1991

Simulation of human memory processes using a system of Artificial Neural Networks (ANNs)

Wayne C. Wood
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_theses

Recommended Citation

Wood, Wayne C., "Simulation of human memory processes using a system of Artificial Neural Networks (ANNs)" (1991). *Master's Theses*. 177.
DOI: <https://doi.org/10.31979/etd.xdmx-yd6j>
https://scholarworks.sjsu.edu/etd_theses/177

This Thesis is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Theses by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

U·M·I

University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313-761-4700 800-521-0600

Order Number 1344323

**Simulation of human memory processes using a system of
Artificial Neural Networks (ANNs)**

Wood, Wayne Curtiss, M.S.

San Jose State University, 1991

Copyright ©1990 by Wood, Wayne Curtiss. All rights reserved.

U·M·I

**300 N. Zeeb Rd.
Ann Arbor, MI 48106**

SIMULATION OF HUMAN MEMORY PROCESSES
USING A SYSTEM OF
ARTIFICIAL NEURAL NETWORKS (ANNs)

A Thesis
Presented to
the Faculty of the Department of Psychology
and
the Department of Mathematics and Computer Science
San Jose State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
Wayne C. Wood
May, 1991

APPROVED FOR THE DEPARTMENT OF PSYCHOLOGY

A handwritten signature in cursive script, reading "Kevin Jordan", written over a horizontal line.

Dr. Kevin Jordan, Committee Chairperson

A handwritten signature in cursive script, reading "Robert Cooper", written over a horizontal line.

Dr. Robert Cooper, Committee Member

APPROVED FOR THE DEPARTMENT OF MATHEMATICS
AND COMPUTER SCIENCE

A handwritten signature in cursive script, reading "Michael Beeson", written over a horizontal line.

Dr. Michael Beeson, Committee Member

APPROVED FOR THE UNIVERSITY

A handwritten signature in cursive script, reading "Serena H. Stanford", written over a horizontal line.

ABSTRACT

SIMULATION OF HUMAN MEMORY PROCESSES USING A SYSTEM OF ARTIFICIAL NEURAL NETWORKS (ANNs)

by Wayne C. Wood

Current models of Human Information Processing often are visualized and implemented as discrete linear models. Following the groundwork laid by the Parallel Distributed Processing (PDP) Research Group some research has been done in highly parallel, distributed representations of human memory. My purpose here is to attempt to create a simulation of memory storage and retrieval using a system of Artificial Neural Networks (ANNs) designed in an overall architecture such as that suggested by Craik and Lockhart (1972). The memory itself will be based upon the Active/Inactive distinction as suggested by Lewis (1979).

In memoriam

Fidel Lam
01/10/63 - 12/26/90

SECTION	TABLE OF CONTENTS	PAGE
INTRODUCTION.....		3
METHOD.....		12
RESULTS.....		31
DISCUSSION.....		33
REFERENCES.....		41
APPENDIX. Source Code.....		45

TABLE	LIST OF TABLES	PAGE
1. Results of Learning Trials		32

FIGURE	LIST OF FIGURES	PAGE
1.	Sensory Level	15
2.	Feature Extraction Level	17
3.	Symbolic Level	19
4.	Semantic Level	24
5.	Activation Function	26
6.	The Delta Rule (Widrow-Hoff Learning Rule)	30

SIMULATION OF HUMAN MEMORY PROCESSES
USING A SYSTEM OF
ARTIFICIAL NEURAL NETWORKS (ANNs)

Wayne C. Wood

San Jose State University

Running head: Network Systems Simulation

copyright 1990 Wayne C. Wood, all rights reserved

Abstract

Current models of Human Information Processing often are visualized and implemented as discrete linear models. Following the groundwork laid by the Parallel Distributed Processing (PDP) Research Group some research has been done in highly parallel, distributed representations of human memory. My purpose here is to attempt to create a simulation of memory storage and retrieval using a system of Artificial Neural Networks (ANNs) designed in an overall architecture such as that suggested by Craik and Lockhart (1972). The memory itself will be based upon the Active/Inactive distinction as suggested by Lewis (1979).

Simulation of Human Memory Processes using a System of
Artificial Neural Networks (ANNs)

Several models of memory have been suggested by many researchers from the various schools of Human Information Processing. Invariably, these models contain partitions corresponding to Short or Long Term Memory (STM and LTM respectively). These models borrow heavily from what is known of computers and their architectures, even though many of them precede the invention of the computer. In the past, various other analogies of the brain have been suggested (telephone exchange, etc.) and one by one they have proven to be too restrictive to be of more than marginal, short-term use.

Craik and Lockhart (1972) proposed a framework for research on human memory that embodied a *Levels of Processing* approach. This approach suggests that the processing in human memory functions occurs in a hierarchy of stages. The basis of this theoretical model is that perception involves the processing and/or encoding of incoming stimuli at various levels. To quote:

Preliminary stages are concerned with the analysis of such physical or sensory features as lines, angles, brightness, pitch, and loudness, while later stages are more concerned with matching the input against stored abstractions from past learning; that

is, later stages are concerned with pattern recognition and the extraction of meaning. This conception of a series or hierarchy of processing stages is often referred to as "depth of processing" where the greater "depth" implies a greater degree of semantic or cognitive analysis. (p. 675)

They further suggest that *elaboration coding* may occur after the stimulus has been recognized by the system. Quoting again:

After the stimulus is recognized it may undergo further processing by enrichment or elaboration. For example, after a word is recognized it may trigger associations, images, or stories on the basis of the subject's past experience with the word. ...

[A]nalysis proceeds through a series of sensory stages to levels associated with matching or pattern recognition and finally to semantic-associative stages or stimulus enrichment. (Craik & Lockhart, 1972, p. 675)

The implication of these statements is that the process of memory storage and retrieval may be decomposed into individual processes involved in a hierarchy of processing levels. Each level contributes to the abstraction of the incoming stimuli with the end result being a semantic symbol used for the retrieval of related items in memory. This need

not be a one-way street. The elaboration coding mechanism discussed previously may also serve to allow connections to form between a new incoming stimulus and those memory patterns that have been previously learned.

Where Craik and Lockhart (1972) provide a framework for the processing/encoding mechanisms of memory storage and retrieval they do not go into the specifics of memory store. The current models of Human Information Processing assume a multi-store arrangement along the lines of Sensory storage (Sensory Short-Term Memory [SSTM], Squire, 1987) or Very Short-Term Memory (VSTM) (Wickelgren, 1970), Short-Term Memory (STM) (Hebb, 1949, in Squire, 1987), Long-Term Memory (LTM) (Anderson, 1976; Squire, 1987), and possibly Working Memory (WM) (Anderson, 1985). Many researchers have investigated the pros and cons of this arrangement (Wickelgren, 1970; Miller, 1956). The most obvious weakness is the transfer of information from STM to LTM (storage) and from LTM to WM (retrieval). Estes (1982) expressed his reservations:

...I had begun to wonder whether the information-processing approach and the computer analogy were perhaps being overdone in some respects. The idea that the human being processes discrete packets of information, usually symbols, through successive stages of filtering, manipulation of symbols in

short-term memory, transfer to long-term storage, and so on, was proving fruitful but also constraining. ...[O]ne could see evidence of more analog-appearing properties of the processing system, with memory storage and retrieval based more on the storage and manipulation of information about distributions or on gradients of information about attributes of events than on the passage of discrete items through memory stores. (p. 129)

In his discussion of the engram Estes cites Lashley's inability to isolate the engram and suggests that it is not localized as a discrete packet of information but may, in fact, be distributed through many areas of the brain. Estes (1982) further cites as evidence the fact that memory impairment is related more to the amount of tissue ablated rather than the location of the lesions. This suggests a distributed representation rather than a localized one. Estes (1982) suggested that a collection of engrams rather than a single memory trace may be involved in the establishment of any particular memory (p. 135).

In his discussion of STM, Estes (1982) posits a *memory element* to explain the storage and retrieval process and also suggests that these elements participate in a *state transition* phenomenon with those elements *activated* by a given stimulus representing the knowledge of that stimulus

with those not related to the stimulus remaining inactive (Estes, 1982, p. 137). This model lends itself well to implementation using Artificial Neural Networks (ANNs). Artificial Neural Networks are matrices designed to mathematically model the activity of small processing units embedded in a network structure (i.e., a structure with multiple connections between the processing units). This clearly defines a model of Natural Neurons.

Finally, in his discussion of the architecture of Associative Memories, Estes (1982) makes the following assertion:

...but we know that memory actually is not such an entity, in fact, not an entity at all, but rather an aspect of the functioning of a complex information-processing system. ... we have no reason to think that between input and output the words, numbers, or propositions are stored in specific locations somewhere in the head. What is in the head is a fantastic conglomeration of interconnected nerve cells. One property important to the functioning of the system is that the input of a message, say a printed word, generates a change in the excitability of some portion of the ensemble of cells with the result that the system responds differently to later input of the same or related messages. ... we may

try instead to comprehend the individual's performance by assuming that it reflects the operation of a network of logical elements. ... then our remark about hierarchical structure refers to the arrangement of logical elements that must be assumed in order to explain performance. (pp. 192 - 193)

This appears to indicate the need for a coordinated, distributed collection of units whose pattern of activation provides the medium by which information is stored and retrieved. It is the changes produced in this network representation which may provide us with the means by which we encode abstractions of incoming stimuli and later retrieve those abstractions when presented with the appropriate cues.

Wickelgren (1970) suggests a *multitrace model* of memory in which STM and LTM are represented by individual but parallel memory traces. He suggests:

The basic properties of Multitrace strength theory are as follows: each event and each association between two events is characterized by a vector of unidimensional strength measures for each of the possible time traces (very-short-term memory, VSTM; short-term memory, STM; intermediate-term memory, ITM; and long-term memory, LTM) in each of an unknown number of modalities (visual, auditory,

speech-motor, abstract-verbal, etc.). Each trace in each modality passes through four phases (acquisition, consolidation, decay, [and] retrieval). The acquisition phase refers to the period of presentation or active rehearsal of events during which the memory trace are initiated. However, acquisition is considered to refer to the establishment of potential traces, not usable (retrievable) traces. The conversion from potential traces is accomplished by the consolidation process, which may be a matter of hours or days for LTM, but is on the order of tens of seconds or seconds for ITM and tenths of seconds for STM. (pp. 66 - 67)

The key to Wickelgren's (1970) model is the suggestion that retrieval of a given item of information is based upon the cumulative strengths of these traces.

In retrieval, the strengths of all traces in all modalities for an event or association are combined into a single total strength. It is this (uni-dimensional) total strength which is judged in the retrieval-decision process. In recognition, only the total strength of the test event or association is judged in relation to a criterion to determine the 'yes-no' response. In multiple-choice or recall, the total strengths of all alternatives are compared,

and the alternative with the maximum strength is selected. (Wickelgren, 1970, p. 67)

This suggestion of a summation of the combined traces of multiple sensory modalities has a direct bearing on the construction of an adequate model, as shall be shown later. The input to the system comes from all modalities whether they are attended-to or not. This multiple modality format must be taken into account when investigating the memory system.

Perhaps the current split into SSTM/STM/LTM/WM is too restrictive for the proper modelling of human memory processes. It is possible that this model is just not precise enough to explain all of the observed phenomena. Other models have been suggested, among those is a split into Primary and Secondary memory.

Waugh and Norman (1965) provide us with a model of Primary Memory in their discussion of James' treatise on Primary and Secondary Memory. Primary Memory (PM) is defined as those events or stimuli which are currently active in our consciousness and are part of our psychological present. In contrast, Secondary Memory is not present in our consciousness and is part of our psychological past. In other words, Primary Memory is currently active and available for manipulation whereas Secondary Memory is inactive and may not be altered. Waugh and Norman also suggest that errors in

retrieval and storage are not so much a function of a time factor as they are a function of the number of events that have occurred. They suggest that the current state of Primary Memory which reflects what is currently in our consciousness, may be altered or interfered with by continuing stimulation.

What is being suggested by Waugh and Norman (1965) is that there is a state to which memories may be attributed. The memories are either in an active state meaning they are part of our consciousness, or they are inactive in which case we are not consciously aware of them. Lewis (1979) provides an excellent review of the literature concerning the STM/LTM division and also provides an overview of the literature for and against this division. In his conclusions he expresses his dissatisfaction with the model and proposes his own model of Active/Inactive Memory. In summary:

Active Memory (AM) is considered as a changing subset of all permanent memories possessed by an organism. At any given time many of the permanent memories, which have the potential for being active, are in a relatively inactive state and have little effect on current behavior. (Lewis, 1979, p. 1065)

The key point in Lewis' (1979) model is the premise that Inactive Memory (IM) and Active Memory (AM) need not occupy different physical locations in the brain:

Although we speak of Active Memory (AM) and Inactive Memory (IM), different stores or locations are not implied; it is doubtful that AM occupies a specific site or sites in the brain. It seems better at present to conceive of AM as a patterned state of neural firing that is not specifically localized, although different memories that are active undoubtedly reflect different densities of firings in different parts of the brain. (p. 1066)

This distinction between Active and Inactive Memory is a key factor which allows the success of an implementation using a system of ANNs. My purpose here is to attempt to create a simulation of memory storage and retrieval using a system of ANNs designed in an overall architecture such as the Levels of Processing approach suggested by Craik and Lockhart (1972). The memory itself will be based upon the Active/Inactive distinction as suggested by Lewis (1979).

Method

Materials

This network was developed on a PC/XT with an 8088-2 CPU, an 8087-2 math coprocessor, and 640 KiloBytes of RAM. The development environment was Borland's Turbo-C (v2.0) running under MSDOS 3.01. It is capable of running under any MSDOS compatible machine with at least 512K RAM and a math

coprocessor. All display screens are displayed in text mode for maximum portability.

Design and Procedure

The network system consists of four discrete levels; the image processing level, the feature extraction level, the symbolic level, and the semantic level. This is in keeping with the basic architecture suggested by Craik and Lockhart (1972).

I have implemented this model using an *image level* to provide the transduction services described above, a *feature extraction level* to provide the extraction of relevant features, and a *symbolic level* to provide identification services. The main departure from the Norman and Rumelhart model (1970) consists in the substitution of the Lewis (1979) AM/IM model of memory for the STM/LTM distinction suggested by Norman and Waugh (1970).

The Image Level. The image level of the system models (in a very primitive sense) the activities of the retina and closely associated neural circuits. The retina of this system consists of ALPHA networks (Rosenblatt, 1962, p. 81) which roughly correspond to the receptive fields of the eye (Kalat, 1984, pp. 164 - 166; Rosenzweig & Leiman, 1982, pp. 271 - 273). Rather than try to capture the inherent complexity of the retina itself (on/off center, off/on surround architecture, lateral inhibition, etc.) I chose to

concentrate on the result of its operation. For this reason I selected Rosenblatt's (1962) ALPHA networks to implement the input functions of my system of networks (Rosenblatt, 1962, pp. 79 - 101).

The key concept behind the ALPHA networks is that they consist of multiple Sensory Units (S-units) connected to Associative Units (A-units) which are connected, in turn, to individual Response Units (R-units). There are several points where I diverge from Rosenblatt's ALPHA architecture, however. Firstly, the input units themselves are the S-units and the A-units. There is no intermediate layer of units between the input and output (or S- and R- units). But the function is essentially the same; an input vector is introduced to the input units, and the corresponding output vector is produced on the output units (R-units). The key to the image layer is the arrangement of the layer and the manner in which it connects to subsequent layers (specifically, the Feature Extraction Layer).

The image layer is organized into *visual areas* (see Figure 1). Each visual area consists of 49 receptor fields arranged in a 7 x 7 matrix. There are nine visual areas. The receptor fields, as I have stated previously, are binary input units which have only two possible values, ON or OFF. The receptor fields are organized into visual areas to provide a framework for the introduction of the selected

```
#####  
#####  
#####  
#####  
#####  
#####  
#####
```

(a) View of a visual area located in the sensory area
(without input).

```
#@@@@@#  
#@####@  
#@####@  
#@@@@@#  
#@####@  
#@####@  
#@####@
```

(b) View of the visual areas located in the sensory area
(with input "A").

Figure 1. Sensory Level

stimulus to the system. For the purposes of this demonstration, nine letter words have been selected as the appropriate stimulus. This explains the reasoning behind the nine visual areas.

When a stimulus (word) is introduced to the image layer it is introduced as a pattern of ones and zeros with the ones representing activated receptor fields and the zeros representing inactive receptor fields. In this manner we are able to simulate the introduction of the image onto the retina.

The Feature Extraction Level. The Feature Extraction Level (FEL) is similar to that proposed by Rumelhart and McClelland (1987c) in their Interactive Activation model. The purpose of the feature extraction level is to provide functions similar to those thought to exist in the visual cortex (Rosenzweig et al., 1982, pp. 273 - 274; Kalat, 1984, pp. 166 - 169). Again, rather than attempt to capture the physical complexity of the brain I have concentrated on functional aspects only.

The Feature Extraction level is organized as nine extractors (one for each visual area) consisting of 14 nodes, each representing a different possible feature and location on the visual area. The nodes in the extractors correspond to the elements defined by the Rumelhart-Siple Font (see Figure 2) (Rumelhart & McClelland, 1987c, p. 210).

(a) View of the Feature Extractors
(without input).

(b) View of the Feature Extractors
(with input "A").

Figure 2. Feature Extraction Level

The purpose of the extractors is to provide a level of abstraction between the raw image and the symbolic representation of that image. The extractors serve to convert the image from a series of activation levels on the retina to line segments which may be used for further processing. In essence, the feature extraction level is an intermediate level for processing of the incoming stimulus. The results of this processing level are then fed into the symbolic level of the system.

The Symbolic Level. The Symbolic Level consists of a 243 node vector (logically, a 9×27 matrix) which is used as a symbolic representation of the incoming stimulus (see Figure 3). The purpose of the symbolic level is to provide a final level of abstraction of the incoming stimulus before it is passed off for information retrieval or storage.

Conceptually, the Symbolic Level provides for the creation of a symbolic representation of the incoming stimuli. The incoming stimulus is known in advance to be a word. For the purposes of this model, a word is defined as a *unique combination of features*. English being the ambiguous language it is (Is *lead* a noun or a verb? Is *read* present or past tense?), the domain of stimulus words used in the stimulus set are restricted to nouns, specifically, the names of animals.

```
#####
#####
#####
#####
#####
#####
#####
#####
#####
```

(a) View of the Symbolic level (without input).

```
#####@###
#@#####
#####@#
#####@#####
#####@#####
#####@###
#####@#####
#####@#####
#####@#####
```

(b) View of the Symbolic level
(with input WAYNEWOOD).

Figure 3. Symbolic Level

The Feature Extraction Level serves to coalesce the input pattern into a combination of lines at given locations on the retina. The Symbolic level further coalesces this information by abstracting the representation from the Feature Extraction level into a representation which consists entirely of feature and position information. Since each pattern at this point will be a unique combination of features and positions, we may assert that we have created a symbolic representation based upon the previous definition of a word.

The architecture of this Network System is very similar to that proposed by Selfridge (1958) in Pandemonium. However, though the architecture is very similar, the method by which learning occurs is different.

Pandemonium employs a hill-climbing algorithm in which it attempts to allow a processing unit to achieve the highest score. This algorithm is faced with the bane of all such algorithms, that of local maxima. It is exceedingly difficult for such an algorithm to determine when it has achieved a local maxima as opposed to a global maxima, which would be the optimum result of learning or recall. Pandemonium attempts to bypass such local maxima by randomly choosing processing units (demons) and randomly adjusting their weighted connections in an attempt to bypass or overcome the local maxima. Such a strategy is acceptable, but does not lend itself well to future analysis.

The architecture I have selected uses a gradient descent method of adjusting the weighted connections of the individual *neurons*. The gradient descent algorithm attempts to minimize the error term encountered during the calculation of the response vector. This could lead to the network being trapped at a local minima (as opposed to the local maxima of Pandemonium). By minimizing the error term in each dimension, the architecture is often able to *sidestep* minima by moving into dimensions in which the minima do not exist, in this manner the architecture is able to achieve the desired goal state.

This symbolic representation is the key to the model. In order for a symbol to be effective it must be unique. The uniqueness of the symbol allows it to be used for *keying* information which has been stored in a network. This naturally leads us to a discussion of the Semantic Level.

The Semantic Level. The Semantic Level represents the final link in the chain running from image impacting on the retina through the representation of that knowledge in the network. Once we have a symbolic abstraction of the incoming stimulus we are able to present this to the network as input to form connections with previously learned symbols.

Conceptually, I am trying to show the relationship between a stimulus and its semantic information. For the sake

of clarity I will use schemata as defined in Rumelhart, Smolensky, McClelland, and Hinton (1987):

The basic idea is that schemata are data structures for representing the generic concept stored in memory. There are schemata for generalized concepts underlying objects, situations, events, actions, and sequences of actions. Roughly, schemata are like models of the outside world. To process information with the use of schemata is to determine which model best fits the incoming information. Ultimately, consistent configurations of schemata are discovered which, in concert, offer the best account for the input. This configuration of schemata together constitutes the interpretation of the input. (p. 18)

But how do we access those schemata? Since we have abstracted a symbolic representation at this point, we should be able to use it as a logical key to query the memory storage. Estes (1982) indicates that what is stored in memory is not a multiplicity of connections between the input and response patterns, but rather a coded representation which serves to specify which set of possible patterns may be retrieved (p. 145).

We have effectively achieved this coded representation with the introduction of the symbolic level to the system. This symbolic abstraction provides us with the necessary

coded symbol(s) to specify which pattern we wish to retrieve at the semantic level.

The semantic level in this network represents the top level understanding provided by the retrieval of schemata keyed from the symbolic representation of the stimulus. By this I mean that each node in the semantic level represents another schema. There are 25 nodes in the semantic level each representing a different concept (schema) about a given aspect of an animal (see Figure 4). The incoming symbolic abstraction provides the key for the activation of the schemata represented at this level.

Network Input Preparation (input formatting). The user is prompted for input strings. Such strings are expected to be animal names no more than nine characters in length. Once this string has been read by the program the characters are converted all uppercase and then reduced to their ordinal values (i.e. '_' or blank = 0, 'A' = 1, 'B' = 2, etc.). This is done so the ordinal values of the characters may be used as an index into the retinal patterns which are stored in a template matrix. In the event the string is less than nine characters long it is buffered with spaces. If the string is longer than nine characters it is truncated.

Presentation to the Retina. Defined in the program is a 27 x 49 matrix which contains the templates for the various characters defined in the set of input vectors. Once the

☐ WarmBlooded	☐ Mammal	☐ Large
☐ Cold Blooded	☐ Bird	☐ Medium
	☐ Reptile	☐ Small
	☐ Fish	

☐ Wings	☐ Carnivore	☐ Hide	☐ Swims
☐ Legs	☐ Herbivore	☐ Feathers	☐ Walks
☐ Fins	☐ Omnivore	☐ Scales	☐ Crawls
☐ None	☐ Unknown	☐ Fur	☐ Flies

Figure 4. Semantic Level

input string has been formatted each formatted character of the string is used as an index into this *image matrix*. Each character (from left to right) is extracted from this matrix and presented to the retina of the corresponding visual area. That is, the extracted image (from the matrix) of the first character of the string is overlaid on the first visual area of the sensory level. This is done for each character of the formatted string in turn. This action of overlaying the image on the visual area corresponds to an image falling on the retina. In this manner, the introduction of the stimuli to the network has occurred. Once the image has been overlaid on the Sensory level it is displayed on the console.

Feature Extraction. The feature extractors in the feature extraction level have a one to one correspondence with the visual areas of the sensory level. A 49 X 14 network connects each visual area with its corresponding extractor. Using the image that has been overlaid on the visual area as input, the activation levels of the various features are calculated using the equation in Figure 5.

The simplification function of equation 1 is possible due to the pre-calculation of the weighted connections in the network. Since the visual system itself is not a concern of this simulation it was decided that no *noisy* data would be entered into the system. This greatly simplified the data input tasks and, as can be readily observed, speeds the

$$A_i = \sum w_{ij} a_j \quad A_i = \begin{cases} 1 & \text{if } \sum w_{ij} a_j \geq \Theta \\ 0 & \text{if } \sum w_{ij} a_j < \Theta \end{cases} \quad (1)$$

A_i = the (calculated) activation of a feature in the feature extractor

a_j = the activation of the receptive field in the given visual area.

w_{ij} = the weighted connection between a_i and a_j

Θ = a threshold function (0.857)

Figure 5. Activation Function

process greatly. So, as a result of this feature extraction through the first level of the network we have our first level of abstraction as per those architectures suggested previously (Craik, et al., 1972; Rumelhart & McClelland, 1987c).

Symbol Formation. Now that we have completed the first level of abstraction, Feature Extraction, we must now construct a unique symbol from these features and thereby acquire a key for the storage or retrieval of the information. The Feature Extractors have left us with a mass of features known to be in given locations. However, the network system has not yet *recognized* these features to be representations of letters and, therefore, words. The transition from a jumble of features to a coherent word *symbol* must now occur. Using the output from the Feature Extraction Level as the input to another network we are able to create a unique combination of features in a unique combination of locations thereby creating a *word*. This is accomplished in much the same way as the conversion from raw input to extracted features. The activation function used in this network is the same as equation 1. A key difference between this network and the one appearing between the Sensory Level and the Feature Extraction Level is that there exist inhibitory weighted connections which insure that the feature combinations only activate those letters which *match*

that combination. All other letters in that extraction are inhibited. Thus, we have extracted a symbolic representation from the sensory input. All that needs be done now is the keying of this symbol to semantic content (i.e., the schemata representing knowledge about that symbol).

Memory Retrieval. Using the output from the Symbolic Level we are now able to access Semantic Memory. We have created a unique symbol to represent the key with which we query the memory storage. Again, we use a mathematical construct similar to equation 1.

After the system has settled into a stable state we will have achieved a pattern of activation across the network which represents the stored knowledge about this specific symbol. For the purposes of this simulation we only represent that activation as higher level nodes. But I must emphasize that this is merely a representation for the purposes of simulation. In its natural counterpart we would expect to find activation from any and all informational representations brought forth by this particular combination of sensory inputs.

So what would happen if this combination of inputs were not to produce a viable activation? For the purposes of this simulation, should a retrieval fail it is interpreted as meaning that particular knowledge is not represented in the

network. If such should be the case then the network automatically enters a phase of supervised learning.

Memory Storage. For the purposes of this simulation, the initial stage of supervised learning is interactive (i.e., the user must respond to questions). The input received from the user is used as the output pattern to be associated with the input pattern produced at the Symbolic Level.

Once the target pattern has been created, the system enters into a learning phase in which the network between the symbolic level and the semantic level is modified to learn the new input. The learning trials proceed with repeated iterations of vector-matrix multiplication. The activation function for this network is once again that presented previously (i.e. equation 1). The *delta rule* (Rumelhart et al., 1987c, pp. 83 - 99; Widrow & Hoff, 1960) is used as the learning rule (see Figure 6).

For each iteration of each known pattern through the learning cycle there is an associated error term. For each pattern this is called the Pattern Error. If the error term is squared then the sign is eliminated but the error term retains its meaning. That is, the ordinal amount of the error is retained, but the sign indicating the direction of the error is eliminated. In this manner we are able to maintain an overall value for the amount of error encountered in each pattern. Therefore, for each pattern we derive a

$$\Delta w_{ij} = \epsilon e_i a_j \quad (2)$$

where:

Δw_{ij} = the change in value of the weighted connections

$e_i = t_i - a_i$ (error term)

a_i = calculated value

a_j = activation value of input layer

t_i = target value

ϵ = learning rate

Figure 6. The Delta Rule (Widrow-Hoff Learning Rule)

Pattern Sum of Squared Error (PSS) term. The idea of the gradient descent learning we perform here is to find some point in the N-dimensional space which minimizes this error across all patterns. So we sum the PSS for each iteration through a learning trial across all patterns and derive a Total Sum of Squared Error (TSS) term. When we have minimized this term then we can be said to have found an error minima in the space which will accommodate all currently held patterns.

The network is forced to repeat these learning trials until exit criterion is met. The exit criterion consists of a change in the Total Sum of Squared Error (TSS) between two learning trials that is less than a value epsilon (epsilon = 0.00001).

Results

The network was allowed to learn input and response patterns corresponding to CAT, DOG, and TUNA. After the learning and recall features of the network were tested these patterns were stored in a disk file so as to maintain a known system-state for future experimentation. Input patterns for 13 other animals were introduced to the network and the corresponding recognition and retrieval features were noted (see Table 1).

The responses generated by the network are divided into three categories: *Learned*, *False Recognition*, and *False*

Table 1.

Results of Learning Trials.

FALSE				
WORD	LEARNED (%)	RECOGNITION (%)	RECALL (%)	
CAT	100%			
DOG	100%			
TUNA	100%			
TIGER		TUNA (100%)		
OCELOT			71%	
SHARK			43%	
OCTOPUS		CAT (86%)		
FLAMINGO			0%	
STORK	100%			
SNAKE		STORK (100%)		
HEMIHABU			0%	
ELK		DOG (100%)		
GNU		DOG (100%)		
IBEX		? (29%)		
GIRAFFE		TUNA (71%)		
SPIDER		STORK (100%)		

False recognition shows pattern that was erroneously recognized and percentage match. False recall shows percentage match.

Retrieval. Learned is the condition that exists when a previously learned pattern is correctly retrieved upon re-introduction of the original stimulus. False Retrieval is the condition of an incomplete or otherwise erroneous retrieval of a previously learned pattern. Finally, False Recognition is the condition that exists when a new (unlearned) pattern elicits a partial response, or elicits a complete response from a previously learned pattern. The network was able to accurately learn and recall four of the patterns introduced, for a 25% learning rate. Seven of the patterns elicited false recognitions (44%) and five of the patterns elicited false recalls (31%).

It may at first appear that a 25% learning rate is exceedingly small for such a simulation. However, the predicted learning rate for such a network is $0.15N$ memory states (Dayhoff, 1990, p. 50) where N = the number of output units (25). Using this formula we see that the network should adequately learn 3.75 patterns. By using orthogonal patterns the network should be able to learn slightly more of them. However, it is exceedingly difficult to find orthogonal patterns in the limited domain allowed by this experiment.

Discussion

The system being modelled here shows, on a primitive level, the activity required to process information from its raw state as sensory input (the sensory image) to a final

semantic form for storage. In attempting to model memory processes it is important to develop an appropriate architecture to support the model. McClelland and Rumelhart (1985) provide descriptions of several properties that should be represented in such architectures of distributed memory.

The first of these properties suggests that the architecture should consist of a network of *simple, highly interconnected units*. The network architecture I have created for the purposes of this simulation has this property. The individual *neurons* of the networks are such units. The simple nature of the learning rule and activation function, plus the interconnectedness of the neurons within their networks meet the specified criteria well.

The second of these properties is *modular structure*. By establishing the model as a *system of networks* I have adhered to the modularity principle. Each processing level of the network system is an individual, self-contained module which serves a single purpose. The retina serves to introduce the stimuli to the system, the feature extractors serve to break the stimulus into its component parts, the symbolic level provides recognition services and the semantic level provides the system with its information storage and retrieval mechanism.

A key property is the third, *mental state as a pattern of activation*. In order to properly model memory processes the

model must be able to represent many different memories. By representing a given mental state (or memory) as a pattern of activation across the connections of the network, we are able to do this. Without adding or deleting nodes from the network, and thereby changing the explicit structure of the architecture, the network was able to learn and retrieve given patterns. This was accomplished simply by altering the pattern of activation, and hence the network's *state*, to achieve the desired result.

The next property concerns the role of the specific units within each pattern of activation. It is important that patterns of activation, and hence the firing of the individual units, have a specific, unique meaning within the system of networks. The pattern of learning of this simulation would appear to indicate that just such a property is present. Unique patterns that had achieved orthogonality, or at least linear independence with the other patterns in the learning set allowed the system to have a correspondingly unique pattern of firing. Thus, unique patterns were able to achieve perfect learning and retrieval, whereas patterns that did not produce a unique pattern of firing within the system produced incomplete or erroneous learning or retrieval.

Another property specified by McClelland and Rumelhart defines *memory traces as changes in the weights*. By this they mean changes in the weighted connections between

processing units of the network correspond to memory traces. Even though the patterns of activation come and go, the changes of the weighted connections caused by the learning of a given set of patterns remain. This is a memory trace.

The final property specified by McClelland and Rumelhart defines *retrieval as reinstatement of prior pattern of activation*. This property follows naturally from that above for the definition of a memory trace and also that which specifies that unique patterns have unique meanings within the network. If a given, previously learned, pattern is re-introduced to the network, then that pattern should reproduce the original pattern of activation across the network.

The system of networks I have created for the purpose of this simulation exhibit these properties. Therefore, the initial requirements for a distributed model of memory have been met.

Norman and Rumelhart (1970) provide valuable insights into the overall structure of the network. They indicate that the system can be conveniently separated into two sections: perception and memory. They indicated that they incorporated the initial transduction of the stimuli from raw form into a sensory image, the extraction of features, and the identification of the given pattern into what they refer to as perception. By memory they refer to those processes that pertain to the storage and retrieval of such images.

The architecture of this system is built precisely upon this suggested model. The individual levels of the network system correspond directly to the processes indicated by Norman and Rumelhart (1970).

Craik and Lockhart (1972) indicated that a primary assumption in their model was the belief that incoming stimuli could be encoded to greater abstractions with each level of processing indicating a greater degree of semantic or cognitive analysis. The modular nature of this system of networks provide an analogy to the levels of processing approach suggested by Craik and Lockhart (1972). Each individual network in the system provides a level of abstraction between the incoming stimulus and the final semantic memory trace.

Craik and Lockhart (1972) also discussed *elaboration coding* as a possible mechanism for the triggering of associative memories. For the purposes of this simulation it was not feasible to attempt such an encoding mechanism. However, a logical extension to the model may provide evidence of such a mechanism.

At the semantic level of the network system there are representations of conceptual information about the individual stimuli. Each of these representations is, in fact, an elaboration of the incoming stimulus. Craik and Lockhart (1972) indicated that we are constantly acquiring

more information through our senses, whether it is attended to or not. This same mechanism can be considered to be at work when we are learning. Along with the information that we are attending to and attempting to learn, there is other, previously held information that may form connections with the new stimulus pattern. This is what I attempted to model (on a primitive scale) with this simulation.

The schemata of the semantic level of the system are representations of other patterns of activity within the brain. Conceptually, they are not only elaborations of the current stimulus pattern, but they also patterns which are possibly being elaborated on by this new incoming pattern. It should also be noted that even though the simulation directs itself toward visual stimuli, the representation could just as well have been applied to any, or even all, of the sensory modalities. The learning and retrieval operations in the simulation were an attempt to represent the activation of other knowledge traces (the elaboration coding of Craik and Lockhart) through the recognition or acquisition of an incoming stimulus pattern.

Kesner (1973) provides a rough model for construction of a network, if you allow for slight modifications to remove constructs used in keeping with the original STM/LTM structures. The nature of the model I am constructing here demands the existence of multiple memory traces in the same

space in the same time frame. The STM/LTM distinction is too restrictive for my purposes. By adapting Kesner's (1973) model by the replacement of the STM/LTM distinction with Lewis' (1979) Active/Inactive model I can also define a sequence of events for the storage and retrieval process. Kesner's generalized model assumes that encoding, registration, and storage is processed in an initial perceptual or cue-access store. The chief characteristic of this storage is the match-mismatch process used between STM and LTM for the purpose of information retrieval.(p. 178)

The architecture I have selected for this implementation involves an abstraction process (as per Craik et al., 1972) with all processes concerning themselves with the match-mismatch process suggested by Kesner (1973).

One further aspect of Kesner's (1973) model lends itself well to implementation. He suggests that the same neural systems are used for both information storage and retrieval (Kesner, 1973, p. 178). The implication of this statement is that the same structures used for the retrieval of information from the system may be used for the storage of new information. The failure to find the match for a given input stimulus would automatically trigger those processes required for the acquisition and storage of new inputs.

Kesner's model suggests a recognition system tied into STM with later consolidation into LTM through a parallel

process operating on separate but equal memory traces. It is at this point that I substitute Lewis' (1979) model of AM/IM.

As may be plainly seen, the structures of the network system provide the services of both transient STM and the more robust LTM. No separate, explicit structures are required for these representations. Rather, I believe that the simulation demonstrates that Short- and Long-term memory representations may be made within the same structure. What has been demonstrated is that no explicit physical separation of the various memory stores is required, and also that the Short- vs. Long-term distinction would be better defined as Active- vs. Inactive memory storage.

References

- Anderson, J. A. (1972) A simple neural network generating an interactive memory. In Anderson, J.A. & Rosenfeld, E.(Eds.). *Neurocomputing: Foundations of Research* (pp. 181 - 192). Cambridge, MA: MIT Press.
- Anderson, J. A. and Rosenfeld, E.(Eds.) (1989). *Neurocomputing: Foundations of Research*, Cambridge, MA: MIT Press.
- Anderson, J. R. (1985). *Cognitive Psychology and Its Implications*, New York: W.H. Freeman & Co.
- Craik, F. I. M. and Lockhart, R. S. (1972). Levels of Processing: A Framework for Memory Research. *Journal of Verbal Learning and Verbal Behavior*, 11, 671 - 684.
- Dayhoff, J. (1990). *Neural Network Architectures, an Introduction*, New York: Van Nostrand Reinhold.
- Estes, W. K. (1982). *Models of Learning, Memory, and Choice: Selected Papers*, New York: Praeger Publishers.
- Kalat, J. W. (1984). *Biological Psychology*, Belmont, CA: Wadsworth.
- Kesner, R. (1973). A Neural System Analysis of Memory Storage and Retrieval. *Psychological Bulletin*, 80(3), 177 - 203.
- Knapp, A. G. and Anderson, J. A. (1984). Theory of categorization based on distributed memory storage. In Anderson, J. A. and Rosenfeld, E.(Eds.). *Neurocomputing:*

- Foundations of Research* (pp. 181 - 192). Cambridge, MA: MIT Press.
- Lewis, D. J. (1979). Psychobiology of Active and Inactive Memory. *Psychological Bulletin*, 86(5), 1054 - 1083.
- McClelland, J. L. and Rumelhart, D. E. (1985). Distributed Memory and the Representation of General and Specific Information. *Journal of Experimental Psychology: General*, 114(2), 159 - 188.
- Miller, G. A. (1956). The magical number seven: Plus or minus two. Some limits on our capacity for processing information. *Psychological Review*, 9, 81 - 97.
- Norman, D. A. (1970). *Models of Human Memory*, New York: Academic Press.
- Norman, D. A. and Rumelhart, D. E. (1970). A System for Perception and Memory. In Norman, D. A. (Ed.). *Models of Human Memory* (pp. 19 - 64). New York: Academic Press.
- Rosenblatt, F. (1962). *Principles of Neurodynamics*, Washington, D.C.: Spartan Books.
- Rosenzweig, M. R. and Leiman, A. L. (1982). *Physiological Psychology*, Lexington, MA: Heath.
- Rumelhart, D. E. and McClelland, J. L. (Eds.) (1987a). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Cambridge, MA: MIT Press, vol. 1.

- Rumelhart, D. E. and McClelland, J. L. (Eds.) (1987b). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Cambridge, MA: MIT Press, vol. 2.
- Rumelhart, D. E. and McClelland, J. L. (Eds.) (1987c). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Cambridge, MA: MIT Press, vol. 3.
- Rumelhart, D. E., Smolensky, P., McClelland, J. L., and Hinton, G. E. (1987). Schemata and Sequential Thought Processes in PDP Models. In Rumelhart, D. E. and McClelland, J. L. (Eds.). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (pp. 7 - 57). Cambridge, MA: MIT Press.
- Selfridge, O. G. (1958). Pandemonium: a paradigm for learning. In Anderson, J. A. and Rosenfeld, E. (Eds.). *Neurocomputing: Foundations of Research* (pp. 117 - 122). Cambridge, MA: MIT Press.
- Smolensky, P. (1987). Information Processing in Dynamical Systems: Foundations of Harmony Theory. In Rumelhart, D. E. and McClelland, J. L. (Eds.). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (pp. 194 - 281). Cambridge, MA: MIT Press.
- Squire, L. R. (1987). *Memory and Brain*, New York: Oxford University Press.

- Waugh, N. C. and Norman, D. A. (1965). Primary Memory.
Psychological Review, 72(2), 89 - 104.
- Wickelgren, W. A. (1970). Multitrac Strength Theory. In
Norman, D. A. (Ed.). *Models of Human Memory* (pp.65 -
102). New York: Academic Press.
- Widrow, B. and Hoff, M. (1960). Adaptive switching circuits.
In Dayhoff, J. (Ed.). *Neural Network Architectures, an
Introduction* (pp. 41 - 44). New York: Van Nostrand
Reinhold.

Appendix

Source Code

```

/*****
**  NAME:  THESIS.C
**  PROGRAMMER:  W.C. WOOD
**  copyright 1990 Wayne C. Wood, all rights reserved
**  HISTORY:  900412 ..created
**             900420 ..network operations & screen
**                  display completed.
**             900429 ..weight and memory pattern files
**                  created. learning fundamentals
**                  operational. schema display completed.
**                  schema format defined.
**             900530 ..converted to linear format
**
*****/

/**** GLOBAL INCLUDE STATEMENTS *****/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>
#include <string.h>
#include <dos.h>
#include <conio.h>

#include "patterns.h"

/**** GLOBAL CONSTANT/MACRO DEFINITIONS *****/

#define NumNodes      49      /* num. of receptor fields/retina*/
#define NumFeatures  14      /* num. of features/detector */
#define NumChars      27      /* num. of possible characters */
#define NumLtrs       9       /* num. of letter positions */
#define NumConcepts   25      /* num. of slots in schema */
#define ON            219     /* screen location on */
#define OFF           176     /* screen location off */
#define THETA         0.857   /* threshold value */

#define LRATE          0.015  /* small steps to feel out *
                             * geometry of the error space. */

#define ECRT          0.00001 /* exit criterion */
#define FALSE          0
#define TRUE           !FALSE
#define EOF            -1
#define PHI            0.2

```

```

/** NETWORK OPERATIONS *****/

void ShowNodes(int x, /* x-coordinate */
               int y, /* y-coordinate */
               int z) /* letter image */
/*
** display retina image on console of incoming letter
*/
{
    int i,j=x;

    for(i=1;i<=NumNodes;i++){
        gotoxy(j++,y);
        printf("%c", (images[z][i-1] > THETA)?ON:OFF);
        if((i%7) == 0){
            j = x;
            y++;
        }
    }
    return;
}

void Retina(char *str) /* str is incoming word */
/*
** display retinal image of incoming word.
** traverses formatted str and shows the retinal
** image of the appropriate letter using ShowNodes().
*/
{
    int i,x;

    for(i=1,x=4;i<=NumLtrs;i++,x+=8) ShowNodes(x,2,str[i-1]);
    return;
}

void ShowFeatures(int x, /* x-coordinate */
                  int y, /* y-coordinate */
                  float *output) /* feature extractor image */
/*
** display the feature-extractor images on the console
*/
{
    int i;

    for(i=0;i<NumFeatures;i++){
        switch(i){
            case 0:
                gotoxy(x,y);
                printf((int) (output[i] > THETA)?"-----":
                       "%c%c%c%c%c",OFF,OFF,OFF,OFF,OFF);break;
            case 1:

```

```

        gotoxy(x,++y);
        printf((int) (output[i] > THETA)?"|":"%c",OFF);break;
case 2:
        gotoxy(x+1,y);
        printf((int) (output[i] > THETA)?"\\":"%c",OFF);
        break;
case 3:
        gotoxy(x+2,y);
        printf((int) (output[i] > THETA)?"|":"%c",OFF);break;
case 4:
        gotoxy(x+3,y);
        printf((int) (output[i] > THETA)?"":"/":"%c",OFF);break;
case 5:
        gotoxy(x+4,y);
        printf((int) (output[i] > THETA)?"|":"%c",OFF);break;
case 6:
        gotoxy(x,++y);
        printf((int) (output[i] > THETA)?"---":
                "%c%c%c",OFF,OFF,OFF);

        break;
case 7:
        gotoxy(x+2,y);
        printf((int) (output[i] > THETA)?"---":
                "%c%c%c",OFF,OFF,OFF);

        break;
case 8:
        gotoxy(x,++y);
        printf((int) (output[i] > THETA)?"|":"%c",OFF);break;
case 9:
        gotoxy(x+1,y);
        printf((int) (output[i] > THETA)?"":"/":"%c",OFF);break;
case 10:
        gotoxy(x+2,y);
        printf((int) (output[i] > THETA)?"|":"%c",OFF);break;
case 11:
        gotoxy(x+3,y);
        printf((int) (output[i] > THETA)?"\\":"%c",OFF);
        break;
case 12:
        gotoxy(x+4,y);
        printf((int) (output[i] > THETA)?"|":"%c",OFF);break;
case 13:
        gotoxy(x,++y);
        printf((int) (output[i] > THETA)?"-----":
                "%c%c%c%c%c",OFF,OFF,OFF,OFF,OFF);

        break;
default: ;
}
}
return;
}

```

```

void Occipital(char *str) /* str is incoming word */
/*
**  extract features from retinal image.
**  use vector*matrix with retinal image and ifweights.
**  display image using ShowFeatures().
*/
{
    int i,j,k,x;
    float output[NumFeatures]={0.0,0.0,0.0,0.0,0.0,0.0,0.0,
                                0.0,0.0,0.0,0.0,0.0,0.0,0.0};

    for(i=0,x=5;i<NumLtrs;i++,x+=8){
        for(j=0;j<NumNodes;j++){
            for(k=0;k<NumFeatures;k++){
                output[k] += (float) images[str[i]][j]* ifweights[j][k];
            }
        }
        ShowFeatures(x,10,output);
        for(k=0;k<NumFeatures;k++) output[k] = 0.0;
    }
    return;
}

void Recognition(char *str) /* str is incoming word */
/*
**  extract position and character information from feature
**  detectors.use vector*matrix with feature image and
**  flweights.
*/
{
    int i,j,k,x,y;
    float output[NumChars]={0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
                             0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
                             0.0};

    for(i=0,x=26,y=16;i<NumLtrs;i++,y++,x=26){
        for(j=0;j<NumFeatures;j++){
            for(k=0;k<NumChars;k++){
                output[k] += features[str[i]][j] * flweights[j][k];
            }
        }
        for(j=0;j<NumChars;j++){
            gotoxy(x++,y);
            printf("%c", (output[j] >= THETA)?ON:OFF);
            output[j] = 0.0;
        }
    }
    return;
}

```

```

void Network(char *str) /* str is incoming word */
/*
** boss procedure for Network operations.
*/
{
    Retina(str);
    Occipital(str);
    Recognition(str);
    return;
}

/** SUPPORT OPERATIONS *****/

void GetWord(char *str) /* str is incoming word */
/*
** gets stimulus word input by user
*/
{
    gotoxy(1,1);
    printf("input the animal name [up to 9 characters] or 'quit'
==> ");
    scanf("%s",str);
    return;
}

void FormatWord(char *str) /* str is incoming word */
/*
** convert characters to upper case and reduce to ordinal
** values.
*/
{
    int i,j;

    for(i=0;i<strlen(str);i++)
        str[i] = (char) ((toupper(str[i]) - 'A') + 1);
    if(i < NumLtrs) for(;i<=NumLtrs;i++) str[i] = 0;
    return;
}

void KeepGoing(void)
/*
** software pause with prompt to let user know
*/
{
    union REGS regs;

    gotoxy(27,25);
    printf("hit any key to continue...");
    getch();
}

```

```

    /* flush keyboard buffer */
    regs.x.ax = 0x0C06;
    regs.h.dl = 0xFF;
    intdos(&regs,&regs);

    return;
}

int Finished(char *str) /* str is incoming word */
/*
**   user indicated he/she wished to terminate the session
*/
{
    return(!strcmp(str,"quit"));
}

void LoadMemory(void)
/*
**   load weight matrix into RAM from disk
*/
{
    FILE *ifp;
    char str[255];
    int i,j,N;

    ifp = fopen("wcwts.wts","r");

    for(i=0,N=(NumLtrs*NumChars);i<N;i++){
        for(j=0;j<NumConcepts;j++){
            fscanf(ifp,"%s",&str);
            wcweights[i][j] = (float) atof(str);
        }
    }

    fclose(ifp);
    return;
}

void SaveMemory(void)
/*
**   save weight matrix to disk
*/
{
    FILE *ofp;
    char str[255];
    int i,j,N;

    ofp = fopen("wcwts.wts","w");
    for(i=0,N=(NumLtrs*NumChars);i<N;i++){
        for(j=0;j<NumConcepts;j++){
            fprintf(ofp,"%f\n",wcweights[i][j]);
        }
    }
}

```

```

    }
}
fclose(ofp);
return;
}

/** CONCEPT OPERATIONS *****/

void Display(float *sym) /* sym is schema */
/*
** display schema image as per format specifications
*/
{
    int i;

    for(i=0;i<NumConcepts;i++){
        switch(i){
            case 0: gotoxy(10,4);
                printf("Warm Blooded %c ", (sym[i]>PHI)?ON:OFF);
                printf("%1.1lf",sym[i]);break;
            case 1: gotoxy(10,5);
                printf("Cold Blooded %c ", (sym[i]>PHI)?ON:OFF);
                printf("%1.1lf",sym[i]);break;
            case 2: gotoxy(2,12);
                printf("Wings..... %c ", (sym[i]>PHI)?ON:OFF);
                printf("%1.1lf",sym[i]);break;
            case 3: gotoxy(2,13);
                printf("Legs..... %c ", (sym[i]>PHI)?ON:OFF);
                printf("%1.1lf",sym[i]);break;
            case 4: gotoxy(2,14);
                printf("Fins..... %c ", (sym[i]>PHI)?ON:OFF);
                printf("%1.1lf",sym[i]);break;
            case 5: gotoxy(2,15);
                printf("None..... %c ", (sym[i]>PHI)?ON:OFF);
                printf("%1.1lf",sym[i]);break;
            case 6: gotoxy(20,12);
                printf("Carnivore... %c ", (sym[i]>PHI)?ON:OFF);
                printf("%1.1lf",sym[i]);break;
            case 7: gotoxy(20,13);
                printf("Herbivore... %c ", (sym[i]>PHI)?ON:OFF);
                printf("%1.1lf",sym[i]);break;
            case 8: gotoxy(20,14);
                printf("Omnivore.... %c ", (sym[i]>PHI)?ON:OFF);
                printf("%1.1lf",sym[i]);break;
            case 9: gotoxy(20,15);
                printf("Unknown..... %c ", (sym[i]>PHI)?ON:OFF);
                printf("%1.1lf",sym[i]);break;
            case 10: gotoxy(55,4);
                printf("Large..... %c ", (sym[i]>PHI)?ON:OFF);
                printf("%1.1lf",sym[i]);break;
            case 11: gotoxy(55,5);

```

```

        printf("Medium..... %c ", (sym[i]>PHI)?ON:OFF);
        printf("%1.1lf",sym[i]);break;
    case 12: gotoxy(55,6);
        printf("Small..... %c ", (sym[i]>PHI)?ON:OFF);
        printf("%1.1lf",sym[i]);break;
    case 13: gotoxy(40,12);
        printf("Feathers.... %c ", (sym[i]>PHI)?ON:OFF);
        printf("%1.1lf",sym[i]);break;
    case 14: gotoxy(40,13);
        printf("Scales..... %c ", (sym[i]>PHI)?ON:OFF);
        printf("%1.1lf",sym[i]);break;
    case 15: gotoxy(40,14);
        printf("Fur..... %c ", (sym[i]>PHI)?ON:OFF);
        printf("%1.1lf",sym[i]);break;
    case 16: gotoxy(40,15);
        printf("Hide..... %c ", (sym[i]>PHI)?ON:OFF);
        printf("%1.1lf",sym[i]);break;
    case 17: gotoxy(60,12);
        printf("Swims..... %c ", (sym[i]>PHI)?ON:OFF);
        printf("%1.1lf",sym[i]);break;
    case 18: gotoxy(60,13);
        printf("Walks..... %c ", (sym[i]>PHI)?ON:OFF);
        printf("%1.1lf",sym[i]);break;
    case 19: gotoxy(60,14);
        printf("Crawls..... %c ", (sym[i]>PHI)?ON:OFF);
        printf("%1.1lf",sym[i]);break;
    case 20: gotoxy(60,15);
        printf("Flies..... %c ", (sym[i]>PHI)?ON:OFF);
        printf("%1.1lf",sym[i]);break;
    case 21: gotoxy(33,4);
        printf("Mammal..... %c ", (sym[i]>PHI)?ON:OFF);
        printf("%1.1lf",sym[i]);break;
    case 22: gotoxy(33,5);
        printf("Bird..... %c ", (sym[i]>PHI)?ON:OFF);
        printf("%1.1lf",sym[i]);break;
    case 23: gotoxy(33,6);
        printf("Reptile..... %c ", (sym[i]>PHI)?ON:OFF);
        printf("%1.1lf",sym[i]);break;
    case 24: gotoxy(33,7);
        printf("Fish..... %c ", (sym[i]>PHI)?ON:OFF);
        printf("%1.1lf",sym[i]);break;
    default : break;
}
}
return;
}

```



```

int Retrieve(char *str, /* str is incoming word */
             float *schema)
/*
**  retrieves previously learned schema.
**  use vector*matrix with position image and wcweights.
**  use Display() to show the schema.
*/
{
    int i,j,k,offset,N;
    int symbol[(NumChars*NumLtrs)];

    clrscr();
    for(i=0,offset = (40 - (strlen(str)/2));str[i] != '\0';i++){
        gotoxy(offset+i,2);
        printf("%c", (str[i] + ('A' - 1)));
    }
    for(i=0;i<NumConcepts;i++)      schema[i] = 0.0;
    for(i=0,k=0;i<NumLtrs;i++){
        for(j=0;j<NumChars;j++,k++){
            symbol[k] = characters[str[i]][j];
        }
    }
    for(i=0;i<NumConcepts;i++){
        for(j=0,N=(NumChars*NumLtrs);j<N;j++){
            schema[i] += wcweights[j][i] * symbol[j];
        }
    }
    Display(schema);
    for(i=0,j=0;i<NumConcepts;i++) if(schema[i] > PHI) j++;
    if(j == 7) return(TRUE);
    else return(FALSE);
}

void GetBlood(float *schema)
/*
**  get interactive input about "bloodedness"
*/
{
    char answer;
    int done=FALSE;

    do{
        printf("is the animal warm-blooded? [y/n]  ");
        answer = tolower(getche());
        switch(answer){
            case 'y': schema[0] = 1.0;
                      done = TRUE;
                      break;
            case 'n': schema[1] = 1.0;

```

```

                                done = TRUE;
                                break;
                        default: break;
                }
        } while(!done);
        return;
}

void GetAppendages(float *schema)
/*
**  get interactive input concerning the creatures
**  appendages
*/
{
    char answer;

    printf("\n");
    if(schema[22]) schema[2] = 1.0;
    else if(schema[24]) schema[4] = 1.0;
    else if(schema[23]){
        printf("does it have legs? [y/n] ");
        answer = tolower(getche());
        if(answer == 'y') schema[3] = 1.0;
        else schema[5] = 1.0;
    }
    else {
        printf("does it have wings, fins, or legs? [w/f/l] ");
        answer = tolower(getche());
        switch(answer){
            case 'w': schema[2] = 1.0;break;
            case 'f': schema[4] = 1.0;break;
            case 'l': schema[3] = 1.0;break;
            default: break;
        }
    }
    return;
}

void GetFood(float *schema)
/*
**  get interactive input concerning the creatures food
*/
{
    int i;
    char answer;

    printf("\n");
    for(i=6;i<8;i++){
        printf("does it eat ");
        switch(i){
            case 6: printf("meat? ");break;

```

```

        case 7: printf("plants? ");break;
    }
    printf("[y/n] ");
    answer = tolower(getche());
    if(answer == 'y')    schema[i] = 1.0;
    printf("\n");
}
if(schema[6] && schema[7]){
    schema[6] = 0;
    schema[7] = 0;
    schema[8] = 1.0;
}
else if(!schema[6] && !schema[7]) schema[9] = 1.0;
return;
}

void GetSize(float *schema)
/*
**  get interactive input concerning the creatures size
**/
{
    char answer;

    printf("\nis it small, medium, or large in size? [s/m/l] ");
    answer = tolower(getche());
    switch(answer){
        case 's': schema[12] = 1.0;break;
        case 'm': schema[11] = 1.0;break;
        case 'l': schema[10] = 1.0;break;
        default: break;
    }
    return;
}

void GetCovering(float *schema)
/*
**  get interactive input concerning the animals skin
**  covering
**/
{
    char answer;

    if(schema[22]) schema[13] = 1.0;
    else if(schema[23]) schema[14] = 1.0;
    else if(schema[24]){
        printf("\ndoes it have scales or hide? [s/h] ");
        answer = tolower(getche());
        if(answer == 's') schema[14] = 1.0;
        else schema[16] = 1.0;
    }
}

```

```

    else if(schema[21]){
        printf("\ndoes it have fur or hide? [f/h] ");
        answer = tolower(getche());
        if(answer == 'f') schema[15] = 1.0;
        else schema[16] = 1.0;
    }
    return;
}

void GetMotion(float *schema)
/*
**  get interactive input concerning the creatures means
**  of locomotion
*/
{
    char answer;

    if(schema[24]) schema[17] = 1.0;
    else if(schema[23]) schema[19] = 1.0;
    else {
        printf("\ndoes it walk, swim, or fly? [w/s/f] ");
        answer = tolower(getche());
        switch(answer){
            case 'w': schema[18] = 1.0;break;
            case 's': schema[17] = 1.0;break;
            case 'f': schema[20] = 1.0;break;
            default : break;
        }
    }
    return;
}

void GetType(float *schema)
/*
**  get interactive input regarding the creatures type
*/
{
    char answer;

    printf("\n");
    if(schema[0]){
        printf("is it a bird? [y/n] ");
        answer = tolower(getche());
        if(answer == 'y') schema[22] = 1.0;
        else schema[21] = 1.0;
    }
    else{
        printf("is it a fish? [y/n] ");
        answer = tolower(getche());
        if(answer == 'y') schema[24] = 1.0;
    }
}

```

```

        else schema[23] = 1.0;
    }
    printf("\n");
    return;
}

void Store(char *str, /* str is incoming word */
           float *schema)
/*
** store the input pattern for future learning
*/
{
    FILE *ifp,*ofp;
    int ch,i;

    ifp = fopen("memory.ltm","r");
    ofp = fopen("temp.tmp","w");
    while((ch = fgetc(ifp)) != EOF) fputc(ch,ofp);
    fclose(ifp);
    unlink("memory.ltm");
    for(i=0;i<NumLtrs;i++) fputc(str[i],ofp);
    for(i=0;i<NumConcepts;i++) fputc((int)schema[i],ofp);
    fclose(ofp);
    rename("temp.tmp","memory.ltm");
    return;
}

void CalculateOutput(int *temp, /* word object or symbol */
                    float *tmp_pat) /* schema image */
/*
** matrix operation... calculate ivector * wmatrix
*/
{
    int i,j,k,N;

    for(i=0;i<NumConcepts;i++){
        for(j=0,N=(NumLtrs*NumChars);j<N;j++){
            tmp_pat[i] += (float) temp[j] * wcweights[j][i];
        }
    }
    return;
}

```

```

float CalculateError(int *temp,
                    float *tmp_pat,
                    float *schema)
/*
** calculate the error term for of the response vector
*/
{
    int i,j,k;
    float err[NumConcepts],pss=0.0;

    for(i=0;i<NumConcepts;i++){
        if(tmp_pat[i] > THETA) tmp_pat[i] = 1.0;
        err[i] = schema[i] - tmp_pat[i];
        pss += (err[i] * err[i]);
    }
    for(i=0;i<NumConcepts;i++){
        for(j=0,k=(NumLtrs*NumChars);j<k;j++){
            wcweights[j][i] += err[i] * LRATE * temp[j];
        }
    }
    return(pss);
}

void Consolidate(void)
/*
** this function handles the learning of new input patterns
*/
{
    int i,j,k,l,ch,temp[(NumLtrs*NumChars)];
    FILE *ifp;
    float tss0=0.0,tss1=0.0,tss2=0.0,pss,tmp,
    tmp_pat[NumConcepts] = {0.0,0.0,0.0,0.0,
                           0.0,0.0,0.0,0.0,
                           0.0,0.0,0.0,0.0,
                           0.0,0.0,0.0,0.0,
                           0.0,0.0,0.0,0.0,
                           0.0,0.0,0.0,0.0};

    clrscr();
    do{
        ifp = fopen("memory.ltm","r");
        tss0 = 0.0;
        do{
            for(i=0,k=0;i<NumLtrs;i++){
                ch = fgetc(ifp);

                /* load word matrix */
                if((ch < 27) && (ch != EOF)){
                    for(j=0;j<NumChars;j++,k++){
                        temp[k] = characters[ch][j];
                    }
                }
            }
        } while(!feof(ifp));
    } while(1);
}

```

```

        else{
            break;
        }
    }

    /* load schema */
    if((ch < 27) && (ch != EOF)){

        for(i=0;i<NumConcepts;i++){
            fscanf(ifp,"%c",&ch);
            schema[i] = (float) ch;
        }
    }

    if((ch < 27) && (ch != EOF)){
        CalculateOutput(temp,tmp_pat);
        pss = CalculateError(temp,tmp_pat,schema);
        tss0 += pss;
    }

    gotoxy(30,10);
    printf("pss      = %5.5f          ",pss);
    } while(ch != EOF);
    fclose(ifp);
    if(ch == EOF){
        tss2 = tss1;
        tss1 = tss0;
    }
    gotoxy(30,15);
    printf("tss(t)    = %5.5f          ",tss0);
    gotoxy(30,16);
    printf("tss(t-1) = %5.5f          ",tss2);
    tmp = tss2 - tss0;
    if(tmp < 0) tmp =1.0;
}while(tmp > ECRIT);
return;
}

void Learn(char *str) /* str is incoming word */
/*
**  this function handles the interactive data acquisition
**  for the learning of new patterns
**
*/
{
    int i;

    do{
        clrscr();
        for(i=0;i<NumConcepts;i++) schema[i] = 0.0;
        GetBlood(schema);
        GetType(schema);
    }

```

```

        GetAppendages(schema);
        GetFood(schema);
        GetSize(schema);
        GetCovering(schema);
        GetMotion(schema);
        clrscr();
        Display(schema);
        gotoxy(30,23);
        printf("is this correct? [y/n] ");
    } while(tolower(getch()) != 'y');
    Store(str,schema);
    Consolidate();

    for(i=0;i<2;i++){
        sound(200);
        delay(100);
        nosound();
        sound(400);
        delay(100);
        nosound();
    }
    return;
}

/**** MAINLINE *****/

int main()
{
    char ch,str[255]; /* str is incoming word */
    int found,
        done;

    found = (int) sin(0.0); /* to keep the floating point
    gods happy */

    clrscr();
    gotoxy(32,10);
    textmode(3);
    textcolor(LIGHTRED+BLINK);
    printf("LOADING MEMORY...");
    LoadMemory();
    textcolor(LIGHTGRAY);
    do{
        clrscr();
        GetWord(str);
        done = Finished(str);
        if(!done){
            FormatWord(str);
            Network(str);

```



```
KeepGoing();
found = Retrieve(str,schema);
KeepGoing();
if(!found){
    gotoxy(22,24);
    printf("INCOMPLETE RECALL -- RELEARN? [y/n] ");
    ch = tolower(getche());
    if(ch == 'y'){
        dirty = TRUE;
        Learn(str);
        KeepGoing();
    }
}
} while(!done);
if(dirty) SaveMemory();
clrscr();

} /* end mainline */
```