NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AN EVALUATION OF HNeT98 (HOLOGRAPHIC QUANTUM NEURAL TECHNOLOGY) SOFTWARE PACKAGE

by

Darryl Langford

June 2000

Thesis Advisor: Carlos F. Borges
Second Reader: Bard Mansager

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE June 2000	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE : An Evaluation Of HNeT98 (Holographic Quantum Neural Technology) Software Package

5. FUNDING NUMBERS

6. AUTHOR(S)
Langford, Darryl

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE
A

13. ABSTRACT ***(maximum 200 words)***

This thesis investigates the properties of a software package called HNeT (Holographic/Quantum Neural Technology) which is based on the use of an artificial intelligence tool called Neural Networks. The basis for the investigation of this software is to establish its reliability, effectiveness and efficiency. Neural technology is a technological replication of the biological neural system designed to learn data patterns and process the data (stimulus) and then generate a response based on the memory of the data. HNeT theory is fundamentally different from the standard Artificial Neural System (ANS) in that it uses complex scalars to evaluate internal mappings of one set of values (stimuli) to another set of values (responses). HNeT employs a process known as *enfolding*, which allows the learning and subsequent recall of many stimulus-response associations to be compressed into a single HNeT neuron cell improving the speed of learning and recall accuracy as well as reducing storage requirements. Whereas the traditional ANS stores stimulus patterns separately as a reference template within a cell and are compared one at a time to a new incoming stimulus response pattern which in this case, requires larger amounts of memory.

14. SUBJECT TERMS
Artificial Neural Networks, HNeT, Enfolding, Adaline, Madaline

15. NUMBER OF PAGES
71

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

AN EVALUATION OF HNeT98 (HOLOGRAPHIC QUANTUM NEURAL TECHNOLOGY) SOFTWARE PACKAGE

Darryl Langford
Captain, United States Army
B.S., Southern University, 1990

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

NAVAL POSTGRADUATE SCHOOL
June 2000

Author: _____
Darryl Langford

Approved by: _____
Carlos F. Borges, Thesis Advisor

_____
Bard Mansager, Second Reader

_____
Michael A. Morgan, Chairman
Department of Mathematics

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This thesis investigates the properties of a software package called HNeT (Holographic/Quantum Neural Technology) which is based on the use of an artificial intelligence tool called Neural Networks. The basis for the investigation of this software is to establish its reliability, effectiveness and efficiency. Neural technology is a technological replication of the biological neural system designed to learn data patterns and process the data (stimulus) and then generate a response based on the memory of the data. HNeT theory is fundamentally different from the standard Artificial Neural System (ANS) in that it uses complex scalars to evaluate internal mappings of one set of values (stimuli) to another set of values (responses). HNeT employs a process known as *enfolding*, which allows the learning and subsequent recall of many stimulus-response associations to be compressed into a single HNeT neuron cell improving the speed of learning and recall accuracy as well as reducing storage requirements. Whereas the traditional ANS stores stimulus patterns separately as a reference template within a cell and are compared one at a time to a new incoming stimulus response pattern which in this case, requires larger amounts of memory.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

The ability of machines to perform accurate function prediction remains an unsolved problem. Although conventional sensors used in military applications provide enough information for a human to predict an events outcome, the extension to automatic prediction by machines is still impractical using current computer designs and methods of construction.

HNeT is a program that is designed to provide alternative methods for predictions of outcomes by way of artificial neural networks. Testing this software will provide information on its reliability, efficiency and whether it will provide an adequate vehicle of improvement for military applications. For instance, if a SCUD missile is fired, will this technology be instrumental in air defense detection through image processing or similarly, if a tank is positioned on the battlefield, will this technology enhance the probability of its detection.

These applications will not be tested directly, however, from examining the software we will be able to provide valuable information that will lead to potential further investigation of the previously mentioned instances.

A. BACKGROUND

Biologists have studied the human brain for many years. The better we understand the brain, the better we can emulate it, and build artificial "thinking machines" that process and respond just as the human brain. As information about the functions of the

human brain accumulated, new technologies arose and the search for an artificial neural network began.

Holographic Quantum Neural Technology (HNeT) is a software package that is modeled to operate in the fashion of a human brain. It supports a framework in which the operation of stimulus-response learning and recall may be performed within single neuron cell structures. The HNeT library houses seven neuron cell types, four of which are modeled after the cerebellum and neo-cortex (the more predominate cells). The predominate cells are the granulate, stellate, pyramidal, and the purkinjee cells.

The mathematical concepts in HNeT are somewhat abstract, however, you do not need a full understanding of the theory to run applications. It is important that you understand how stimulus–response information is presented to the system, and how the various types of holographic/quantum neural cells interact with each other.
The number of scalars within that matrix can be no larger than the number of scalars within only one stimulus pattern.

B. OBJECTIVES

The objectives for this thesis are to determine the differences between traditional artificial neural networks (ANNs) and HNeT and to evaluate the software's response to two mathematical concepts. The first is simple function evaluation. It will be determined if the software is capable of learning to emulate various simple functions. The reason for starting with these functions is to test HNeT's capability of generating response-recall of common functions that is familiar to most. After HNeT is evaluated using these

functions, the error of the output will give adequate information on this software's capability of evaluating simple functions.

The second concept is evaluating the random number phenomenon provided by California Lotto data. This is a more sophisticated operation in that HNeT will be required to guess a response given a set of values (stimulus) from randomly generated numbers. There will be several versions of testing of this concept to verify the software's reliability. The objective is to determine if HNeT can discern the statistically meaningful content of the stimulus.

THIS PAGE INTENTIONALLY LEFT BLANK

II. NEURAL NETWORKS AND HNET: A COMPARISON

A. WHAT IS A NEURAL NET?

1. Artificial Neural Networks

An artificial neural network is a system for information processing whose performance characteristics are analogous to biological neural networks. Artificial neural networks have been developed as generalizations of mathematical models of neural biology, based on assumptions that information processing occurs at many simple neuron elements, connecting links are used to pass signals between neurons, each connecting link has an associated weight which attenuates the signal transmitted in a typical neural net, and each neuron applies an activation function (nonlinear in most cases) to its net input to determine its output signal.

Characterizations of a neural network are (1) its pattern of connections between the neurons (*architecture*), (2) its method of determining the weights on the connections which is called its *training* or *learning* algorithm, and (3) its activation function.

A large number of simple processing elements called neurons, units, cells, or nodes are the primary makeup of a neural net. Each neuron, with an associated weight, is connected to other neurons by means of directed communication links. The weights represent information being used by the net to solve a problem. Neural nets can be applied to a wide variety of problems, such as storing and recalling data or patterns,

classifying patterns, performing general mappings from input patterns to output patterns, grouping similar patterns, or finding solutions to constrained optimization problems.

Each neuron has an internal state which is a function of the inputs received. These internal states are called its activation or activity level. A neuron typically sends its activation one signal at a time to several other neurons.

Fausett [Ref. 1] provides the following example of a neuron activation. Consider a neuron **Y**, illustrated in Figure 1, that receives inputs from neurons $\mathbf{X}_1$, $\mathbf{X}_2$, and $\mathbf{X}_3$. The activations (output signals) of these neurons are x_1, x_2, and x_3, respectively. The weights on the connections from $\mathbf{X}_1$, $\mathbf{X}_2$, and $\mathbf{X}_3$ to neuron **Y** are w_1, w_2, and w_3, respectively. The net inputs, y_in, to neuron **Y** is the sum of the weighted signals from neurons $\mathbf{X}_1$, $\mathbf{X}_2$, and $\mathbf{X}_3$, i.e.,

$$y\_in = w_1 x_1 + w_2 x_2 + w_3 x_3 = \sum_{i=1}^{3} x_i w_i \,. \tag{1}$$

The activation y of the neuron **Y** is given by some function of its input, $y = f(y\_in)$, e.g., the logistic sigmoid function (an S-shaped curve)

$$f(x) = \frac{1}{1 + \exp(-x)} \tag{2}$$

or any of a number of other activation functions. Now suppose further that the neuron **Y** is connected to neurons $\mathbf{Z}_1$ and $\mathbf{Z}_2$, with weights v_1 and v_2, respectively, as shown in Figure 2.

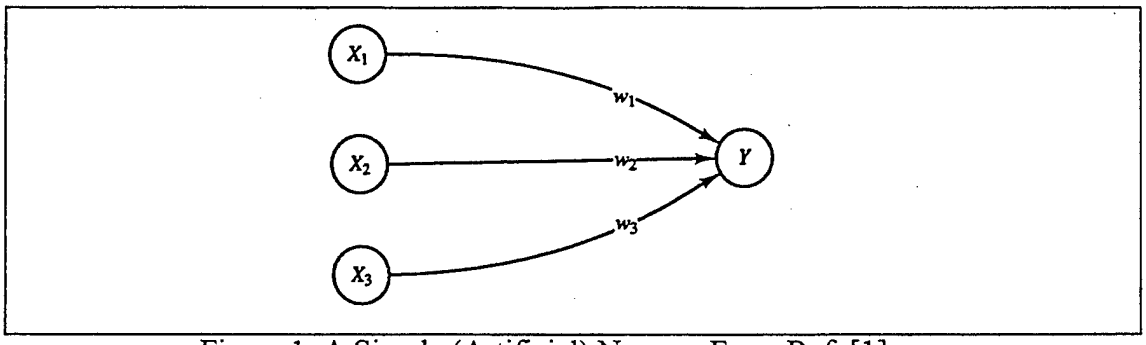

Figure 1. A Simple (Artificial) Neuron. From Ref. [1].

Neuron **Y** sends its signal y to each of these units. However, in general , the values received by neurons $\mathbf{Z}_1$ and $\mathbf{Z}_2$ will be different, because each signal is scaled by the appropriate weight, v_1 or v_2. In a typical net, the activations z_1 and z_2 of neurons $\mathbf{Z}_1$ and $\mathbf{Z}_2$ would depend on inputs from several or even many neurons, not just one, as shown in this simple example.

Although the neural network in Figure 2 is very simple, the presence of a hidden unit, together with a nonlinear activation function, gives it the ability to solve many more problems than can be solved by a net with only input and output units. On the other hand, it is more difficult to train (i.e., find optimal values for the weights) a net with hidden units.

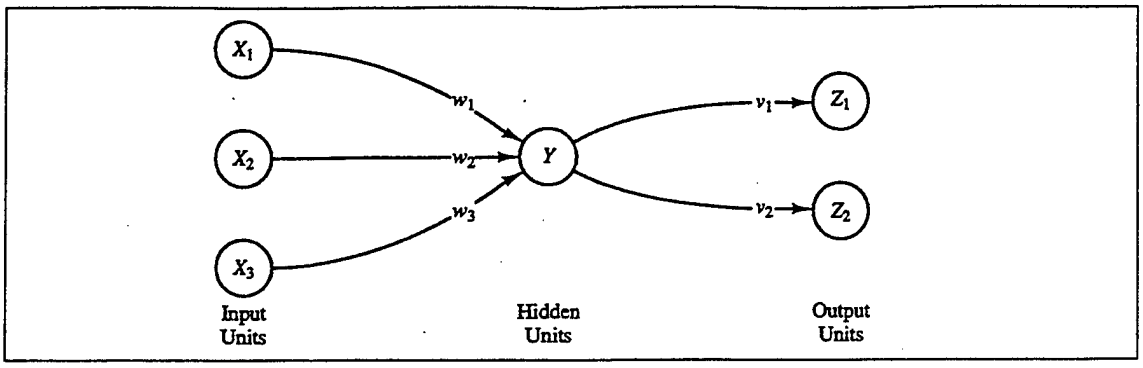

Figure 2. A Very Simple Neuron Network. From Ref. [1].

2. Biological Neural Networks

The extent to which an artificial neural network models a particular biological neural system varies, which causes much concern for some researchers. For others, the ability of the net to perform useful tasks is the focal point of continued research rather than the biological plausibility of the net.

There is a close analogous relationship between a biological neuron (brain or nerve cell) and an artificial neuron (processing element). There are three components of a biological neuron that are of particular interest in understanding an artificial neuron. Those components are *dendrites, soma,* and *axon.* The dendrites (many in number) receive signals from other neurons. The signals that are transmitted are electric impulses that travel by means of a chemical process across a synaptic gap. The action of the chemical transmitter modifies the incoming signal (typically, by scaling the frequency of the signals that are received) in a manner similar to the action of the weights in an artificial neural network.

The cell body, known as the *soma*, sums the incoming signals. When sufficient input is received, the cell transmits a signal over its axons to other cells. The frequency of

transmitting varies and can be viewed as a signal of either greater or lesser magnitude. This process is closely matched to looking at discrete time steps and summing all activity (signals received or sent) at a particular point in time.

The transmission of the signal from a particular neuron is accomplished by an action potential resulting from distinctive concentrations of ions on either side of the neuron's axon sheath (the brain's "white matter"). Potassium, sodium, and chloride ions are most directly involved.

Figure 3 shows a generic illustration of a biological neuron, together with axons from two other neurons (from which the illustrated neuron could receive signals) and dendrites for two other neurons (to which the original neuron would send signals).

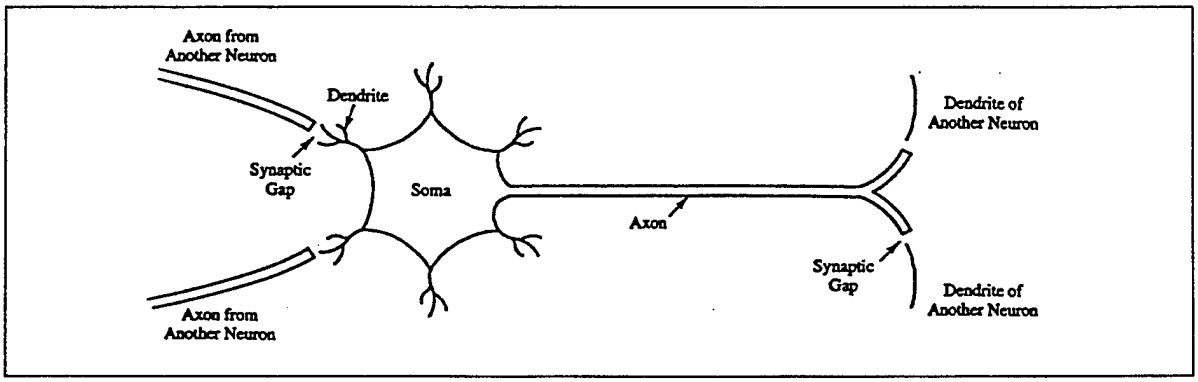

Figure 3. Biological Neuron. From Ref.[1].

Fausett [Ref. 1] gives several key features of the processing elements of artificial neural networks suggested by the properties of biological neurons, viz., that:

1. The process element receives many signals.
2. Signals may be modified by a weight at the receiving synapse.
3. The process element sums the weighted input.
4. Under appropriate circumstances, (sufficient input), the neuron transmits a signal output.

5. The output from a particular neuron may go to many other neurons (the axon branches).
6. Information processing is local (although other means of transmission, such as the action hormones, may suggest means of overall process control).
7. Memory is distributed:
   a. Long term memory resides in the neurons' synapses or weights.
   b. Short term memory corresponds to the signals sent by the neurons.
8 A synapse's strength may be modified by experience.
9. Neurotransmitters for synapses may be excitatory or inhibitory.

Artificial neural networks share another important characteristic with biological neural systems which is called *fault tolerance*. Biological systems are fault tolerant in two respects. First, we are able to recognize many input signals that are somewhat different from any signal we have seen before. An example of this is our ability to look at a picture we have not seen before and recognize a person in that picture or to recognize a person after a long period of time.

Second, we are able to tolerate damage to the neural system itself. Johnson & Brown [Ref. 2] state that humans are born with as many as 100 billion neurons. Most of these are in the brain, and most are not replaced when they die. In spite of our continuous loss of neurons, we continue to learn.

3. Adaline and Madaline

We will begin this section by first introducing some activation functions most common to artificial neural networks. An activation function is defined as a function that

transforms the net input to a neuron into its activation. It is also known as a transfer, or output function.

The first function to be discussed was introduced previously. It is known as the *binary sigmoid* or the *logistic sigmoid.*

$$f(x) = \frac{1}{1 + \exp(-\sigma x)}. \tag{3}$$

$$f'(x) = \sigma f(x)\,[1 - f(x)]. \tag{4}$$

This function is illustrated in Figure 4 for two values of the *steepness parameter* σ. This function is used for neural nets in which the desired output values either are binary or are in the interval between zero and one. It is important to point out that as σ approaches infinity, the function becomes a binary step function.

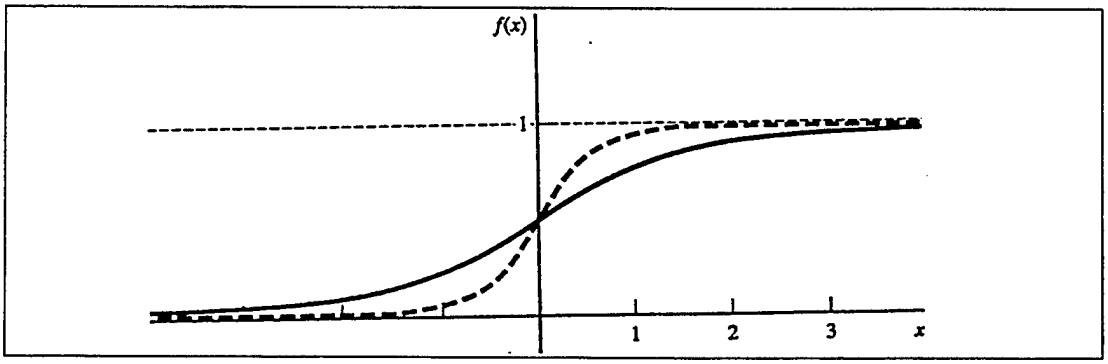

Figure 4. Binary Sigmoid Steepness Parameter σ =1 and σ = 3. From Ref. [1].

The bipolar sigmoid is another activation function that is very common to neural nets. This function is often used as an activation function when the desired range of output values is between minus one and one. It is illustrated in Figure 5 for σ = 1.

$$g(x) = 2f(x) - 1 = \frac{2}{1+\exp(-\sigma x)} - 1 \qquad (5)$$

$$= \frac{1-\exp(-\sigma x)}{1+\exp(-\sigma x)}.$$

$$g'(x) = \frac{\sigma}{2}[1+g(x)][1-g(x)]. \qquad (6)$$

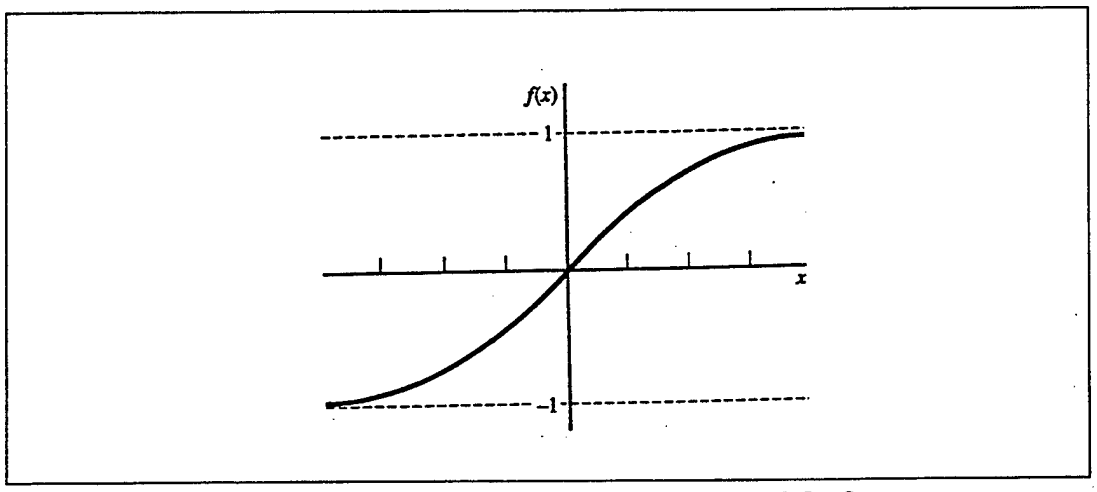

Figure 5. Bipolar Sigmoid. From Ref. [1].

The bipolar activation function is the function most commonly used with the *Adaline* device. Adaline was originally conceived by Widrow and Hoff initially called the ADAptive LInear NEuron but later became the ADAptive LINear Element. It is a feed forward net consisting of a single processing element (neuron). It receives input from several units. Figure 6 depicts the Adaline structure. It is almost identical to the simple (artificial) neuron previously described, but it has two modifications that make it the Adaline. The first is the addition of a connection with weight, w_0 which is called the **bias**

term. This term is a weight on a connection whose input value is always equal to one. The second modification is a bipolar activation function on the output.

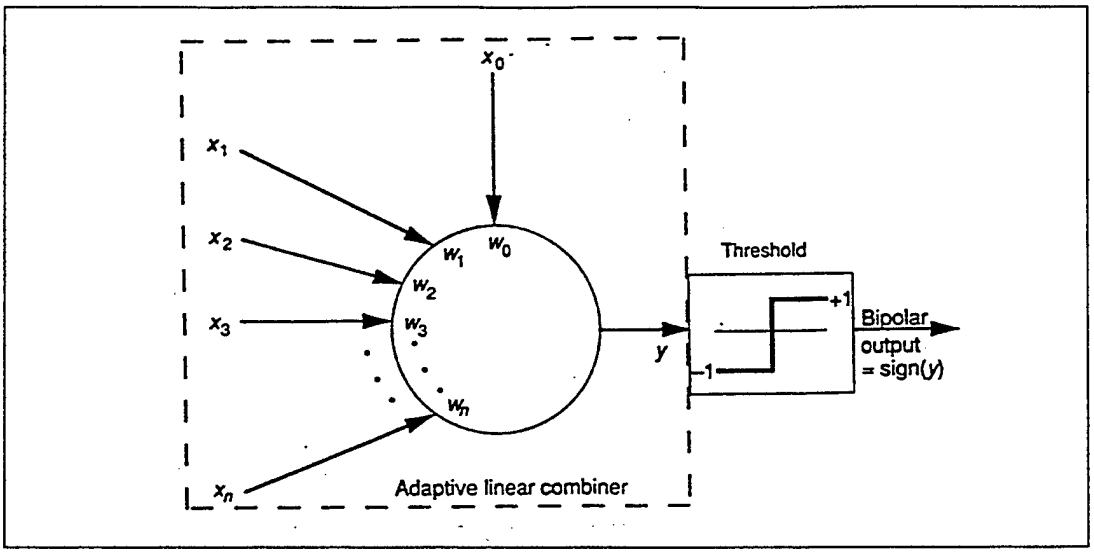

Figure 6. Adeline Structure. From Ref. [3].

There is a part of the Adaline called the **adaptive linear combiner** (ALC). It is pictured in Figure 6. If the output of the ALC is positive, the Adaline output is positive one. If the ALC output is negative, the Adaline output is negative one. The process done by the ALC is very similar to that of the processing element previously discussed in the first section. It produces a sum-of-products calculation using the input and weight vectors, and applies an output function to get a single output value thus giving us the following equation.

$$y = w_0 + \sum_{j=1}^{n} w_j x_j \tag{7}$$

In this equation, w_0 is the bias weight. If we set $x_0 = 1$, then we rewrite the previous equation as

$$y = \sum_{j=0}^{n} w_j x_j \tag{8}$$

or its corresponding vector notation,

$$y = \mathbf{w}^{\mathbf{T}}\mathbf{x}. \tag{9}$$

In this particular case, the output is the identity function as well as the activation function. This means that the output is the same as the activation, which is the same as the net input to the unit.

The Adaline can be trained using the **least mean squares** (LMS) rule also known as the delta rule. It is a method used in finding the desired weight vector. This rule minimizes the mean squared error between the activation and the target value. Because of this rule, the net can continue learning on all training patterns. It is mathematically described bellow as:

$$\mathbf{w}(t+1) = \mathbf{w}(t) + 2\mu\varepsilon_k \mathbf{x}_k \tag{10}$$

where: μ = the learning rate
ε_k = error value
$\mathbf{x}_k$= input vector.

If there is an instance where Adalines are combined so that outputs from some of them become inputs for others of them, then the net becomes multilayer or many Adalines. This net is known as Many ADAptive LInear NEurons or *Madaline*. Figure 7 below shows the structure of Madaline.

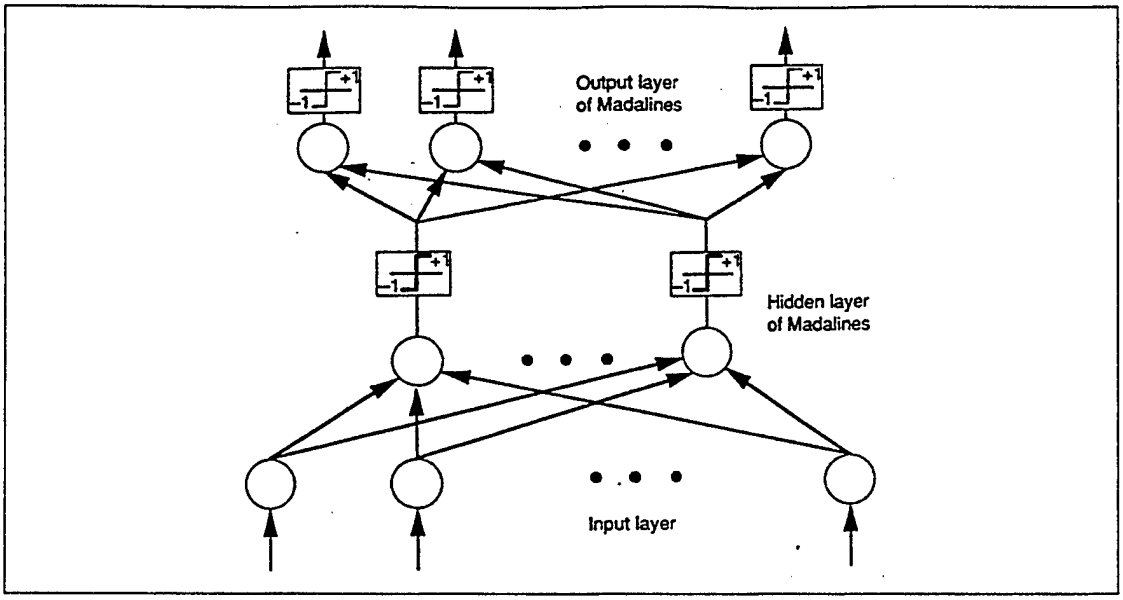

Figure 7. Madaline Structure. From Ref. [3].

There are three basic training algorithms for Madaline of which two will be discussed. The first is MADALINE RULE I (MRI). This algorithm was made such that only the weights for the hidden Adalines were adjusted. The second is MADALINE RULE II (MRII). This algorithm provides a method for adjusting all weights in the net. The aim of this algorithm is to cause as little disturbance as possible to the net at any step of the learning process, in order to cause as little "unlearning" of patterns for which the net had been previously trained. Freeman & Skapura [Ref. 3] embodied this principle in the following algorithm:

1. Apply a training vector to the inputs of the Madaline and propagate it through the output units.
2. Count the number of incorrect values in the output layer; call this number the error.
3. For all units on the outer layer,
   a. Select the first previously unselected node whose analog output is closest to zero. (This node is the node that can reverse its bipolar output with the least change in its weights-hence the term *minimum disturbance.*)

b. Change the weights on the selected unit such that the bipolar output of the unit changes.
c. Propagate the input vector forward from the inputs to the outputs once again.
d. If the weight change results in a reduction in the number of errors, accept the weight change; otherwise, restore the original weights.

4. Repeat step 3 for all layers except the input layer.
5. For all units on the outer layer,
   a. Select the previously unselected pair of units whose analog outputs are closest to zero.
   b. Apply a weight correction to both units, in order to change the bipolar output of each.
   c. Propagate the input vector forward from the inputs to the outputs.
   d. If the weight change results in a reduction in the number of errors, accept the weight change; otherwise, restore the original weights.
6. Repeat step 5 for all layers except the input layer.

Steps 5 and 6 are a modification of the algorithm attempting to modify pairs of units at the first layer after all of the individual modifications have been attempted. If necessary, the sequence can be repeated with triplets of units, or quadruplets of units, or even larger combinations, until satisfactory results are obtained.

The next rule that will be discussed is called the GENERALIZED DELTA RULE (GDR). This rule is used when performing **back propagation** . This network addresses problems requiring recognition of complex patterns and performing nontrivial mapping functions. Freeman & Skapura [Ref. 3] states that the network is embodied by the following description:

1. Apply an input vector to the network and calculate the corresponding output values.
2. Compare the actual outputs with the correct outputs and determine a measure of the error.

3. Determine in which direction (+ or -) to change each weight in order to reduce error.
4. Determine the amount by which to change each weight.
5. Apply the corrections to the weights.
6. Repeat items one through five with all the training vectors until the error for all vectors in the training set is reduced to an acceptable value.

B. HNET

1. Mathematical Concepts

AND [Ref. 4] states that HNeT theory is fundamentally different from the standard already discussed Artificial Neural System (ANS) theory. The neuron cell within the holographic/quantum neural model follows a non-connectionist model, which implies that learning and subsequent recall of stimulus-response associations are performed within single neuron cells. This is simply saying that the operational features exhibited by connectionist neural networks can be compressed into a single HNeT neuron cell thus improving the speed of learning and recall accuracy for one HNeT cell over traditional connectionist models.

The stimulus and response information or patterns may be represented by a set of values, each value residing within some analog range. These sets of values represent data values measured within an external environment with conditions such as pressure, brightness, temperature, etc. During stimulus-response learning, neural cells "map" one set of values (stimulus) to another set of values (responses). The neural cell subsequently operates in a manner that by exposing the cell to the stimulus, generation of the second field is produced (i.e. a response recall).

The mathematical basis for HNeT permits such stimulus-response patterns to be learned or "mapped" within a single matrix comprised of complex or real valued scalars. The number of scalars within that matrix can be no larger than the number of scalars within only one stimulus pattern.

The following linear model (11) using summation as well as inner product notation is an example of the operation that takes place within the holographic/quantum neuron cell where X_n is the cortical memory element.

$$X_n = \sum_t \lambda_{n,t}\gamma_t e^{i(\phi_t - \theta_{n,t})} \qquad (11)$$

$$= \langle S_n | R \rangle \text{ integrated over time } (t)$$

where in this case $S_n = \{\lambda_{n,t}e^{i\theta_{n,t}}, \ldots\}, R = \{\gamma_t e^{i\phi_t}, \ldots\}$

λ = the assigned confidence level for the stimulus
γ = the assigned confidence level for the response
θ = phase orientation for the stimulus
ϕ = phase orientation for the response

Once the initial stimulus-response has cycled through, the above operation superimposes or *enfolds* information pertaining to all of the stimulus-response patterns onto one scalar which is stored by the cortical memory element X_n. This process is characterized by the following equation (12) which introduces the response recall operation. In this case, the form is similar,

$$R' = \Omega_S \langle S^* | X \rangle \text{ summed over the cortical memory element } (n) \qquad (12)$$

however, the inner product is formed over the cortical memory elements (indexed by n) instead of time and R' is the response generated by HNeT. A new stimulus S^* is transformed through all of the stimulus-response memories enfolded within the cell's cortical memory elements and normalized by Ω_s which is generally some function of the stimulus field. This process just described is referred to as linear search which is one approach to pattern recognition.

In the more standard approach to pattern recognition (i.e. artificial neural networks), stimulus patterns "memory" are stored separately as a reference template. These reference templates are compared one at a time to a new incoming stimulus pattern during a response recall operation as we saw in the previous section. From this fact, AND [Ref. 4] implies that this method would require large amounts of memory, is computationally intensive, and rather limited in its generalization capabilities. The linear search performed by this process indicates only a level of closeness (i.e. pattern variance) for each of the stored reference templates against the new input. Recognition problems are encountered given slight deviations in the input pattern, these deviations most often incurring a large increase in the computed pattern variance.

HNeT, by comparison, performs a process that is similar in function, but not in origin to the pattern variance calculation. Instead of separately storing the reference template as ANN does, they are *enfolded* on to the same space. To explain enfolding, it is simply a process of superimposing large numbers of stimulus-response patterns upon the same set of scalars. Equation (12) best explains the mathematical process of this phenomenon. New stimulus-response patterns are generated, but instead of being placed

in a different storage location, they are placed back over the previously computed scalars. This reduces the storage requirements for all of the reference patterns to that required to store only one reference template. For the HNeT cell, storage requirements are decreased over the linear search method in direct proportion to the number of stimulus-response patterns learned. The time requirements to perform a response recall operation for an HNeT neural cell is also reduced relative to the proportion of the number of stimulus-response patterns learned.

Within the HNeT process, analog stimulus-response patterns are most often presented to the neuron cell as sets of complex scalars (values). Each complex scalar has a phase angle $e^{i\theta_j}$ which may represent some form of measurement (i.e. temperature, pressure, etc.) and magnitude λ_j which represents a degree of confidence in that measurement. This confidence (magnitude) regulates the influence of the input signal during both learning and recall operations. The confidence component may also exist within the generated response recall signal, in this case providing a measure of recognition that the cell has for the stimulus signal received, that is to say that the new (recycled) stimulus S^* will be regulated within that cell as well.

2. Representation of Mathematical Information

Current theories in computational neural dynamics follow from an idea known as the Hebb hypothesis. Kartalopoulos [Ref. 5] says that in 1949, Donald Hebb stated that when an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes place in firing it, some growth process or metabolic change takes place in one or

both cells such that A's efficiency, as one of the cells firing B, is increased. Thus, the synaptic strength (known as weight *w*) between cell A and cell B is modified according to the degree of correlated activity between input and output. This type of learning is called **Hebbian learning.** Most recent neural theory has expressed this in great detail leading to the current abundance of gradient-descent type algorithms, whose core aspects are largely built upon linear representations or real-valued inner product.

As previously mentioned, holographic/quantum model is different in that it uses both a complex scalar and nonlinear representation for stimulus information. At the most basic level, one element of information within the HNeT model is represented by a complex scalar. Complex scalars are a superset of real scalars, which is an indication that HNeT covers a wider range of numbers and has a computational advantage over more conventional neural networks. These scalars operate with two degrees of freedom represented by phase angle and magnitude. Illustrated in Figure 8 is information represented by the complex scalar where (θ) represents information pertaining to phase orientation, and (λ) represents magnitude or the confidence one has in that information (which can also be referred to as weighting). The magnitude component typically will vary between zero and one, although they may practicably extend over any range.

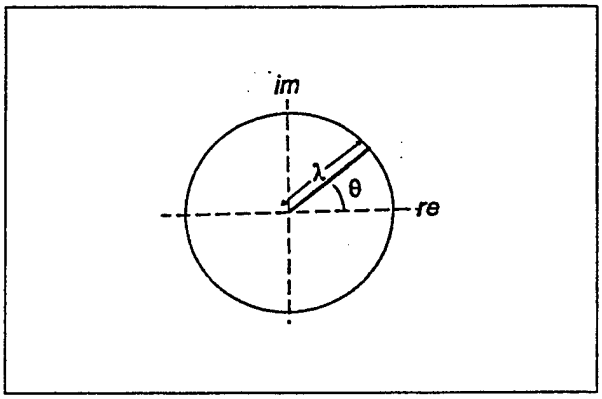

Figure 8. Illustration of the Information Element. From Ref.[4].

There are some aspects of the HNeT process that are similar to conventional processes in respects to *intracellular-transmission* of signals between biological neuron cells which give support to complex scalar representation. In Figure 9 (representation of a cortical cell) a large amount of input lines receive pulse modulated signals in which another responding pulse modulated signal delivers the cell's response output in a manner analogous to the axonal process. Although these signals may be interpreted in a variety of ways, the more predominant interpretation is that pulse modulated signals are translatable into real-valued scalar quantities.

The actual signal transmission characteristics, illustrated in Figure 9 suggest an equally feasible view point which is that pulsed signals could transmit complex scalar quantities via single line transmission. Frequency pulse modulation and amplitude modulation of a wave form is one direct means of analog transmission of complex scalars along a single of line transmission where frequency modulation could be interpreted as phase orientation, and amplitude modulation interpreted as the magnitude component of the complex scalar. However, more important are the operational features observed

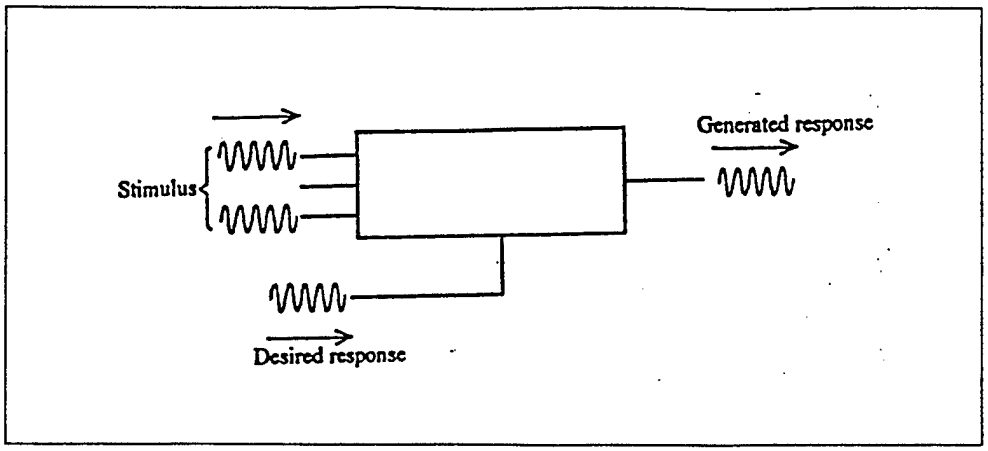

Figure 9. Block Diagram of a Cortical Cell. From Ref. [4].

within single cell structures when information is both represented and transmitted as complex scalars as well as other aspects of the HNeT neural process.

The following form is the basic unit of information within the HNeT system:

$$\lambda e^{i\theta} \tag{13}$$

Before moving on, a point to make is that in cases where Fourier conversion is applied as a preprocessing operation, the phase angle orientation represents the phase shift for that frequency harmonic, and the magnitude represents the power of the harmonic. Similarly, low power stimulus inputs have less influence over the stimulus-response mappings learned by the cell, as well as less influence in the generation of the response signal during recall.

For the following discussion, let's assume that all stimulus and response scalars have unit magnitude for simplification. Phase angle differences are the source for building up the memory generated within the cell for such stimulus-response associations. For instance, one element of a stimulus may be represented by phase orientation $e^{i\theta_j}$, and

the associated response by orientation $e^{i\phi_k}$. The quantity generated mathematically for associating stimulus (s) element j to response (r) element k is noted as follows:

$$\overline{s_j} \cdot r_k \tag{14}$$

where: $s_j = \lambda_j e^{i\theta_j}$

$r_k = \gamma_k e^{i\phi_k}$

This produces the phase angle difference $e^{i\theta diff}$ where:

$$\theta_{diff} = \phi_k - \theta_j \tag{15}$$

The quantity $e^{i\theta diff}$ is the representation of a fundamental "quanta" of information, or many measured amounts of information which stores the portion of an associative mapping that is learned by one cortical memory element for one pattern, which in turn, connects one element of the stimulus to one element of the response. A very important aspect to make note of is that learning capacity is primarily based on the number of cortical memory elements (i.e. complex scalars), and this learning capacity is directly proportional to the number of cortical memory elements within the cell. To clarify further, one cortical memory element is physically represented by one complex valued scalar. An example would be that if a cell has 100 memory elements, it is capable of learning and storing 100 stimulus response associations.

The HNeT cell also has the ability to respond to the space of unknown or unlearned stimuli, commonly referred to as the "test set" in the *Supervised Learning Platform* within the system, through generation of low magnitude (or weight/confidence) in the generated response. There are fundamental properties that exist within the complex

number domain that support the concept of enfolding of information, and the corresponding increase in the density of information within cortical memory elements. Let (A) be defined as any point in the complex plane. This point may also describe any path from the complex origin to point A giving us the following mathematical equality:

$$A = \lambda e^{i\theta} = \sum_{j=1}^{M} \alpha_j e^{i\theta_j} \tag{16}$$

Simply put, the above scalar quantity defines any path within the set of all possible paths leading from the origin to point A. Figure 10 illustrates the above process. Within one cortical memory element, each of the component scalars which define the path leading to point A represent one association "quanta" that has been learned.

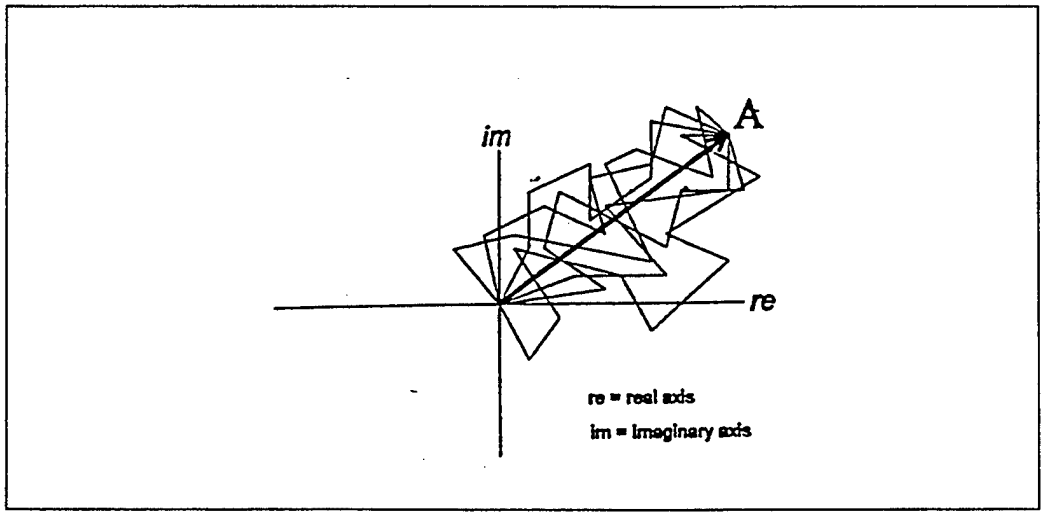

Figure 10. Multiple Pathways Defining a Complex Scalar. From Ref.[4].

3. Conversion Formats

We discussed previously that HNeT's stimulus [S] and response[R] sets are represented by vectors comprised of complex scalars. Using the complex exponential, these sets are as follows:

$$[S]=\{\lambda_1 e^{i\theta_1},\lambda_2 e^{i\theta_2},\lambda_3 e^{i\theta_3},\lambda_4 e^{i\theta_4},\ldots,\lambda_N e^{i\theta_M}\} \tag{17}$$

and

$$[R]=\{\gamma_1 e^{i\phi_1},\gamma_2 e^{i\phi_2},\gamma_3 e^{i\phi_3},\gamma_4 e^{i\phi_4},\ldots,\gamma_M e^{i\phi_N}\} \tag{18}$$

External stimuli and response actions are predominantly represented by real numbers, therefore HNeT is required to convert the external real numbers to internal complex values. A generic representation of this mapping is as follows:

$$s_j \rightarrow \lambda_j e^{i\theta_j} \tag{19}$$

The **sigmoidal conversion** is one manner in which this conversion may be applied. The following is a mathematical representation of the sigmoidal process:

$$s_j \rightarrow \lambda_j e^{i\theta_j}$$

$$\text{where: } \theta_j = 2\pi\left(1+e^{\frac{\mu-s_k}{\alpha}}\right)^{-1} \tag{20}$$

μ = mean of distribution over input s; k=1 to N
σ = variance of distribution of s
λ_j = the assigned confidence level.

This representation is very similar to the sigmoidal process discussed in the previous section, however, there are some changes to compensate for phase orientation and HNeT's complex process.

The sigmoidal function as described above maps real valued signals within an external environment to sets of phase angle orientations in most cases having unit magnitude. These complex scalars are then read directly into the HNeT cell. Figure 11 illustrates the sigmoid conversion format. In the illustration, notice that the real number domain extends along the horizontal axis and varies from $-\infty \Rightarrow +\infty$. The phase axis extends along the vertical and is bounded by 0 and 2π.

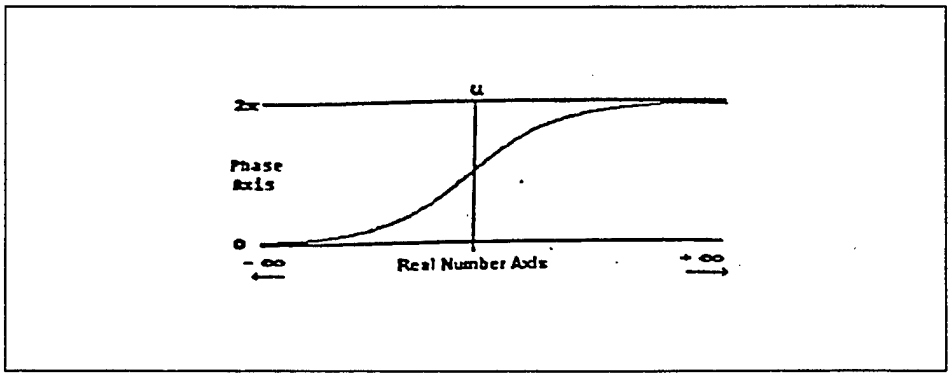

Figure 11. Sigmoidal Conversion. From Ref. [4].

This particular format allows one to assign some value to the complex magnitude (λ_j) or confidence level associated with each real valued input signal. This is done on the **SL Platform** by the use of the **Parameter Search Settings/Learning Rate**. This rate ranges between zero and one. Assigning confidence levels to input signals gives it control over its level of influence during both learning and recall operations. We can also describe this as phase angle information being "weighted" in proportion to its associated magnitude. For instance, a stimulus element with a magnitude of 0.0 assigned will have no influence in the learning or recall of a stimulus-response pattern. However if a confidence level of

1.0 is assigned, an influence of at least equal weighting will be established for all other values contained within the stimulus input array.

The next conversion we will discuss is the **histogram based conversion**. This conversion is used for data that is derived from artificial sources. In other words, data is not derived from environmental sources or human factors thus the distribution may vary considerably from a Gausian or normal distribution. In order to achieve a reasonable level of symmetry, one may be required to build a custom conversion. Constructing a distribution histogram will aid in determining a measure of the distribution profile that records and sorts bins appropriately centered about the mean of the distribution. Once the distribution profile is determined, a polynomial must be obtained to arrive at a best fit for the distribution profile. One fit technique that is effective is the Chebychev function or polynomial given as follows:

$$C_n(X) = \cos(n \arccos x) \tag{21}$$

This may be combined with trigonometric identities to yield:

$$C_{n+1}(x) = 2xC_n(x) - C_{n-1}(x) \tag{22}$$
$$\text{for } n \geq 1$$

It is important to note that each polynomial has n zeros in the interval [-1,1], and they are located at the points:

$$x = \cos\left(\frac{\pi(k-.5)}{n}\right) \tag{23}$$

$$\text{for } k = 1, \ldots, n$$

The distribution density histogram must be selected over an appropriate range and centered about the points indicated in (23). The input range is from $-\infty \Rightarrow +\infty$, scaling this to [-1,1] via the arctan function. Once this is done, distribution bins are set for optimal placement and data is place such that the distribution format is symmetrically formed.

The **Fourier conversion** is a method of calculation that performs a one to one translation between a time domain series and a frequency domain series. The benefit of the Fourier transform is that it transforms real valued *vectors* or surfaces directly into complex value sets. It is important to note this method helps eliminate discontinuity within problems that may not be solved by the faster conversion methods previously mentioned. This conversion allows for better generalization because data often assumes a more orthogonal (i.e. symmetrical) state when expressed in frequency domain, and data reduction is possible at the input. The transform is described mathematically below:

$$H(f) = \frac{1}{\sqrt{n}} \sum_{t=0}^{n-1} h(t) e^{i\frac{2\pi f}{n}} \tag{24}$$

The coefficients *H(f)* are fed into the cortical cell at which time stimulus to response mappings are performed within the complex domain. When values are read back out from the cell, the inverse Fourier transform is performed given as such:

$$h(t) = \frac{1}{\sqrt{n}} \sum_{f=0}^{n-1} H(f) e^{i\frac{2\pi f}{n}} \tag{25}$$

An illustration of the Fourier conversion is provided below.

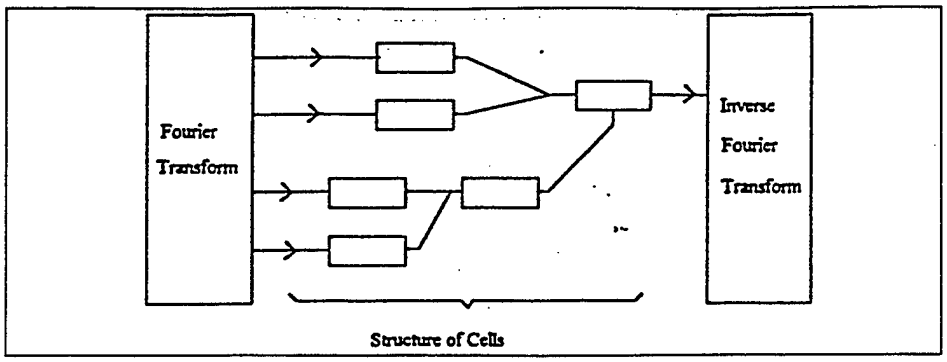

Figure 12. Fourier Conversion on Data. From Ref. [4].

4. Generalization

Within the HNeT model, generalization refers to aspects concerning the topology or characteristics relating to the interpolation of trained stimulus-response pattern sets within an HNeT neural cell. A basic depiction of the generalization properties within the HNeT system is that they form a nonlinear complex based polynomial in which the cell similarly behaves. That is to say that the function interpolates smoothly between the fit points (i.e. pattern sets used to train). For example, if we trained data and the polynomial fit is similar to that of a fourth order product of terms, as seen in Figure 13

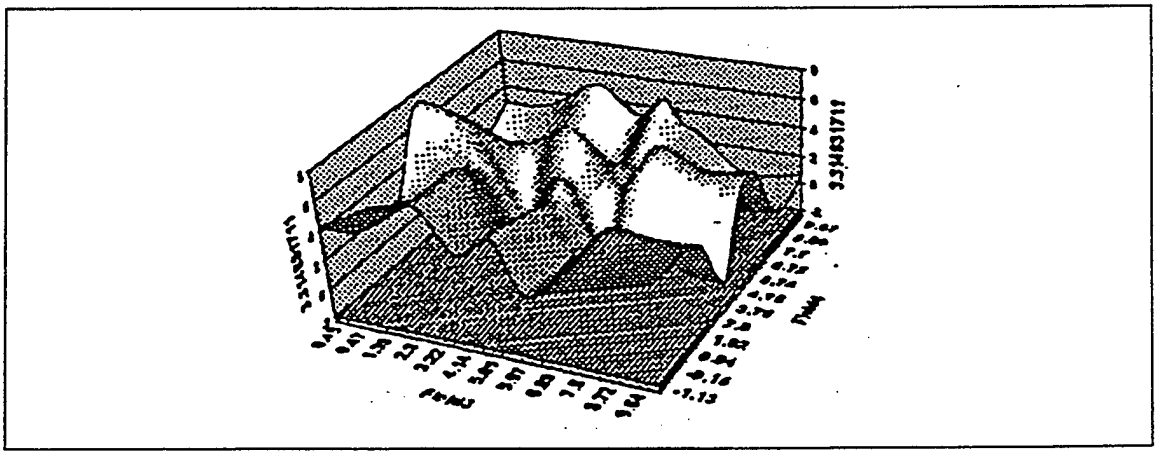

Figure 13. Topology of Cell of Two Input Dimensions (Fourth Order). From Ref. [4].

(trained on 100 patterns), the topology would be somewhat smoother than that of fifth order terms. For comparison, in Figure 14 below (trained on 500 patterns) , is a topology created when using sixth order terms. One sees that over a time scale, the system creates a more complex topology in that the greater slopes and narrower summit provides for a more complex topology which accommodates a greater stimulus-response pattern storage density.

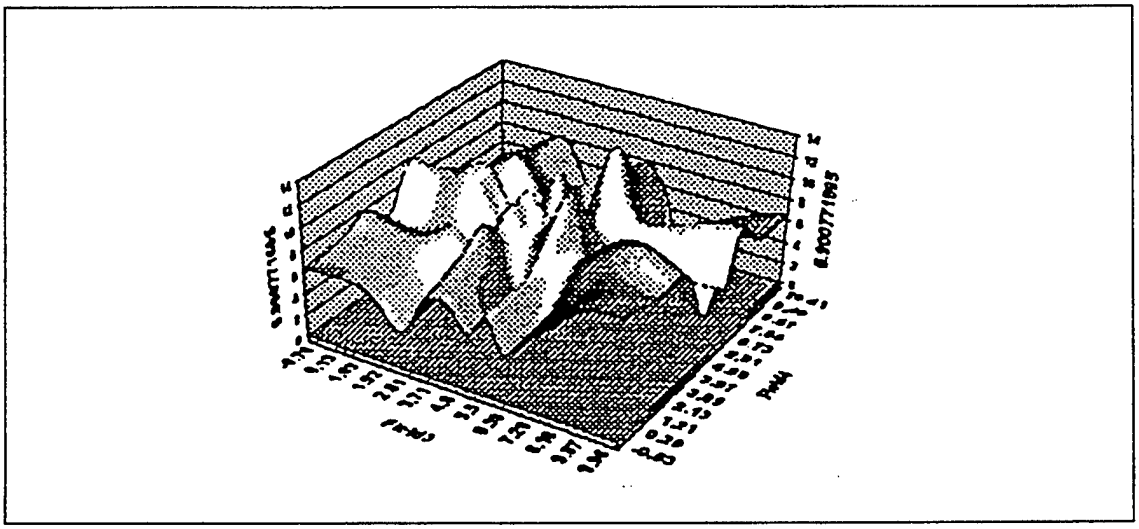

Figure 14. Topology of Cell of Two Input Dimensions (Sixth Order). From Ref. [4].

THIS PAGE INTENTIONALLY LEFT BLANK

III. SIMPLE FUNCTION EVALUATION

Now that a general description of the HNeT process has been given, we will begin to investigate its properties on simple mathematical functions. Random numbers will be generated and used as inputs to each function. The random inputs, which we will call $\mathbf{X}_1$ (stimulus), and data produced by these functions using the stimulus, $\mathbf{X}_2$ (response), will be applied to HNeT (e.g. for $f(x) = \cos(x)$, we have $\mathbf{X}_2 = \cos(\mathbf{X}_1)$). The error produced by the system will help to evaluate some of the properties of HNeT.

A. EVALUATING TRIGONOMETRIC FUNCTIONS

1. Cosine and Cos(1/x)

The first set of tests were conducted on various trigonometric functions to establish whether HNeT could learn to emulate the most basic functions in mathematics. This is necessary because most technological applications require the use of various functions such as cosine, tangent and many others.

The cosine function was first function that was tested. This function was tested using 300 values of generated random input data, $\mathbf{X}_1$, and the corresponding outputs $\mathbf{X}_2 = \cos(\mathbf{X}_1)$. For the purpose of our test, we will split this data into training and testing sets with 80% in the training set and 20% in the testing set (240 and 60 records respectively). This test was conducted with 4 *epochs* (training exposures per cycle) and a learning rate (magnitude/confidence) of .25. The results of the test yield a mean absolute error of .0061

for the training set and .0075 for the test set. This was found in 5 cycles using 777 memory elements out of a total 1000. This is expected because the training set has more data available so that HNeT can learn to recognize more of the presented cases. The test set, however, uses fewer cases but still has a reasonable error, which is a good indication.

If we continue to observe the results, we will find that as the parameter search continues to cycle, the error for the test and training results increase. This is indicated by the error graph in Figure 15. The results show that at the 85th cycle, the test error (error #1) is .1091, and the training error (error #2) is .1157. This was done in 23 memory

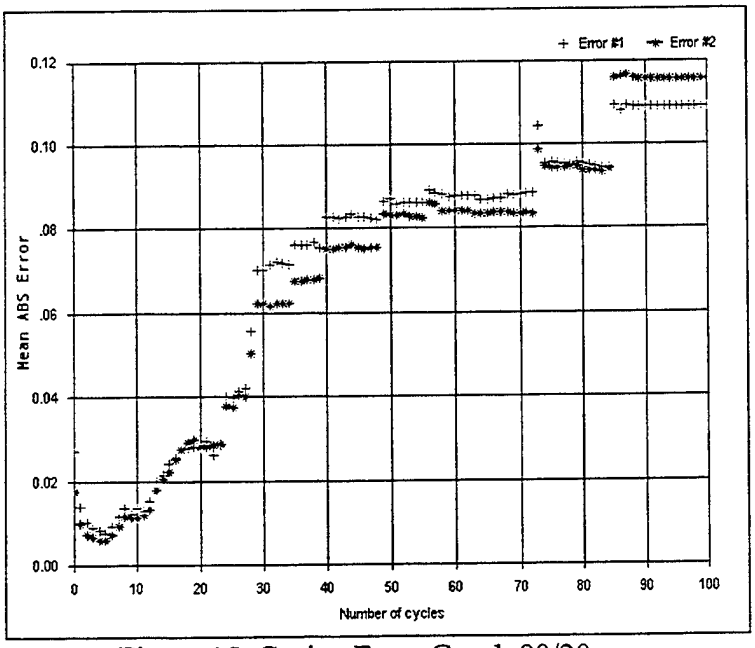

Figure 15. Cosine Error Graph 80/20.

elements out of 1000. The explanation for this is that as HNeT cycles through the data, the process performs a function analogous to the pattern variance calculation, however, the reference templates are *enfolded* on to the same storage space. As the

storage requirement reduces for all of the reference patterns to only that amount of memory required to store a single reference template, the average error on response recall gradually increases.

A second evaluation was performed on the same set of data, this time splitting the data into training and test sets of equal size (150 records each). This time, the results yielded a mean absolute error of .0290 for the test set and .0257 for the training set. This was executed in 16 cycles using only 470 memory elements out of 1000. The resulting test set error was much closer to the training set error. Due to increased data of the test set, its pattern recognition capability increased. Overall, both sets of error increased which is believed to be caused by a decrease in training records evaluated for the training set, and in the test set, HNeT has incorrectly used data as the enfolding process took place thereby limiting its generalizing capability. The fact that fewer memory elements were used limited generalizing capabilities causing increased error for both sets. As HNeT continued to run through its parameter search, its minimum memory elements were 20, yielding an error of .1148 for the test set and .1153 for the training set at the 90^{th} cycle. This is similar to the outcome in the previous test, which shows for these two evaluations for this particular function, there is a tendency to converge to the same error near the same cycle. Figure 16 captures the error data for the second test in graph form.

We will next introduce the trigonometric function, *Cos(1/x),* for further investigation of the properties of HNeT. This function is important to test because of its unbounbed variation. That is, as x approaches zero, the oscillations become extremely

rapid, indeed the function oscillates between minus one and one an infinite number of times in any open interval (0, ε) for any ε >0. When evaluating this function, we took a similar approach to that of the *cos(x)* function. Initially, we started with 200 data values with 80% of the data to be evaluated as the training set and 20% of the data for the tested set. As before, we assigned $\mathbf{X_1}$ (the generated random input value) as the stimulus and $\mathbf{X_2}$ (the corresponding output value) as the response. This resulted in a significant increase in error. The test set data resulted in an error of .4587, and the training set resulted in an error of .0863 in the 9th cycle utilizing 635 memory elements out of a possible 1000. Considering the range of the numerical value of the data, this error is considerably large. The data for this test set range from roughly -2.0 to +2.0 for the stimulus value and –1.0 to +1.0 for the

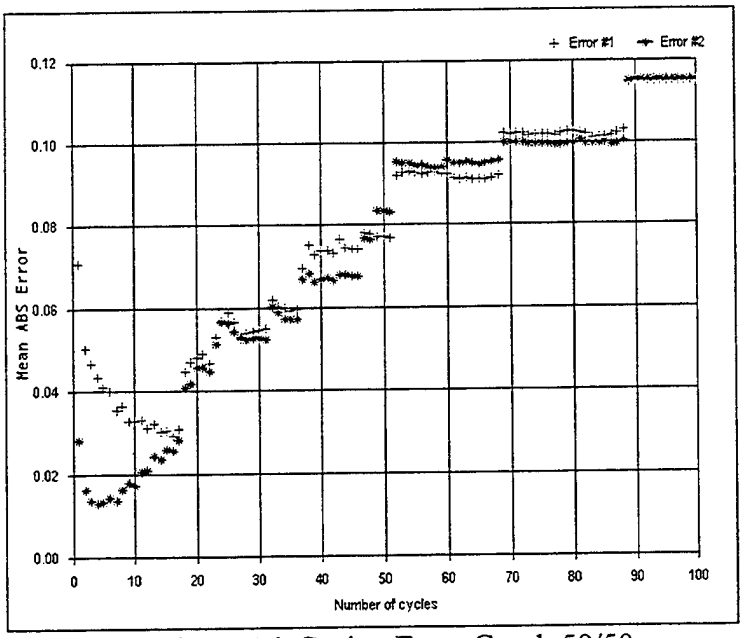

Figure 16. Cosine Error Graph 50/50.

response. In an attempt to decrease the error, we increased the number of *epochs* (the number of training exposures in one cycle) from four to 15 and the learning rate (magnitude/confidence) remained the same at .25. There was a slight decrease in both errors. The decrease was due to increased training exposures, giving HNeT more chances to learn the data. The training set error result was .0693, and the test set error was .4058 in the 6^{th} cycle, however, the result is still rather large. HNeT is expected to have difficulty with this function due to its unbounded variation. The system can not keep up with the rapid fluctuations of this function. The error graph in Figure 17 gives visual evidence of HNeT's difficulty in learning this data, as shown by the test set error.

To determine if the tests of the function are valid for both the training set and the test set, there is a procedure in HNeT known as *validation*. In this procedure, HNeT learns to emulate the entire set of data, and generates the mean absolute error for the entire data set as a whole. It also gives a pictorial view of the function graph on the **Supervised Learning Platform** as a whole verses individually by sets. The validation procedure was conducted for the previous test and the resulting combined mean absolute error was .1369. We will continue to investigate the properties of HNeT using another trigonometric function to determine its capabilities relative to discontinuity.

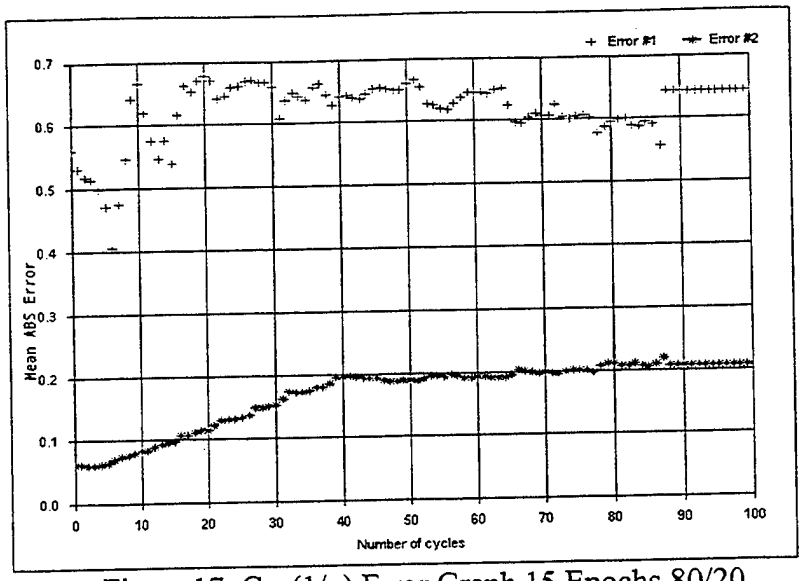

Figure 17. Cos(1/x) Error Graph 15 Epochs 80/20.

2. Tangent

The *tangent* function unlike the cosine function is one that is not defined for all x. Since $\tan x = \sin x / \cos x$, the tangent function is undefined whenever $\cos x = 0$; this occurs when x is an odd integer multiple of $\pi/2$, that is when

$$x = \pm\frac{\pi}{2}, \pm\frac{3\pi}{2}, \pm\frac{5\pi}{2}, \ldots\ldots$$

Table 1 shows some values generated by the tangent function from randomly generated numbers. Notice that as x approaches $\pi/2 = 1.571$, $\tan x$ starts to increase sharply. This is because $\sin x$ is approaching one, while $\cos x$ is approaching zero. From this data we will investigate how HNeT responds with the sharp increases near the singularities $\pm\pi/2$.

x	Tan x
1.191	2.504
1.254	3.050
1.290	3.470
1.339	4.247
1.415	6.372
1.443	7.813
1.472	10.13
1.478	10.85
1.488	12.12
1.535	28.04
1.553	56.96
1.592	–45.15
1.623	–18.93
1.692	-8.180
1.864	-3.305
1.957	-2.456
2.112	-1.663
2.183	-1.423

Table 1. Tangent Values for x.

For the testing of the tangent function, we generated 200 data values of $\mathbf{X}_1$ (stimulus) and $\mathbf{X}_2$ (response). We will use 80% of the values for the training set and 20% of the values for the test set. From the first test, using the default setting of 4 epochs and learning rate of .25, the training set error was 1.424 and the test set error was 5.682 in the 15^{th} cycle with 904 memory elements used. In an attempt to decrease error, the number of epochs and learning rate were adjusted until the best results were found. After numerous trials, the error remained relatively unchanged, however, the memory elements decreased almost two fold. By increasing the epochs to 15 and slowing the learning rate to .01, HNeT was able to use less memory to accomplish these results. The error for the

training set was 1.619 and the test set was 5.695 in the 15th cycle, using 494 memory elements. The following table is an example of the results near the singularities. Notice that HNeT didn't respond very well near the singularities because of the sharp

X_1	tan x	HNeT	Error
1.415	6.372	8.319	-1.947
1.443	7.813	8.617	-0.803
1.472	10.13	8.852	1.285
1.478	10.85	8.894	1.958
1.488	12.12	8.949	3.172
1.535	28.04	9.084	18.95
1.553	56.96	9.074	47.88
1.592	-45.15	8.931	-54.08
1.623	-18.93	8.711	-27.64
1.692	-8.180	7.912	-16.09
1.864	-3.305	4.838	-8.144
1.957	-2.456	3.052	-5.508
2.112	-1.663	0.455	-2.118
2.183	-1.423	-0.524	-0.899

Table 2. HNeT Results of the Tangent Function.

increase in the values. Even as a result of the decrease in the learning rate, HNeT still produced high error. The pattern for this function is one with predominantly low values for each stimulus. As HNeT recognizes the pattern, it was not able to adjust to the sharp increases in the values. Further examination is required to adjust for the sharp increases of this function.

For the purpose of decreasing the error in HNeT, we will treat the values near the singularities as outliers and eliminate them. After eliminating values, the remaining data

gives 172 stimulus and response values to test. Using the same 80/20 ratio as before, the results of the evaluation yielded an error of .0433 for the training set and .5844 for the test set in the 26th cycle utilizing 271 memory elements. The number of epochs remained the same, however, HNeT responded better with an increase in the learning rate to .50.

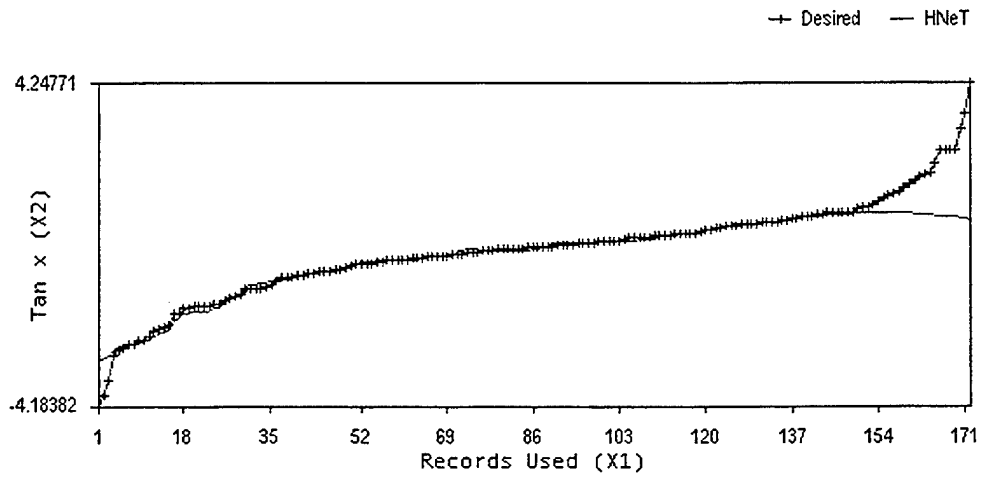

Figure 18. Validation Function Graph With Outliers Eliminated.

When adjusting the ratio to 50/50 and increasing the epochs to 30, HNeT decreased its error even further with a training set error of .0304 and test set error of .3878 in the 17th cycle, utilizing only 192 memory elements. The validation for this data has an error of .2091. Below is the function graph of the data with outliers removed. Notice, the right tail of the function shows that HNeT (test set) doesn't respond well as the data increases, which is the cause for the majority of the error.

There are other functions that we will investigate however, there is some limited evidence that HNeT has problems in reproducing functions with sharp increases or

decreases in data that produces sharp curves. In the following evaluations, we will give examples of HNeT's capability in evaluating functions whose data produce curves that are smoother and more gradual (up or down).

B. EVALUATING NATURAL LOGARITHMIC, EXPONENTIAL, AND POLYNOMIAL FUNCTIONS

1. Natural Logarithm

The natural log function ($\ln x$) is one that can be described as a function that has a smooth curve with a gradual increase as x increases. Some properties to note about this function are that:

$$\begin{array}{lll} \ln x > 0 & \text{if} & x > 1 \\ \ln x < 0 & \text{if} & 0 < x < 1 \\ \ln x = 0 & \text{if} & x = 1. \end{array}$$

Furthermore, this function is undefined for $x \leq 0$.

For the purpose of our evaluation of HNeT, we will use 300 data values. The results of the test produced very low error for this function. The training set error was .0052 and the test set error was .0071 in the 4^{th} cycle. These results were accomplished with 15 epochs using 817 memory elements. The error graph (Figure 19)

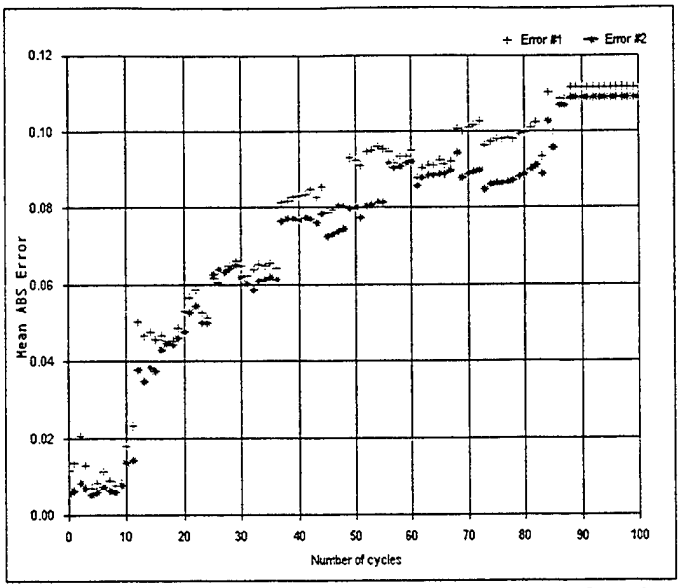

Figure 19. Natural Log Function Error Graph.

shows how well the test set remains close to the training set despite the lower amounts of data (60 out of 300) to evaluate. HNeT responds very well to the log function. However, we will continue to examine other functions to further investigate its properties.

2. Exponential Function

The exponential function (e^x) is another function that produces a smooth curve. The exponential function is continuous for all values of x. In our case, $x > 0$ and has the following properties:

$e^x > 0$ for all x
limit e^x as $x \to +\infty = +\infty$
limit e^x as $x \to -\infty = 0$.

The function was also tested using 300 data values with the same format as the natural log function. Again, the results of this function produced very low error for both sets of data. During this particular evaluation, the memory elements were considerably lower than those of the log function. The memory elements used were 384 out of 1000 total. This was accomplished with 15 epochs. The error was .0013 for the training set and .0014 for the test set in the 19^{th} cycle. Numerous evaluations for this function were conducted with various epoch changes as well as learning rate changes. For each evaluation, the error produced was relatively low with the currently mentioned evaluation having the lowest error.

As HNeT tried to emulate this function and the enfolding process took place, the error for the test set began to grow at a slower rate than that of the training set around the 27^{th} cycle indicating that the test set was doing a better job of reproducing the function than the training set. This trend is shown in Figure 20.

HNeT continues to show that for functions with smooth curves with gradually increasing data, it performs rather well. The next function to be introduced to the system is the polynomial function. With this function, we will continue to examine HNeT's properties.

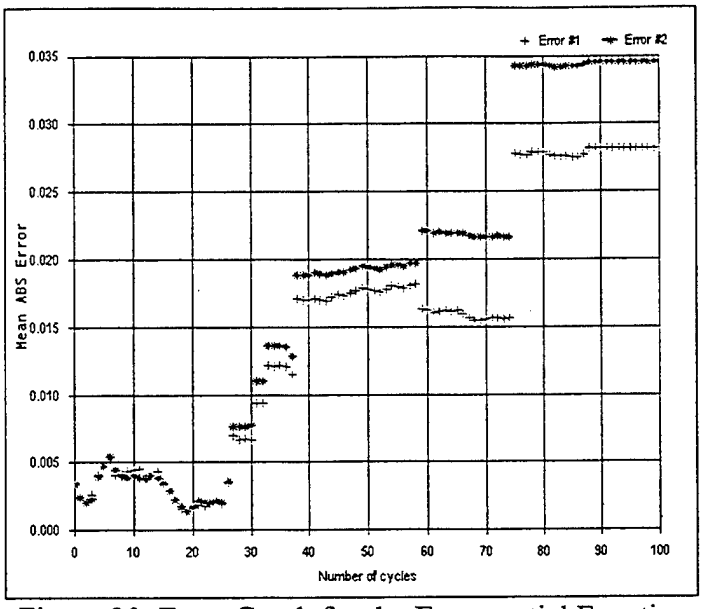

Figure 20. Error Graph for the Exponential Function.

3. Polynomial

Given $f(x) = x^3 + 4x^2 - 7x - 10$, this randomly selected polynomial is a function that is continuous for all x. In other words, the domain of the polynomial is $(-\infty,+\infty)$. This polynomial will be evaluated in the HNeT system to examine HNeT's capability with this type of function.

The first tests was conducted with 300 data values, again using the same format as the exponential function. We began with the default setting of 4 epochs and a learning rate of .25. The results for this evaluation yielded an error of .0059 for the test set and .0073 for the training set in the 10^{th} cycle. The memory elements used were 285 out of 1000. Continuing on with more evaluations, the best results yielded a test set error of .0018 and a training set error of .0026 in the 12^{th} cycle. This was accomplished with 15

epochs and 546 memory elements. For this evaluation the test set error was lower than the training set for all evaluations conducted. The error graph for this function was very similar to the error graph of the exponential function. This is an indication of how well HNeT responds to these functions. It responded to untrained data better indicating that HNeT was able to easily learn and recognize smooth and gradually increasing data .

In the next section, HNeT will be examined on truly random data. That is, there will be no associated function to determine a particular pattern to follow.

IV. RANDOM DATA EVALUATION

The California Lottery began on October 14, 1986. In its beginning, the draw was based on picking six numbers from 1 to 49. However, the numbers increased from 49 to 53 with six picks on June 21, 1990. On December 15, 1991, the draw was changed and is known as the "Super Lotto". To this current day, the numbers drawn range from 1 to 51 with six picks.

The first set of tests are based on the collection of California Lottery results. This is a truly random data set that is publicly available via the Internet. It is an ordered collection of the results of every draw of the California lottery from October 14, 1986 up to the current drawing. Each record consists of 6 ordered distinct integers whose values range from 1 to 51. Since the drawing is conducted physically (with numbered balls), the data are truly random representatives of their particular distribution. This is crucial for testing neural network software since pseudo-random sequences often have underlying structure that can produce statistically abnormal results in such tests. Initial test were conducted with all drawings dating back to October 14, 1986, however, since there were differences in the range of numbers drawn, the results were erroneous. For the purpose of our current tests, we trimmed the data set to those dating back to December 15, 1991. We will use data from 868 recent draws that will be split into training and testing sets of equal size (434 records each).

The values of the 6 numbers in the drawing are random variables; we will call them $\mathbf{X_1}$, $\mathbf{X_2}$, $\mathbf{X_3}$, $\mathbf{X_4}$, $\mathbf{X_5}$, and $\mathbf{X_6}$ respectively. Note that the knowledge of the value of some $\mathbf{X_i}$ gives information that can be used to narrow down the possible values of the others. For instance, if we are given that $\mathbf{X_5} = 40$ then we can infer that $\mathbf{X_6}$ must take on a value between 41 and 51 with each possible value being equally likely. This is simply a conditional probability, in particular $\Pr\{ \mathbf{X_6} = x \mid \mathbf{X_5} = y\} = 1/(51-y)$ if $x>y$ and zero otherwise.

In the first test, we attempt to train HNeT to guess $\mathbf{X_6}$ given the value of $\mathbf{X_5}$. This is an interesting test probabilistically since $\Pr\{ \mathbf{X_6} \mid \mathbf{X_1}, \mathbf{X_2}, \mathbf{X_3}, \mathbf{X_4}, \mathbf{X_5}\} = \Pr\{ \mathbf{X_6} \mid \mathbf{X_5}\}$, that is, $\mathbf{X_6}$ is conditionally independent of $\mathbf{X_1}$ through $\mathbf{X_4}$ if you are given $\mathbf{X_5}$ (this is similar to a Markov property in a stochastic process).

The best linear unbiased estimate (BLUE) of $\mathbf{X_6}$ given that $\mathbf{X_5} = y$ is $\mathbf{X^*} = (52 + y)/2$. In Table 3, we show the $\mathbf{X^*}$ as well as the estimate generated by HNeT, denoted $\mathbf{X^H}$, for all possible values of $\mathbf{X_5}$. Note that in this table HNeT was trained using only the values of $\mathbf{X_5}$ and $\mathbf{X_6}$. From the data, it is apparent that the response generated by HNeT attempts to approach the optimal response although it is far from it in some areas. The mean absolute error over the training set was 3.320 and 3.611 over the test set. HNeT found this solution in one cycle using 100 memory elements out of a possible 100.

Another test was conducted using $\mathbf{X_4}$ as the stimulus and $\mathbf{X_6}$ as the response. The results of this test yielded a higher error value for both sets. The mean absolute error for

X_5	X^*	X^H	Error
5	28.50	34.37	-5.87
6	29.00	30.12	-1.12
7	29.50	30.43	-0.93
8	30.00	36.27	-6.27
9	30.50	29.01	1.49
10	31.00	31.88	-0.88
11	31.50	31.05	0.45
12	32.00	35.95	-3.95
13	32.50	34.23	-1.73
14	33.00	34.28	-1.28
15	33.50	33.55	-0.05
16	34.00	33.51	0.49
17	34.50	36.04	-1.54
18	35.00	36.33	-1.33
19	35.50	37.10	-1.60
20	36.00	44.44	-8.44
21	36.50	36.76	-0.26
22	37.00	34.12	2.88
23	37.50	36.74	0.76
24	38.00	37.36	0.64
25	38.50	38.71	-0.21
26	39.00	37.27	1.73
27	39.50	37.59	1.91
28	40.00	32.70	7.30
29	40.50	40.61	-0.11
30	41.00	40.44	0.56
31	41.50	41.33	0.17
32	42.00	42.53	-0.53
33	42.50	42.28	0.22
34	43.00	40.71	2.29
35	43.50	45.15	-1.65
36	44.00	43.80	0.20
37	44.50	39.19	5.31
38	45.00	43.70	1.30
39	45.50	39.77	5.73
40	46.00	45.49	0.51
41	46.50	43.93	2.57
42	47.00	44.60	2.40
43	47.50	47.61	-0.11
44	48.00	45.16	2.84
45	48.50	40.63	7.87
46	49.00	48.20	0.80
47	49.50	41.59	7.91
48	50.00	49.67	0.33
49	50.50	49.44	1.06
50	51.00	51.31	-0.31

Table 3. BLUE Comparison. After Ref. [6].

the training set is 4.060 and 4.353 for the test set. However, this was found using 24 memory elements in 37 cycles, much less memory than the previous test. The error was expected to be higher because as stated before, if we are given that $X_4 = 30$ and $X_5 = 40$, X_6 must take on a value between 41 and 51 with each possible value being equally likely, rendering X_4 statistically unimportant.

For the third part of this test we let HNeT train using the full data. That is, attempt to guess the value of X_6 given the values of X_1, X_2, X_3, X_4, and X_5 . When presented with this new task we get very interesting results. The resulting artificial neural network (ANN) had a mean absolute error over the training set of only 3.128 but 3.836 over the test set. This was found in 6 cycles and used 77 memory elements. The interesting phenomenon here is that the error improves over the training set because there is more data available so that HNeT can learn to recognize more of the presented cases. However, the error is worse over the tests set. This happens because in this test HNeT has incorrectly used data in constructing the ANN that is statistically unimportant. In the following graphs we will discuss the results based on generalization.

The first graph shown (Figure 21) is a representation of the response (Y axis) verses X_4 (X axis) and X_5 (Z axis). Notice that the gradients are not as steep at most points but steep for some. This is a representation of HNeT responding well to most stimuli because X_5 and X_4 have data that is relatively close in value to the response data X_6. However, there are also steep gradients that are caused by low values in X_4 and X_5. For the most part, the graph shows that the stimuli interpolates much smoother than that

of the lower values $\mathbf{X_1}$. The second graph (Figure 22) shows the response verses $\mathbf{X_1}$ (Z axis) and $\mathbf{X_5}$ (X axis). Since most values in $\mathbf{X_1}$ represent values between one and ten, and $\Pr\{ \mathbf{X_6} \mid \mathbf{X_1}, \mathbf{X_2}, \mathbf{X_3}, \mathbf{X_4}, \mathbf{X_5}\} = \Pr\{ \mathbf{X_6} \mid \mathbf{X_5}\}$, $\mathbf{X_1}$ has little or no effect on the outcome of the response. This is evident in the graph that shows the Z axis being relatively flat, whereas the X axis has a much more linear fit.

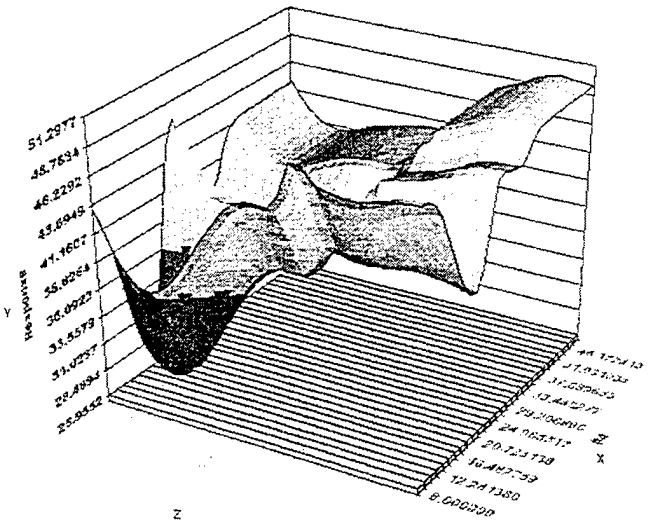

Figure 21. Topology of Cell Input $\mathbf{X_4}$ and $\mathbf{X_5}$.

The test conducted with $\mathbf{X_1}$ as the stimulus and $\mathbf{X_6}$ as the response backs up the graphical evidence that $\mathbf{X_1}$ is statistically unimportant. The test set error was 4.450, and the training set error was 4.810 using 72 memory elements in 7 cycles which gives further evidence of its unimportance.

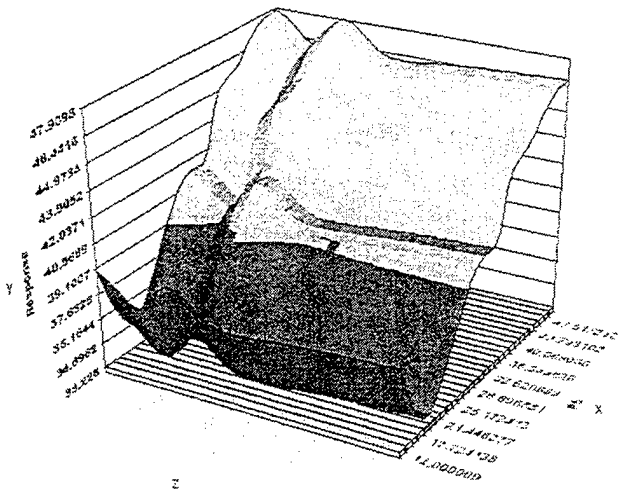

Figure 22. Topology of Cell Input $\mathbf{X_1}$ and $\mathbf{X_5}$.

V. FINDINGS AND CONCLUSIONS

A. FINDINGS

The HNeT system is an artificial neural network that has shown considerable evidence of being a more reliable, efficient, and effective tool than some of the more traditional artificial neural networks. This is evident by its performance in evaluating simple functions and random data. Not only was it fairly accurate in its evaluations, it provided many other means of determining the accuracy of its outcomes; for instance, error graphs, topology graphs, function graphs, and validation capabilities. HNeT is also a very user friendly system. The ability to easily change settings within each evaluation will reduce time constraints for finding solutions to research problems.

Its ability to learn many stimulus-response patterns with a single cell comprised of complex scalars decreases its storage capacity requirements which allows for a decrease in the time to perform a response recall operation. This capability will allow for increased volumes of problem solving and evaluating.

Although in most cases the error increased as the memory elements decreased, HNeT had already found its best case evaluation, usually in an early cycle. This shows its capability responding to stimuli accurately and early.

B. CONCLUSIONS

HNeT is a system that provides a valuable means for problem solving. Although the results from the evaluations conducted in this thesis were favorable, other evaluations should be conducted to further examine this system's capabilities. Image processing is an application that is well worthy of consideration for future evaluations. This will provide great insight into its true capabilities as it relates to military applications.

Most military doctrine, whether friendly or enemy, is established based on some repetitious pattern of war. In other words, our enemy's pattern of warfare dictate how we fight. From these patterns, and our previous evaluations, HNeT has the capability to learn and provide information based on these previous patterns.

LIST OF REFERENCES

1. Fausett, L. V., *Fundamentals of Neural Networks*, Prentice-Hall, Inc.,1994.

2. Johnson, R. C. & C. Brown, *Cognizers: Neural Networks and Machines that Think*, John Wiley & Sons, Inc., 1988.

3. Freeman, J. A. & D. M. Skapura, *Neural Networks: Algorithms, Applications, and Programming Techniques*, Addison-Wesley, Inc., 1991.

4. AND Artificial Neural Devices Corp., *HNeT^{tm} Application Development System: User's Guide*, AND Corp. 1992-1999.

5. Kartalopoulos, S. V., *Understanding Neural Networks and Fuzzy Logic: Basic Concepts and Applications*, Institute of Electrical and Electronics Engineers, Inc., 1996.

6. Borges, C. F., "Preliminary Evaluation of HNeT Software", *Technical Report*, Monterey, Ca, 1999.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, Ste 0944
   Fort Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, California 93943-5101

3. COL David C. Arny, Chairman 1
   Department of Mathematics
   United States Military Academy
   West Point, NY 10996

4. Professor Carlos Borges Code MA/Bc 2
   Naval Postgraduate School
   Monterey, CA 93943

5. Professor Bard Mansager Code MA/Ma 1
   Naval Postgraduate School
   Monterey, CA 93943

6. Professor Michael A. Morgan Code MA 1
   Naval Postgraduate School
   Monterey, CA 93943

7. CPT Darryl Langford 1
   311C Winans Place
   West Point, NY 10996