986) demonstrates that an artificial neural network can indeed be used to form such abstractions, also called internal representations, and that

they can be retrieved when needed. In NETtalk these internal representations are formed structurally in the network.

Many models of distributed processing use a large number of very simple processing units. Like neurons in the brain these processing units take a number of inputs from different units and compute a function of these inputs. Since these are viewed as very simple computational models of a neuron's input output behavior they are sometimes referred to as 'neurons' and a network of such processing units is sometimes referred to as a 'neural network'.

2.1 Neural Networks

A computing unit receives a number of inputs. It computes some function of these inputs called the 'transfer function'. The transfer function maybe a threshold logic unit or a sigmoidal transfer function.

Different networks can be formed based on different connectivity patterns (i.e., interconnections among the computing elements) and different firing rules (i.e., the particular function computed by the computing element).

A network of such computing elements can be formed in different layers such that computing elements in each layer send their output to each unit in the next layer. This is a feed-forward network. There are no interconnections within a given layer. Units in the first layer receive input from outside the network. This input is a vector that is to be associated with an output vector of the last layer of the network. In particular, an input to the first layer is clamped. Based on the input the units in the first layer produce some output which is the input to the next layer. This

Input in turn produces some output at the second layer which is fed forward in the same manner until an output is produced at the final layer.

2.2 Description of NETtalk

NETtalk employed a three-layered feed-forward network to associate a moving window of seven characters with the correct phoneme. The second and third layer in this network has modifiable weights on the connections between the layers. Every computing element in the first layer (Input layer) sends its output to every computing element in the second layer. Every computing element in second layer sends its output to every computing element in third layer (Output layer). Since the second layer is not accessible from outside, it is called the hidden layer .

Input to the input layer is from a character window where center character is mapped to the corresponding phonemic output in the output layer. Initially an input is applied to the first layer and after the network settles to a particular output it is compared with a corresponding correct training instance of the output. If there is any error it is back-propagated to adjust the weights of neurons using the back propagation of error rule (or the generalized delta rule) developed by Rumelhart, Hinton, and Williams (1986).

NETtalk demonstrates that it is possible for a network to be trained to form internal representations of the interrelationships in a training set. There have been some other studies which report good generalizations (e.g. PARSNIP, Hanson and Kegl, 1987).

NETtalk leads to a host of questions concerning the network architecture most suited to problems of this nature, the most appropriate strategies to be used for training such networks, and whether the performance of these distributed processing models compares favorably with sophisticated systems like MITalk (Allen, 1985). A question of particular interest concerns whether it is possible to devise a system based on distributed representations that will be able both to form abstractions and to translate this learned relationship to other test data (i.e., to give good generalization).

THREE

Sparse Distributed Memory

Sparse Distributed Memory (SDM) is a distributed model of memory proposed by Kanerva (1984). It is capable of handling enormously large address spaces and is capable of associative recall in the presence of noise.

The realization of the memory is attained through an actualization of a small subset of the address space. This subset is a random sample of the address space. The strategy for storing a pattern consists of storing it in a distributed manner. In the simplest case, the input pattern is stored at all the locations whose addresses are sufficiently similar to the input pattern. Hamming distance is used as a metric of similarity.

Reading from the memory consists of pooling the information contents from addresses most similar to a specified read address and taking a majority decision for each of the features of the pooled information to arrive at the output pattern.

3.1 How SDM Works

SDM can be viewed as a black box, with two inputs and an output. One of the inputs is an address pattern and the other input is the pattern to be stored. That is, the memory operates by storing a pattern at an address. In the read mode, given an address pattern the memory retrieves a related data pattern.

The internal structure of this black box is just that of a random access memory (RAM). It possesses a set of addresses and associated storage bins at these addresses. It is different from RAM in that not all possible addresses of a contiguous address space are present. Only a small subset of the address space is present. Storage in a conventional RAM consists of bit registers. In SDM it is instead a set of counters (one counter corresponds with each bit in a data register of a RAM). There is also a similarity indicator. The memory works by storing a pattern at similar addresses. Hamming distance is used as a measure of similarity.

Figure 1 shows the addresses for storage locations on the left and the actual associated storage bins on the right.

SDM operations can be stated in terms of three primitives.

1. Selecting locations similar to pattern X.
2. Storing pattern Y at Pattern X.
3. Retrieving a pattern given a probe X.

3.1.1 Selecting locations similar to pattern X

We start with some similarity criterion. Let us first consider the concept of Hamming distance. We say that patterns x 1 and y 1 are a distance d apart if they differ in d positions. Thus, the smaller the number of positions in which two patterns differ, the more similar they are. In this example given in Figure 2 the Address X is 011101. All addresses which do not differ in more than r positions from address X are considered to be similar to address X. These are shown shaded. Each of these are at distances indicated in the distance column from

address *X*. For example, the first address 010011 differs from address *X* in the third fourth and fifth positions. So the total number of positions in which it differs from address *X* is 3. If the distance between the location's address and the address *X* is less than or equal to the radius then the address is selected. This is shown by a 1 in the select column for the selected addresses and a 0 for those which have not been selected. All the selected addresses are shown in gray. The parameter *r* is called the *select radius*. (It indicates, in fact, the maximum allowable *dissimilarity* in selecting the addresses). In the example shown, the select radius r has a value of 2. Thus the first address has not been selected. Display 1 gives a formal statement of the select operation.

3.1.2 Storing a pattern

When storing a pattern *Y* at an address *X* we first select locations given *X*. To store *Y* at these selected locations we proceed as follows. If a bit in *Y* is one, we increment the counters for all the selected addresses. If a bit is 0, we decrement the counters at those addresses. This is done for all bits in *Y*.

In the example shown in Figure 3, pattern 001110 is to be stored at 011101. First we select locations that have addresses similar to 011100 (that differ from 011100 in no more than 2 positions, as the radius *r* has a value 2). These are the the locations marked in gray.

In this example, the first bit in the pattern to be stored, i.e. 001110 is 0. So, for all the selected locations the counter in the first position is decremented. The second bit also happens to be 0, so counters in the second position for all the selected locations are decremented. The third bit is 1 so counters in the third

position for the selected locations are incremented. Following this method all counters of the selected locations are updated. Figure 3 shows the situation after updating all the counters in the selected locations. Display 2 shows a formal statement of the write operation.

3.1.3 Retrieving a pattern

Given a probe pattern *X* we wish to retrieve an associated pattern. We first select the addresses that are similar to *X*. Figure 4 shows selected locations in gray. For each position in the selected locations, we pool the contents. This is the pooled sum shown in Figure 4 at the bottom right.

For each of these positions we now threshold the sum. If the sum is above the threshold, we output a 1 in the corresponding position otherwise we output a zero. Display 3, shows a formal statement of the retrieve operation.

3.2 SDM Modes of Operation

In its simplest mode of operation, SDM works as a pattern recognizer. In each write operation SDM modifies the abstraction of the stored pattern. With SDM the problem of learning tasks is transformed to storing and retrieving encoded tasks.

3.2.1 Auto-associative Mode

In an auto-associative mode a pattern *X* is stored at address *X*. This gives SDM an ability to use iterative reads to enhance fault-tolerance. That is, given *X* 1 we retrieve *Y1*. Then reading at address *Y* 1 we retrieve *Y* 2. Continuing

in this fashion we find that under certain conditions we will converge at the correct stored pattern. That is *X 1* is stored at *X 1*, then under conditions of low noise and with relatively few patterns in memory, we will be able to retrieve *X 1* by reading at *X 2*, which may be slightly different from *X 1*. If the probe is sufficiently near the stored pattern, the reading procedure is guaranteed to converge if the number and nature of stored patterns is such that the signal-to-noise ratio remains within acceptable limits. If the probe pattern is farther out, the reading procedure is not guaranteed to converge. This property of SDM can be used for tasks of pattern completion or simple fault-tolerant applications.

SDM works well in this fashion when the number of stored patterns is less than about 10% of the number of locations (Kanerva, Cohn, and Keeler, 1986). It is necessary to have the addresses distributed randomly throughout the address space in order to get good predictable performance.

3.2.2 Sequential Mode

SDM can be used in another mode to store sequences. To store a sequence '*X1, X2, X3, X4 ,...., X n*', store *X 2* at address *X1* , store *X 3* at address *X 2*, store *X 4* at address *X 3*, and so on. The sequence can be retrieved by using a probe pattern *X* .

Retrieving sequences with this scheme may run into problems when two sequences have an identical beginning. To counter problems of this nature, Kanerva proposes a modification of SDM incorporating the use of "folds" (see Kanerva, 1984). Sequential mode and operation of folds are not relevant to the study described in this report.

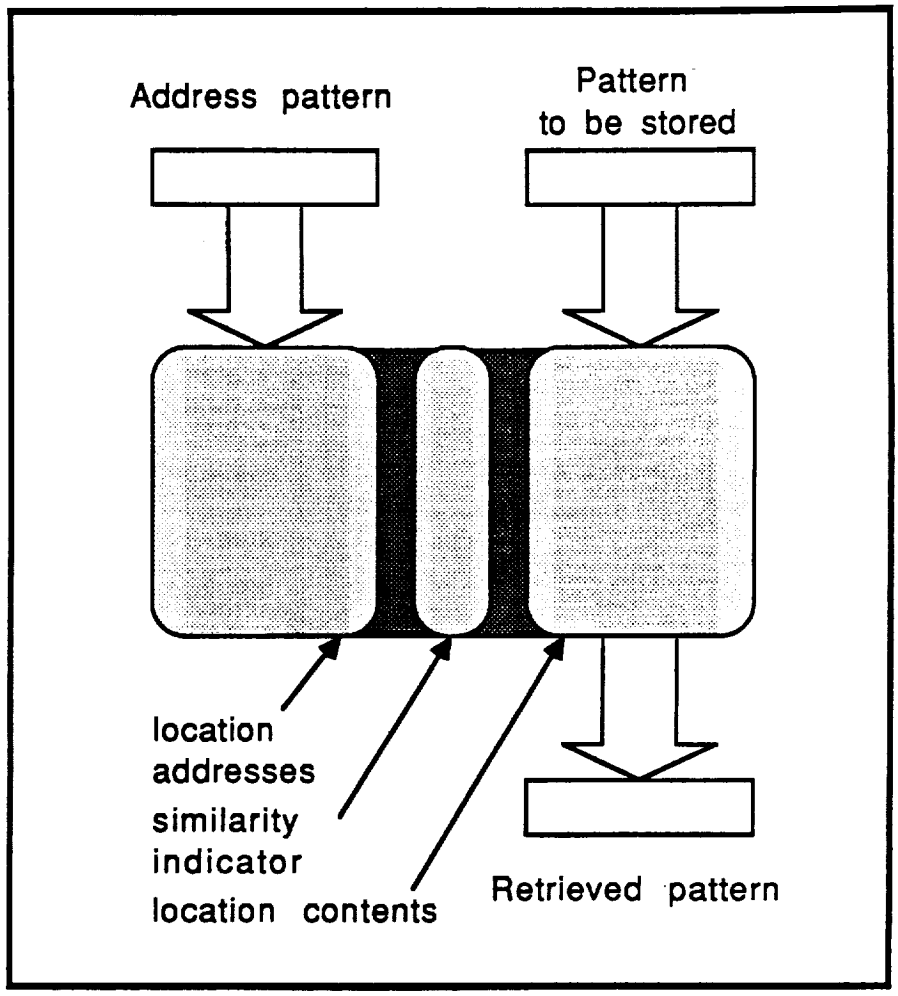

Figure 1 - Internal Structure of the Memory.

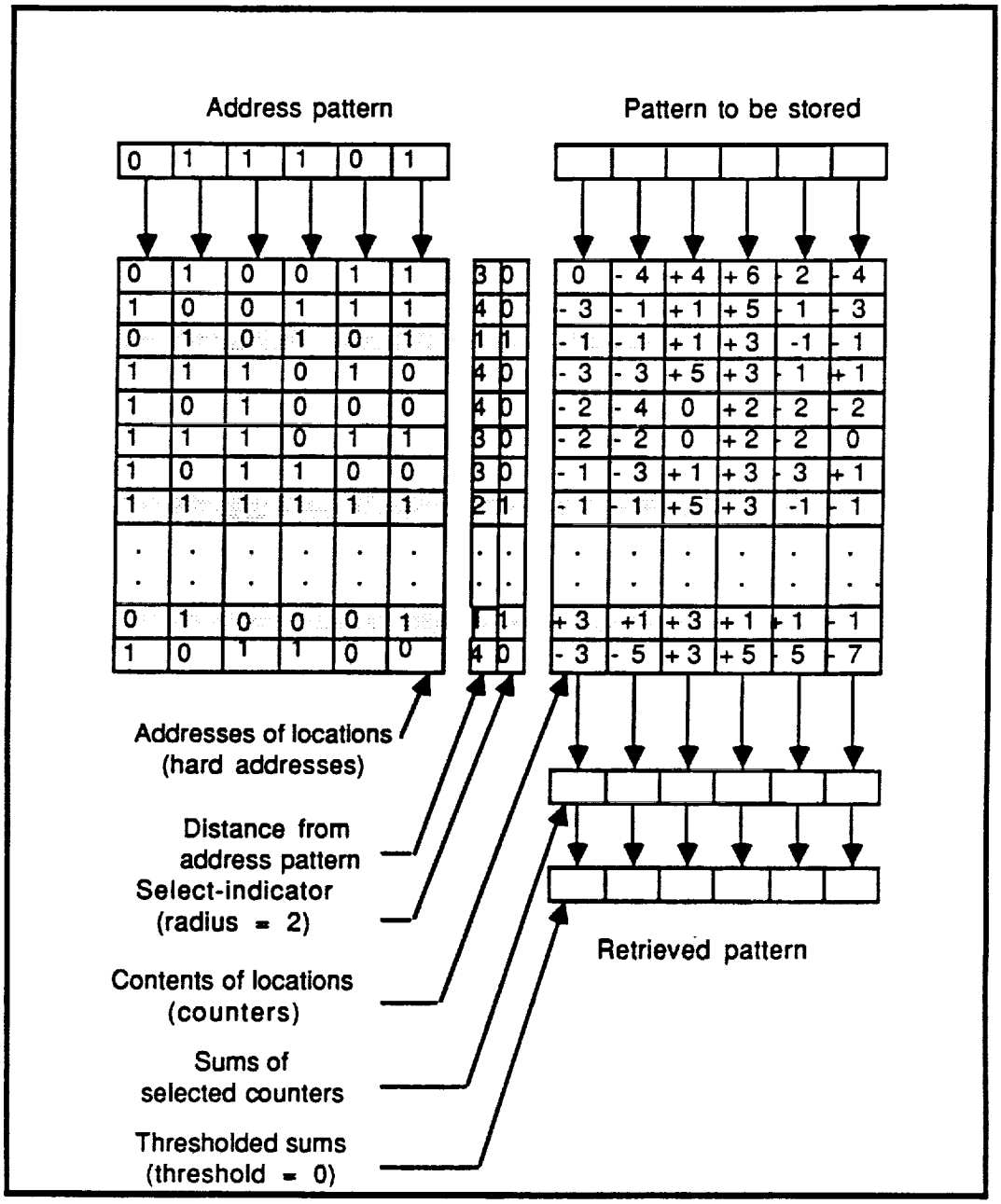

Figure 2 - Selecting Locations. Locations which do not differ in more than two places from the address pattern are selected. They are shown in gray tone.

Display 1- Selecting Locations.

Let

M be the set of actual memory locations,

T be the reference address,

n be the number of bits in the address,

r be the select radius,

d(x, y) be the distance between x and y:

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i| .$$

Then

S(T), the set of selected locations, is given by

$$\mathbf{S(T)} = \{ L \mid L \in \mathbf{M} \ \wedge \ d(L, \mathbf{T}) \leq r \}.$$

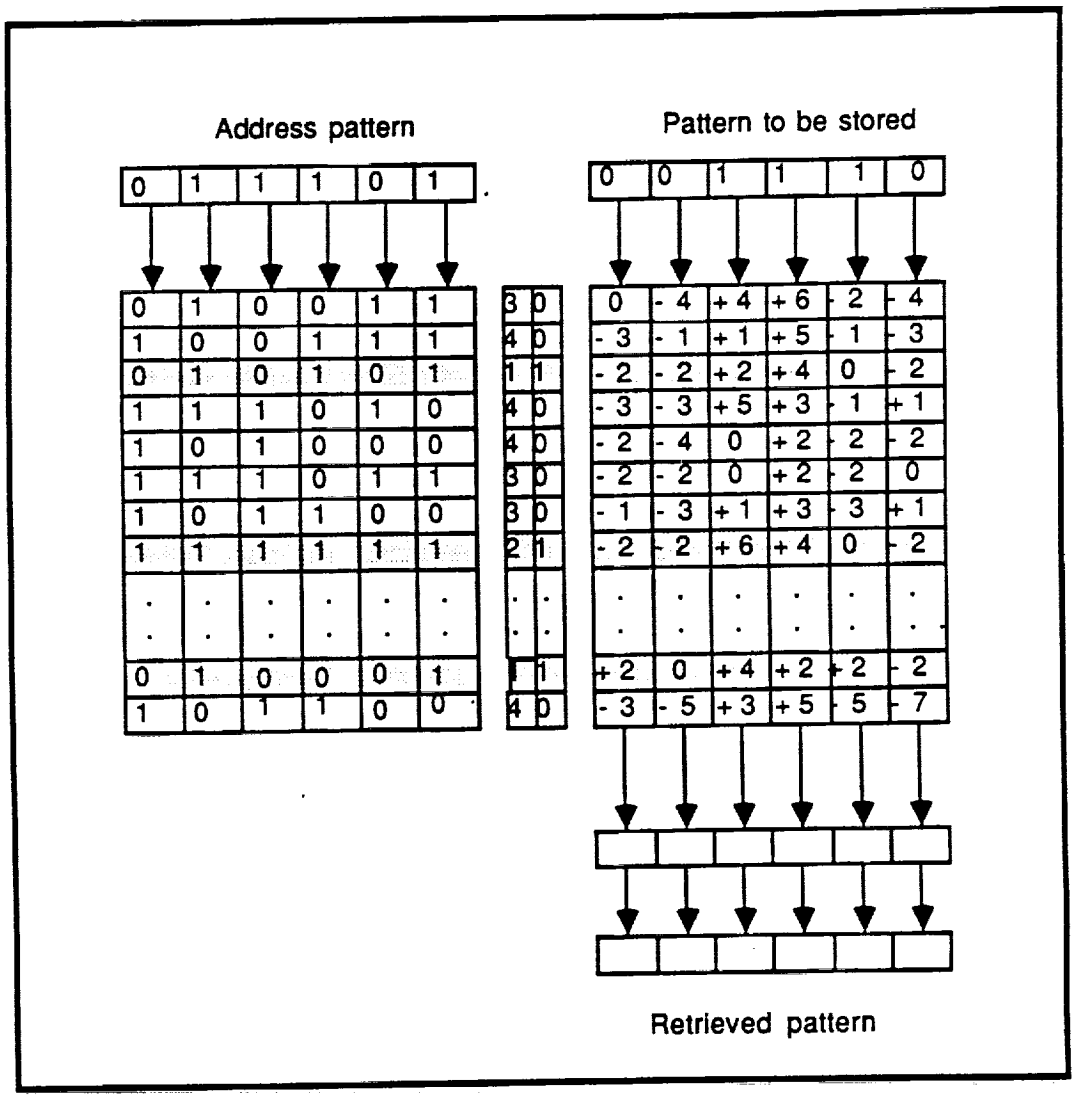

Figure 3 - Storing a Pattern. Locations similar to the address pattern are selected. These are shown in gray. The counters at the selected locations are componentwise incremented or decremented if the respective components of the pattern to be stored are 1 or 0.

Display 2 - Writing to SDM .

Autoassociative mode

Let

$\mathbf{T}_j$ be the j^{th} bit of the target pattern $\mathbf{T}$,
$\mathbf{C}_{ij}$ be the j^{th} counter of memory location $\mathbf{L}_i$.
Then writing the pattern $\mathbf{T}$ implies that

$$\forall_{\mathbf{L}_i \in \mathbf{S(T)}}$$

$$\mathbf{C}_{ij} := \mathbf{C}_{ij} + 1 \text{ if } \mathbf{T}_j = 1$$

$$\mathbf{C}_{ij} := \mathbf{C}_{ij} - 1 \text{ if } \mathbf{T}_j = 0$$

$$(j = 1, .. , \mathbf{n}).$$

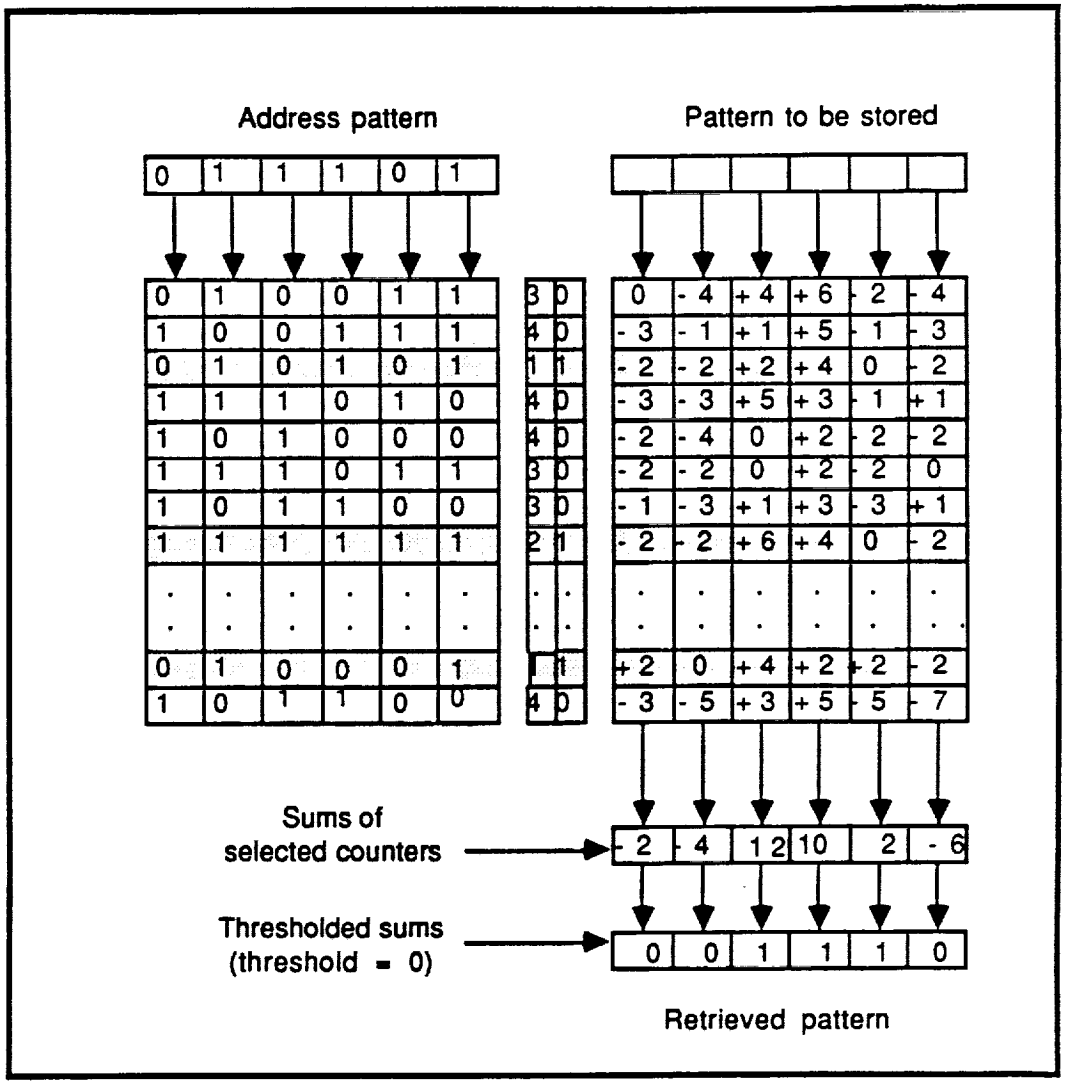

Figure 4 - Retrieving a Pattern . First, select locations similar to the target address. These are shown in gray tone. Then pool the counters at the selected locations and threshold these pooled sums to retrieve the pattern. In the example above as well as in display 3, the value of the threshold is zero.

Display 3 - Reading from SDM.

Let

T be the probe pattern,

S(T) be the set of locations selected with probe **T**,

N(T) be the number of locations selected with probe **T**.

Then reading with probe **T** implies that

$$\forall_{L_i \in S(T)}$$

$$\mathbf{Sum}_j = \sum_{i=1}^{N(T)} \mathbf{C}_{ij}$$

$$j = 1, .., n$$

$$\mathbf{Output}_j = 1 \quad \text{if } \mathbf{Sum}_j > 0,$$

$$\mathbf{Output}_j = 0 \quad \text{otherwise.}$$

FOUR

Learning to Read Aloud

4.1 Introduction

In this chapter, we describe the simulations performed in an attempt to solve the text-to-phoneme mapping problem using SDM as the network model. In what follows SDM is treated as a multi-layer network. A modification of the generalized delta rule is used to train SDM to perform the desired mapping.

Work described in this report is empirical in nature. The main results obtained in these experiments include:

1. A demonstration that an error-correcting iterative training scheme can teach SDM the desired mapping. This demonstration is based upon simulation results. The learning algorithm is described in detail in the later parts of this section. While the results are empirical in nature the learning algorithm is based on the delta rule. The delta rule is modified to account for the differences between SDM and the multi-layer model used in NETtalk.
2. A scheme to handle correlated data sets. Simulation results show that the scheme gives good results. We believe that this scheme can provide distributed representation of the mapping rules as a function of similarity.

3. A demonstration that the performance can be further improved by using a two-stage model. This is shown through simulation results.

4.2 Details of the learning mechanism

4.2.1 Thresholds

SDM is similar to many matrix models. In simple matrix models of associative memory , one can recall the stored vectors accurately if they are orthogonal . Under some other conditions the vectors can still be retrieved if they are not orthogonal as long as they are linearly independent. For correlated vectors, retrieval is still possible by adjusting the thresholds (Stone, 1986).

As more and more patterns are stored in SDM, the effective radius from which a pattern can be retrieved decreases. This occurs as the system starts moving from a low noise state to a state with high level of noise. (Here, noise refers to the interference in a a stored signal from one pattern due to storage of other vectors). One approach to solving this problem is to estimate the noise and adjust the thresholds accordingly. If the input and target output patterns are randomly chosen, the noise is distributed with mean zero. When the input and output patterns are not random, the associations can be retrieved better by adjusting the bias to that of the mean of the counters (see display 4). This simple scheme is equivalent to having a dummy location which is always selected during storing and retrieval and its weight is adjusted by the number of other counters that are selected in the select operation. This is analogous to the dummy unit that is always on as used in NETtalk .

This still does not correct for the fact that input addresses are not randomly chosen. A scheme to take account of correlated address patterns is discussed later.

Another way to estimate the correct thresholds is to pose it as a multi-dimensional search problem with retrieval as the objective function to be maximized. It can then be solved by methods such as simulated annealing or stochastic iterative genetic hillclimbing (Ackley, 1987). For a discussion of various search methods in a multidimensional space and their relative merits, see Ackley (1987).

4.2.2 Learning Mechanism

The learning mechanism consists of exposing the pattern associator with a pattern to be associated and minimizing the error between the actual output pattern and the desired output pattern. This is accomplished by feeding back a small portion of the error, in an error-correcting manner, to the counters that have taken part in producing the error. This corresponds to a gradient-descent search on the error surface such that traversal on the error surface is in the direction of lower error. Many training procedures in artificial neural systems take this approach (Rumelhart and McClelland , 1986).

4.2.3 Nonlinear Activation Function

At first, a scheme similar to a simple "perceptron learning procedure " (Rosenblatt, 1961), was used to adjust the counters. This learning was found to

be unstable because of the discontinuity at the threshold. One way to overcome this problem is to use a sigmoidal transfer function that makes it possible to obtain a desired change in output by choosing the proper input. The important characteristic of a sigmoid function is that it is a differentiable, nondecreasing function of its input and it approximates the threshold logic unit (a threshold logic unit is an infinite-gain sigmoid).

The use of a sigmoid can be further supported by the fact that it can model the input output characteristics of biological neurons to a certain extent. Some characteristics of a sigmoid function that appear to be similar to the biological neurons are:

1. Noise suppression.
2. Limited dynamic range.
3. Nonlinear, nondecreasing response.

With the sigmoidal transfer function the activation is computed as shown in Display 5. The output is then computed as shown in Display 6. The actual feedback amount is computed by the learning rule as shown in display 7. This is just the delta rule as applied to Sparse Distributed Memory. In keeping with the basic characteristic of SDM, learning is restricted to changes in the counters. The scheme of selecting similar addresses to store similar entities is preserved. The feedback amount as shown in display 7 is the quantity δ for the output units multiplied by the learning rate λ. In the generalized delta rule this would be multiplied by the activation of units from the preceding layer. In our case the activation of these units is 1 and hence the feedback amount does not show this

multiplicand. Appendix A shows how Sparse Distributed Memory is a special case of three-layer networks.

4.3 Details of the Experiment

Carterette and Jones (1974) prepared a database of transcriptions of informal speech for four age groups. The youngest of these were first grade children. Informal speech drawn from first grade transcriptions was chosen as the data set. The training set consisted of 1028 words. The test set consisted of 915 words. The symbols in the alphabet of the text set were the 26 letters of English. These were augmented with two symbols: full stop and word boundary. The symbols in the alphabet of the phoneme set were the 45 phonemes (only those which occur in the training and test sets) augmented with a symbol for the sentence boundary, a symbol for the word boundary and a symbol for unpronounced letters. Thus, the alphabet of the orthographic language had 28 symbols and the alphabet of the phonemic language had 48 symbols. The problem to be solved is to map a string of symbols from one language (orthographic language) to a symbol in another language (phonemic language). The grammars of the two languages are closely related. For an interesting example where the two languages differ, see R. B. Allen (1987). He describes an experiment in which a mapping from English to Spanish is taught to a network using a supervised learning procedure (i.e., the network learns to translate English text to corresponding Spanish text).

The orthographic stream was properly aligned with the phonemic stream. Figure 5 shows examples of segments of aligned orthographic and

phonemic streams. Appendix B describes the symbols used in the phonemic stream.

In the simulations being discussed here the window consisted of 7 characters as in the NETtalk study. The 7-character window was coded by giving different weights to different character positions The weights for the characters in the window were - 1, 2, 4, 8, 4, 2, 1 (Figure 6). These weights, which were subjectively chosen, represent the relative importance of input characters in determining the output. After weighting, the characters were coded with a compact binary code (i.e. five bits were used to code each character). Similarly, the phonemes were coded with a 10-bit Hamming representation of a six-bit compact binary representation. (One-bit error detection and one-bit correction code).

4.4 Training the Network

Let **Tr** = {<t1, p1>, <t2, p2>, ..., <tn, pn>} be the set of pairs in the training set, where < ti, pi> represents the i^{th} pair of text window ti and the corresponding phoneme pi. The network was trained using set **Tr** as follows:

Step 1: Store the training set by storing p1 at t1, p2 at t2, ..., pn at tn.

Step 2: Compute thresholds using the equation shown in display 4.

Step 3: For each pair <ti, pi> in **Tr**,

Read at ti. Let the output be oi.

(Use Equations in display 5, and display 6 to compute the output).

Compute the error for all positions in the retrieved vector oi as compared to the desired vector pi.

(this is the componentwise difference between each vector).

Compute the feedback amount.

(Use the learning rule in display 7).

Accumulate the feedback for each of the selected counter separately.

Step 4: Feedback the accumulated error to all the counters.

Repeat steps 2 to 4 until number of correctly retrieved vectors does not increase with further training.

In the actual training that was carried out a vector pi was considered to have been correctly retrieved if it matched in at least 9 of the 10 positions with the output vector oi. Use of Hamming code in coding pi allows an error in any one position. This can be deterministically detected and corrected.

Step 1, in the procedure described above is not essential in training the network. One could as well proceed without it. However by including the first step in the training procedure the percent correctly retrieved start at a higher initial value.

Figure 7 shows the schematic of the network in training. Initially the training set was stored in one pass. Then in each successive pass, response to the vectors in the training set was noted. The learning rule was then used to feed back a small portion of any noted error.

4.5 Simulation Results

We now describe the simulation results in the following sections. The next section describes the results obtained with a network which was constructed with randomly chosen hard locations (i.e., addresses). Later sections describe improvements aimed at obtaining better performance.

4.5.1 Results with Randomly Chosen Locations

Figure 8 shows the performance of the training scheme used, when the addresses of the locations are randomly chosen. The peak performance was about 74% correct on the training set after 65 passes through the training set. The training was still increasing the percent correctly retrieved at the end of the experiment, although the marginal gain was not enough to justify further training. The memory contained 800 addresses in this simulation.

4.5.2 Countering the Problems of Correlated Data

Usually, real world data are highly correlated. If one uses SDM with randomly generated addresses, its performance deteriorates as the distribution of data points is not random. One way to solve this problem is to select addresses from the distribution of the problem domain. Keeler (1987) suggests such an approach. He considers SDM from Kanerva's original formulation to consider the case of correlated input patterns. He shows that if the input set of correlated patterns (i.e. the addresses) and the distribution of Hamming distances between any two randomly chosen patterns from this set is known a priori, then choosing the addresses from the distribution of input patterns, and

using the proper radius of similarity, SDM will show the same ability to retrieve a given associated vector as output, as in the original formulation. He suggests that if this distribution is not known, the above procedure can still be followed, if the distribution could be learned by some means. Rather than finding techniques to learn this distribution in some way, we feel that it would be better to draw the addresses from the data points themselves. Keeler's scheme was introduced in the original Kanerva formulation which did not use any iterative supervised learning. We believe, however, that it can be extended to include the case where the memory is trained using the supervised learning. We now assume that the training set is sufficiently representative of the population of input vectors in the problem domain (See the discussion in chapter 6 of learnable tasks and related training set size). Thus, we propose that the addresses be drawn from the training set.

Figure 9 shows the performance as a function of training when the hard addresses are drawn from the training set. In this example, 800 training vectors were randomly chosen without replacement from the training set as addresses of locations. In these simulations, the peak performance was about 81% correct after 300 passes through the training set.

Continuing our discussion further, let us now consider some interesting improvement. Assume that we have M data points (i.e., training vectors). If we chose a memory of M cells by drawing these addresses from the M data points without repetition, we will have a memory with addresses identical to the data points. If they are all distinct, then with a zero radius-of-select, this corresponds to the model of Baum, Moody, and Wilczek (1986), where each

address is a grandmother-cell representation of itself (Baum et al. call this a unary representation). Thus, we will get 100% yield on the training set retrieval. This case is of little interest, since it is equivalent to memorizing the training set, and the model will not have any ability to generalize. Also, there will be no damage resistance.

Interesting behavior can be observed as we start increasing the radius-of-write. As the radius-of-write starts increasing, the signal-to-noise ratio will begin to decrease. For a small radius the retrieval on the training set would still be fairly high, and the system's damage resistance will start increasing. The system's ability to generalize will also start increasing.

A more intriguing possibility involves finding a functional relationship between the addresses and the data. This may be better than the connectionist approach of analyzing the weights on the hidden units in a 3-layer feed-forward model. Since a given address from the training set will correspond to a hard address in the memory, statistical analysis of counters in the immediate neighborhood may reveal a functional relationship between the addresses and the data. More specifically, since address *A* in the training set corresponds to address *A* of a hard location, one can just take addresses in the training set that are similar to this and perform statistical analysis on their respective data counters, thereby obtaining a more concise representation of letter-to-sound rules. This method can provide these distributed letter-to-sound rules as a function of similarity. This, we believe, is the main advantage of the scheme Generalization can be improved further by choosing a majority of addresses from the training set and augmenting them with many addresses from possible

test sets (i.e., randomly chosen character windows selected from different text passages). This would show higher generalization as long as the radius is non zero.

Figure 10 shows the performance of the networks when hard locations are chosen corresponding to each training vector and these are further augmented as previously suggested with vectors from possible test sets.

The performance on the test set is much higher in figure 10 (i.e. the case where the addresses of hard locations correspond to the training set and these are further augmented with addresses from randomly drawn character windows). Peak performance in figure 9 on the test set is about 65% whereas the peak performance on the test set in figure 10 is about 71%.

4.5.3 Improving the Performance Using a Two-stage Model

It is possible to improve the performance further by using the following scheme. Let **Tr** = {<t1, p1>, <t2, p2>, ..., <tn, pn>} be the set of pairs in the training set, where < ti, pi> represents the ith pair of text window t1 and the corresponding phoneme p1. First, train SDM to its best possible mapping capability As described in the section - 'Training the Network'. Let the best output of the memory be {f1, f2, ..., fn} Now create a new memory and train it with the training set, **Tr2** = {<f1, p1>, <f2, p2>, ..., <fn, pn>}.

Thus, the output of the first stage is used as input to the second stage, such that the desired output (target) is stored at the output of the first stage. This second stage is then trained with respect to target output in the same way as the first stage. This leads to a dramatic improvement in performance.

Figure 11 shows results of simulations when first stage SDM was trained as previously shown in figure 9 (i.e., The addresses of locations were drawn from the training set). The peak performance now reached about 87% as opposed to 81% obtained using only one stage.

Figure 12 shows the improvement in performance when the first stage hard locations correspond to the training set (Not just a small sample of the training set). The peak performance improved from about 89% to 93% in 120 further passes through the training set. The gain may seem insignificant but this is because the first stage performance was quite high. The retrieval starts at a lower value than the maximum for the first stage; but this very rapidly rises to above the highest in the first stage.

At present, we cannot offer a clear explanation of why this scheme shows an improvement in performance over a single-stage model. We can only speculate about it.

The basic learning scheme that is chosen in a single-stage model is based on SDM's similarity based storage and retrieval mechanism. Hamming distance is used as the metric of similarity. For some problems this criterion is clearly inadequate. Consider the 'parity problem ' or the 'clumps problem '. The learning mechanism as described in the single-stage model is incapable of solving problems of this kind . In the first problem above, we are interested in learning to find the 'parity' of binary vectors. In the second problem we are interested in detecting the number of clumps of "1"s in a binary vector.

We tested our single-stage learning mechanism on the clumps problem and the parity problem (which is just the generalized XOR problem). As

expected, learning mechanism was unable to solve the clumps problem. Training improved performance over the training set however the performance on test set was hopeless (about same as random guessing). Thus, training could only help SDM memorize the training set but it was unable to generalize. The performance on the parity problem was very good but that is because of the quirk of the select mechanism. SDM select mechanism is such that SDM behaves as though it is hard wired to solve this problem. Appendix C explains this behavior.

The improved performance in a two-stage model may be explained as follows. The performance in the first stage can be thought of as the maximum obtainable performance from the first order statistics. After the first stage has separated the outputs in various categories, the second stage can be thought of as utilizing this knowledge in further separating the outputs. Multi-layer networks with more layers of hidden units are able to learn higher order predicates . NETtalk experiment showed that with zero hidden units the performance was poorest as it corresponded to learning from first order statistics. SDM is a special case of multi-layer feed-forward networks (see Appendix A). With stacking of SDM stages this becomes a network with two layers of hidden units. It must, however, be pointed out that the training in two stages does not proceed simultaneously. The first stage has been trained completely before the creation of the second stage.

In a sense, the second stage can be thought of as an interpreter of what the first stage has found. However, it is not limited to being an interpreter otherwise a simple table lookup would suffice as an interpreter. It is an adaptive

interpreter where the learning is stored in a distributed fashion. Consequently, it shows the robustness of a distributed representation .

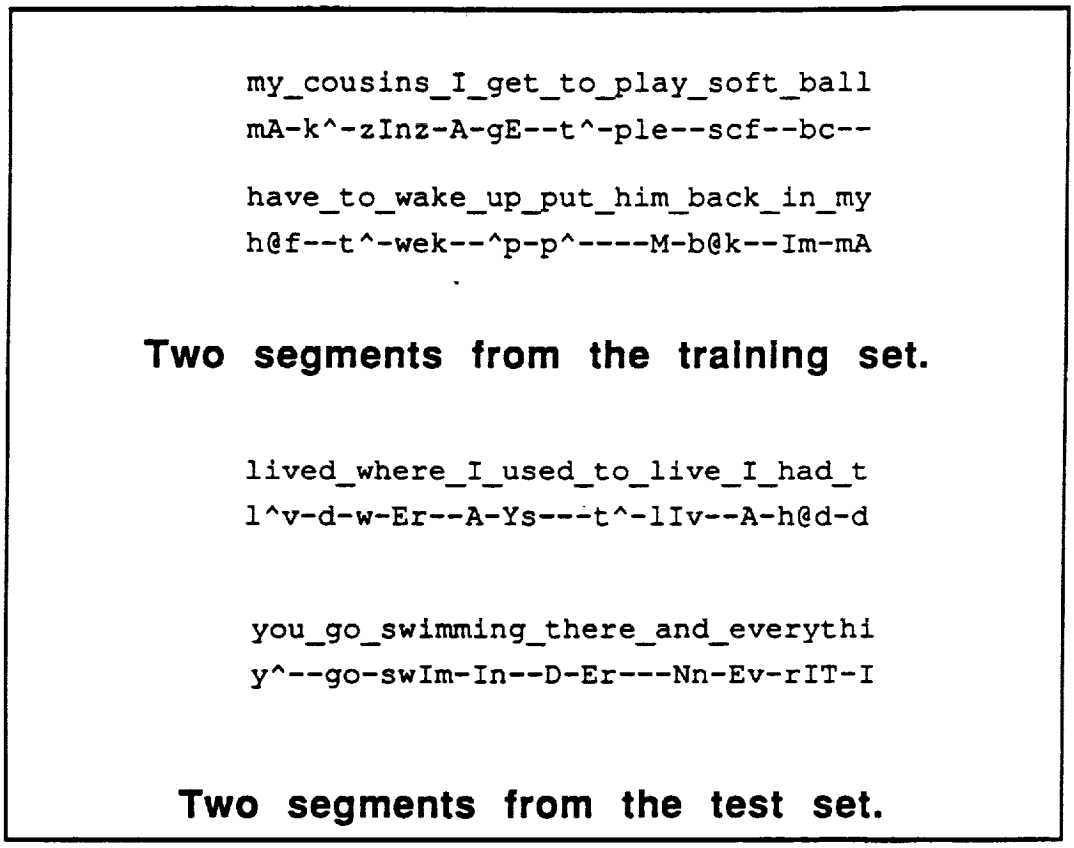

Figure 5 - Aligned Orthographic and Phonemic Streams.

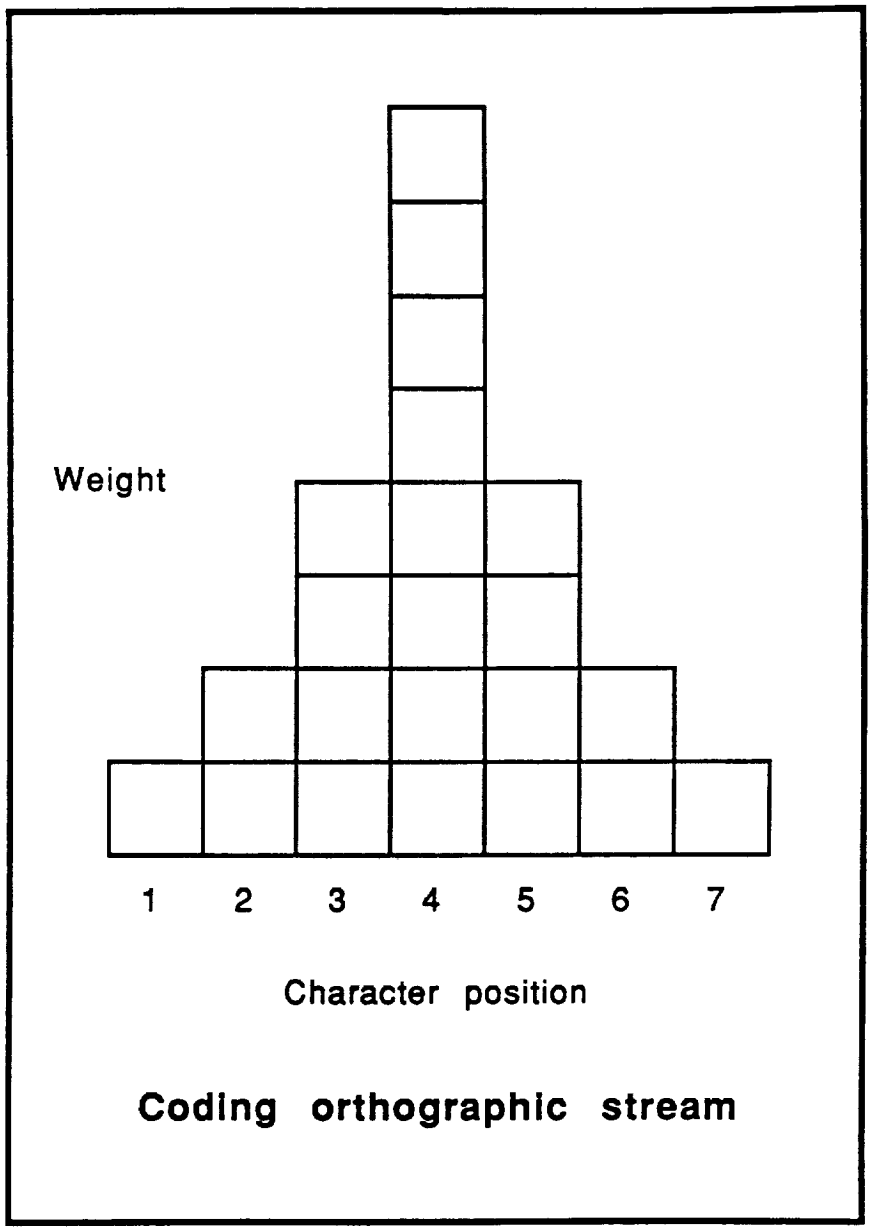

Figure 6 - Coding Orthographic Stream . Central character in the window is the character being mapped in the context of other six characters. It is given the highest weight with the weights reducing as you go away from the center. Each character is then coded using a five-bit code.

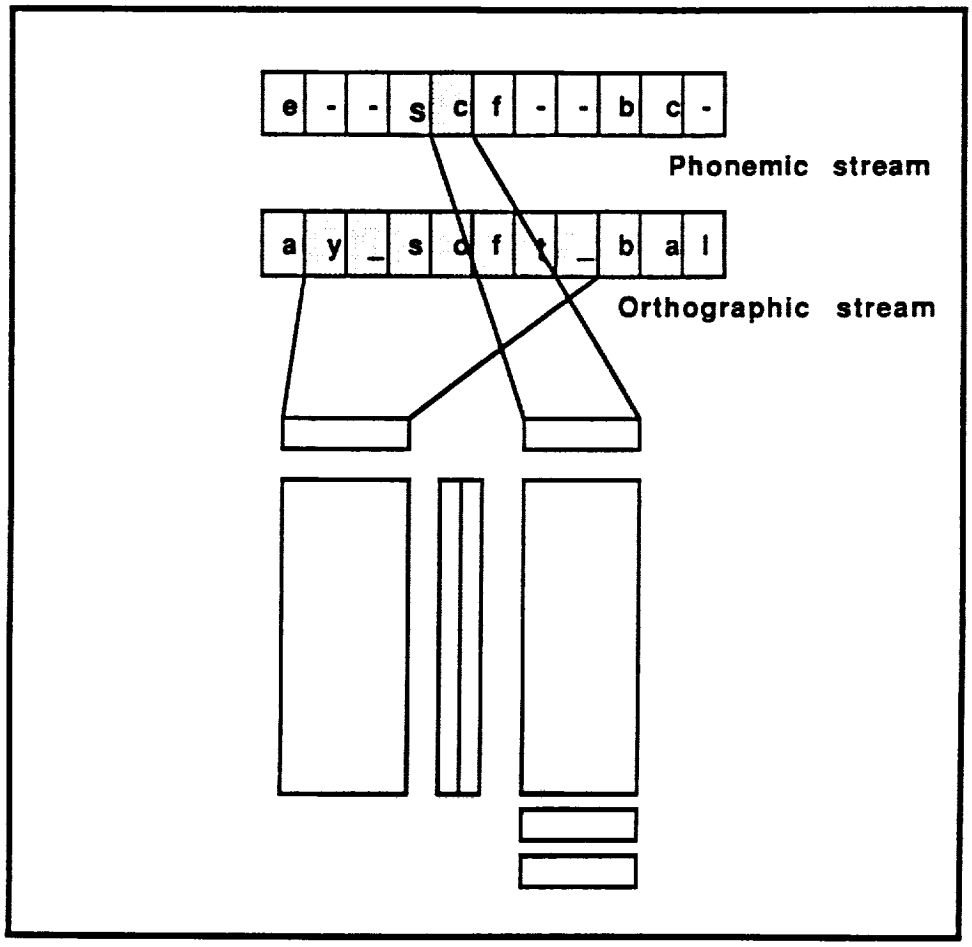

Figure 7 - A Snapshot of Training. Phoneme /c/ is the target phoneme for the character window: "y_soft_". Character "o" in the context of "y_s" and "ft_" is mapped to target phoneme /c/.

Display 4 - Computation of Thresholds.

Let,

$\mathbf{m}$ = The number of locations in the memory.

$\mathbf{n_a}$ = The number of bits in the address.

$\mathbf{n_d}$ = The number of bits in the data.

θ_i = Bias for computing i^{th} bit of activation.

$i = 1, \ldots, \mathbf{n_d}$

Then,

$$\theta_i = \sum_{j=1}^{\mathbf{m}} \frac{\text{Counter}_{ji}}{\mathbf{m}}$$

Display 5 - Computation of Activation .

Let,

T be the reference address,

S(T), be the set of selected locations,

N(T), be the number of locations selected,

$\overline{\mathbf{C}}_i$, be the mean counter value,

over the selected locations.

i.e.,

$$\overline{\mathbf{C}}_i = \frac{\sum_{\mathbf{S(T)}} \text{Counter}_i}{\mathbf{N(T)}}$$

Then the i^{th} component of the activation vector, $\mathbf{a}_i$, is given by

$$\mathbf{a}_i = \frac{1}{1 + e^{-(\overline{\mathbf{C}}_i - \theta_i)}}$$

Display 6 - Computation of Output.

Let

$\mathbf{a}_j$ = j^{th} component of the activation vector.

Then

$\mathbf{Output}_j$ = 1 if $\mathbf{a}_j \geq 0.5$, 0 otherwise.

Display 7 - Learning Rule . The learning rule is same as the generalized delta rule for the output units. As the output of the selected locations is 1, it is not shown explicitly. Output of units not selected is zero so they do not take part in learning. Thus, only the counters of selected units are adaptively changed.

Let

t be the target vector,

a be the activation vector,

e be the error in the output

Then the componentwise error in the activation vector is given by

$$\mathbf{e}_j = \mathbf{t}_j - \mathbf{a}_j$$

The learning rule reduces this error by feeding back a small fraction of this error to the counters that contribute to producing this error.

Let λ be the coefficient of learning $(0 < \lambda < 1)$

Then the error reducing signal for the j^{th} bit, $\mathbf{b}_j$, is given by

$$\mathbf{b}_j = -\lambda \mathbf{a}_j (1 - \mathbf{a}_j) \mathbf{e}_j .$$

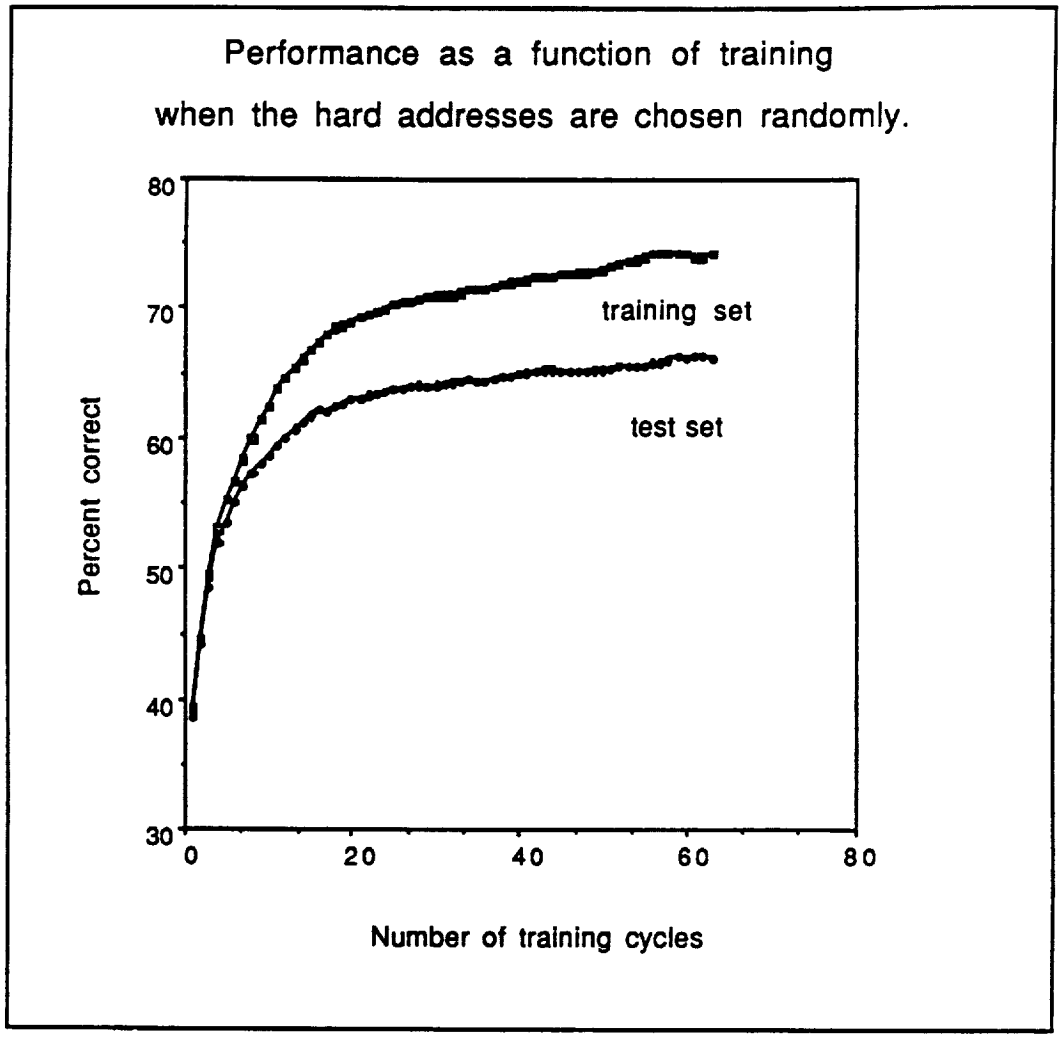

Figure 8 - Network Performance With Randomly Chosen Addresses. The memory contained 800 hard locations. Addresses of these hard locations were chosen randomly.

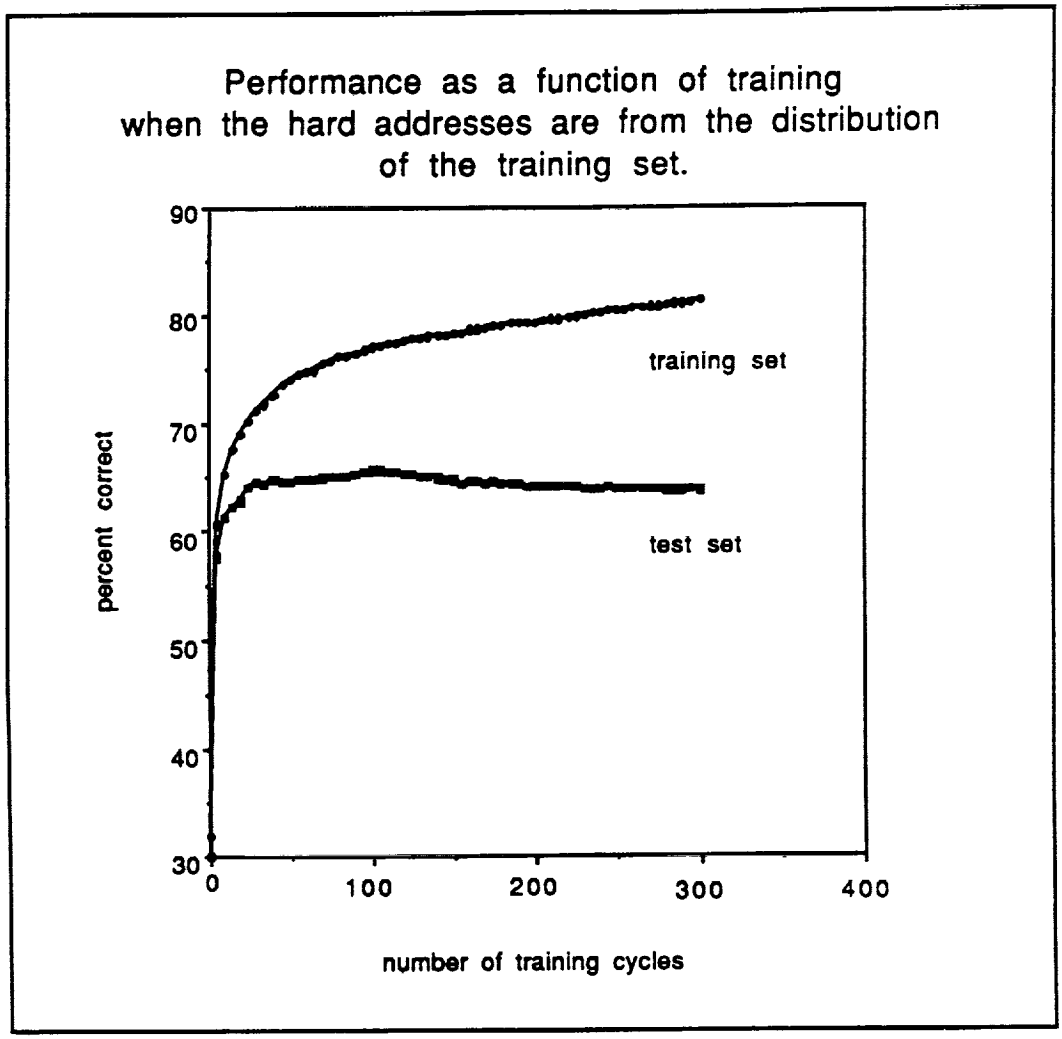

Figure 9 - Network Performance When Addresses Are Chosen From the Training Set. In these simulations the addresses of hard locations were chosen from the training set without repetition. The memory contained 800 hard locations.

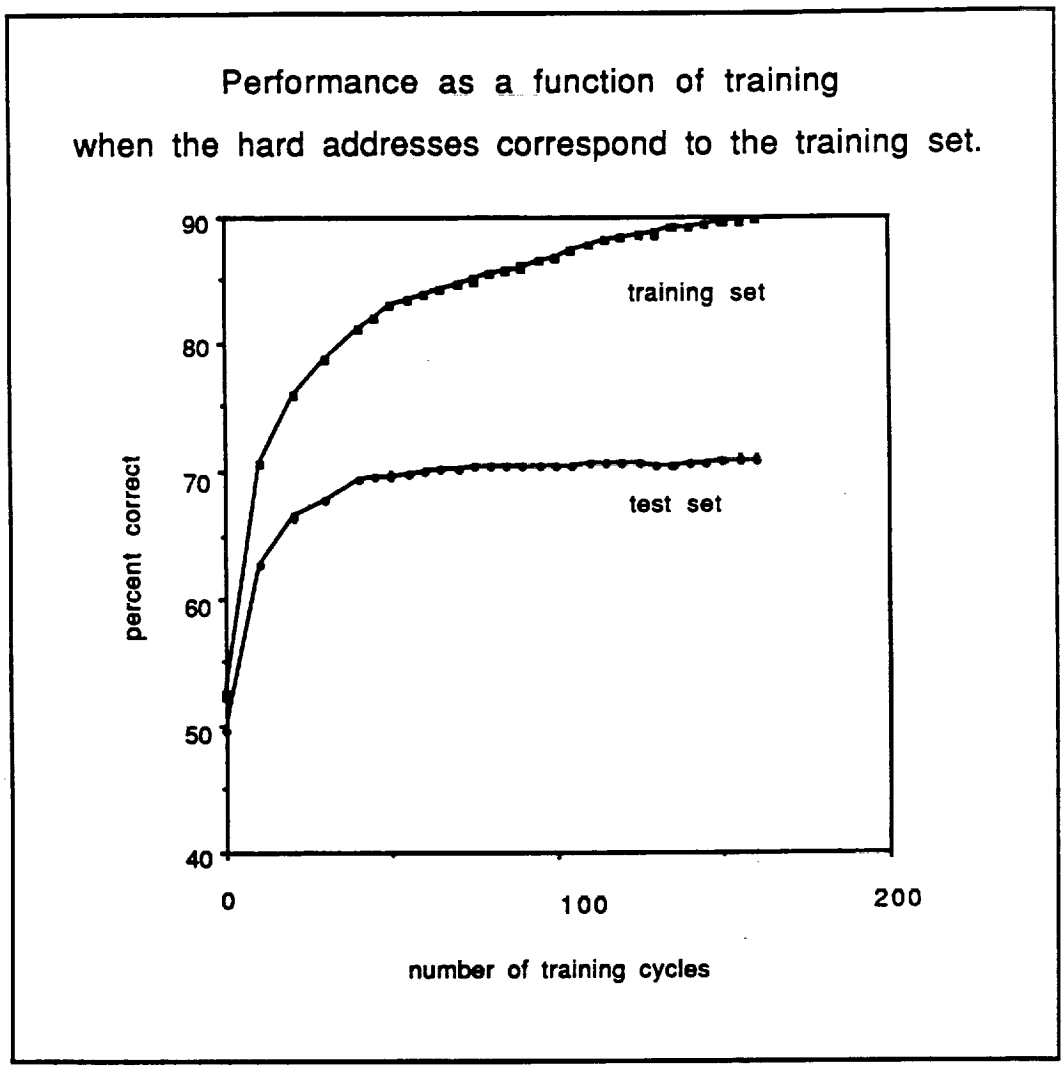

Figure 10 - Network Performance When Addresses Correspond to the Training Set. In these simulations the memory contained a hard location corresponding to each unique vector in the training set. These were further augmented with randomly generated character windows as explained in the text.

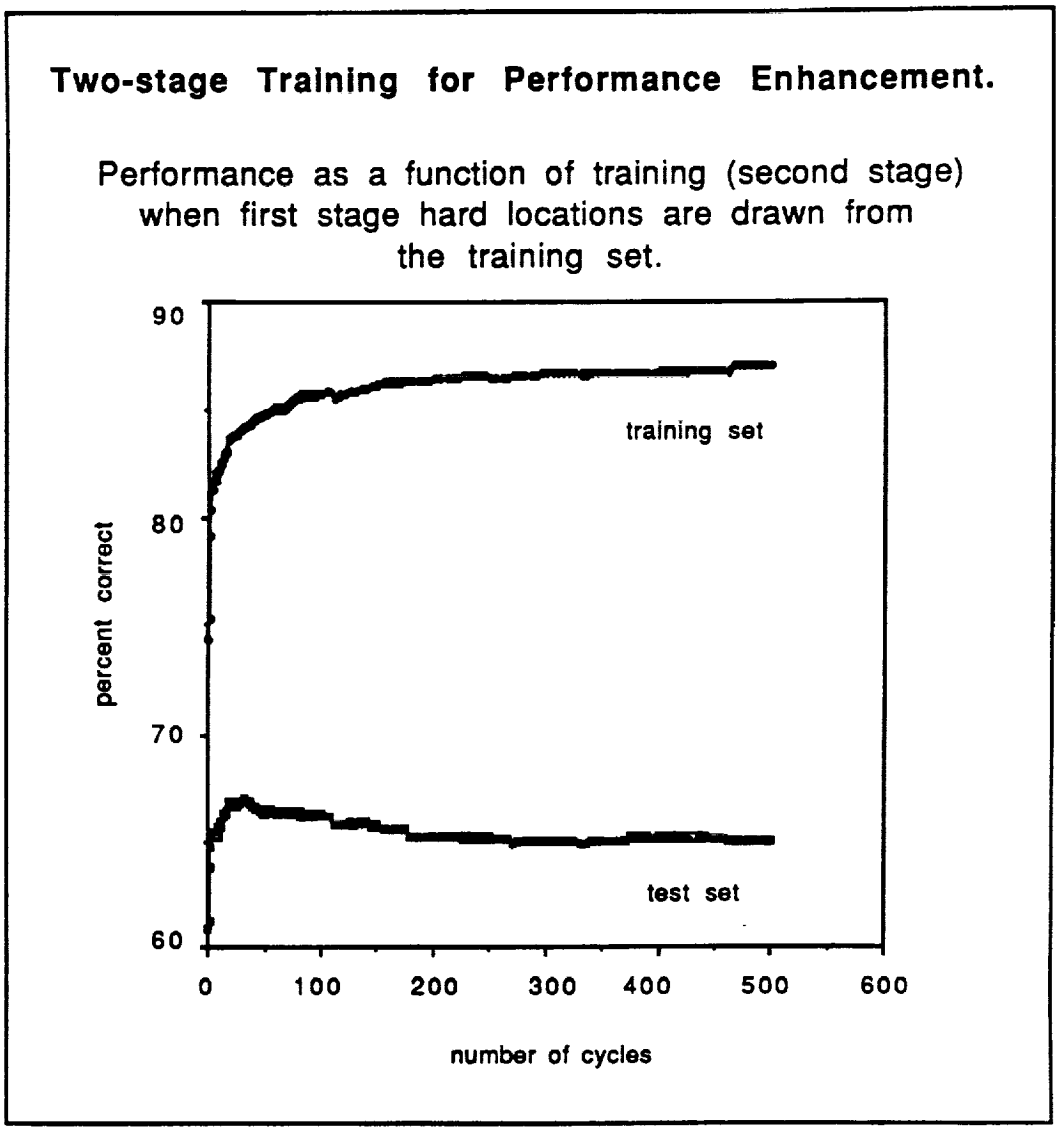

Figure 11 - Two- stage Training. Figure shows the performance of the second stage as a function of training. Peak output of the network (shown in figure 9) was used to form a new training set as explained in section 4.5.3.

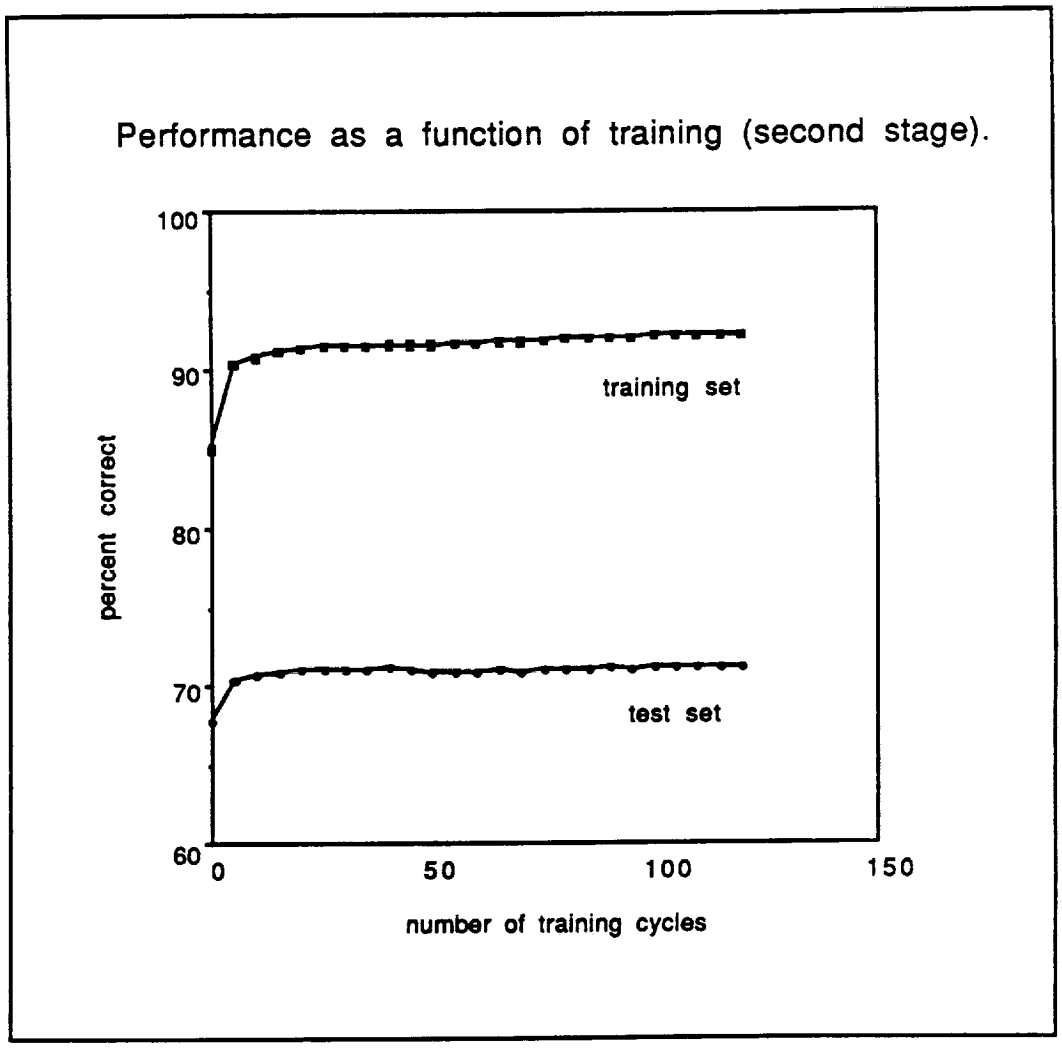

Figure 12 - Two-stage Training With Full Training Set. Figure shows the second-stage performance of the network. In these simulations, the training set was formed by taking the peak output of the network from figure 10.

FIVE

Design Decisions

5.1 Introduction

In this section we describe various design decisions and contrast these with the ones made in NETtalk in particular and some other systems in general. Differences in NETtalk and the simulations performed using SDM include:

1. Network architecture.
2. Learning mechanism.
3. Coding.
4. Preprocessing and post processing.
5. Measuring the performance

5.1.1 Network Architecture

The architecture of SDM is in many respects different from the multi-layer network used in the NETtalk study. (For a complete mapping from SDM to the the network used in NETtalk study see Appendix A). Major differences include:

1. In SDM connections between the first and the second layer are fixed but are modifiable between the second and the third layer. NETtalk used a network where all the connections between the units were modifiable.

2. In the modified SDM used in the present study only the output units have real valued activations. All the computing units in NETtalk had real valued activation.

5.1. 2 Learning Mechanism

The learning mechanism that was used in the present study differs from the one used in NETtalk in many respects.

1. In the present study, the learning takes place only between the second and the third layer, while in NETtalk all the connections are plastic. There are some other studies (see for example, Huang and Lippmann, 1987) which report experiments in multi-layer networks with a few fixed sets of connections and remaining modifiable connections.
2. An a-priori choice is made in choosing the connections in the first layer. When these correspond to the distribution of the training set the performance of the network improves. When these correspond to the examples in the training set the performance improves further. It can then, also provide a distributed representation of mapping rules. NETtalk has no mechanism to arbitrarily fix some connections. In NETtalk the network learns these connections over many training cycles.
3. NETtalk was restricted to using extremely small learning rates and using momentum terms in the learning rule in order to have a stable learning curve. It follows from the scheme of exposing

one pattern at a time and then making a change in the strengths of the connections. The present study does this training in parallel. (i.e., changes in the connection strengths are made only after a complete pass through the training set). Hinton calls this 'batch' mode of training. This requires a global memory to store the changes required until a pass is completed through the training set. Thus, this fails as a neural model of learning.

4. In the present study, a two-stage model is shown to improve the performance of the network. NETtalk scheme did not have a similar setup.

5.1.3 Coding

In the present study, in coding the input a weighted input scheme was chosen (see Figure 6). The weights were arbitrarily chosen. They were meant to reflect the fact that the importance of each character in conveying the information required, for finding the correct mapping, decreases as the distance of the character increases, from the center of the window of characters. This is reflected in the work of Lucassen and Mercer (1984). NETtalk did not use such a weighted input scheme. All positions in the input stream were considered to have the same influence in determining the output.

In the present study, the characters were first coded with a compact binary code using 5-bits to code each symbol in the orthographic stream. Similarly, in coding the output, a compact binary code was used. Each symbol

in the phonemic stream was initially coded with a 6-bit code. These were later processed through an error-correcting scheme.

On the other hand, NETtalk used articulatory features to code the output units. In this scheme, units are either on or off, indicating presence or absence of a particular feature. One unit is used for complete information about a particular feature. For coding the input NETtalk used local representation. In this scheme one out of 29 units (26 letters and 3 punctuation marks) is switched on to indicate the particular input character. In the distributed representation the information is coded using many units. If each unit participates in the representation of many entities, it is said to be *coarsely tuned* (Rosenfeld and Touretzky, 1987) and the pattern is called *coarse-coded pattern*. Thus, any particular unit cannot give complete information about the presence or absence of any feature.

In the particular scheme that has been adopted in the present work (viz., using a compact binary representation), units that may be on do not bear any particular resemblance to the meaning of the patterns they encode. Thus, they are patterns for the symbols they encode and the scheme is similar to what Rosenfeld and Touretzky refer to as *coarse-coded symbol memories*. For a study of the coarse-coded symbol memories, their strengths and weaknesses, see Rosenfeld and Touretzky (1987).

The coding method employed makes the coding more general and hence brings it closer to a situation in which an expertise in the domain is not necessary. This is not to say that there is no role for the expert. The role of the expert is limited to making sure that the set of examples is internally consistent

and that the errors in the examples are minimized. By trying to reduce the role of expert as much as possible, the system has been taken more and more in the general direction such that it should be possible to transfer the whole learning apparatus to a problem in a different domain with little, if any, change. For an example of completely random coding , where randomly chosen vectors acts as symbols for the entities they encode, see Elman (1988). In the present study, the coding is as good as random with the size determined by the number of symbols in the phonemic language.

5.1.4 Preprocessing and Postprocessing

In the present study the output was preprocessed and postprocessed using Hamming error-correction coding. NETtalk did not use any such scheme. The phonemes were initially coded using six-bit code. These were further recoded to ten-bit Hamming representation of these six-bit codes. Hamming codes are one of many different codes that have evolved out of a need for reliable information transmission. Different coding techniques use built-in redundancies to detect and in some cases, as in the present case, deterministically correct an allowable error in transmission. Redundancies in the code-words have been used extensively in distributed representations, however, coding theory uses redundancies in a systematic way.

With a compact six bit code it is impossible to detect (let alone correct) an error in the output as the legal code-words are separated by a Hamming distance of 1. If the code-words are n-bit long, Hamming transformation separates the code-words by adding k "parity" bits such that the code-words

are separated by a Hamming distance of 3. This allows for the detection and correction of any one-bit error in retrieval. For an excellent introduction to ideas behind error-correcting codes, information theory and cybernetics, see Jagjit Singh (1966).

At lower stages of yield, separating the legal code-words as described above, improves the performance. The gain drops as training reduces error in retrieval. Even at the peak retrieval this scheme improves the retrieval.

This result is really not surprising as separating the code-words will always result in a higher yield.

5.1.5 Measuring the Performance

In the present study the performance was measured in the following way. If the output of at least nine bits matched the desired output then the vector was scored as having been correctly retrieved. For any particular bit the output was considered to be 1 if it was greater than 0.5 and 0 if leas than 0.5 as shown in the output rule. A stricter criterion would be to consider output as 1 if it was greater than 0.9 as done in the NETtalk study and 0 if less than 0.1. This stricter criterion was used in some experiments and the results followed those with the not so strict criterion but required many more training cycles.

NETtalk scheme judged performance according to a perfect-match and a best-guess criterion. The output is 1 if the activation value is greater than or equal to 0.9 and 0 if it is less than or equal to 0.1. If the activation value was between 0.1 and 0.9 then for the purpose of finding perfect match the output was considered to be undefined (i.e., it required further training to find if it would

stabilize to the proper extreme values). If all the bits of an output vector matched the desired output, it was scored as a perfect match. Best-guess criterion classified the output vector by mapping it to the nearest legal code making the smallest angle with the output vector.

This procedure is somewhat similar to the idea of error-correcting codes. However, it can give misleading results. (Hamming error correction scheme separates the legal code so that any one-bit error can be deterministically detected and corrected by pushing the output vector to the nearest correct legal code). Dahl (1987) shows that the idea of using the nearest-match criterion in measuring the networks performance can give misleading results. While the approach may intuitively appear to be similar to minimal error, a class of examples has been found for which it is not the case. In particular, the nearest-match criterion is satisfied but the error is not minimized.

SIX

Discussion

6.1 Introduction

In this section we discuss the simulations performed with SDM as the network model and in the later part we discuss some issues which are common to different connectionist models.

6.2 General Discussion

What follows is a general discussion of the simulations performed. This discussion is limited to the present study without a particular reference to NETtalk in every instance, since in many cases the discussion is not applicable to NETtalk and in other cases there is no information available regarding some of these points from NETtalk study.

6.2.1 Some Comments About the Learning Mechanism

The following points need to be noted about the learning mechanism.

1. The input and output vectors are in a discrete space.
2. The learning error correction scheme is in a continuous space.
3. The output plots show number of vectors correctly (with error correcting codes) retrieved.

4. The criterion could have been number of bits correctly retrieved but the error-correcting code corrects errors in vectors whereas the learning error correction scheme corrects errors in bits.

6.2.2 Character-Window Sizes

From the studies performed by Lucassen and Mercer (1984) it appears that a seven-character window may be appropriate though a smaller five-character window may be a good approximation. An et al. (1988) take a different approach and experiment with windows of different sizes to arrive at the proper text-to-phoneme mapping.

6.2.3 Damage to Counters and Its Effect on Retrieval

Distributed representations manifest a remarkable tolerance to failure of individual elements. Performance is not affected to the same degree as the damage if the damage is not extensive. To test this, some damage to the counters was introduced artificially. A certain percentage of counters were randomly chosen and set to zero. Figure 13 shows the performance of memory as a function of the percentage of damage. Figure 14 shows behavior of second stage in the presence of damage to the counters. In these simulations the peak trained setups were taken from figures 8 and 10 respectively.

6.2.4 Relearning After Damage to Counters

A network was taught using the learning scheme discussed earlier. It was exposed to some damage and again trained. This was expected to show

performance similar to simulated annealing (Kirkpatrick et al., 1983). The network regained its peak performance after training. NETtalk study reported a similar finding.

6.2.5 Inconsistencies in the data sets

The data used in the present study contained a few inconsistencies. This affected the peak performance and the number of training cycles required to attain the peak performance. Details of inconsistencies in the data in the case of NETtalk study were unavailable.

6.3 Limitations of the present study

The present study is limited by many of the assumptions and simplifications. It is an oversimplification to assume that a given size of window of the orthographic stream has enough information to find the appropriate phonemic output. The present study also ignores the effect of co-articulation. No attempt has been made to account for syntax or semantics. For this problem, Hamming distance may be an inappropriate metric of similarity.

6.4 Related Issues

In what follows, we extend the discussion of issues that are common to different connectionist models. These include:

1 Scaling of the learning algorithms with respect to different parameters.

2 Generalization.

3 Behavioral and neural plausibility.

6.4.1 Scaling

Most of the present day learning algorithms used in the connectionist models do not scale well with size of the problem. Thus, while they may show some dramatic results on toy problems, they are far from a stage when they can be used in useful practical applications.

Fogelman et al. (1987) investigate the back-propagation algorithm to study memorization and generalization on two tasks to study the scaling behavior of the network with the ratio of training-set size to the total set size.

Tesauro (1987) describes the scaling behavior of a back-propagation scheme in a three layer network. He investigates scaling behavior with respect to the size of the training set, in the context of learning the "parity problem " with 32-bit vectors. In considering problems where generalization is possible, the required number of presentations of each example should decrease as the size of the training set increases. Thus, the total training time required should increase at a less than linear rate. Sejnowski and Rosenberg (1987) showed that NETtalk learning scheme followed a power law and observed such sublinear scaling. In the present study, the scaling behavior has not been tested yet.

If the task is learnable, the learning time would remain constant after a given size of representative training set. For a learnable task, a way to reduce the required training time, in terms of number of cycles of training, is to use higher order correlations (Psaltis et al., 1988).

Tesauro, and Janssens (1988a) describe the scaling relationship with predicate order as the criterion.

6.4.2 Generalization

In a task of learning from examples, generalization may be loosely defined as the ability to respond to a novel stimulus with a correct response, with the help of the knowledge gained from a set of examples. This is inductive learning. Clearly, from a given set of examples, it may not be possible to give a unique correct response to a particular stimulus. Thus, it may be necessary to specify some additional criterion of correctness. Pavel et al (1988) view this additional criterion as posing some additional constraints. These constraints may be by way of restricting the connectivity of the network, by a choice of coding of inputs and outputs, or by constraining the learning algorithm in some way.

Let us consider some of the ways in which generalization can be aided. Consider the "clumps problem". Given a binary string the problem is to determine the number of clumps of "1" s that exist in the string. Fully connected neural networks are not suited to solving this problem without a change in the architecture. How does one, then, teach a network to solve this problem? A possible solution involves interconnections limited to adjacent units (to reflect the geometry of the problem).

Due to the particular connectivity pattern, each one of the units in the second layer can detect if its two inputs are the same or different, which is essentially the solution to detecting the clumps of "1" s.

Another approach would be to represent it properly. Many studies of expertise in psychological literature show that experts perceive their domain differently. They develop better representation of particular environments (Smolensky, 1986). Thus, a clearer understanding of the domain can be reflected in the proper coding of inputs and outputs to solve this problem.

Another way of course is to have a learning/storage algorithm that accounts for higher order correlations. For schemes that incorporate higher order correlations see Smolensky (1986), Baldi and Venkatesh (1987), and Psaltis et al. (1988). It must however be pointed out here that learning from higher order correlations quickly runs into a problem of combinatorial explosion.

6.4.3 Biological and Behavioral Plausibility

If the parallel distributed processing models are to serve as computational models of neural systems they have to take into account observed biological and behavioral phenomenon.

The iterative learning scheme involving gradient descent in error space does not have any known biological counterpart. A major weakness of this work, however, is the fact that it involves a supervised learning scheme (in so far as it concerns iterative error-correction learning). Living organisms do not have a "teacher" in every walk of life, teaching every single association, by providing an error vector after retrieval of every association.

A step closer to reality would be to provide a scalar measure of the error as a teaching signal. A better way would be to have a learning scheme that is behaviorally more justified by learning through the success or failure of a learned

association. This would be like the reinforcement learning scheme of Williams (1986) or the ARP (Associative Reward Penalty) learning scheme of Barto and Jordan (1987).

However giving a scalar error signal increases the search space and thus increases the search time. For some simulation results describing these problems associated with a scalar measure of error see Alspector et al. (1987).

6.5 Future directions

If connectionist models are to serve as cognitive models they have to step out of simplistic worlds of toy problems. This is one of the problems the field of Artificial Intelligence has faced for a long time.

One of the highly unrealistic simplification which is often made in connectionist models is assuming that real world inputs are quantized. This is manifested in the use of fixed width vectors as inputs and outputs. Real world is not so nicely quantized. Inputs in real world vary both in time and space.

Another problem is that many of these models do not account for time dependent phenomenon. Some new schemes solve this problem through the use of innovative architectures (see Jordan, 1986) For some interesting studies using Jordan's model of network, see (Elman, 1988), (Allen R .B., 1988).

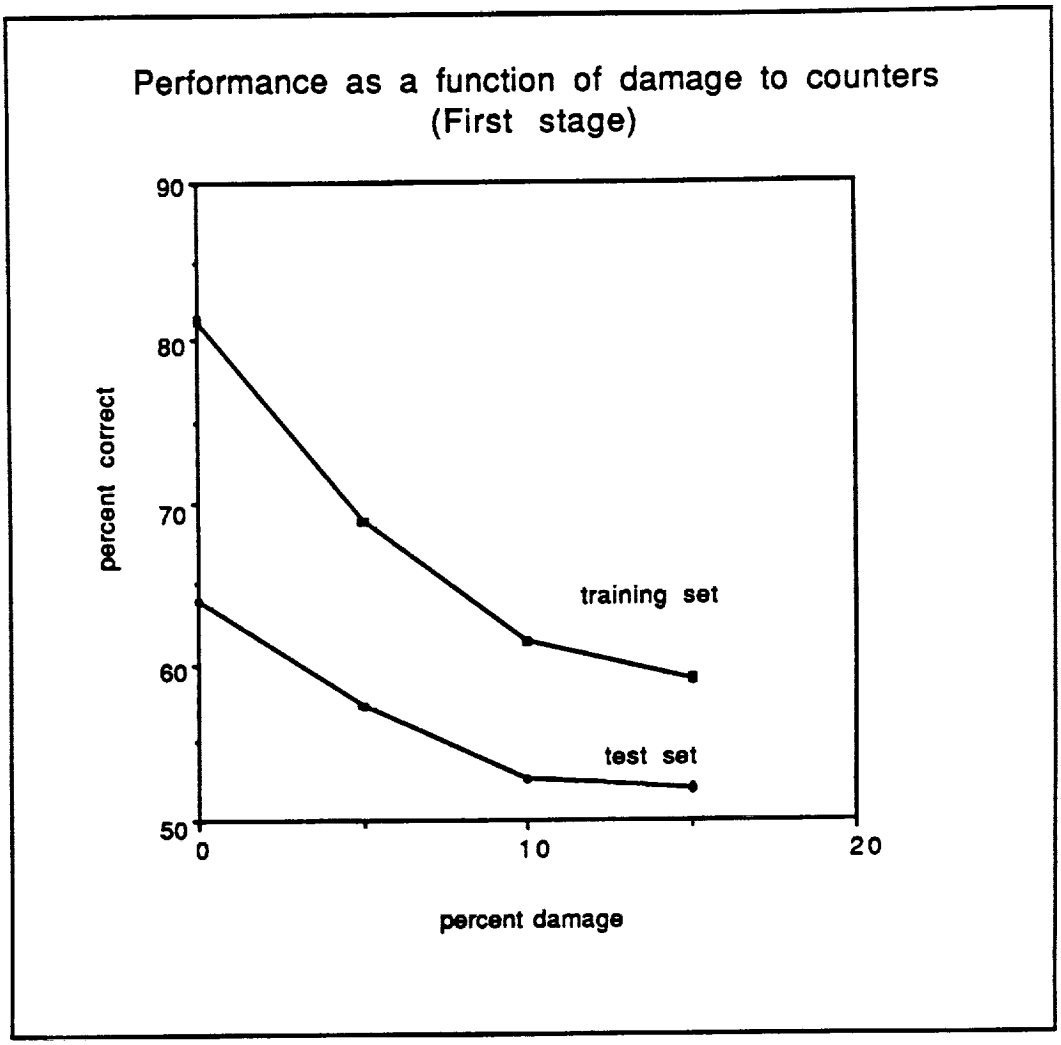

Figure 13 - Damage Resistance (First Stage). Performance as a function of damage in the first stage. The network that was trained as shown in figure 8, was used as a starting network. 5%, 10%, and 15% counters were randomly erased for these simulations.

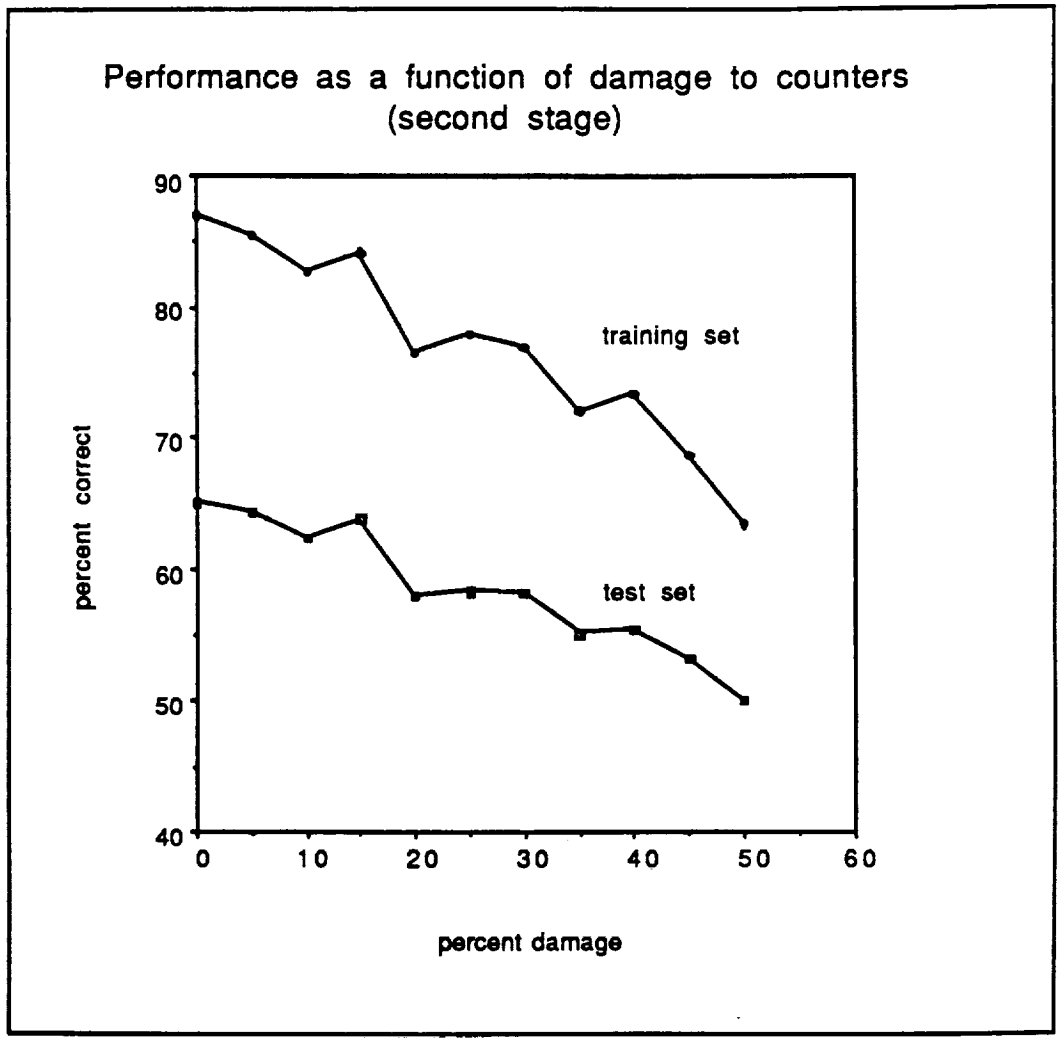

Figure 14 - Damage Resistance (Second Stage). Performance as a function of damage in the second stage. The network that was trained as shown in figure 10, was used as a starting network. Random damage was introduced in stages of 5% increment. The number of locations in the second stage were a significant fraction of the total address space. This may partly explain the better damage resistance in the second stage. In the first stage the number of locations were an extremely small fraction of the total address space.

APPENDIX A

SDM as a three-layer feed-forward network

SDM can be viewed as a three-layer feed-forward network. First we describe a three-layer feed-forward network. Figure 15 shows a three-layer feed-forward network, similar to the one used in the NETtalk study.

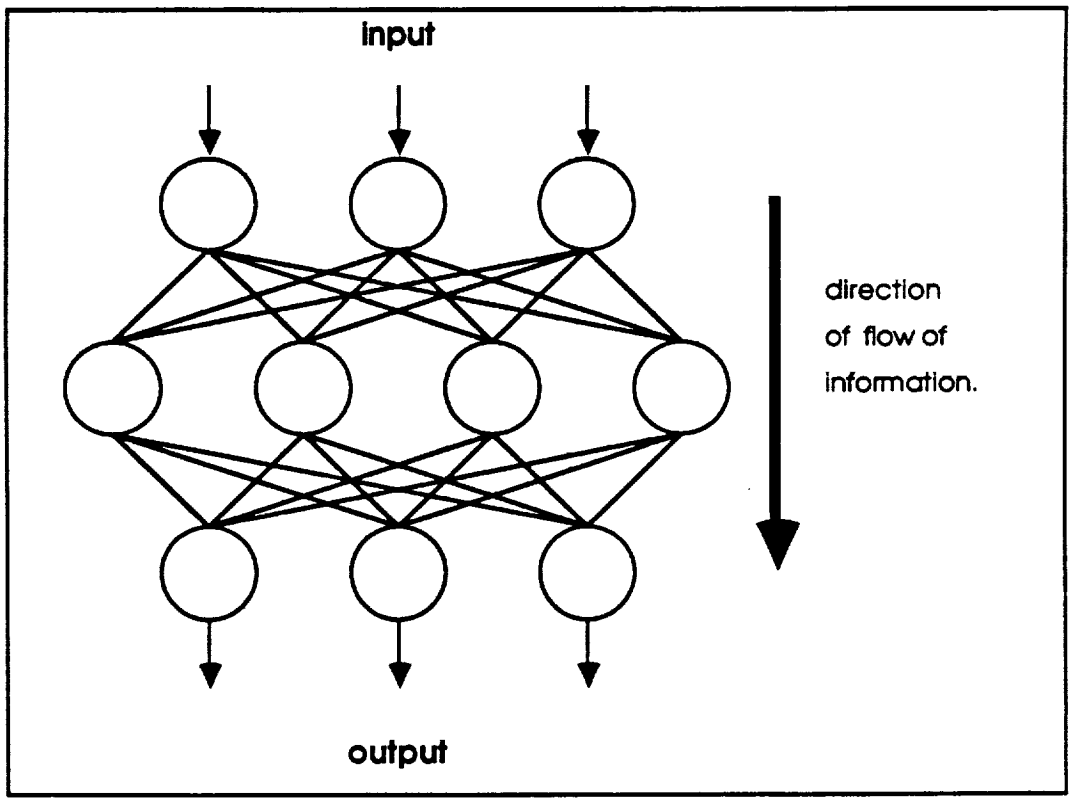

Figure 15 - A three-layer feed-forward network.

There are three layers of computing units. Units in the first layer take one input and compute the identity function. Units in the second and third layer take input from all the units of the previous layer. They all have real valued outputs. Also the weights on the connections between the units are real valued. These weights are all modifiable.

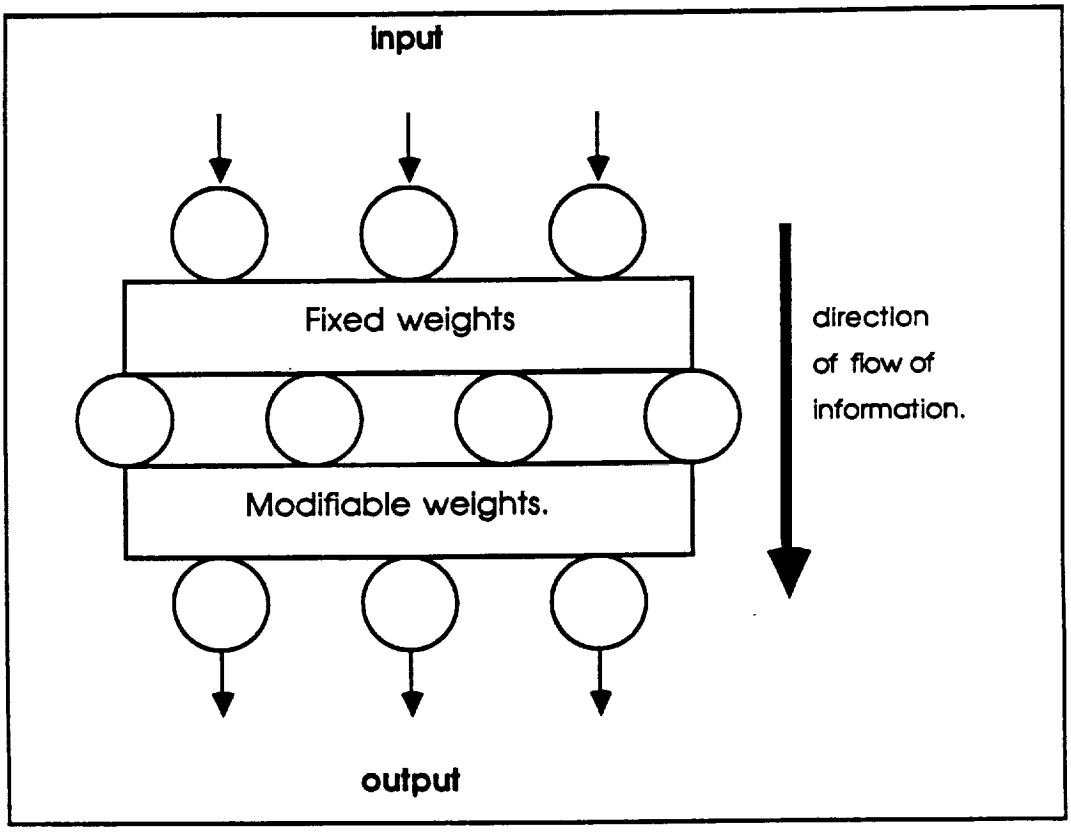

Figure 16 - SDM as a three-layer network.

Figure 16 shows SDM as a 3-layer feed-forward network. In many respects it is different from the network illustrated in figure 15. A major difference

is that the connections between the first and the second layer are fixed and the connections between the second and the third layer are modifiable.

Consider the connections from the first to the second layer. Figure 17 shows these connections in greater detail. **L1** is the first layer or the input layer. There are n units in this layer which take input from outside plus one dummy unit which does not take any input. Units in this layer have a fan-in of 1 input. If X is the input and Y is the output of these units then $Y = +1$ if $X = 1$ and $Y = -1$ if $X = 0$. The dummy unit represents a unit which takes no input and always produces an output = 1.

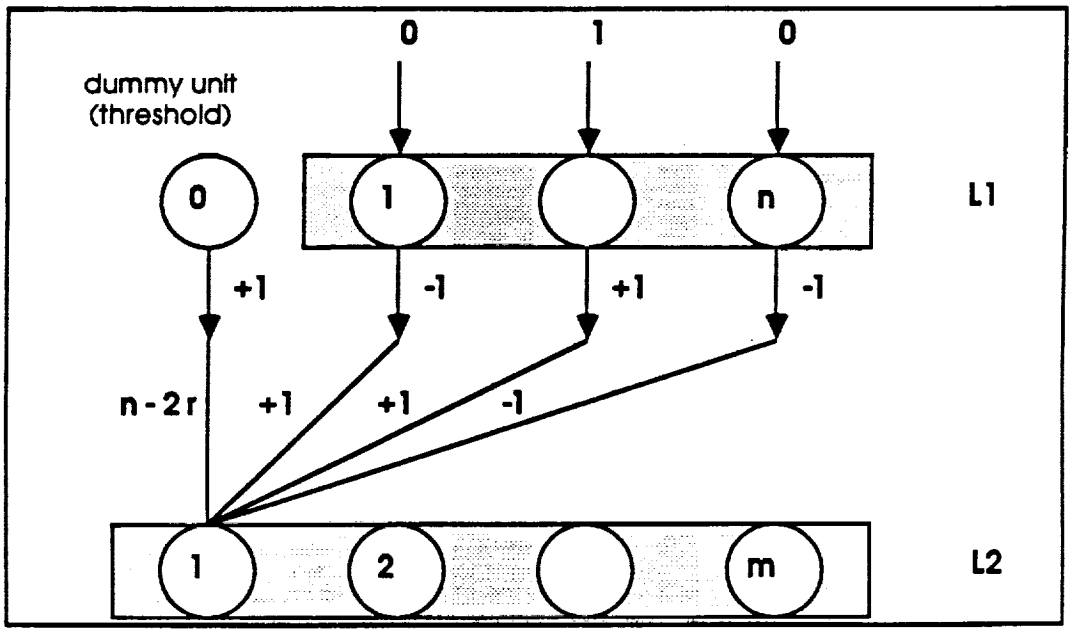

Figure 17 - Fixed weights from the first to the second layer.

The dummy unit and the units in **L1** are connected to all the units in **L2**, i.e., the second layer. There are m units in the second layer. The connections from **L1** to **L2** are binary, either +1 or -1. These connections are randomly chosen. These correspond to the addresses of the hard locations in figure 1. The connection from the dummy unit is an integer which represents the threshold. By keeping this value fixed outputs of different units in **L2** can be set to 1, in response to different inputs to layer 1. This will occur if the connections to a unit in **L2**, from all the units in **L1**, are sufficiently similar to the inputs to units in **L1**. By choosing the strength of the connection from the dummy unit to be n-2r, we can select any unit in **L2** (i.e., force its output to be 1) if the weights on its connections to the units in **L1**, do not differ in more than "r" places from the output of units in the first layer.

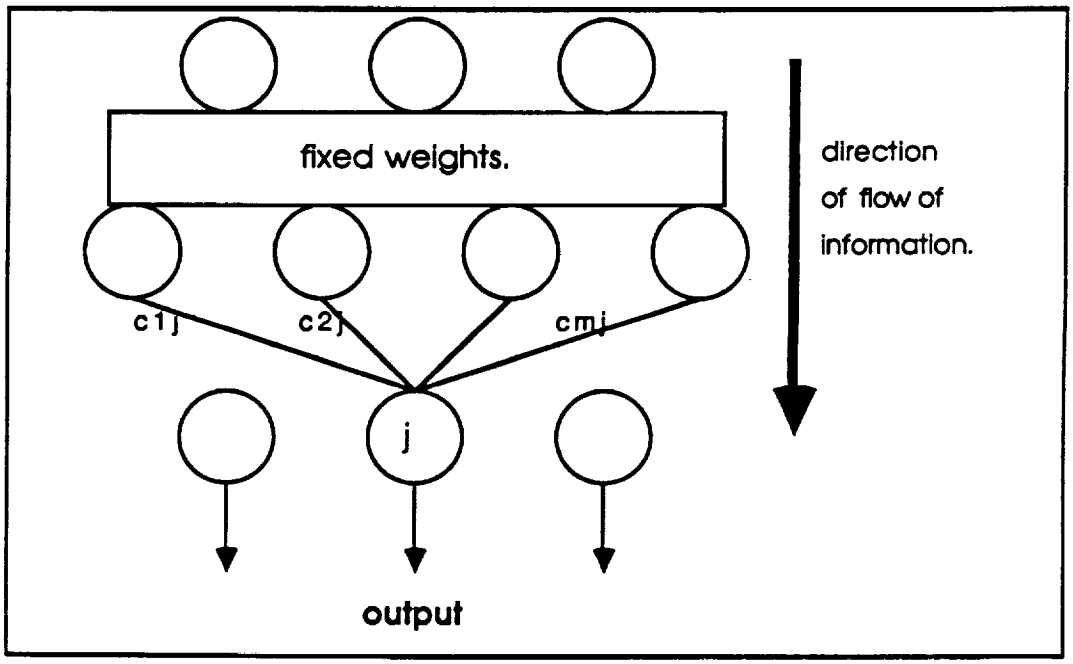

Figure 18 - Modifiable weights.

Units in the second layer are threshold logic units. Their output is "1" if the connections are sufficiently similar to output of the first layer, otherwise the output is "0".

Units in the second layer send their output to all the units in the third layer. The third layer units are also threshold logic units. The connections from the units in the second layer to those in the third layer are integers.

Figure 18 shows the modifiable connections between the second and the third layer. These correspond to the contents of the hard locations in figure 1. Connections c1j, c2j, ..., cmj represent the j^{th} position of each of the counters. Assume that there are n units in the third layer. If the k^{th} hard location (i.e., the k^{th} unit in the second layer) is selected then all the connections from it viz. ck1, ck2, ..., ckn will take part in producing the outputs o1, o2, ..., on respectively. On the other hand if the k^{th} hard location is not selected then the output of the k^{th} unit in the second layer will be zero, hence the connections ck1, ck2, ..., ckn will not take part in producing the outputs o1, o2, ..., on respectively.

In the simulations reported, a few changes to the above architecture have been proposed to facilitate an iterative supervised learning scheme. These include:

1. making the transfer function of the third layer units , a sigmoid.
2. making the connections from the second to third layer real valued. This allows small changes in the values of connections so that the network can be iteratively trained.

Modifying the generalized delta rule for the SDM case, error is propagated back from the third layer to the second layer only. (In the case of NETtalk error is back-propagated all the way to the first layer). The learning rule covers only the selected locations as only they have a nonzero output. Since the output of the selected units is "1", it is not explicitly shown as a multiplicand in the learning rule. **bj**, the amount to be fed back is thus the same as the δ in the generalized delta rule multiplied by the coefficient of learning λ.

The computation of thresholds can again be explained as a dummy unit in the second layer which is always selected and thus participates in producing the outputs o1, .., on.

In addition to showing this similarity between SDM and three-layer feed-forward networks, and thus proposing a learning mechanism for SDM, the present study shows that further improvement in the performance is possible by at least two mechanisms:

1. Choosing connections from the first to the second layers from the training set (i.e. from the set of examples). If they correspond to the examples then they can provide distributed mapping rules.
2. Another improvement suggested is to stack up two stages of SDM by first fixing connections through training in the first stage and then training the second stage.

APPENDIX B

List of symbols used in the phonemic stream.

Following table describes the transcription symbols used in the phonemic stream. First column lists the symbol, second column shows the symbol as it appears in a word in the phonemic stream and the third column contains the same word as it appears in the orthographic stream.

Consonants

p	pu-l	pool
b	blu-	blue
f	fu-d	food
v	vErl	very
m	mI-n	mean
w	wI	we
T	T-IGk	think
D	D-En	then
t	tu-	two
d	de-	day
s	sIK	sick
z	nO-z-	noise
n	nA--t	night
l	lAk-	like

r	r^n	run
C	m^C-	much
J	J^st	just
S	S-i	she
Z	da-ZInt	doesnt
	(As in rouge	and beige)
y	yEt	yet
k	kold	cold
g	gEts	gets
G	T-IG-	thing
?	?M	um
h	hom-	home

Vowels

I	S-i	she
I	wIT	with
e	ple-	play
E	wEnt	went
@	D-@t	that
A	mA	my
^	^-	uh
a	nat	not
u	tu-	two
U	fUl-	full

o	D-o---	though
O	bO-	boy
c	wc-k	walk
W	hW-	how

Combinations

M	-M	um
N	iv-N	even
L	lld-L-	little
Y	-Y-	you
X	sIX	six
*	*n-	one

APPENDIX C

SDM's performance on parity problem.

Parity Problem

This is the generalized XOR problem. The problem is to determine the parity of the input vectors. The training set contained randomly drawn 16-bit vectors as input and their correct parity as the output. Simulations were conducted with different sizes of memory, different learning rates and different training sets. This is a problem that cannot be learned from examples. The performance was, however, unexpectedly very high. With little or no training, the performance on both the training set and the test set was very high. This can be explained by the select mechanism.

As explained earlier, SDM is based on a similarity based storage and retrieval scheme. The locations are selected according to their similarity (or rather, maximum dissimilarity) from a target address. Consider the total address space of n bit vectors. This is given by 2^n. Let, $\mathbf{N}(r)$ be the number of locations selected with select radius r.

$$\mathbf{N}(r) = \sum_{i=0}^{i=r} \frac{n!}{i!\,(n-i)!}$$

Thus, it is clear that for $0 < r < n/2$, a majority of locations selected are exactly at a distance r from the target address. These locations are responsible for influencing the output. Dr. Louis Jaeckel pointed out that a

majority of the selected locations are exactly at a distance of select radius from the target address.

Expanding on Dr. Jaeckel's explanation, we give additional arguments in support of his reasoning.

Two vectors which are separated by a Hamming distance of "1" will have opposite parity. Those separated by a Hamming distance of "2" units will have the same parity. In general two vectors separated by a hamming distance of odd units will have opposite parity and those separated by a Hamming distance of even number of units will have the same parity. Assume that the select radius is even. As pointed out by Dr. Jaeckel, a majority of the selected addresses will be exactly at a distance of "r" from the target address. They will all have the same parity as the target address. In addition, there will be addresses at a distance of exactly "r-2" units, "r-4" units, "r-6" units, ..., down to "0" distance if r is even. Thus, an overwhelming majority of addresses will have the correct parity stored in their data counters. The memory will organize itself with a majority of locations containing correct signal for each new vector that is stored. Similar argument can be given for the case, when the radius of select is odd. Thus, as long as the radius of select is fixed (i.e. write and read operations are performed with the same radius), the memory will always compute correct parity of the target address. In the actualization of the memory, a random sample of the address space is taken to serve as actual addresses. For small values of n and r it may be possible to get a wrong output for a very small number of vectors. But the training procedure quickly eliminates even this error. As the value of n and r

increases the memory starts giving correct output even without training in almost all cases.

Thus, it is clear that the selection mechanism of SDM makes it behave as if it is hard-wired to solve the parity problem.

References

Ackley , D. A. (1987) *Stochastic iterated genetic hillclimbing,* CMU-CS-87-107, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213.

Allen, J. (1985) "Speech synthesis from unrestricted text." - in *Computer Speech Processing.* Frank Fallside and William Woods, (Eds.), Prentice Hall International (UK) Ltd.

Allen, R. B. (1987) Several Studies on Natural Language and Back-Propagation *Proceedings of the IEEE International Conference on Neural Networks,* **Vol.2,** pp.225-341, June 21-24, 1987, San Diego, Ca.

Allen, R. B. (1988) Sequential Connectionists Networks for Answering Simple Questions about a Microworld, to appear in *the proceedings of the annual meeting of the Cognitive Science Society,* August 1988, Montreal, Canada.

Alspector , J., Allen, R. B., Hu, V., and Satyanarayana, Srinagesh (1987) *Stochastic Learning Networks and their Electronic Implementation.* in the proceedings of IEEE Conference on Neural Information Processing Systems. Denver, Colorado, 1987, pp. 9-21, Anderson, D. (Ed.) American Institute of Physics, New York, 1987.

An, Z. G., Mniszewski, S. M., Lee, Y. C., Papcun, G., and Doolen, G.D. (1988) *HIERTALKER: A Default Hierarchy of High Order Neural Networks that Learns*

to Read English Aloud. Presented at the First Annual Meeting of the International Neural Network Society, Boston, September, 6-10, 1988.

Baldi , P., and Venkatesh, S. S. (1987) Number of stable points for spin glasses and neural networks of higher orders. *Physics Review Letters,* **58**(9), 913

Barto, A. G., and Jordan M. I., *Gradient following without back-propagation in layered networks* in proceedings of IEEE ICNN, **Vol 2,** pp 629-636, 1987

Baum, E. B., Moody, J., and Wilczek, F. (1986) *Internal Representations for Associative Memory.* Technical Report, NSF-ITP-86-138. Institute for Theoretical Physics, University of California, Santa Barbara, Ca. 93106.

Bourland, H., and Wellekens, C. J. (1987) *Multilayer Perceptrons and Automatic Speech Recognition.* In proceedings of IEEE First International Conference on Neural Networks, **Volume 4,** pp 407-416, San Diego Ca. June 1987.

Carterette, E. C., and Jones, M. H. (1974) *Informal Speech.* University of California Press, Los Angeles, California.

Cohen, M. A., Grossberg, S., and Stork, D. (1987) *Recent Developments in a Neural Model of Real-Time Speech Analysis and Synthesis.* In proceedings of IEEE First International Conference on Neural Networks, **Volume 4,** pp 443-453, San Diego Ca. June 1987.

Dahl, E. D. (1987) *Accelerated Learning using the generalized Delta Rule.,* in proceedings of IEEE ICNN 1987, **Vol 2,** pp. 523-530

Dreyfus, H. , and Dreyfus, S. (1986) Why skills cannot be represented by rules. - In *Advances in Cognitive Science 1,* 315-335, Sharkey N. E. (Ed.) Ellis Harwood Ltd., Chichester, West Sussex, England.

Elman, J. L., and Zipser, D. (1988) Discovering the hidden structure of speech. *Journal of Acoustical Society of America.* In press.

Elman, J. L. (1988) *Finding structure in time.* CRL Technical Report 8801. Center for Research in Language, University of California, San Diego.

Fanty, M. (1985) *Context-Free Parsing in Connectionist Networks.* Computer Science Department, Technical Report 174, University of Rochester, Rochester, New York, 1985.

Fogelman Soulie, F., Gallinari, P., Le Cun, Y., and Thira, s., (1987) *Evaluation of network architectures on test learning tasks.* In proceedings of IEEE First International Conference on Neural Networks, **Volume 2,** pp 653-660, San Diego Ca. June 1987.

Hanson, S. J., and Kegl, J. (1987) PARSNIP: A connectionist Network that Learns Natural Language Grammar from Exposure to Natural Language Sentences. *The Ninth Annual Conference of The Cognitive Science* Society, 106-119, Lawrence Hillbaum associates, Hillsdale, New Jersey.

Huang, W.P., and Lippmann, R. P. (1987) *Neural Net and Traditional Classifiers.* In the proceedings of IEEE Conference on Neural Information Processing Systems: Natural and Synthetic. Denver, Colorado, 1987, pp.387-396, Anderson, D. (Ed.) American Institute of Physics, New York, 1987.

Jagjit Singh (1966) *Great Ideas in Information Theory, Language and Cybernetics* Dover Publications, Inc., New York.

Jordan, M. I. (1986) *Serial Order: A parallel distributed processing approach.* Institute for Cognitive Science Report 8604. University of California, San Diego.

Kanerva, P. (1984) *Self-propagating Search: A Unified Theory of Memory.* (Rep. No. CSLI-84-7) Stanford: Center for the Study of Language and Information. To appear as a book by Bradford Books/MIT Press (est. 1988.)

Kanerva, P., Cohn, D., and Keeler, J. (1986) *Capacity of Sparse Distributed Memory.* Unpublished.

Keeler, J. A. (1987) *Comparison Between Sparsely Distributed Memory and Hopfield-type Neural Network Models.* RIACS Technical Report 86.31 , Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffett Field, Ca. 1987.

Kirkpatrick, S., and Gelatt, C. D., Jr., and Vechhi, M. P. (1983) Optimization by simulated annealing. *Science,* **220,** 671-680.

Lucassen, J. M., and Mercer, R. L. (1984) An information theoretic approach to the automatic determination of phonemic baseforms. *IEEE ICASSP, 1984,* 42.5.1-42.5.4.

Michalski, R. S., and Chilausky, R. L. (1980) Learning by Being Told and Learning From Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis. *International Journal of Policy Analysis and Information Systems,* **4**, 2.

Mitchell , T. M. (1982) Generalization as Search. *Artificial Intelligence,* **18**, 2, based on Ph. D. Thesis Stanford University, Stanford, Ca. 1978.

Pavel, M., and Moore, R. T. (1988) *Constraints on Generalization By Adaptive Networks.* Presented at the First Annual Meeting of the International Neural Network Society, Boston, September, 6-10, 1988.

Plaut, D. C., and Hinton, G. E. (1987) Learning sets of filters using back-propagation. *Computer Speech and Language*, **2**, 35-61.

Psaltis, D. , Park, C. H., and Hong, J. (1988) Higher Order Associative Memories and Their Optical Implementations. *Neural Networks*, **1**(2), 149-163

Reggia, J. A., and Berndt, R. S. (1985) *Modelling Reading Aloud and its Relevance to Acquired Dyslexia.* In proceedings of the Ninth Annual Symposium on Computer Applications in Medical Care, Nov. 10-13, 1985, Baltimore, MD, pp. 252-256, IEEE Computer Society, Washington, D.C. , 1985.

Rosenblatt, F. (1961) *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms.* Washington, D.C. : Spartan, 1961.

Rosenfeld, R., and Touretzky, D. A. (1987) *Four Capacity Models for Coarse-Coded Memories.* Tech. Report No. CMU-CS-87-182, Computer Science Department, Carnegie Mellon University, Pittsburgh, Pa. 15213

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986) Learning Internal Representations by Error Propagation In *Parallel Distributed Processing Vol. 1: Foundations* David Rumelhart, Jay McClelland (Eds.) MIT Press, Cambridge, MA 1986.

Rumelhart, D. and McClelland, J. (1986) *Parallel Distributed Processing Vol. 1: Foundations* David Rumelhart, Jay McClelland (Eds.) MIT Press, Cambridge, MA 1986.

Sejnowski, T. J., and Rosenberg, C. R. (1986) *NETtalk: A Parallel Network that Learns to Read Aloud.* Johns Hopkins University, Technical Report - JHU/EECS-86/01.

Sejnowski, T. J., and Rosenberg, C. R. (1987) Parallel Networks that Learn to Pronounce English Text. *Complex Systems,* **1,** (1987) 145-168.

Smolensky, P. (1986) Information Processing in Dynamical Systems: Foundations of Harmony Theory. In *Parallel Distributed Processing Vol. 1: Foundations* David Rumelhart, Jay McClelland (eds.) MIT Press.

Stone, G. O. (1986) An Analysis of Delta Rule and Learning of Statistical Associations in *Parallel Distributed Processing Vol. 1: Foundations* David Rumelhart, Jay McClelland (Eds.) MIT Press, Cambridge, MA 1988.

Tank, D. W., and Hopfield, J. J. (1987) *Concentrating Information in Time: Analog Neural Networks with Applications to Speech Recognition Problems.* In proceedings of IEEE First International Conference on Neural Networks, **Volume 4,** pp 455-468, San Diego Ca. June 1987.

Tesauro, G. (1987) Scaling relationships in back-propagation learning: dependence on training set size. *Complex Systems,* **1,** (1987) 367-372

Tesauro, G., and Janssens, R. (1988a) *Scaling relationships in back-propagation learning: Dependence on Predicate Order.* Technical Report CCSR-88-1, Center for Complex System Research, University of Illinois at Urbana Champaign.

Tesauro, G., and Sejnowski, T. J. (1988b) *A Parallel Network that Learns to Play Backgammon.* Technical Report CCSR-88-2, Center for Complex System Research, University of Illinois at Urbana Champaign.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. (1987) Phoneme recognition using time-delay neural networks. *ATR Technical Report TR-I-0006,* Japan: ATR Interpreting Telephony Research Laboratories.

Williams, R. J., (1986) *Reinforcement Learning in Connectionists Networks: A Mathematical Analysis* ICS Report 8605, June 1986, Institute for Cognitive Science, University of California, San Diego

Winston, P.H. (1975) Learning Structural Descriptions from Examples. in *The Psychology of Computer Vision* . Winston, P. H. (Ed.) McGraw Hill Book Company, New York. based on Ph. D. Thesis M.I.T. Cambridge, Ma. (1970).

Index