NEURAL NETWORK EMULATION OF TEMPORAL SECOND ORDER LINEAR DIFFERENCE EQUATIONS

by

JOSEPH CHARLES PELLEGRINI
B.S. Aerospace Engineering, M.I.T.
(1989)

SUBMITTED TO THE DEPARTMENT OF
AERONAUTICS AND ASTRONAUTICS
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

MASTER OF SCIENCE IN AERONAUTICS AND ASTRONAUTICS
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June, 1991

Signature of Author _____
Department of Aeronautics and Astronautics
April, 1991

Certified by _____
David Akin
Asst. Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by _____
Professor Harold Wachman, Chairman
Departmental Graduate Committee
Department of Aeronautics and Astronautics

Abstract

A recently developed simulated neuron called a Sequential Associative Memory was used to construct a set of neural networks in order to determine if it is possible to create a trainable network that is capable of recognizing and acting upon temporal or frequency dependent events in a signal that is continuous in space and discrete in time. The specific test involved attempts at emulating a simple second order (two term) linear difference equation with constant coefficients.

A comparative study was performed on three groups of networks. The first group was the baseline composed networks that did not utilize any type of feedback connections and therefore should exhibit no short term memory. The second group, the Sequential Associative Memory group, utilized internal feedback connections within the neural network. The third group used a variant architecture that utilized external feedback connections, but had no feedback connections internal to the network.

Each group of networks was tested using different topologies with regard to number of neurons and the number of connections. These networks were trained to emulate a two term linear difference equation. The performance of each topology was measured by the mean and maximum errors observed after the networks' weight matrix had achieved near steady-state behavior. These performance results were compared to judge relative performance for each general group and for the specific topologies.

It was found that the SAM networks exhibited much better performance than any of the baseline networks, demonstrating that feedback connection could indeed create a mechanism for short-term memory. It was also found that only a small number of the most complicated SAM networks could approach the performance of the variant networks, which were generally simpler in topology.

It was also shown that SAM networks are sensitive to the ratio of the coefficients in the linear difference equation and perform poorly when the [k-1]th term is greater in magnitude than the [k]th term. The variant architecture was found to be insensitive to this issue.

Thesis Supervisor: Dr. David Akin
Title: Asst. Professor of Aeronautics and Astronautics

Introduction

In 1988, Robert Sanner of MIT's Space Systems Lab published a Master's thesis that described his work on the synthesis of neuromorphic control systems. In this work Sanner demonstrated that a neural system with no prior knowledge of a second order dynamic system was capable of learning how to control the system. [Sanner, 1989]

Sanner's networks worked well and helped open a new approach to controlling dynamic systems. These networks had a drawback: they had no sense of prior state, and hence were incapable of emulating anything more complicated than a proportional feedback control system. Admittedly, these proportional control systems were non-linear and adaptive, but the networks were still incapable of emulating something as simple as a lead-lag compensator.

To address this limitation, this research considers the application of feedback connections within a neural network to achieve some type of short-term memory and thus to provide a mechanism for remembering a previous state. The inspiration for this work came from recognizing the similarities between neural networks and combinational logic circuits. It should be possible to emulate a finite state machine in a neural network, based on the fact that a finite state machine implemented with digital electronics is simply a combinational logic circuit attached to a bank of flip-flops. Since a flip-flop can be constructed using a pair of logic gates wired with feedback, it was reasoned that applying feedback to a neural network would result in a behavior similar to that of a finite state machine.

In 1989, Stephen Gallant and Donna King of Northeastern University, published a paper describing a variant artificial neural cell called a Sequential Associative Memory (SAM) cell. In their paper Gallant and King describe how several neural networks were trained to exhibit finite state machine behavior. [Gallant and King, 1989] This work was limited to the analysis of networks with binary inputs and binary

outputs and did not address the viability of training networks with continuous inputs and continuous outputs as would be required for a linear feedback control system.

The approach taken in this research was to create an artificial neural network that would attempt to emulate a multiple term finite difference equation. The first question that needed to be resolved was what form the artificial network would take, either an analog or digital hardware implementation or a software simulation. The goal was to emulate a finite difference equation that was discrete in time and continuous in space. A truly continuous system would require the construction of an artificial neural network from analog components. Due to cost and experience issues, digital simulation was chosen as a more desirable approach. The continuous nature of the system was maintained in the simulation by selecting digital representations with high enough resolution that the discretization error would be negligible.

Minor modifications were made to the structure of the binary SAM cells to accommodate higher resolution signals. Modifications were also made to the standard back propagation learning algorithm that facilitated training of the feedback connections. An attempt was made to see if these new networks could emulate a second order finite difference equation. This was demonstrated as reported in the body of this thesis.

Chapter 1
Application of Artificial Neural Networks to Adaptive Control

Chapter Summary

- Neural networks provide a relatively simple mechanism for creating an adaptive control system.
- Neuromorphic control systems have been found to be very effective in cases where the behavior (transfer function) of the controlled plant is largely unknown.
- Adaptive control systems constructed using linear digital filters usually require some knowledge of the controlled plant.
- Digital filters higher than first order exhibit some sense of past state.
- Unidirectional sequential neural networks do not exhibit behavior based upon past state.

1.1 Artificial Neural Networks

The term artificial neural network applies to a wide range of systems. For the purpose of this investigation the discussion will be limited to the sequential simulation of a neural network on a digital computer. Despite the limited scope of this investigation, the discussion and results presented here should be generally applicable to other types of artificial neural networks including those implemented as massively parallel digital or analog electronic devices.

One of the primary characteristics of an artificial neural network is its ability to adapt. As time progresses the relationship between a neural network's input and output will change. Under ideal conditions this change follows a course that produces a desired behavior.

This adaptive behavior is so universal and so powerful that many researchers prefer to classify the behavior as learning. The perceived ability of a neural network to learn is consistent with other characteristics that provide a base of commonality between artificial neural networks and their biological counterparts.

Artificial neural networks derive their overall decision power from the parallel action of many simple elements. Human beings derive their complex and powerful deductive powers from several billion neurons acting in unison. [Crick and Asanuma, 1986] The current theories imply that this type of massive parallelism may lead to computational power which is not bound by the limitations of current artificial systems.

Artificial neural networks are fault tolerant. Aberrant behavior in a small number of elements does not cause large changes in the nominal behavior of the network. This characteristic is derived from the large number of elements and their vast interconnectivity. In a biological system, aberrant behavior of individual elements is unavoidable as oxygen and nutrient supplies are not perfectly uniform.

Artificial neural networks are also failure tolerant. Destruction or disconnection of a small number of elements does not cause large changes in the nominal behavior of the network. It is known that humans can lose millions of brain cells without a noticeable change in behavior or ability. An artificial neural network can be retrained in real-time to work around substantial failures.

Artificial neural networks are capable of extracting relevant information from a noisy signal. This is crucial in many applications

since all signals have some level of noise. Neural networks can reject noise without explicit knowledge of the type of noise they are dealing with. This is quite different from a simple filter that requires information about the spectral character of the noise. A neural network's noise rejection capability is inherent in the adaptive (learning) properties of the network.

The elements in an artificial neural network are massively connected to each other. A single element can be connected to any number of other elements. The complexity of the connections in a network are largely responsible for its behavior. The greater the number of connections, the more unusual (or unpredictable) the behavior may be. Since the possible number of connections increases exponentially with the number of neurons, a neural network's behavior can quickly become very intricate and complex. This complexity is best exemplified in animal (particularly human) behavior.

In an artificial neural network, the structures of all the elements are essentially identical. Even in a biological system different types of neural cells appear to operate in a similar fashion. These elements are also simple in their operation. A single element has very little computational power when it is considered by itself, but a network of elements working together can produce vast computational power. This characteristic is important because it make the simulation and construction of artificial networks very simple. It is much easier to create thousands of simple things than it is to make hundreds of complicated things.

The synthesis of an artificial neural network usually follows one of two popular courses. The first course is to construct a device whose elements and structure imitate natural neural systems. These systems are usually electronic in nature and can be entirely analog, entirely digital, or a hybrid of both. Hardware implementations of this type have one very large benefit: the parallel nature of neural computation is preserved, leading the way to great computational power. On the

negative side, hardware implementations are difficult to reconfigure: once an overall structure for a neural system has been determined and implemented in hardware, it is difficult to make major changes in the topology of a network since connections are implemented physically.

The second class of artificial neural networks are implemented as sequential simulation. These simulations are usually performed on a digital computer with a single processing element. In a sequential simulation, the neural network elements and topology are represented purely through software. Since there is only one processing element, the benefit of massively parallel computation is lost. However, sequential simulations are relatively easy to reconfigure; not only can the topology be changed, but also the characteristics of the neural elements can be modified. This level of flexibility makes sequential simulation the natural choice for research applications.

Recently a hybrid mechanism for creating artificial neural networks has emerged. Sequential simulations are implemented on computers that utilize multiple parallel processing elements. Unlike the pure hardware implementation, the ratio of processing elements to neural elements in the hybrid implementation is not one-to-one. As a matter of fact, the number of processing elements is usually much smaller than the number of neural elements. The result is a system that takes advantage of the parallel nature of neural systems, but is still bounded in computational power. The hybrid system retains the flexibility found in the purely sequential solution, since the neural elements and neural topology are defined entirely in software. The drawback of the hybrid system is cost. At the time of this writing a typical system utilizing four parallel processors would cost approximately $15,000. The increased performance (roughly proportional to the number of processors) usually does not justify the cost when a personal computer costs around one tenth as much. The experiments described in this paper were carried out using a sequential simulation on a computer with a single processor.

1.2 Artificial Neural Elements

Artificial neural elements or neurons can be divided into four sections:

- Dendrites (input connections and weighting values)
- Left body (summation function)
- Right body (activation function)
- Axon (output connections)

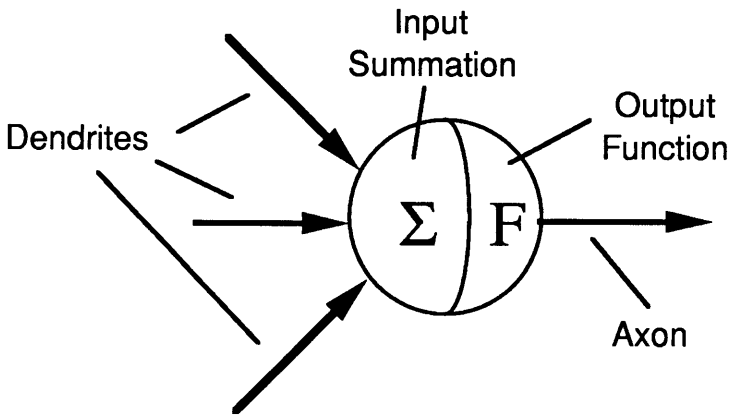


Figure 1.1 - A Neural Element

A given neuron can have any number of dendrites, receiving signals from the axons of other neurons. Each dendrite connects to the axon of one other cell. In addition to accepting the raw input signals, each dendrite has its own weighting value. These weights vary with time and are partially responsible for the behavior of a neural network's input/output relationship. It is common to represent the dendrite weights as a matrix. The following notation will be used henceforth:

$$W_j^i$$

(the weight of dendrite i of neuron j)

The inputs, transmitted from the axons of other neurons, can also be represented as a matrix.

$$E_j^i$$

(the input to dendrite i of neuron j)

The summation function is used to derive a value for the neuron that is commonly referred to as the NET value.

$$N_j = \sum_{i=1}^{k} W_j^i \, E_j^i$$

In the above expression, k is the number of dendrites on neuron j.

The activation function is used to determine the output value for the neuron using the value computed in the summation function. The output of a neuron is fed through the axon and becomes the input to other neurons. There are three common types of activation functions: threshold, linear activation, and sigmoidal activation.

Threshold functions compare the value of Nj to some threshold value and produces one of two possible outputs depending on whether Nj is greater than or less than the threshold. The following is an example of a threshold function.

$$O_j = F(N_j) = \begin{cases} +1 & \text{if } N_j \geq 0 \\ -1 & \text{if } N_j < 0 \end{cases}$$

In a neuron of this type the threshold value can be predetermined and fixed, or it can be randomly set and time varying. The discrete output values are usually predetermined and fixed. Obviously a neuron with this type of output function would have a non-linear input/output relationship. It would also follow that a network containing neurons of this type would also have a non-linear input/output relationship.

Networks containing threshold units are suitable for tasks where the desired output behavior is discrete; that is, where the number of possible outputs is limited. This type of behavior is usually associated with the act of making decisions or picking a selection from a limited number of choices.

A linear activation function simply takes the value of Nj and multiplies it by some constant, Mj.

$$O_j = F(N_j) = M_j N_j$$

It has been shown that a network made up solely of linear neurons will itself be linear. It has also been shown that linear networks are incapable of solving some relatively simple problems [Minsky and Papert, 1969]. On the other hand, as will be shown later in more detail, linear networks are well suited for the task of emulating linear digital filters.

The third common type of output function is the sigmoidal activation function. The sigmoidal function has several interesting characteristics. Unlike the linear function, the sigmoidal function is bounded; even if the magnitude of the Nj value approaches infinity, the output of a sigmoid neuron will remain within a defined bound. Unlike the threshold function, the sigmoidal function is continuous. This provides access to a training algorithm (known as the Modified Delta Rule or Back Propagation) that requires knowledge of the slope of the activation function. Finally, the sigmoidal function can approximately mimic either the linear function or the threshold function, depending on how the dendrites of a given neuron are weighted. The sigmoidal output function takes the following form:

$$O_j = F(N_j) = \frac{1}{(1 + e^{-N_j})}$$

During previous investigations into neuromorphic control of dynamic systems [Sanner, 1989], neurons with sigmoidal activation functions were placed on the final output stage. It was decided in Sanner's work that the bounded output of a sigmoid matched well with the physical limitations of the system's actuators.

1.3 Generation of Artificial Neural Networks

The way in which a neural network is connected -- its topology -- is also an important factor in the behavior of the network. Conceptually, most artificial neural networks are arranged in layers. Each layer contains some number of neurons. The outputs of the lower layers feed into the inputs of the higher layers. The information flow during neural operation is unidirectional: moving from the input layer through to the output layer.

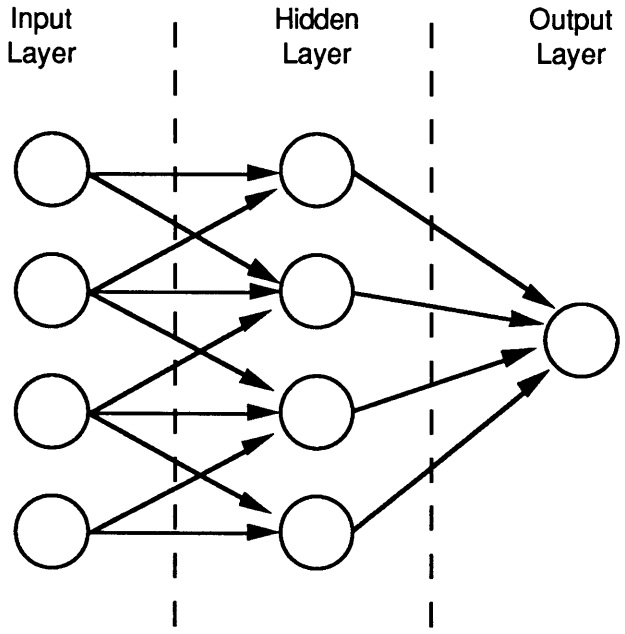


Figure 1.2 - A Unidirectional Network

In a typical operation an input vector is applied to the first layer of neurons. This layer is referred to as the input layer. The input layer's axons are excited in reaction to the input signals and these axons supply input signals into the dendrites of the higher levels. The signals propagate forward through higher and higher levels until every level has been reached. The output of the network is taken as a vector composed of the outputs of the neurons in the highest level. These output neurons are classified as belonging to the output layer.

Neurons that are neither classified as input neurons nor as output neurons are simply called hidden neurons (or neurons that exist in hidden layers). In several papers [Sanner, 1989; Grossberg, 1976, 1982, 1987; Kohonen, 1984] neural researchers have theorized that the hidden layers in a neural network create a complicated internal representation of the outside world. Creating a system to trace and comprehend these internal representations has been a desire of neural researchers for quite some time. The fact that one cannot understand why a particular neural network correctly performs its job prevents the researcher from guaranteeing that such a network will continue to do so for all cases. This characteristic has in many cases disqualified neural networks from being implemented in applications where such a guarantee is critical.

The connections between axons and dendrites are responsible for defining the topology of the network. Each dendrite is connected to the axon of a single neuron. If a unique identification number is assigned to every neuron in a network, the topology of that network can be described using a connectivity matrix D.

$$D_j^i$$

(the identification number of the axon
connected to dendrite i of neuron j)

The dendrites that receive the input signals to the network can be connected to a layer of linear input neurons whose outputs are set (rather than evaluated) to correspond to the input vector.

When the signals in a neural network progress unidirectionally through increasing layers with synaptic and neural evaluations proceeding under a fixed natural sequence, that neural network is classified as a sequential unidirectional network. In a sequential unidirectional neural network there are no feedback paths for synaptic signals.

If it is assumed that the identification numbers of the neurons increases monotonically with level and the order of firing progresses naturally, the lack of feedback can be described with the following rule:

$$D_j^i < j \ (\forall \ i,j)$$

In a sequential network the dendrites of any given neuron are only connected to axons that have already had an opportunity to fire. This "no feedback rule" is implemented because the typical learning algorithms are not designed to coherently train feedback connection. One of the purposes of this thesis is to examine how this "no feedback rule" can be successfully violated.

It is important to recognize the fact that sequential networks and unidirectionally propagating signals are constructs unique to artificial sequential simulations of neural networks. In natural neural networks the flow of information is not necessarily unidirectional, but instead tends to create complicated closed paths, where signals may follow intricate loops. In addition, most artificial sequential networks evaluate the neuron activations synchronously: this is opposed to the character of natural neural systems and hardware mimics that are essentially asynchronous. In a natural neural network each individual

neuron works completely independent from the other neurons; there is no necessary pattern to the firing order.

The asynchronous character of natural neural networks is a curious and promising thing. It is because of this asynchronous nature that one can hope to obtain maximum benefit from massive parallelism without the crippling effects of having to synchronize the computational nodes (the neurons).

The ultimate purpose of this investigation is to take a sequential simulation of a neural network and violate the classical restriction against feedback paths for information, while retaining the synchronous evaluation sequence. By doing this a neural network with a workable and trainable short-term memory capable of emulating second and higher order digital filters was created.

1.4 Training of Artificial Neural Networks

The primary appeal of neural networks is that the global application of relatively simple rules can easily cause a network to converge to a desirable behavior. This behavior adaption is achieved by making modifications in the weight matrix associated with the network's dendrites.

The adaptive process is usually referred to as training, since it utilizes a course of action analogous to a training routine:

- Apply a set of sample input vectors
- Allow the network to produce an output
- Compare the actual output vector with the desired output vector for that set of inputs
- Apply an algorithm that adjusts the weight matrix based upon the difference between the output vectors, the input vector, and the current weight matrix.

Since the adaptive process is driven by the error in the output vector, the change in the weight matrix can be loosely associated with a punish/reward system. Larger changes are made to weights of neurons whose output is perceived as being incorrect and smaller changes are made to neurons whose output is correct. Theoretically, the system will eventually settle to a point where the input/output mapping is optimal and the weight matrix will remain relatively fixed.

There are two strategies for the application of training:

Strategy #1 -- TRAIN and LEAVE

1. Setup the artificial neural network into the proper topology
2. Initialize the weight matrix to a series of small random values
3. Present a series of training sets and adjust weights according to some algorithm.
4. When desired behavior is apparent, cease training and fix weights
5. Utilize the neural network with the fixed weights

Strategy #2 -- CONTINUOUS TRAINING

1. Setup the artificial neural network into the proper topology
2. Initialize the weight matrix to a series of small random values
3. Present a series of training sets and adjust weights according to some algorithm.
4. When desired behavior is apparent, set the neural network to work on the real problem.
5. Continue to evaluate, train, and adjust weights using the spontaneous (real world) inputs and outputs.

The first strategy takes advantage of the neural network's learning ability, but foregoes the capability of continuous adaptation. This is common in real-time applications where there is not sufficient computational power to operate and train in real-time. In this case

the training will occur off-line by using a simulator that also may not be real-time.

There are many different algorithms for implementing the punish/reward mechanism. Most utilize a formula that causes the system to follow the gradient of the output error to some minimum. There are also other training algorithms that rely on more than simple punish/reward rules and utilize other mechanisms, such as simulated annealing. These Boltzman machines, as they are called, overcome some problems that simple error gradient following algorithms encounter, most notably the potential to get caught in local error minima. However, these Boltzman machines have some negative behaviors of their own, particularly unpredictable behavior during training and potential network paralysis [Hinton and Sejnowski, 1986].

In this investigation the simplest and most well known punish/reward training algorithm is used. It is known as the modified delta-rule or the back propagation algorithm. This algorithm was chosen for this study for the following reasons:

1. It is a relatively simple algorithm to implement.
2. It is a well known algorithm that has become a yardstick against which other training algorithms are measured.
3. While no neural network training algorithm is perfectly repeatable, back propagation is more repeatable than algorithms that use simulated annealing or other stochastic methods.
4. The back propagation algorithm was easily modified to work with the variant neural network topologies that are the central focus of this thesis.

The back propagation algorithm was originally reported by Werbos (1974), but it wasn't until 1986 when Rumelhart, Hinton, and Williams rediscovered the process that it achieved widespread recognition and use.

The back propagation training algorithm is often referred to as the generalized delta-rule. The original delta rule is a training algorithm developed in 1960 by Widrow and Hoff. The delta rule was developed to train networks of perceptrons. Perceptron networks were simple single layer networks composed of linear discriminant neurons. It was shown [Rosenblatt, 1962] that a network of perceptrons trained using the delta-rule could eventually learn any function that it could represent.

The original delta-rule is applicable only to single layer networks. The training sequence and algorithm are as follows.

1. Apply a training input pattern and allow the network to calculate the output vector, **O** , of the network.

2. For each output neuron, compute a delta value which is the difference between the desired output, **T** , and the actual output.

$$\delta_j(n) = [T_j(n) - O_j(n)]$$

3. For each dendrite of a given output neuron, the weight is adjusted by a product of the neuron's delta value, the activation level of the neuron, and a learning rate.

$$\Delta_j^i(n)=\eta\ \delta_j(n)\ E_j^i(n)$$

$$W_j^i(n+1)= W_j^i(n)+ \Delta_j^i(n)$$

A training "momentum" term is often appended to the above equation. The momentum is modulated by a constant momentum rate, α.

$$W_j^i(n+1)= W_j^i(n)+ \Delta_j^i(n) + \alpha\Delta_j^i(n-1)$$

4. Repeat step (3) for each output neuron.
5. Repeat steps (1-4) until satisfactory behavior is observed.

In 1969, Minsky and Papert demonstrated the fundamental limitation of simple perceptron networks trained by the original delta-rule by showing that single layer networks could only represent linearly separable functions. This eliminated the potential for single layer perceptron networks to represent an entire class of functions,

including functions as simple as the binary "exclusive or" (XOR) function.

Since the original delta-rule was limited to training single layer networks, an extension to the algorithm was developed to handle multiple layer networks, and the back propagation algorithm was created.

The problem with the original delta-rule was that it did not provide a mechanism for training internal or "hidden" neurons. The original delta-rule equations required that each neuron have a target or training value (Tj) for comparison to its output. Since these training values only applied to the output neurons, there was nothing that could be presented to the hidden neurons.

The back propagation algorithm is aptly named, since it solves the above problem by propagating the delta values for the neurons backwards through the network. This mechanism provides effective training of the hidden neurons in a multiple layer network.

The back propagation training sequence and algorithm are as follows.

1. Apply a training input pattern and allow the network to calculate the output vector, **O**, of the network.

2. For each output neuron, compute a delta value which is the difference between the desired output and the actual output.

$$\delta_j(n) = [T_j(n) - O_j(n)]$$

3. For each dendrite of a given output neuron, the weight is adjusted by a product of the neuron's delta value, the activation level of the neuron, and a learning rate.

$$\Delta_j^i(n) = \eta\, \delta_j(n)\, E_j^i(n)$$

$$W_j^i(n+1) = W_j^i(n) + \Delta_j^i(n) + \alpha \Delta_j^i(n-1)$$

4. Repeat step (3) for each output neuron. Once all of the output neurons have been evaluated, continue to step (5) and evaluate the hidden neurons.

5. For each hidden neuron, compute a delta value that is a weighted sum of the delta values of the neurons attached via the axons of the hidden neuron.

$$\delta_j(n) = \frac{\partial O_j}{\partial N_j}(n) \sum_q [\delta_q(n)]*[W_j^q(n)]$$

For a sigmoidal neuron the partial derivative of the output is given by the following -

$$\frac{\partial O_j}{\partial N_j}(n) = O_j(n)*[1 - O_j(n)]$$

6. For each dendrite of a given hidden neuron, the weight is adjusted the same as with the output neurons.

$$\Delta_j^i(n) = \eta\, \delta_j(n)\, E_j^i(n)$$

$$W_j^i(n+1) = W_j^i(n) + \Delta_j^i(n) + \alpha \Delta_j^i(n-1)$$

7. Repeat steps (5-6) until the input layer is reached.

8. Repeat steps (1-7) until satisfactory behavior is observed.

1.5 Information in Neural Networks

Neural networks are information engines: they absorb information about their environment both through signals received at their input neurons, and through feedback provided by training algorithms. Neural networks take the information they collect and manipulate it producing information that is (hopefully) in a more useful form than that which was collected. Neural networks do not simply manipulate information; they also store it internally for future use.

In the type of artificial neural networks described in this thesis there are two mechanisms for information storage. The first place where information can be stored is in the excitation signals produced by the axons of each neuron in a network. At any point in time, the axons are at some excitation level. This excitation level can change as often as every evaluation cycle in the case of sequentially simulated networks.

The information stored in the axon excitations is often referred to as **short term memory**. This label is applied since in the most common networks -- sequentially simulated networks with unidirectional signal flow -- the information contained within these excitations is flushed and replaced every evaluation cycle. In a **SUD** (**S**equential **UniD**irectional) network there is no direct relationship between the information contained in the axon excitations at time **t** and the information contained in the same excitations at time $\mathbf{t} + \Delta\mathbf{t}$ where $\Delta\mathbf{t}$ is some an increment of time equal to one evaluation period.[1]

The second mechanism for information storage in an artificial neural network is in the weight matrix associated with the network's

[1] While there is no direct relation between the axonic information across evaluation periods, there can be a rather tenuous relationship in cases where the SUD network is undergoing training. In this case changes to the weight matrix are indirectly dependent upon axonic excitation at time **t** and these changes are relevant to the axonic excitations at time $\mathbf{t}+\Delta\mathbf{t}$. This relationship is so tenuous and insignificant that it is hardly worth mentioning. It is very difficult to imagine coherent information being transmitted in this way.

dendrites.[1] It is this weight information that determines the static input/output relationship of the network. Unlike the axonic excitations, the information in the weight matrix is relatively long lived. In the extreme, a well trained network's weights may be held constant or may only change by an infinitesimal amount depending on whether or not training is still active.

The information contained in the weight matrix is usually classified as **long term memory** due to its longer lifetime with respect to axonic excitations.

It is not difficult to calculate the total information content of both the long term and short term memory in a digitally simulated neural network. Usually, the information pertaining to axonic excitation and dendrite weights are digitally stored in static resolution variables; the number of bits used to represent each quantity is fixed during all operation. The total short term information storage can be calculated by multiplying the number of bits used to represent the axonic excitation and multiplying it by the number of axons in the network. The total long term information storage can be calculated by multiplying the number of bits used to represent each element in the weight matrix and multiplying by the number of dendrites in the network.

While computing the information content of a neural network, it is important to remember that only a small fraction of the capacity of the network is actually used. Neural networks (artificial and natural) are notoriously inefficient at packing information tightly. Therefore, one would find that it will usually take many more neurons and synaptic connections than would be minimally necessary to solve a problem.

---

[1] It can also be argued that there is information stored in connectivity matrix, but for simplicity this information can be grouped collectively with the information contained in the weight matrix.

Chapter 2
Sequential Associative Memory

Chapter Summary

- Sequential Associative Memory (SAM) cells have been shown to provide state-like behavior in networks of linear discriminant neurons.

- SAM networks utilize feedback connections to implement short term memory.

- SAM networks using linear neurons can emulate second and higher order linear finite difference equations.

- Simple modifications to the standard back-propagation algorithm provide a mechanism for training the feedback connections in a SAM network.

2.1 What is Sequential Associative Memory?

In 1988 Stephen Gallant and Donna King of Northeastern University's College of Computer Science, published several papers on a type of neural network construction referred to as a *Sequential Associative Memory* [Gallant & King, 1988]. The object of their research was to improve the short term memory capability of artificial sequential neural networks to the point where their networks could emulate the types of behavior common to digital electronic finite state machines.

Gallant and King succeeded in producing networks with interesting finite state type behavior. One network solved the sequential parity problem. This network received a sequence of bits (inputs of 0 or 1) and produced an output (0 or 1) that reported whether the current sum (sequentially added in time) was even or odd. Another of their

networks was able to add an arbitrarily long sequence of digits, represented as binary numbers, that were presented over a period of time. They also performed experiments in robot action, where a short signal triggered a long series (in time) of network outputs.

Gallant and King's networks achieved "finite state machine" behavior through the action of feedback paths in the neural topology of the network and in an adjustment to the evaluation rules applied to the calculation and firing of axonic signals. These rules [Gallant and King, 1988] are reproduced here:

---

1. Initialize SAM cell activations and output cells activations to 0.

2. Set the activations of inputs cells to the {first/next} set of inputs.

3. For each SAM cell, compute its activation value, but do not change that activation until all other SAM cells have computed their new values. After computing the new activation values for all SAM cells, modify all activations accordingly.

4. For each output cell, compute its new activation value and immediately change its activation to that value. These activations are the {first/next} network outputs for the inputs previously applied in 2.

5. Go to step 2.

It was observed by this researcher that the following characteristics were typical of Gallant and King's SAM networks:

1. The neurons were all threshold discriminant; their input and output signals were essentially binary.

2. The networks were composed of three layers: an input layer, one hidden layer, and an output layer.

3. All feedback connections led to the cells in the hidden layer; it is these hidden layer cells that Gallant and King refer to as SAM cells.

4. The weights of the dendrites involved in feedback connections were randomly set and permanently fixed at the time of network initialization. In other words the feedback connections were not affected by the training procedure.

5. The random initialization of the feedback weights included a gain factor that increased the average initial magnitude of the feedback weights with respect to the other non-feedback weights. This gain factor was typically set to about 10.

6. According to Gallant's evaluation rules, the SAM cells can be evaluated and updated in parallel. This characteristic makes the SAM architecture attractive for parallel processing hardware implementation.

In this investigation two changes were made to the Gallant-King model. The first change was related to observation number 1. Where Gallant and King used discriminant functions with binary signals, the SAM networks examined in this research utilize linear activation

functions and sigmoidal activation functions producing high resolution (almost continuous) signals[1].

The second deviation from the Gallant-King model was to modify the back propagation training algorithm so that coherent training of the feedback weights could occur. The details of these changes are discussed in detail later in this chapter.

Despite the fact that feedback weights were allowed to adapt, the gain factor mentioned in rule 5 was maintained. Experimentally, this seemed to increase the speed and success of convergence for the SAM networks.

[1] The output signals in these SAM networks were encoded as 8 byte (64 bit) floating point number. This provided the equivalent to an analog signal with a signal-to-noise ratio of approximately 10^{15} .

2.2 Example of a SAM Network Exhibiting Short Term Memory

The following diagram shows a SAM network that will implement the sequential parity function.

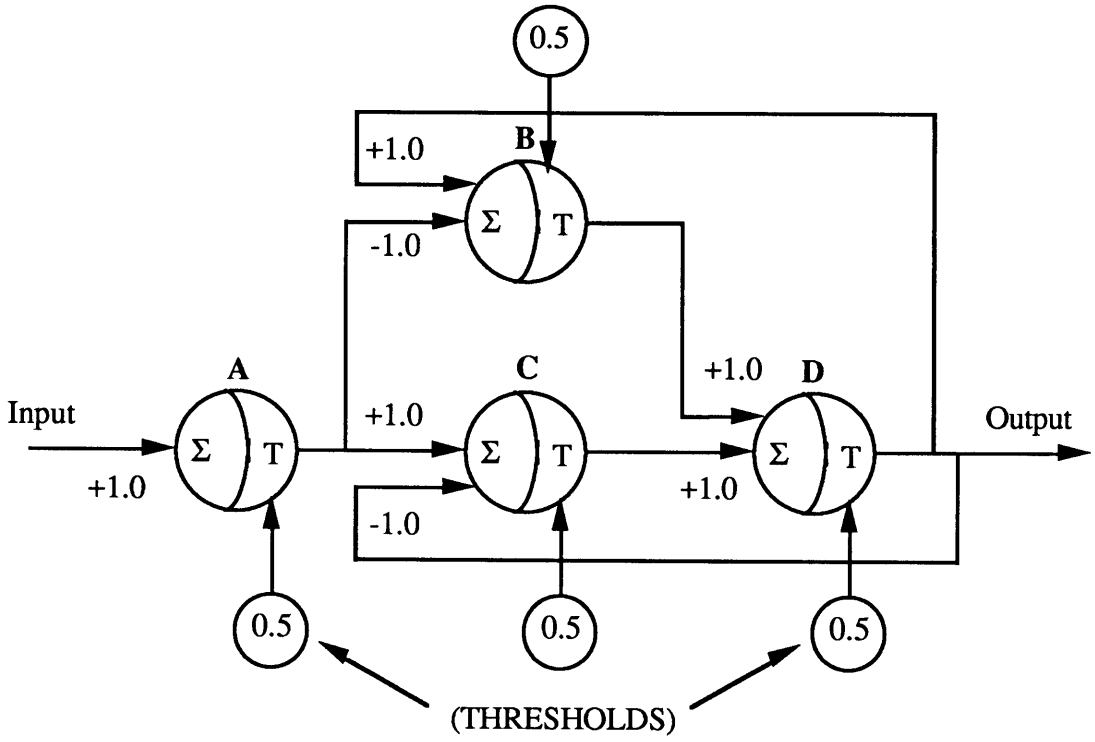


Figure 2.1 - A SAM Network Implementing Sequential Parity

The sequential parity function is described as follows, where E(n) is a sequence of binary inputs and O(n) is a sequence of binary output.

$$O(n) = \begin{cases} 0 \text{ if } \sum_{k=0}^{n} E(n) \text{ is even} \\ 1 \text{ if } \sum_{k=0}^{n} E(n) \text{ is odd} \end{cases}$$

This summations in the above function seem to imply that a record of all previous inputs needs to be kept in order to compute the correct

output. However, it is only necessary to remember one previous output.

$$O(n) = \begin{cases} 0 \text{ if } [O(n-1) + E(n)] \text{ is even} \\ 1 \text{ if } [O(n-1) + E(n)] \text{ is odd} \end{cases}$$

The memory of the previous output, O(n-1), is an indication of internal state storage. This state is represented in the above network through the feedback connection that leads from the output neuron back into the middle neurons.

The sequential parity function can also be written in binary logic notation as follows:

$$O_n = XOR(E_n , O_{n-1}) = E_n \otimes O_{n-1}$$

Breaking down this function, one can analyze the structure of the neural network shown in the figure 2.1.

$$O_n = E_n \otimes O_{n-1} = [E_n \cup \overline{O_{n-1}}] \cap [\overline{E_n} \cup O_{n-1}]$$

Neuron A simply passes the input, E(n) to neurons B and C. Neuron B implements the logical function $[E_n \cup \overline{O_{n-1}}]$. Neuron C implements the logical function $[\overline{E_n} \cup O_{n-1}]$. Neuron D, performs an AND function to complete the computation of O(n).

Although it can be shown that such a simple network as the one given above will implement desired behavior, as with all neural networks it is generally necessary to implement a solution in a network that is larger than the minimum size required. The reason for this need lies in the nature of the training mechanisms. Back-propagation can really be described as a very inefficient error gradient following algorithm. Due to its inefficient nature, it is unlikely that training will cause a

minimally connected network to converge in an acceptable period of time.

Therefore, in practice, many more neurons and connections are typically used than would be minimally necessary to implement a desired function.

2.3 Short Term Memory and Second Order Linear Digital Filters

The ultimate goal of this thesis is to show that it is possible to train a neural network to emulate the behavior of a linear second order digital filter. From a functional standpoint, the goal is to develop a network that will reproduce the effects of a two term difference equation, such as:

$$\mathbf{u}(k) = \mathbf{c_0}\,\mathbf{e}(k) + \mathbf{c_1}\,\mathbf{e}(k-1)$$

or

$$\mathbf{u}(k) = \mathbf{c_0}\,\mathbf{e}(k) + \mathbf{c_1}\,\mathbf{u}(k-1)$$

The first equation describes a filter whose output is dependent on a weighted sum of the current input and the input from the previous time step. The following neural network, composed entirely of linear neurons, will reproduce this function exactly. The feedback path shown in the diagram produces a one time step delay that allows the system to produce the behavior linked to the second term in the equation.

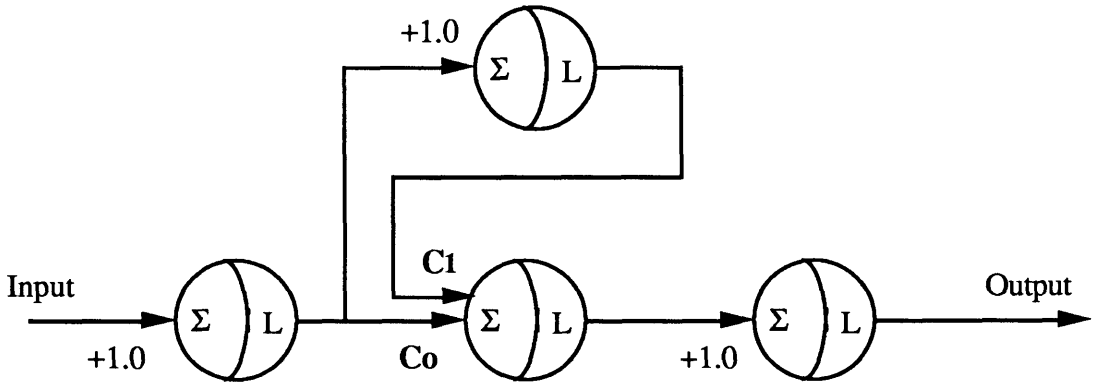


Figure 2.2- A SAM Network Implementing a Finite Difference Equation

The second equation describes a filter whose output is dependent on a weighted sum of the current input and the output from the previous time step. The following neural network, composed entirely of linear neurons, will reproduce this function exactly. Once again, the feedback path shown in the diagram produces a one time step delay that allows the system to produce the behavior linked to the second term in the equation.

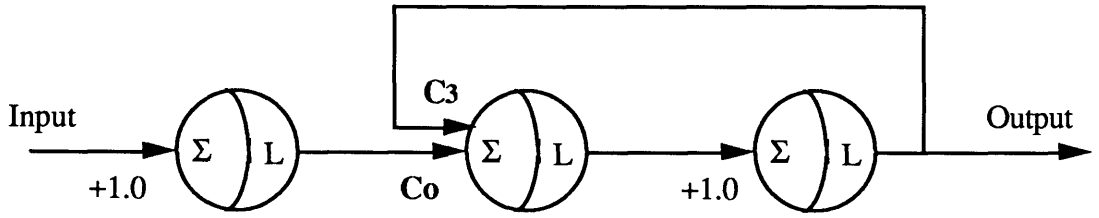


Figure 2.3- Another SAM Net Implementing a Finite Difference Equation

Unfortunately, observations made in this thesis indicate that it may not be possible to train the above networks. Application of the back-propagation training algorithm (the original and as modified for this thesis) produces diverging training sessions whenever there is a closed loop of connections that does not include at least one neuron with a bound output. In other words, it may not be possible to train a SAM network composed entirely of linear neurons.

This leads to a very important point about the characteristics of neural networks:

Just because it can be shown that a particular neural architecture can emulate a certain behavior does not mean that such a network can be trained, starting at a random state, to emulate such behavior.

It also appears that the requirement for having neurons with bounded outputs in all closed feedback paths makes it impossible to perfectly emulate second (or higher) order linear difference equations. There will always be some limit to how well a non-linear network can emulate a linear function. Theoretically, it should be possible to create an arbitrarily large and complex non-linear network where this limit is so close to perfection that it is of the same order as the signal-to-noise ratio of the signals (or the discretization error for the case of digital signals). In the cases examined later in this thesis, this limit to performance was measured and explicitly noted.

2.4 Short Term Memory and Higher-Order Linear Digital Filters

It is also possible to cascade the feedback loops to obtain an arbitrarily long time delay. For example, the following network reproduces the behavior of the equation

$$\mathbf{u}(k) = \mathbf{c_0}\,\mathbf{e}(k) + \mathbf{c_3}\,\mathbf{u}(k-3)$$

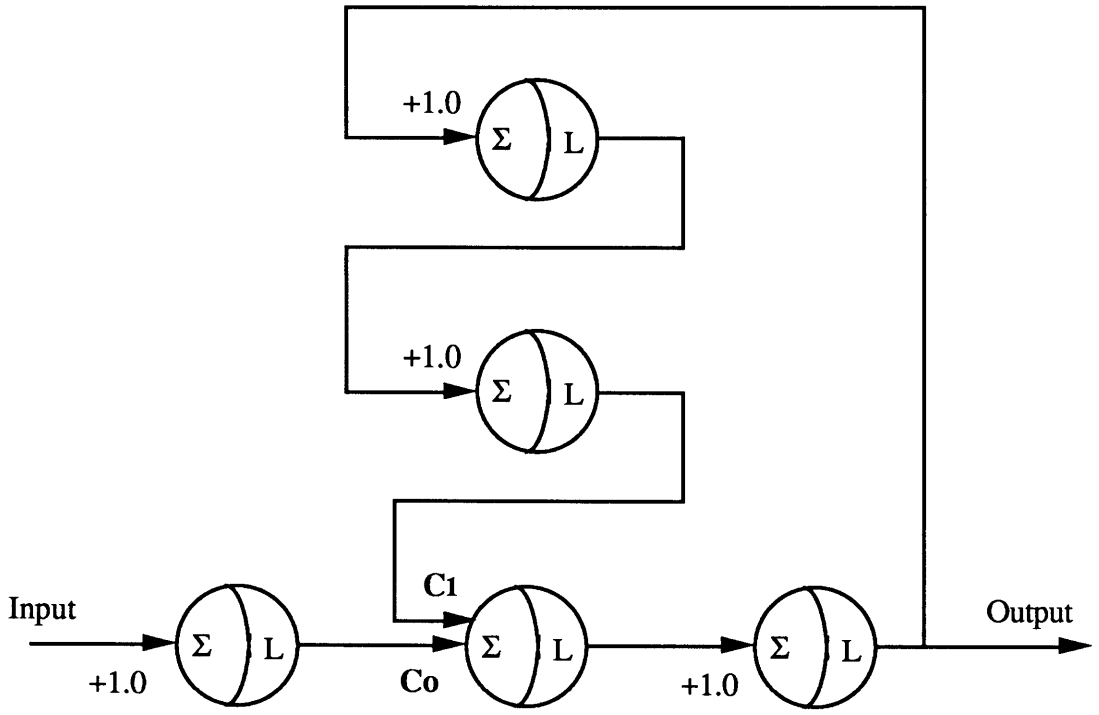


Figure 2.4 - A SAM Net Implementing a 3-Step Time Delay

Unfortunately, observations made in chapter 4 of this thesis indicate that although it is possible to train a network to emulate higher order equations (and time delays), it may take an unacceptably long training time to achieve a satisfactory behavior. This brings up a second important point about the nature of neural networks:

Just because it can be shown that a particular neural network can be trained, starting at a random state, to emulate a desired behavior does not mean that such a network can be trained in an acceptable period of time.

2.5 Modification to the Back Propagation Algorithm for SAM Networks

Earlier in this chapter, it was discussed how Gallant and King's original concept for SAM networks assumed that the weights on the feedback connections would remain fixed. Their presumption was that the other weights in the network would reorganize and scale themselves to work within the bounds set by the fixed feedback weights.

While these presumptions provided satisfactory results, a decision was made by this researcher to attempt a modification of the back-propagation algorithm to allow training of the feedback weights as well as the feedforward weights.

The problem with applying standard back propagation to a feedback path is that the equations in the typical implementation of the back propagation algorithm require information that is either unavailable or not appropriate for the training of feedback connections. To fix these problems, two changes to the back propagation algorithm were required.

The first change to the back propagation algorithm is applied to the computation of a delta value for hidden neurons. The unmodified equation is:

$$\delta_j(n) = \frac{\partial O_j}{\partial N_j}(n) \sum_q [\delta_q(n)]*[W_j^q(n)]$$

This equation computes the delta value for each hidden neuron by computing a weighted sum of the delta values of the neurons attached to the axons of the hidden neuron. The assumption made in the original algorithm is that axons only connect to cells in higher levels. In the case of a feedback path, *the axon might be connected to a neuron whose current delta value has not yet been computed.* The

current modification to this part of the algorithm is to simply ignore such feedback paths in the computation of the hidden delta values. This changes the above equation to

$$\delta_j(n) = \frac{\partial O_j}{\partial N_j}(n) \sum_q \begin{cases} [\delta_q(n)]*[W_j^q(n)] & \text{if (level of q) > (level of j)} \\ 0 & \text{if (level of q) } \leq \text{ (level of j)} \end{cases}$$

The second change to the back propagation algorithm applies to the formula that calculates the change to each dendrite weight. The unmodified formula is presented as

$$\Delta_j^i(n) = \eta\ \delta_j(n)\ E_j^i(n)$$

This equation computes the change in each weight by multiplying the delta value of the neuron by the current excitation level of the dendrite and a constant learning rate. This equation, taken by itself, is perfectly satisfactory for application to feedback connections. It is only on application that this formula may corrupt the algorithm. The reason for this lies in the mechanism typically used implement the above formula in software code. In order to eliminate the need for redundant storage, the following assumption is typically made:

$$E_j^i(n) = O_j(n)$$

Usually, it is assumed that the excitation level of a dendrite is equal to the current excitation level of the axon it is connected to. In a discrete time digital simulation, this assumption only applies to feedforward connections. For feedback connections the above equation does not hold. The correct equation for a feedback connection would be

$$E_j^i(n) = O_j(n-1)$$

In the simulation used in this investigation, the integrity of the algorithm was maintained at the expense of some additional memory by eliminating the above assumption and having each dendrite "remember" its own excitation state.

Chapter 3
Convergence Properties of Sample SAM Architectures

Chapter Summary

- Closed feedback paths containing only linear neurons will cause instability during training when using the modified back-propagation algorithm.

- A network attempting to emulate a linear equation utilizing a sigmoidal output neuron will always have some inherent error induced by the non-linear nature of the sigmoid.

- A sequential unidirectional network without feedback cannot emulate a second order digital filter.

- SAM networks are capable of producing relatively low errors when trying to emulate some second order linear difference equations.

- Variant networks utilizing a unidirectional network and external feedback connections through a FIFO buffer can emulate second order difference equations as well or better than most SAM networks.

3.1 Experimental Network Architectures and Labeling System

In this chapter a presentation is made of several sample neural network architectures. The first two architectures are typical unidirectional sequential networks. These unidirectional networks are presented as "baseline" cases for judging the success of the networks with feedback. The focus of this chapter is the development

of ten feedback architectures utilizing SAM cells to achieve temporal behavior.

In addition to the baseline cases and the SAM networks, studies were also performed on a set of networks having internal sequential unidirectional topology with feedback implemented via an external (non-neural) memory. This variant architecture is offered as an alternative to the purely neural SAM architectures.

For each of the architectures, the number of neurons and the number of connections was varied to determine the sensitivity of training convergence on network size. The following diagrams describes the general architectures of the networks examined in this study. Within each architecture the number of neurons in the hidden layer was varied which also varied the total number of connections contained in the network. In the diagrams the arrows describing connection paths indicate multiple connections to and from every neuron in the connected layers:

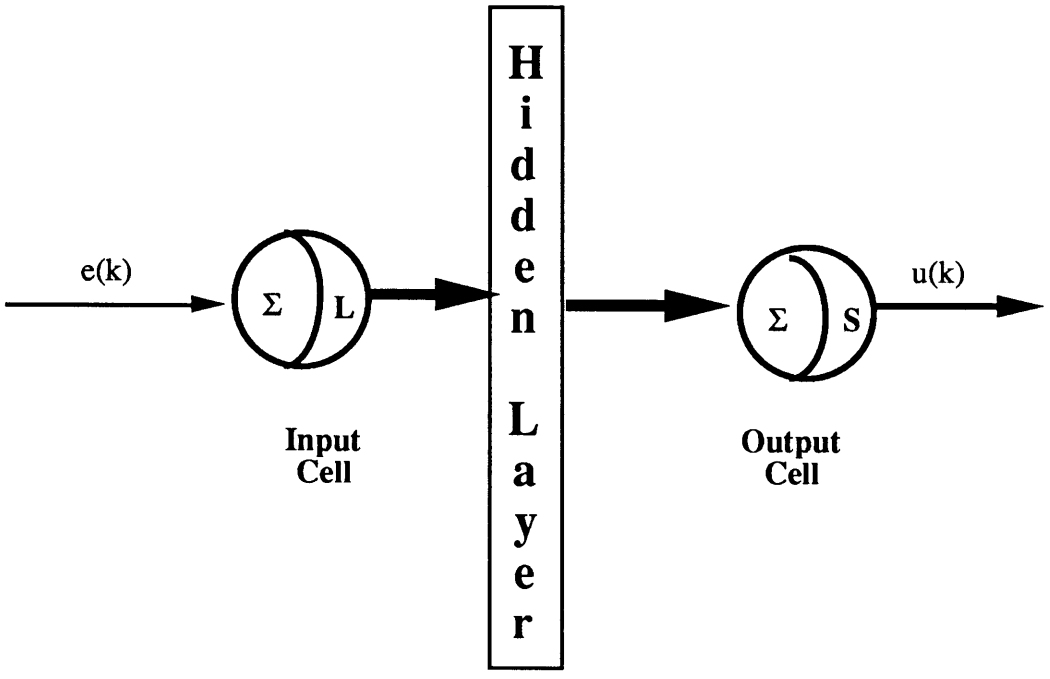


Figure 3.1 - General Form of the Sequential Unidirectional Architecture (No Feedback)

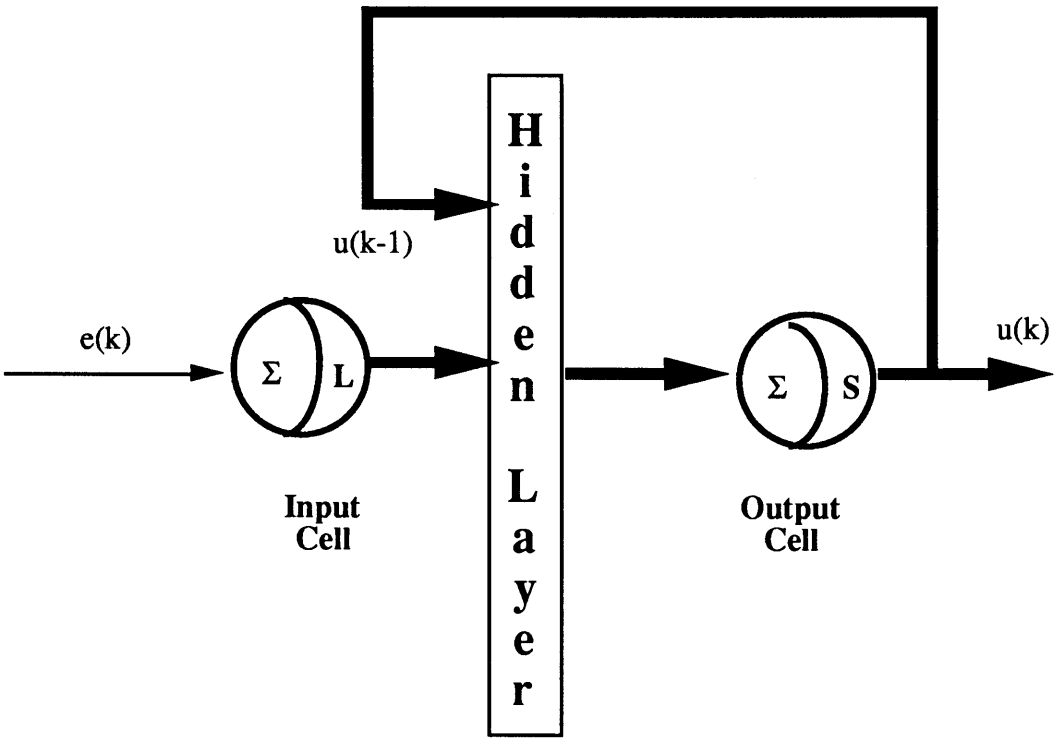


Figure 3.2 - General Form of the SAM Architecture with Output Feedback

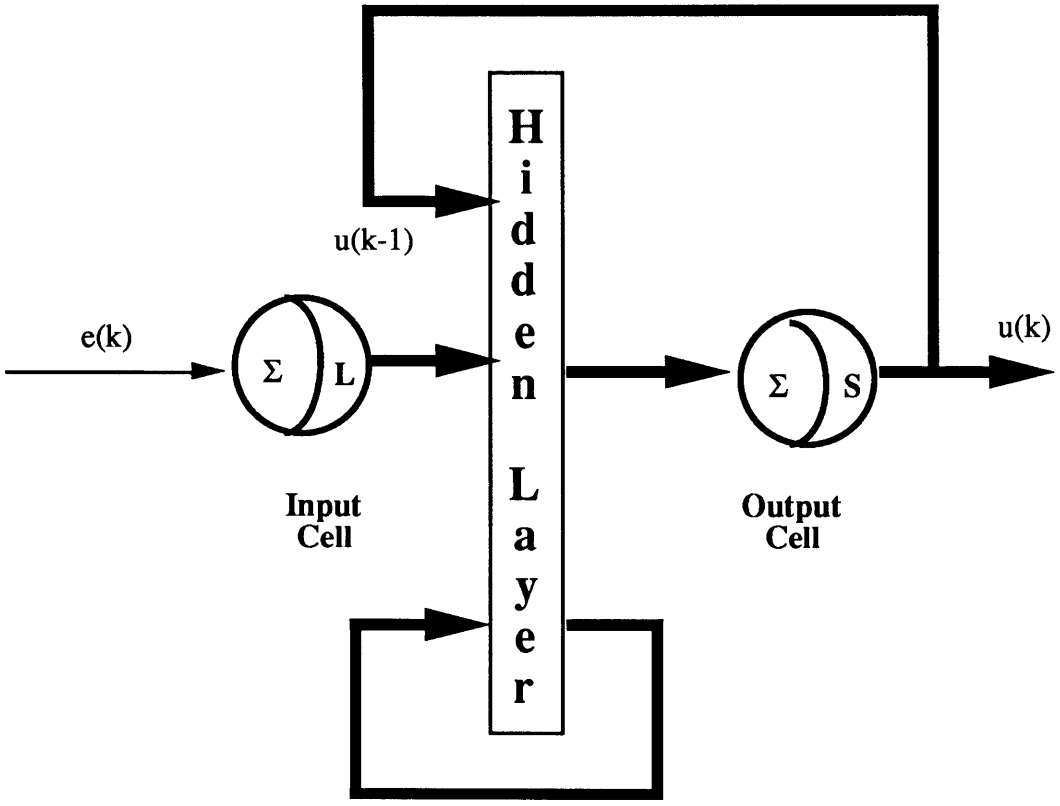


Figure 3.3 - General Form of the SAM Architecture with Global Feedback

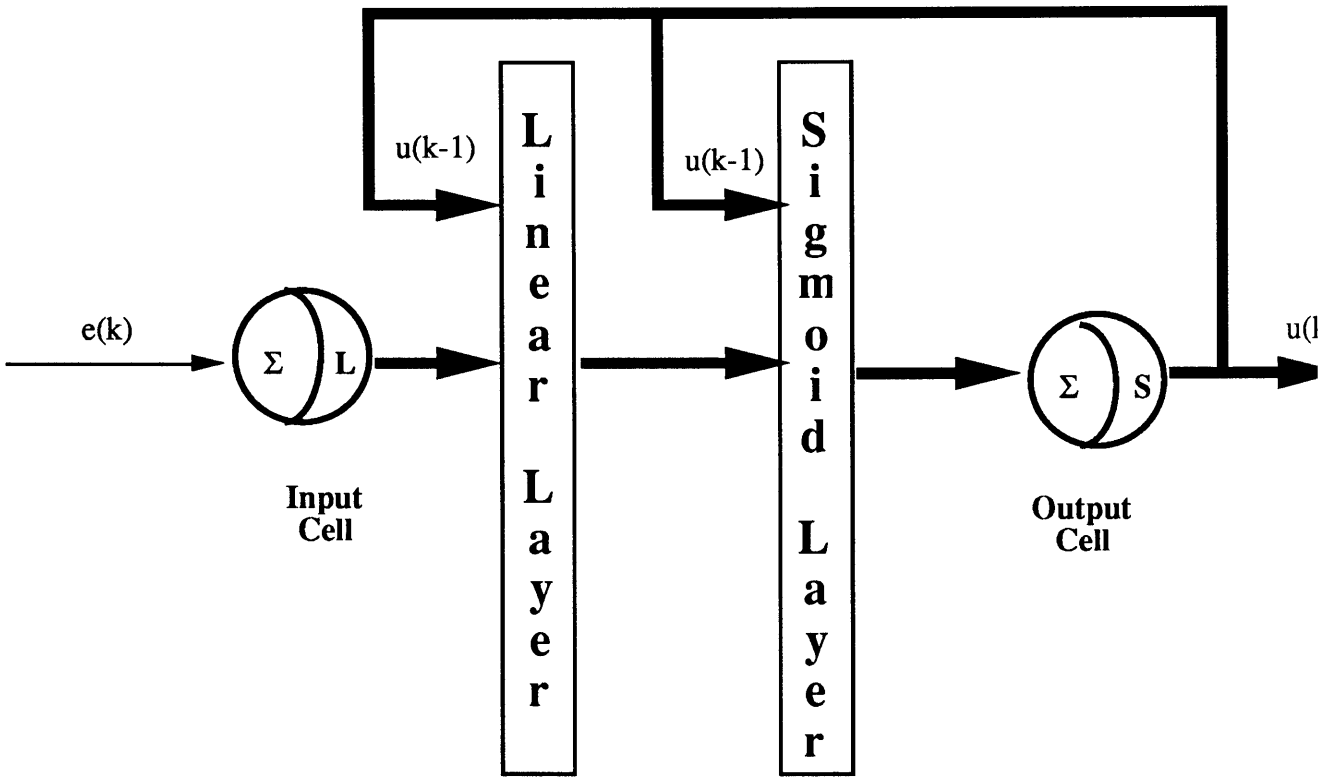


Figure 3.4 - General Form of the Hybrid SAM Architectures with Output Feedback

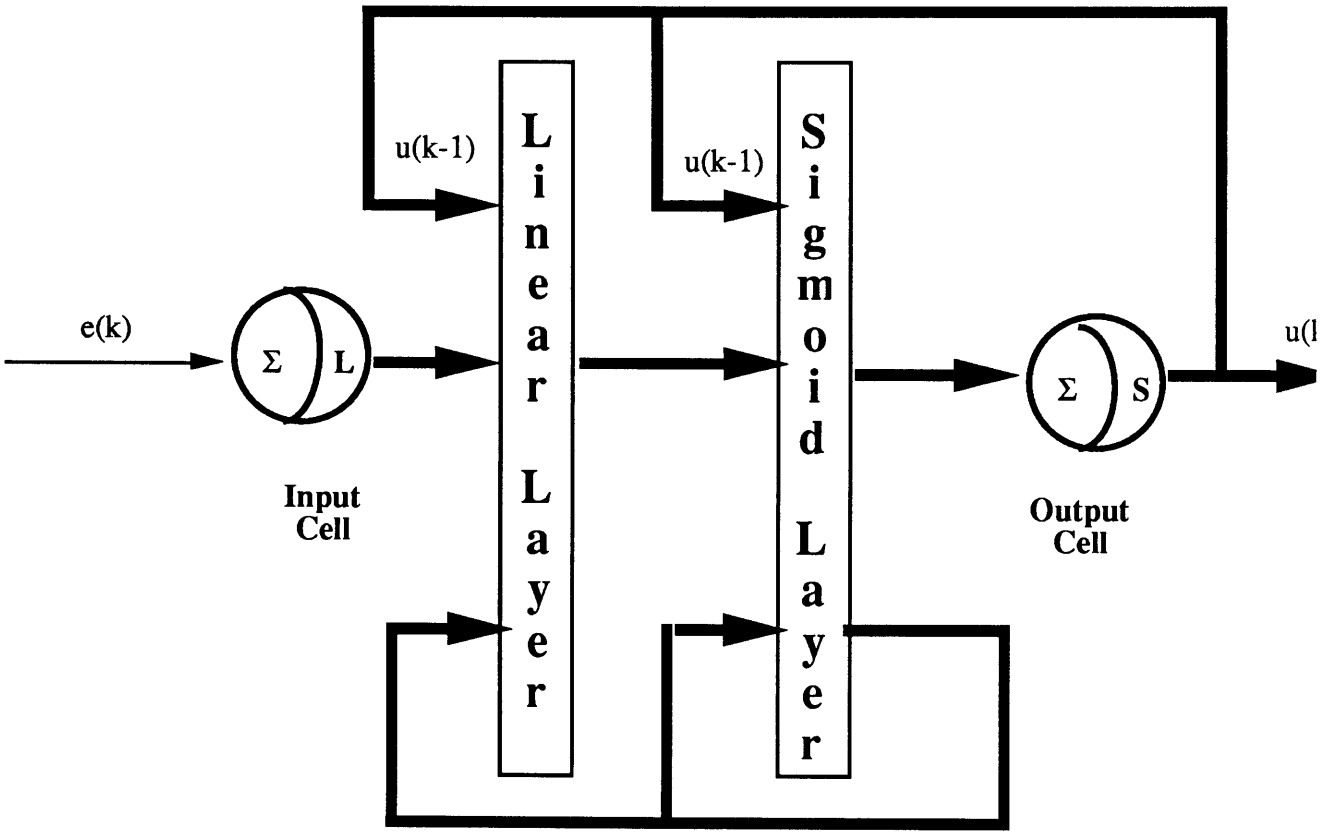


Figure 3.5 - General Form of the Hybrid SAM Architectures with Global Feedback

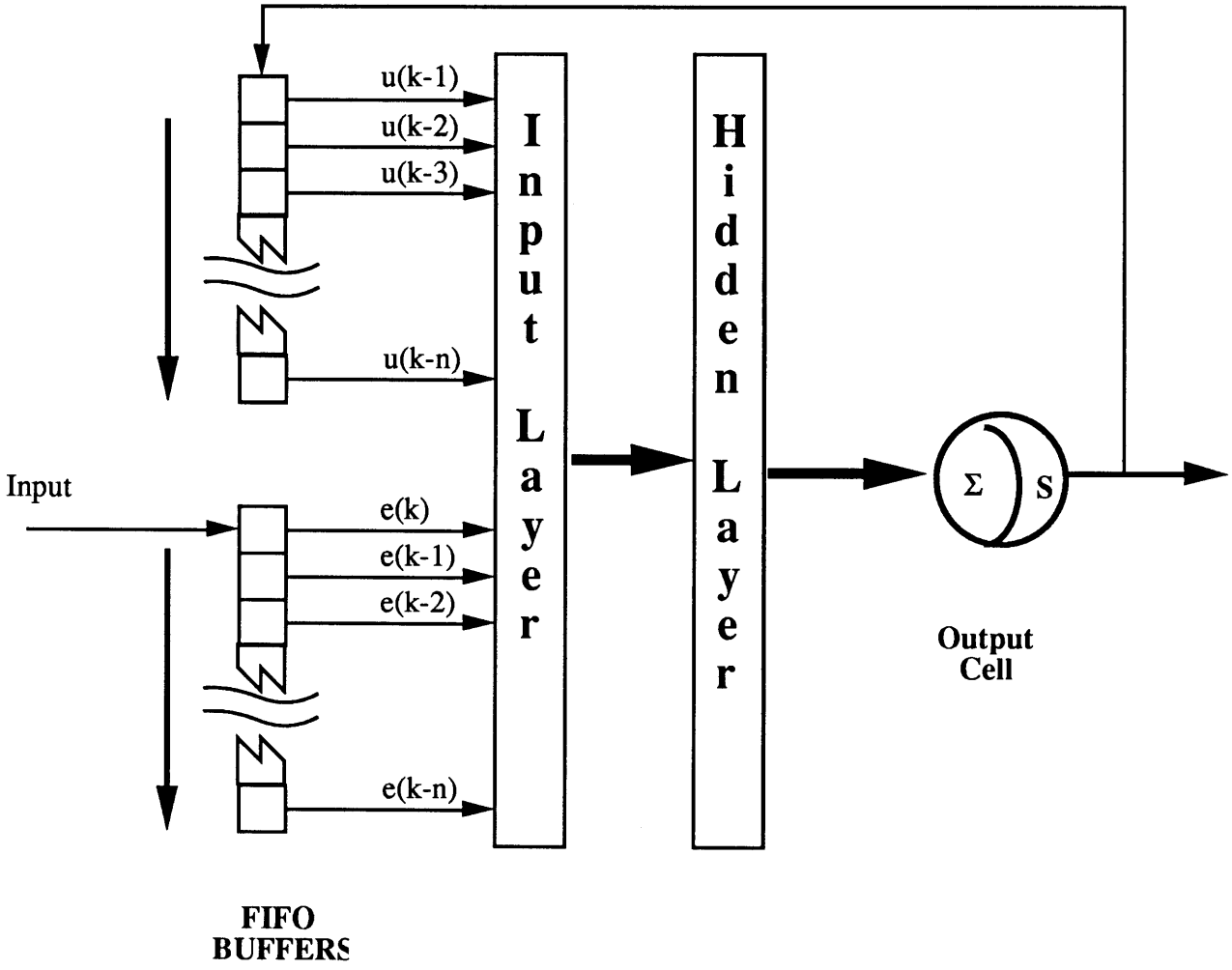


Figure 3.6 - General Form of the Variant Architectures with External Feedback

In all of the networks, two types of neural cells were utilized in the hidden layer: neurons with linear activation functions and neurons with sigmoidal activation functions. For record keeping purposes, each network architecture was assigned a unique label. This label is composed of four parts: the type of neural cells in the hidden layer (SAM layer), the base topology, the number of cells in the hidden layer, and a bias flag.

The following label: **LIN-OUT-20**

indicates a network with twenty (20) neurons in the hidden layer, the hidden neurons all utilize linear activation functions, and the output is fed back into each of the SAM cells in the hidden layer.

Another example: **SIG-GL-20-(B)**

indicates a network with twenty (20) neurons in the hidden layer and the hidden neurons utilize sigmoidal activation functions. The 'GL' label stands for GLOBAL feedback and indicates that signals from all neurons (output and hidden) are fed back into each of the SAM cells in the hidden layer. The '(B)' at the end of the label indicates that every hidden cell in the network is connected to a bias cell whose output is always one (1.00).

The activation function for the neurons defined as LINEAR have a slope of +1/2. The activation function for the neurons defined as SIGMOID have a range from 0 to +1 with a maximum slope (at OUT = +1/2) of +1. The sigmoidal output function takes the following form:

$$O_j = F(N_j) = \frac{1}{(1 + e^{-N_j})}$$

The inputs [e(k)] have a range of 0 to +1. With the coefficients for the difference equation shown above, the range of the training outputs will

be between 0 and +1, equivalent to the output range of the sigmoidal neurons.

The output layer of each network was comprised of a single neuron with a sigmoidal activation function. The sigmoidal output neuron was used for two reasons:

1. Networks with linear output neurons would usually diverge during training. The sigmoid output neuron allowed an architecture to have a linear hidden layer when output feedback was used.

2. The bounded output of a sigmoid matched well with the limitations placed on the output of a control system with respect to the limitations of a typical actuator like a servo-motor.

The following table describes the set of unidirectional and SAM architectures that were simulated. The network types were defined based upon the relative mix of sigmoid and linear neurons in the hidden layer(s) and the number of feedforward and feedback connections. The input layer was always composed of a single linear neuron and the output layer was always composed of a single sigmoid neuron. For each network type two or three networks of varying size were generated and tested. The table below also contains the steady-state average error which concisely describes the relative performance of the particular network; the lower the number, the better the performance.

	Size of Network		
Network Type	Small	Medium	Large
LIN-NONE	5 linear neurons	20 linear neurons	NOT GENERATED
No bias	10 connections	40 connections	
No feedback	Avg. Err: 0.16522	Avg. Err: 0.16974	
SIG-NONE	5 sigmoid neurons	20 sigmoid neurons	NOT GENERATED
No bias	10 connections	40 connections	
No feedback	Avg. Err: 0.07840	Avg. Err: 0.16970	
LIN-OUT	5 linear neurons	20 linear neurons	40 linear neurons
No bias	15 connections	60 connections	120 connections
Output feedback	Avg. Err: 0.13476	Avg. Err: 0.13392	Avg. Err: 0.13231
LIN-OUT-(B)	5 linear neurons	20 linear neurons	40 linear neurons
With bias	20 connections	80 connections	160 connections
Output feedback	Avg. Err: 0.03167	Avg. Err: 0.03151	Avg. Err: 0.03113
SIG-OUT	5 sigmoid neurons	20 sigmoid neurons	40 sigmoid neurons
No bias	15 connections	60 connections	120 connections
Output feedback	Avg. Err: 0.0363	Avg. Err: 0.0357	Avg. Err: 0.0369
SIG-OUT-(B)	5 sigmoid neurons	20 sigmoid neurons	40 sigmoid neurons
With bias	20 connections	80 connections	160 connections
Output feedback	Avg. Err: 0.0278	Avg. Err: 0.0270	Avg. Err: 0.0268
SIG-GL	5 sigmoid neurons	20 sigmoid neurons	NOT GENERATED
No bias	40 connections	460 connections	
Global feedback	Avg. Err: 0.02626	Avg. Err: 0.01405	
SIG-GL-(B)	5 sigmoid neurons	20 sigmoid neurons	NOT GENERATED
With bias	45 connections	480 connections	
Global feedback	Avg. Err: 0.02097	Avg. Err: 0.00779	
HYB-OUT	3 sigmoid neurons	10 sigmoid neurons	20 sigmoid neurons
No bias	3 linear neurons	10 linear neurons	20 linear neurons
Output feedback	18 connections	60 connections	120 connections
	Avg. Err: 0.0329	Avg. Err: 0.0328	Avg. Err: 0.0346

Table 3.1 - Unidirectional and SAM Networks Generated

	Size of Network		
Network Type	Small	Medium	Large
HYB-OUT-(B) With bias Output feedback	3 sigmoid neurons 3 linear neurons 24 connections Avg. Err: 0.0327	10 sigmoid neurons 10 linear neurons 80 connections Avg. Err: 0.0284	20 sigmoid neurons 20 linear neurons 160 connections Avg. Err: 0.0358
HYB-GL No bias Global feedback	3 sigmoid neurons 3 linear neurons 27 connections Avg. Err: 0.0157	10 sigmoid neurons 10 linear neurons 160 connections Avg. Err: 0.0174	NOT GENERATED
HYB-GL-(B) With bias Global feedback	3 sigmoid neurons 3 linear neurons 30 connections Avg. Err: 0.0167	10 sigmoid neurons 10 linear neurons 180 connections Avg. Err: 0.0166	NOT GENERATED

Table 3.1 (cont'd) - Unidirectional and SAM Networks Generated

3.2 Parameters of the Neural Simulation and Learning Algorithm

The task at hand is to emulate the behavior exhibited by the following second order linear difference equation:

$$\mathbf{u}(k) = c_0\mathbf{e}(k) + c_1\mathbf{e}(k-1)$$

where $c_0 = 0.7$

$c_1 = 0.3$

The above coefficients were arbitrarily chosen with the following constraints in mind:

1) The desired output range of the network should match the output range attainable by the network. In this case, that range is from 0 to 1.

2) The coefficient of the (k-1)th term should be significant with respect to the coefficient of the kth term in order to guarantee that temporal errors could be distinguished from the relatively small convergence errors and the inherent sigmoidal errors.

It should be noted that since the sigmoidal function is non-linear the output of these networks will never be perfect; there will be some discrepancy caused by the non-linear nature of the output neuron's sigmoid attempting to match the linear character of the difference equation. Therefore, it is important in viewing the results of the following experiments to examine the relative behavior of the different networks.

The input sequence used in training is generated using a pseudo random number generator[1]. Each network is presented with several hundred thousand inputs in a training sequence. The performance of each network is judged by two criterion. The first is the average

[1] Specifically, the **rand()** function supplied with MS-DOS Microsoft C version 5.1.

magnitude of its output error. This average is taken at each point using the 100 previous presentations. **Therefore, the error at k=100 is the average of the magnitudes of the errors for presentations 1 through 100**. The second criterion was the maximum error magnitude in the previous 100 presentations.

Evaluations can be based upon three standards:

1) Accuracy of the network (average error, maximum error)
2) Number of presentations required to achieve steady-state behavior
3) Number of connections in the network

The first two standards judge the ability of each network to learn a function when given a certain number of training samples. The third standard provides an adjustment that takes into account the relative training times required under a sequential simulation[1].

All dendrites were initialized with random weights between -0.05 and +0.05. The SAM connections were adjusted with a gain bias[2] of 10. For all simulations the training rate, η , was set at 0.5 and the momentum term, α, was set to 0.9.

In some cases the networks with linear nodes diverged from a satisfactory solution and caused floating point overflow errors on the simulation system[3]. This behavior was also observed in [Sanner, 1989]. In these cases, it was observed that the divergence was coupled with closed feedback loops composed entirely of neurons with linear output functions. It was further observed that placing neurons with bounded

---

[1] If the artificial network was implemented with parallel processing, the computational penalty for many connections is lessened since training of these connections can occur simultaneously.

[2] See chapter 2 for a definition of gain bias.

[3] These usually occur when an operation produces a floating point number with too great a magnitude. This is usually taken as an indication that the back-propagation algorithm has become unstable.

output functions (sigmoids) into all closed feedback loops prevented divergence during training.

3.3 Analysis of Error Induced by Sigmoidal Output Neuron

As discussed previously, the sigmoidal output neuron will introduce some small discrepancy between the desired output and the actual output when one is attempting to emulate a linear function, even in cases where the neural weights have been adjusted to optimal values. For a given network with a sigmoidal output, the average magnitude of this error should be dependent on the distribution of output values. When expected output values fall close to (or beyond) the limits of the sigmoid's possible output range, a neuron with a sigmoidal output function would be hard pressed to accurately produce the full range of expected values in a linear fashion. This is due to the fact that the sigmoid function is highly non-linear near its boundaries. While it is almost linear in the middle of its range, the slight non-linearity will cause errors there as well. To quantify the magnitude of this sigmoidal error, several tests were run using the following two layer network.

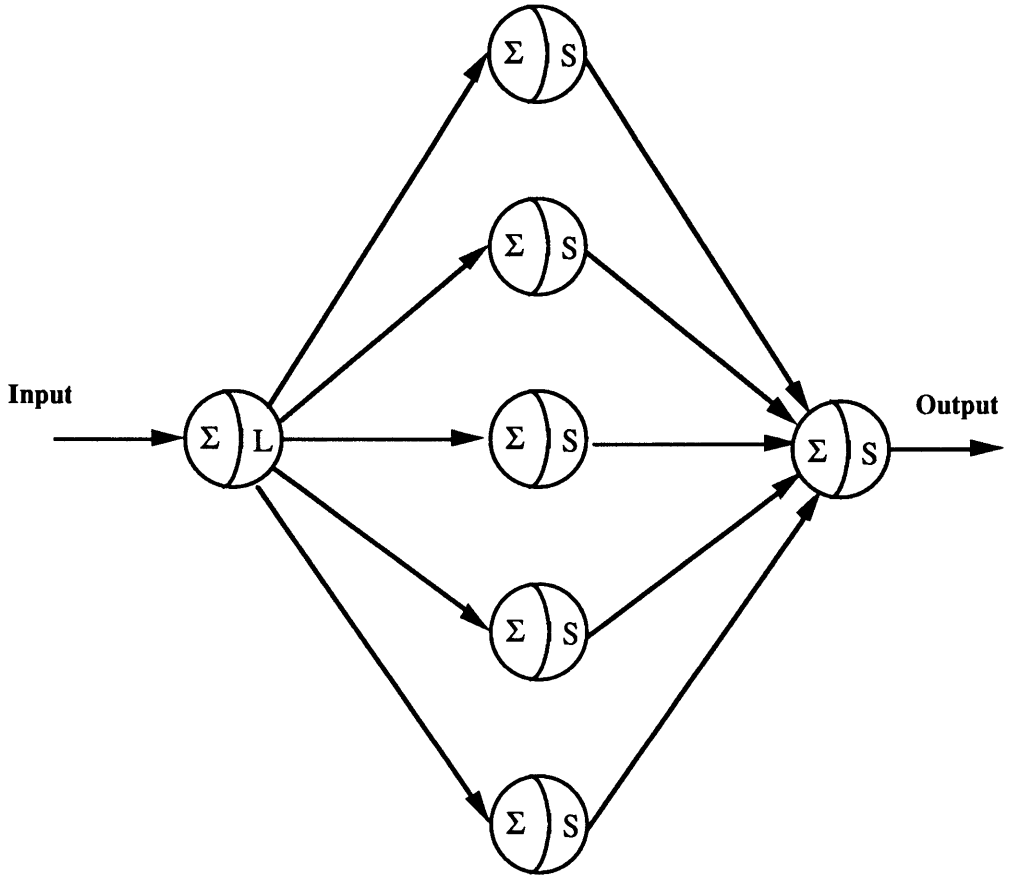


Figure 3.1- Network Used to Measure Inherent Sigmoid Induced Error

The above network was trained to recognize the following finite difference equation:

```
u(k) = e(k)      ; Output equals input
```

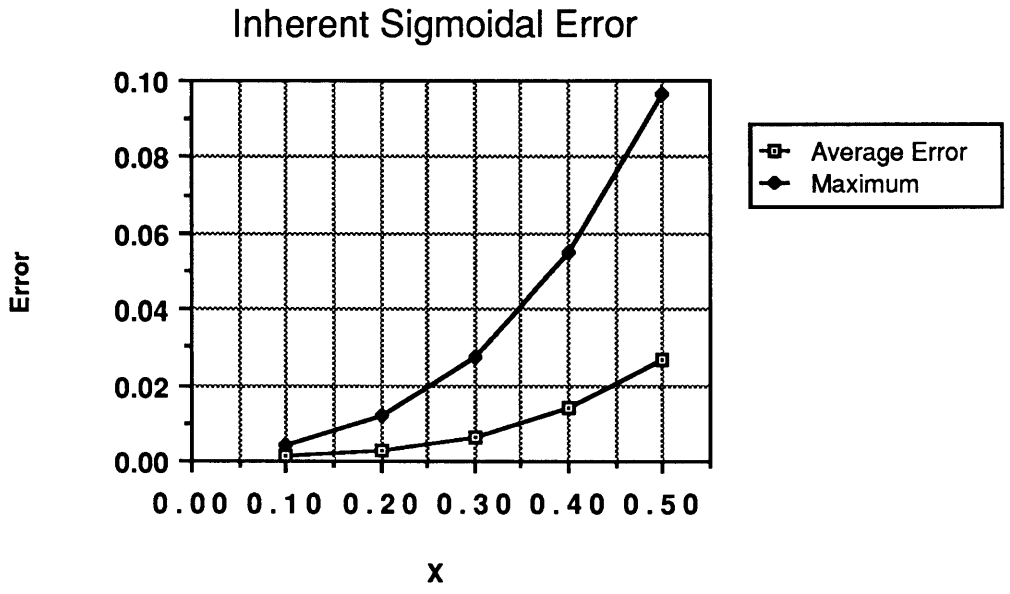
The same learning algorithm and neural parameters documented earlier in this section were here as well. Five simulations were executed with the range of possible inputs being varied between runs. The input range was centered about 0.5 such that the input range could be described with a single parameter **x**:

```
(0.5 - x) < e(k) <  (0.5 + x)
```

The network was trained with sequences of x=0.1, x=0.2, x=0.3, x=0.4, x=0.5. These correspond to the following input ranges:

x	e(k)		
0.1	0.4	<-->	0.6
0.2	0.3	<-->	0.7
0.3	0.2	<-->	0.8
0.4	0.1	<-->	0.9
0.5	0.0	<-->	1.0

Within a given range the presentation inputs were generated randomly. For each of the five ranges the network was trained for 50,000 presentations. The average error magnitude and the maximum error magnitude for each case were recorded after training was completed. The following chart conveys the relationship between the inherent sigmoidal errors and the parameter **x**. Data is given for both average observed error and maximum observed error.



For purposes of comparison with the test results in the remainder of this chapter, one should consider the error value at **x**=0.5 as being most significant, since the networks to be demonstrated were all trained using presentation sets where the range of output conformed to **x** being 0.5.

At x=0.5 the following errors were observed:

Average Error	.0265
Maximum Error	.0967

The above numbers are considered **nominal expected behavior** for networks with sigmoidal output neurons. These results are used for comparison to results in the rest of this chapter. Any network whose steady-state errors were comparable or better than the nominal expected behavior could be considered successful at converging on the desired behavior. Since the network used to determine the nominal errors had only five neurons and ten connections, it is possible for a network with more neurons or more connections to achieve better than "nominal" results.

3.4 Convergence Results of Target Neural Architectures

The sequential unidirectional networks and the SAM networks listed in table 3.1 were tested as described in sections 3.1 through 3.3. The convergence history and the steady-state errors were recorded for each network as a basis judging both absolute and relative success of the various architectures and network sizes. Tables and charts detailing the convergence histories of the various architectures can be found in Appendix A.

The results can be summarized as follows:

1) All the sequential unidirectional networks, with one noted exception, performed poorly in attempting to emulate the finite difference equation. This result was expected since these networks had no mechanism for representing past state.

2) Many of the SAM networks produced clearly positive results in their attempts at emulating the difference equation. Most notably, the networks using sigmoid neurons in the hidden layer and global feedback connections (the SIG-GL group) performed well, with the performance improving as network size increased.

3) The networks utilizing linear neurons in the hidden layer without bias connections performed remarkably poorly. However, the addition of bias connections to this architecture produced positive results.

4) No benefit was discovered in using a hybrid architecture (mixed linear and sigmoid neurons in the hidden layer) when output feedback alone is used. Empirical data indicates that in these cases the hybrid architecture does not offer any benefit over a purely sigmoidal architecture with an equal number of neurons and connections.

5) Utilizing a hybrid architecture with global feedback did provide some benefit over a global architecture with only sigmoid neurons in the hidden layer.

The following pages contain more specific commentary on the individual architecture groups that were tested:

Linear with no feedback (LIN-NONE):

The large values observed for steady-state average error and steady-state maximum error agree with what was expected for this architecture. Without any kind of feedback connections, these networks were unable to form an impression of the second term in the difference equation. Comparing these results with those for the networks with feedback connections, it is clear that the existence of feedback has a significant positive effect on the behavior of the networks.

Sigmoid with no feedback (SIG-NONE):

As with the linear case, the observed errors are large and agree with what is expected of a network with no feedback.

The results for SIG-NONE-5 represent a significant anomaly. Somehow, this network performed noticeably better than the other non-feedback networks. While the results for this network are still rather poor, they are significantly better than the results for the other non-feedback networks. The SIG-NONE-5 network was reexamined with greater scrutiny. This examination indicated that the observed behavior is consistent and was not a random aberration. Despite the careful verification, this researcher believes that the most likely explanation is an error somewhere in the setup of the simulation. It is also possible that this architecture somehow "learned" to predict the

random number sequence in such a way that it was able to achieve the observed behavior.

Linear with output feedback (LIN-OUT and LIN-OUT-(B)):

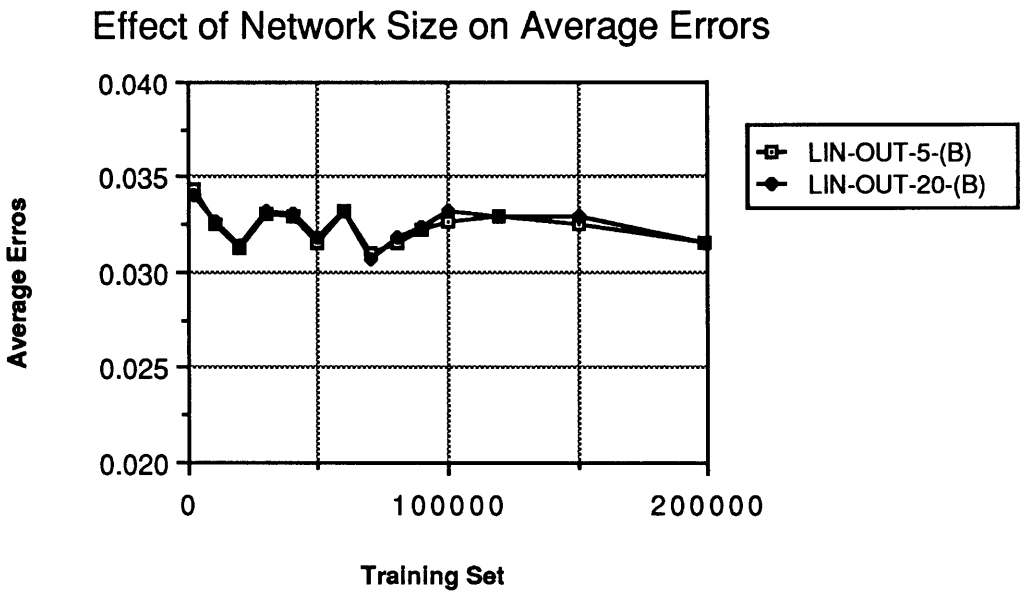
Three things are clear when one examines the steady-state results for networks with linear hidden neurons and simple output feedback:

1) Networks of this type without bias produced results that were close to the theoretically worst results possible regardless of network size. This indicates an overall failure for this particular architecture.

2) Networks of this type that included bias connections produced results that were relatively close to the nominal values established by the test on inherent sigmoidal error. This indicates a general success for this particular architecture.

3) Network size (above 5 hidden neurons) appears to have no effect on the behavior of any of the architectures utilizing linear hidden neurons and simple output feedback.

In the chart showing the effect of network size on training convergence, it is apparent that the training paths were almost identical. It was hypothesized that the limited feedback (output only) is responsible for this architecture's reduced sensitivity to hidden layer size. There simply is not enough information coming back to the hidden layer that would require processing.



From the training histories, it also apparent that the convergence of the LIN-OUT-n-(B) architectures are rather quick. They all converged to within a few percent of their steady-state behavior in less than 2000 presentations.

Sigmoid with output feedback (SIG-OUT and SIG-OUT-(B)):

As with the LIN-OUT architecture, the bias connections had a greater effect on the performance of the SIG-OUT architecture than the network size. Unlike the LIN-OUT architecture, both the biased and unbiased networks achieved reasonably good behaviors.

1. For the networks with bias connections, the steady-state average and maximum errors were around (0.027/0.095), which were almost exactly the nominal expected errors (0.027/0.097).

2. For the networks without bias connections, the steady-state average and maximum errors were around (0.036/0.160), which were near the nominal expected errors (0.027/0.097).

3. For both bias and non-bias networks, the SIG-OUT architecture did noticeably better than the LIN-OUT architecture.

Sigmoid with global feedback (SIG-GL and SIG-GL-(B)):

The results for the SIG-GL architectures are very positive. All of the networks based on these architectures performed better than nominally expected. The SIG-GL-20-(B) produced steady-state average and maximum errors of (0.008/0.059), which are only a fraction of the nominally expected values of (0.027/0.097). This indicates that this particular network would do an exceptional job of emulating the finite difference equation in question.

This superior performance is achieved at a very high computational cost. The SIG-GL architecture requires $(\mathbf{n}^2 + 3\mathbf{n})$ connections vs. $(3\mathbf{n})$ connections for the comparable SIG-OUT or LIN-OUT architectures. The SIG-GL-(B) architecture requires $(\mathbf{n}^2 + 4\mathbf{n})$ connections vs. $(4\mathbf{n})$ connections for the comparable SIG-OUT or LIN-OUT architectures. The following table of connections illustrates this

point. The sequential computational requirement is approximately proportional to the number of connections in the network.

Size and Type of Hidden Layer	# Connections (Output Feedback)	# Connections (Global Feedback)
5	15	40
5-(B)	20	45
20	60	460
20-(B)	80	480
40	120	1720
40-(B)	160	1760

Intuitively, these results make sense. As a network becomes more complicated (indicated by the number of connections) its ability to reduce steady-state error increases, but the computational cost does not scale linearly with the benefit. If one compares the SIG-OUT-20-(B) network with the SIG-GL-20-(B) architecture, one will find the ratio of steady-state average errors to be about 3:1 versus a ratio of 6:1 for the number of connections. While there was no recognizable pattern to these ratios for other comparative cases, it was clear in all cases that within a given network architecture a doubling of accuracy required greater than a doubling of the number of connections.

Hybrid with output feedback (HYB-OUT and HYB-OUT-(B)):

The purpose of the hybrid architectures was to mix linear and sigmoid neurons in the hidden layers with the hope that such a mix would cause complementary performance. In the case of the hybrid architecture utilizing output feedback there did not appear to be any such performance improvement. Overall, the performance for the HYB-OUT networks was relatively positive with all the tested variations coming reasonably close to the nominal expected steady-state average and maximum errors (0.027/0.097), but no significant performance breakthroughs observed were observed. The hybrid networks with bias and output feedback performed nearly equal to the linear and sigmoid networks with bias and output feedback. Overall, the hybrid networks with output feedback did not have any remarkable or unique performance characteristics.

The HYB-OUT architectures didn't display any strong dependence on either bias connections or network size. The steady-state errors actually increased slightly going from 20 hidden cells to 40 hidden cells -- an unusual occurrence.

HYB-GL and HYB-GL-(B): Hybrid with global feedback

Where the hybrid architectures with output feedback proved to be a disappointment, the ones with global feedback showed extreme promise. Amongst the networks tested that had internal feedback, the HYB-GL architectures did consistently well. In particular, the HYB-GL-6 and the HYB-GL-6-(B) networks displayed the best performance of all such networks when network size (number of connections) is taken into account. For example, the HYB-GL-6 network achieved a steady-state average error of 0.0157 using 27 connections, while the SIG-GL-20 network achieved a steady-state average error of 0.01405 using 460 connections.

Unfortunately, the performance of the HYB-GL architectures did not appear to scale with network size; the average steady-state errors were about the same for all the HYB-GL networks tested independent of network size or the existence of bias connections. However, there is significant promise here to warrant further investigation into this particular architecture. This researcher believes that the HYB-GL architectures potentially provide superior performance versus computational cost when compared with any of the purely neural solutions investigated in this work.

3.5 Variant Architecture Using Random Access FIFO Buffer and a Sequential Unidirectional Network

The SAM networks discussed in the previous sections attempted to achieve desired temporal behavior by providing internal feedback connections within the network. While the above systems provide a "purely neural" solution, it seems that for the purpose of emulating finite difference equations their performance and predictability were not always satisfactory.

The networks can be given a better sense of past state by providing them with the exact information they need. It is possible to create an external memory buffer that "remembers" the previous inputs and outputs of the network. This buffer can be implemented as a FIFO (first-in first-out) structure that will automatically cycle values into the correct inputs of a network. The following is a diagram of neural network utilizing this variant architecture.

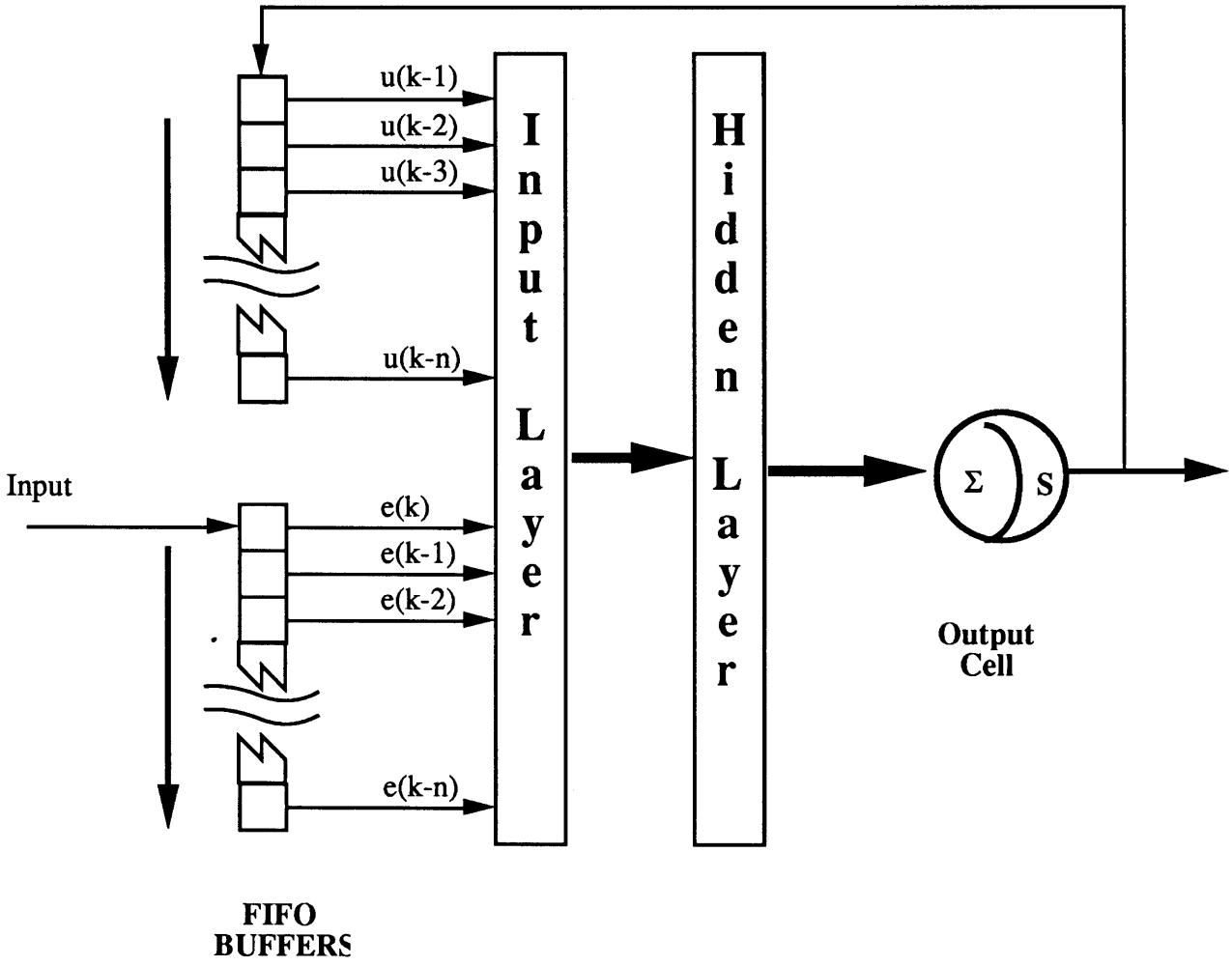


Figure 3.7- Sequential Unidirectional Network with External Memory

The only drawback to this variant architecture is that the number of terms in the finite difference equation needs to be known prior to network setup. Compared to the limitations observed in SAM architectures attempting to emulate higher order equations (presented in the next chapter), the setup requirements for the variant architecture appear to be minor.

Another benefit of the variant architecture is that the number of connections is kept relatively low, when compared to some of the SAM networks.

3.6 Comparison of the Variant Architecture with the SAM Architectures

Several versions of the variant architecture were generated and tested in the same manner as the sequential unidirectional and SAM architectures. The following table shows the high degree of success that was achieved. Note the extremely low steady-state average errors for the smaller networks.

	Size of Network		
Network Type	Small	Medium	Large
SIG-VAR	5 sigmoid neurons	20 sigmoid neurons	40 sigmoid neurons
No bias	10 connections	40 connections	80 connections
External feedback	Avg. Err: 0.0165	Avg. Err: 0.0160	Avg. Err: 0.0160
SIG-VAR-(B)	5 sigmoid neurons	20 sigmoid neurons	40 sigmoid neurons
With bias	15 connections	60 connections	120 connections
External feedback	Avg. Err: 0.0032	Avg. Err: 0.0029	Avg. Err: 0.0090

Table 3.2 - Variant Networks Generated

It is clear from this data that the variant architecture performs much better than most of the isolated SAM networks. Additional observations made in chapter 4 of this thesis tend to indicate that the variant architecture is also far more robust than any of the purely neural solutions.

It should be noted that with the variant architecture the presence of bias connections produced significant benefit. In fact, one should notice that bias connections almost universally produce the most benefit at the least cost (compared to adding more neurons, more standard connections, or more SAM connections) in all of the tested architectures. For this reason, the utilization of bias connections is

highly recommended in any neural network attempting to emulate a function like the one presented in this thesis.

3.7 Comparative Training Convergence Properties of the Tested Architectures

In summary the following observations were made:

1) The variant architecture with bias connections performed much better than nominal for all examined architectures. In fact, this general architecture was judged to be the best overall.

2) The global feedback architecture with all sigmoidal neurons and bias connections produced the best pure neural solution to the presented problem. This was done, however, at a high cost in terms of the number of required connections.

3) The output feedback architectures, with all linear neurons and no bias connections produced very poor results which were comparable to those observed with networks having no feedback connections at all.

4) Almost all of the feedback architectures with the exception of those mentioned in #3, performed much better than the non-feedback networks; many performed better than nominally expected.

Chapter 4
More Properties of SAM Networks Emulating Higher Order Linear Digital Filter

Chapter Summary

- Observations indicate that SAM networks may have difficulty emulating second order finite difference equations where the magnitude of the [k-1]th coefficient is greater than the magnitude of the [k]th coefficient.

- Unidirectional networks utilizing external memory appear to be insensitive to coefficient ratios.

4.1 Effects of Coefficient Ratios on Convergence

In the experiments carried out in chapter 3, a single finite difference equation was utilized

$$\mathbf{u}(k) = c_0\mathbf{e}(k) + c_1\mathbf{e}(k-1)$$

where $c_0 = 0.7$

$c_1 = 0.3$

It was found that for the networks with internal feedback loops, the ratio of the above coefficients had an effect on the training convergence of the network. As the ratio c_1/c_0 increased, the SAM networks found it harder and harder to converge on a satisfactory solution. This behavior was not observed in the variant architectures where convergence appeared to be independent of coefficient ratios.

These observations were made while carrying out the following tests. Two sample architectures were selected for testing. They were HYB-GL-6-(B) and LIN-VAR-5-(B). These networks were trained under the same parameters used in chapter 3.

The networks were tested to see how long it took them to converge to a solution for different finite difference equations. Convergence was judged as occuring when the maximum observed errors became less than 0.15. The finite difference equation was structured as before with

$$\mathbf{u}(k) = c_0\mathbf{e}(k) + c_1\mathbf{e}(k-1)$$

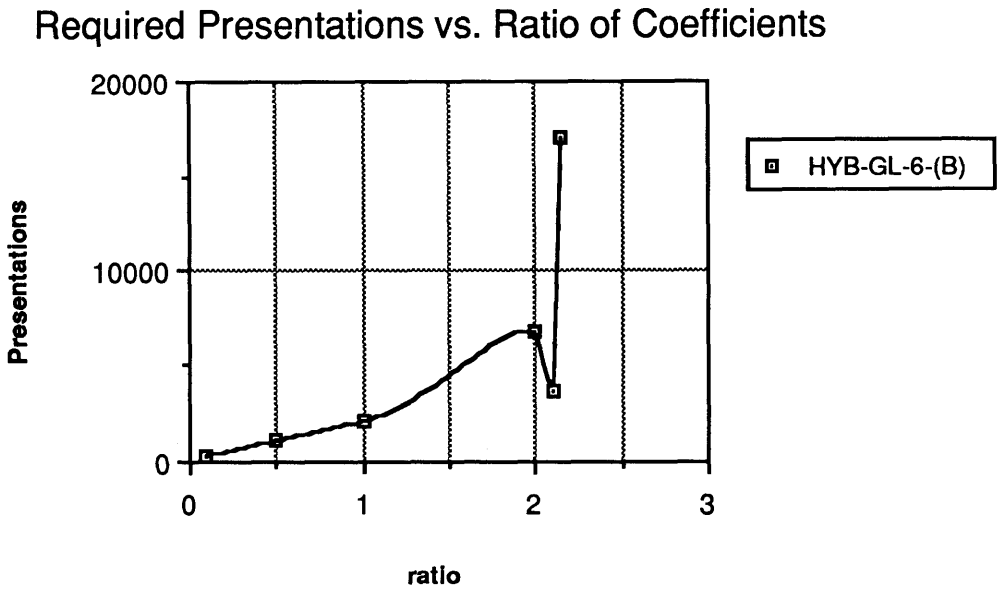
except the coefficients were presented with various ratios, using the following constraint equations.

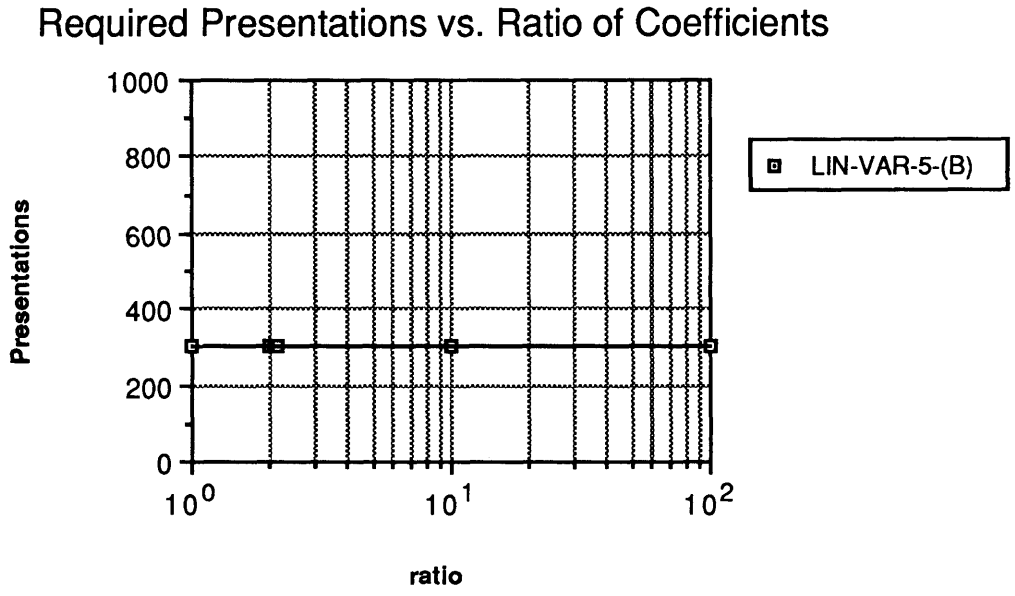
$$c_0 + c_1 = 1.0$$

$$\text{ratio} = \frac{c_1}{c_0}$$

As the ratio approaches zero, the finite difference equation begins to describe a first order equation. As the ratio approaches infinity, the equation begins to describe a pure time delay.

The following two charts show the convergence properties of the two architectures with respect to the coefficient ratio of the equation they are trying to emulate.





It is apparent from the above evidence that the SAM architecture found it increasingly more difficult to converge on a solution as the ratio of the coefficients went over 2.0. As a matter of fact, it failed to converge after 200,000 presentations on any equation with a ratio greater than 2.15. On the other hand, the convergence of the variant architecture demonstrated very little dependence on the ratio of coefficients. The variant architecture even succeeded in converging

on a pure time delay given by the following equation with a ratio of infinity.

$$\mathbf{u}(k) = 1.0 * \mathbf{e}(k-1)$$

These results clearly place a limit on the usefulness of the SAM architectures with respect to emulating digital filters (finite difference equations). The modified back-propagation algorithm as applied to the SAM networks does not provide a robust mechanism for extracting temporal behavior from neural networks.

Despite this, one must remember that the observed limitation is purely a function of the chosen training algorithm. It was shown in chapter 2 that the SAM networks had the ability to represent almost any finite difference equation; it was the process of learning to represent an arbitrary function via modified back-propagation that was flawed. This leaves open the possibility of someone developing a training algorithm that will be better at training networks with temporal capabilities.

Chapter 5
Summary of Conclusions and Suggested Further Research

5.1 Summary of Conclusions

1) It is possible for neural networks to emulate second and higher order finite difference equations by utilizing internal feedback connections. This implies a massive potential for the application of neural networks to an impressive list of problems including robotics, adaptive/learned control, speech recognition, and pattern recognition of moving objects.

2) The simple modification to the back propagation algorithm presented in this thesis was successful in some cases, but was not sufficiently robust for general application. Specifically, it was shown that a pure neural network utilizing feedback connections was capable of emulating arbitrary finite difference equations including time delays, but the proposed learning algorithm was only successful with a very limited set of applications.

3) Augmenting sequential unidirectional neural networks with external memory and external feedback connections apparently provides a robust platform for emulating second and higher order finite difference equations. This approach to achieving temporal behavior from a neural system is highly recommended despite the fact that it is not a "purely neural" solution.

5.2 Suggestions for Future Research

1) It should be possible to devise a new type of learning algorithm that is specifically designed to exploit the temporal capabilities of neural networks. Such an algorithm might take into consideration the fact that punish/reward information is not

always available in parallel with the synapse firings that initiated the result. There should be some way for a punish/reward event to coherently utilize a short history of synaptic firings instead of just the most recent firings.

2) There were some unusual results that were observed in this thesis. Most notably was the behavior of the SIG-NONE-5 architecture. This network should not have exhibited temporal behavior, but it somehow performed considerably better than all of the other sequential unidirectional networks. It was hypothesized in the body of chapter 3 that perhaps this network had learned to predict the output of the pseudo-random number generator. Even though this conclusion is not necessarily true in this particular case (personal error has not been entirely ruled out), the issue of neural networks predicting pseudo-random events is a worthwhile topic of investigation. Particularly, such research may lead to the conclusion that one must be very particular in choosing a pseudo-random number generator for the purpose of testing neural networks.

3) No attempt was made in this thesis to characterize the specific manner in which information flowed through the feedback paths in the various architectures. Such an investigation might provide some insight into the "how" and "why" of the behaviors observed. For now, this work merely provides a base of empirical observations that might serve as an initial roadmap for future researchers in this area.

Appendix 1
Architecture and Convergence Data for Target Neural Networks

Linear with No Feedback

LABEL: LIN-NONE-**n** ; No bias
LIN-NONE-**n**-(B) ; Bias node attached to all hidden cells

ARCHITECTURE: 3 Layers, Single Input/Output, **n** Hidden Cells

Connections: Layer 1 --> Layer 2
Layer 2 --> Layer 3

Cells: Layer 1 : Linear Input
Layer 2: Linear Hidden
Layer 3: Sigmoid Output

Number of Connections: **C(n)** = 2**n** (3**n** for bias)

STEADY-STATE RESULTS
(errors after 200,000 presentations)

	Average Error	Maximum Error
LIN-NONE-5	0.16522	0.48800
LIN-NONE-20	0.16974	0.49106

Sigmoid with No Feedback

LABEL: SIG-NONE-**n** ; No bias
SIG-NONE-**n**-(B) ; Bias attached to hidden cells

ARCHITECTURE: 3 Layers, Single Input/Output, **n** Hidden Cells

Connections: Layer 1 --> Layer 2
Layer 2 --> Layer 3

Cells: Layer 1 : Linear Input
Layer 2: Sigmoid Hidden
Layer 3: Sigmoid Output

Number of Connections: **C(n)** = 2**n** (3**n**+1 for bias)

STEADY-STATE RESULTS
(errors after 200,000 presentations)

	Average Error	Maximum Error
SIG-NONE-5	0.0784	0.2539
SIG-NONE-20	0.1697	0.4910

Linear with Output Feedback

LABEL: LIN-OUT-**n** ; No bias
LIN-OUT-**n**-(B) ; Bias attached to hidden cells

ARCHITECTURE: 3 Layers, Single Input/Output, **n** Hidden Cells

Connections: Layer 1 --> Layer 2
Layer 2 --> Layer 3
Layer 3 --> Layer 2

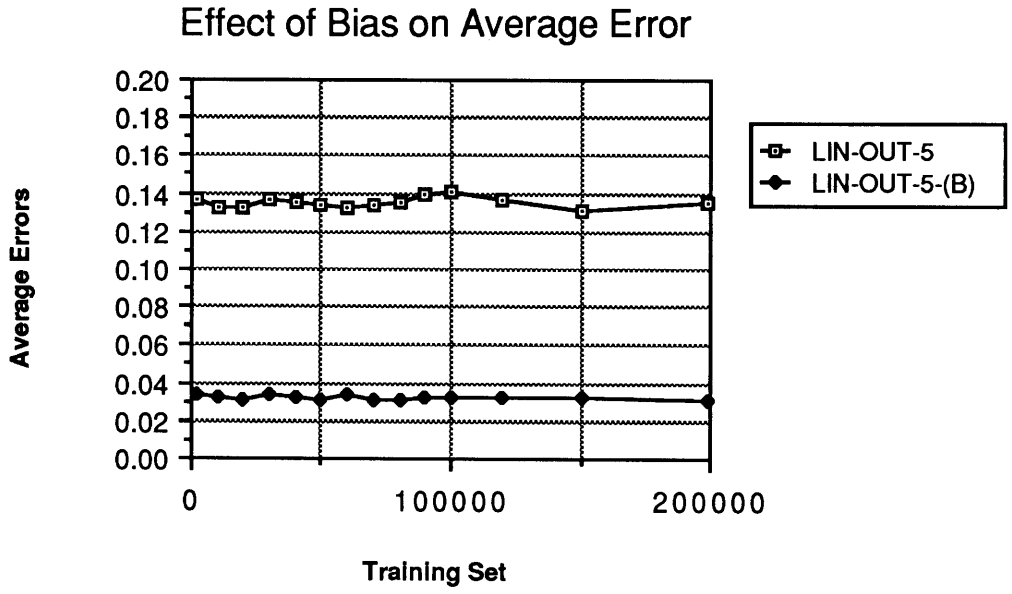
Cells: Layer 1 : Linear Input
Layer 2: Linear Hidden
Layer 3: Sigmoid Output

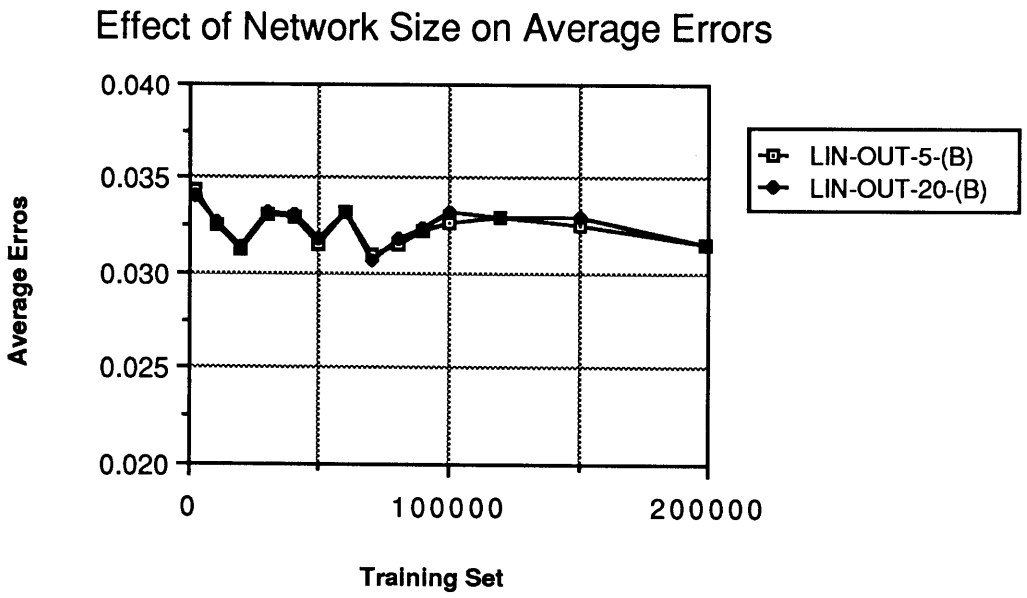
Number of Connections: **C(n)** = 3**n** (4**n** for bias)

STEADY-STATE RESULTS
(errors after 200,000 presentations)

	Average Error	Maximum Error
LIN-OUT-5	0.13476	0.40524
LIN-OUT-5-(B)	0.03167	0.14998
LIN-OUT-20	0.13392	0.40524
LIN-OUT-20-(B)	0.03151	0.14999
LIN-OUT-40	0.13231	0.40524
LIN-OUT-40-(B)	0.03113	0.14999

TRAINING RESULTS (LIN-OUT-n)





Sigmoid with Output Feedback

LABEL: SIG-OUT-**n** ; No bias
SIG-OUT-**n**-(B) ; Bias attached to hidden cells

ARCHITECTURE: 3 Layers, Single Input/Output, **n** Hidden Cells

Connections: Layer 1 --> Layer 2
Layer 2 --> Layer 3
Layer 3 --> Layer 2

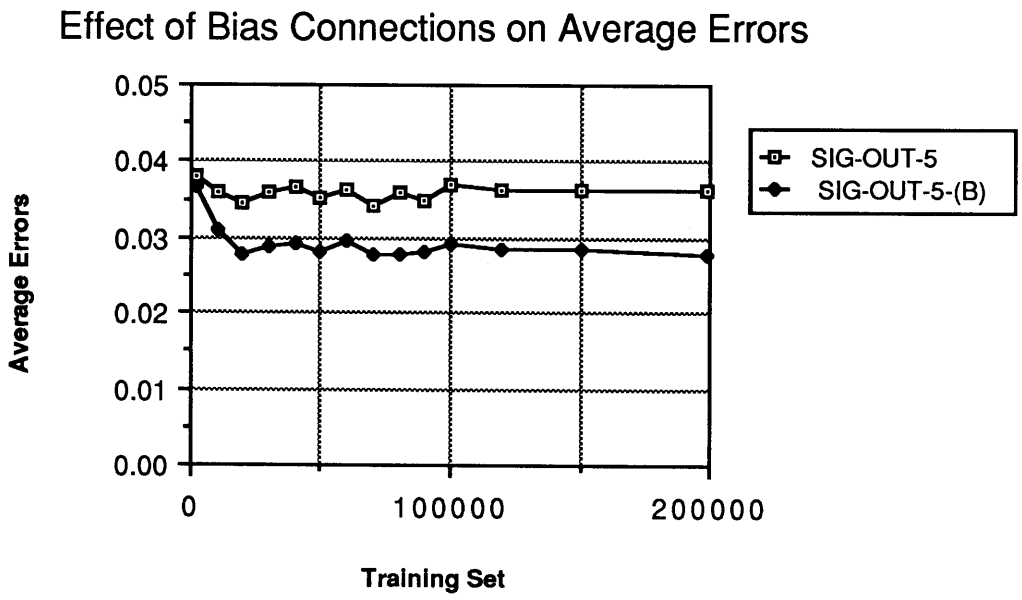
Cells: Layer 1 : Linear Input
Layer 2: Sigmoid Hidden
Layer 3: Sigmoid Output

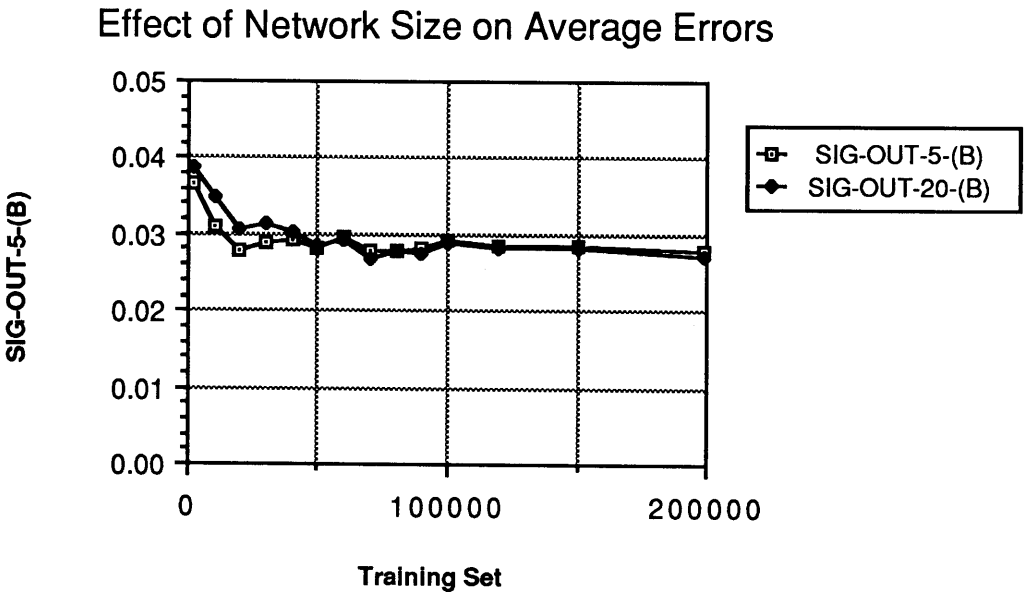
Number of Connections: **C(n)** = 3**n** (4**n** for bias)

STEADY-STATE RESULTS
(errors after 200,000 presentations)

	Average Error	Maximum Error
SIG-OUT-5	0.0363	0.1429
SIG-OUT-5-(B)	0.0278	0.1017
SIG-OUT-20	0.0357	0.1642
SIG-OUT-20-(B)	0.0270	0.0929
SIG-OUT-40	0.0369	0.1657
SIG-OUT-40-(B)	0.0268	0.0935

TRAINING RESULTS (SIG-OUT-n)





Pure Sigmoidal with Global Feedback

LABEL: SIG-GL-**n** ; No bias
SIG-GL-**n**-(B) ; Bias attached to hidden cells

ARCHITECTURE: 3 Layers, Single Input/Output, **n** Hidden Cells

Connections: Layer 1 --> Layer 2
Layer 2 --> Layer 3
Layer 2 --> Layer 2
Layer 3 --> Layer 2

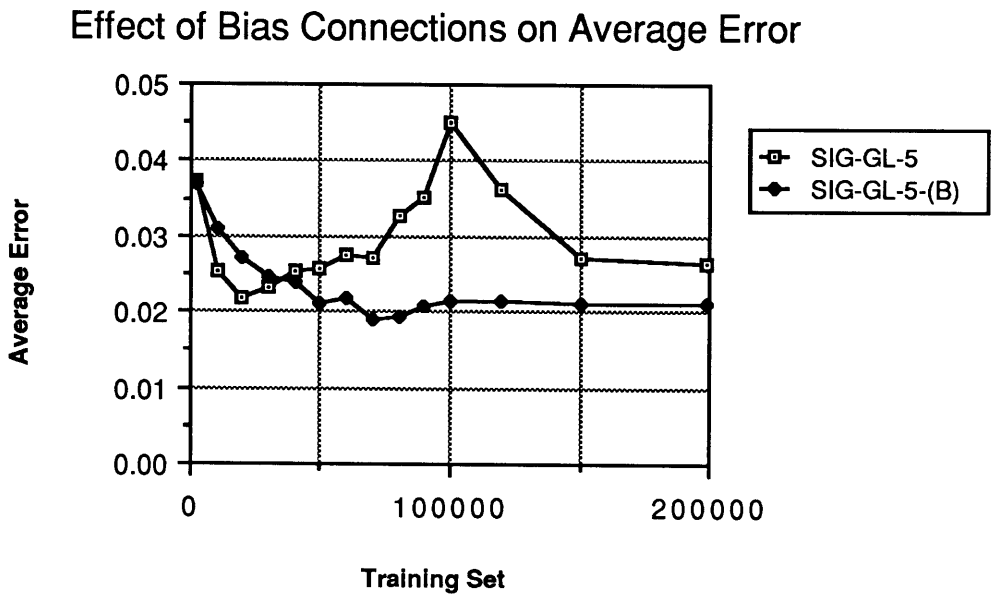
Cells: Layer 1 : Linear Input
Layer 2: Sigmoid Hidden
Layer 3: Sigmoid Output

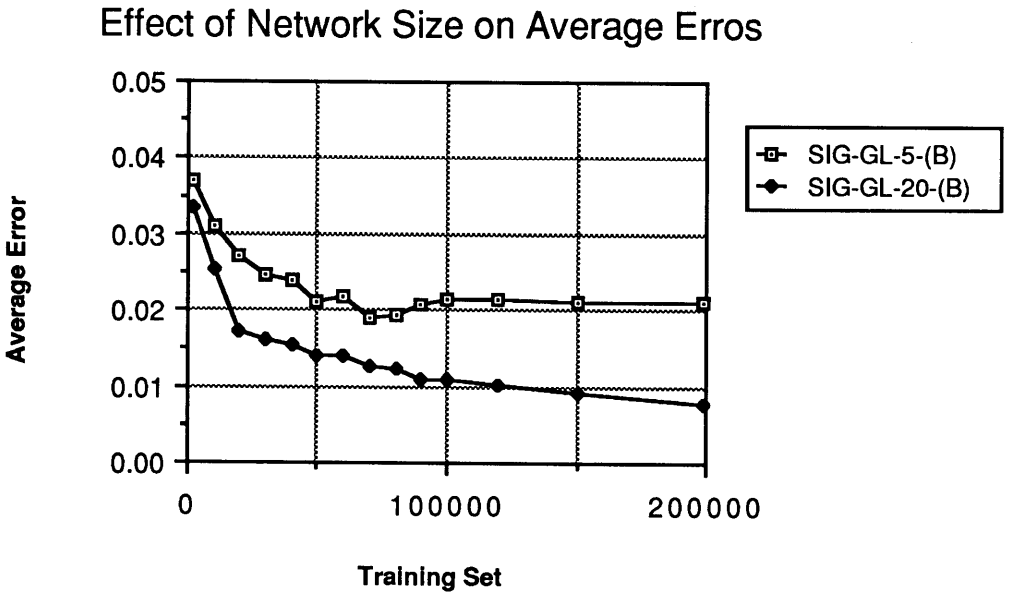
Number of Connections: **C(n)** = $\mathbf{n}^2$ + 3**n**; $\mathbf{n}^2$+4**n** for bias

STEADY-STATE RESULTS
(errors after 200,000 presentations)

	Average Error	Maximum Error
SIG-GL-5	0.02626	0.10449
SIG-GL-5-(B)	0.02097	0.08228
SIG-GL-20	0.01405	0.06281
SIG-GL-20-(B)	0.00779	0.05857

TRAINING RESULTS (SIG-GL-n)





Hybrid with Output Feedback

LABEL: HYB-OUT-**n** ; No bias
HYB-OUT-**n**-(B) ; Bias attached to hidden cells

ARCHITECTURE: 4 Layers, Single Input/Output, **n** Hidden Cells

Connections: Layer 1 --> Layer 2
Layer 1 --> Layer 3
Layer 2 --> Layer 3
Layer 2 --> Layer 4
Layer 3 --> Layer 4
Layer 4 --> Layer 2

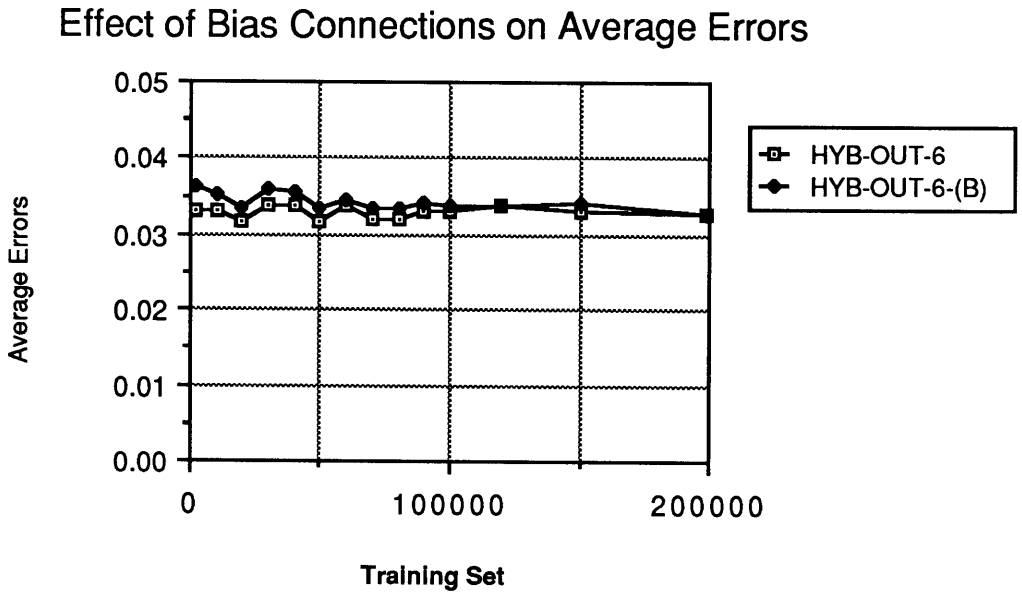
Cells: Layer 1 : Linear Input
Layer 2: Sigmoid Hidden (1/2 **n** cells)
Layer 3: Linear Hidden (1/2 **n** cells)
Layer 4: Sigmoid Output

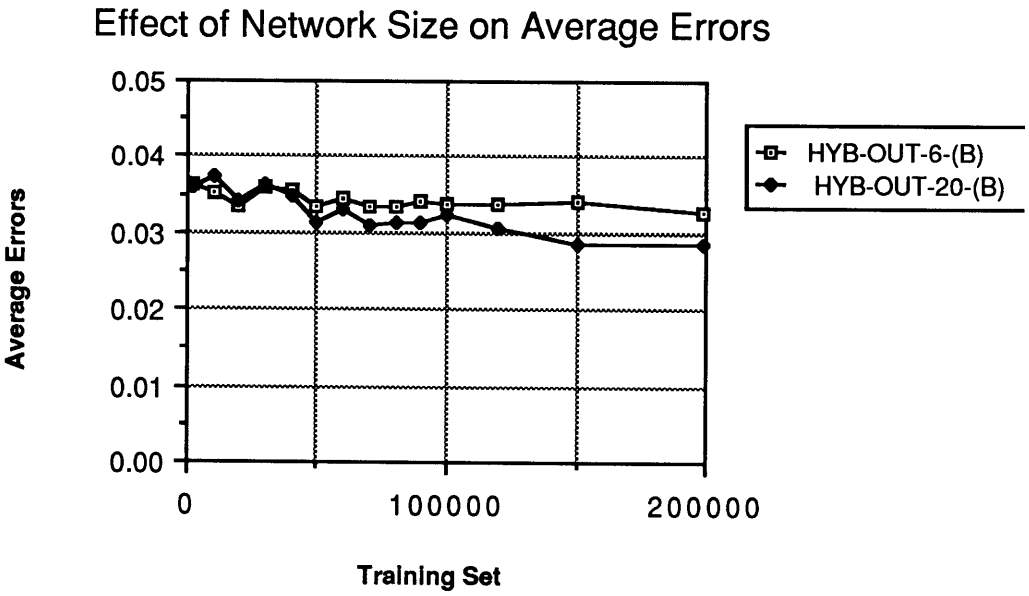
Number of Connections: **C(n)** = 3**n** (4**n** for bias)

STEADY-STATE RESULTS
(errors after 200,000 presentations)

	Average Error	Maximum Error
HYB-OUT-6	0.0329	0.1089
HYB-OUT-6-(B)	0.0327	0.1067
HYB-OUT-20	0.0328	0.1446
HYB-OUT-20-(B)	0.0284	0.1011
HYB-OUT-40	0.0346	0.1303
HYB-OUT-40-(B)	0.0358	0.1567

TRAINING RESULTS (HYB-OUT-n)





Hybrid with Global Feedback

LABEL: HYB-GL-**n** ; No bias
HYB-GL-**n**-(B) ; Bias attached to hidden cells

ARCHITECTURE: 4 Layers, Single Input/Output, **n** Hidden Cells

Connections:
Layer 1 --> Layer 2
Layer 1 --> Layer 3
Layer 2 --> Layer 3
Layer 2 --> Layer 3
Layer 2 --> Layer 4
Layer 3 --> Layer 4
Layer 4 --> Layer 2

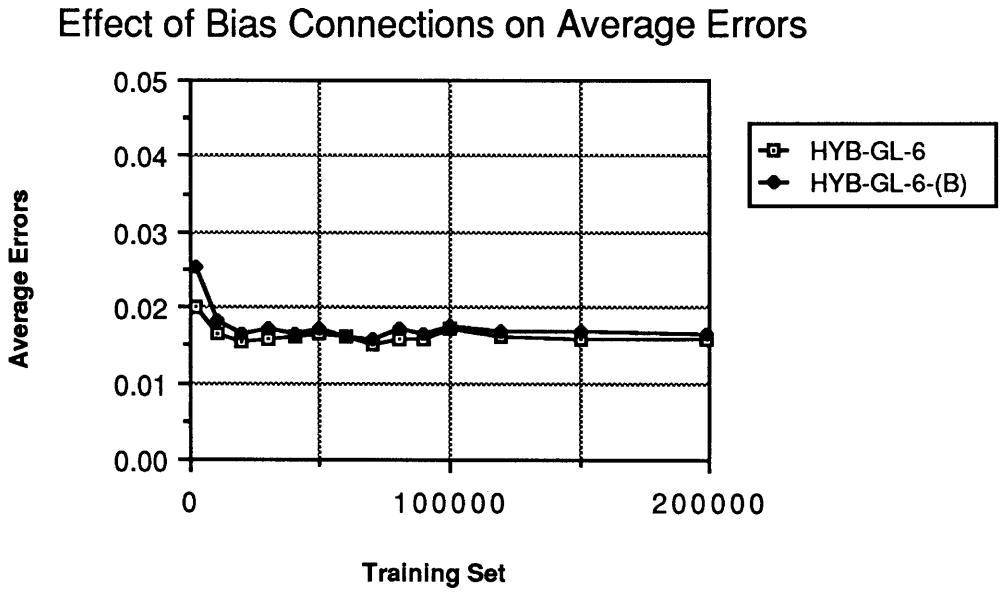
Cells:
Layer 1 : Linear Input
Layer 2: Sigmoid Hidden (1/2 **n** cells)
Layer 3: Linear Hidden (1/2 **n** cells)
Layer 4: Sigmoid Output

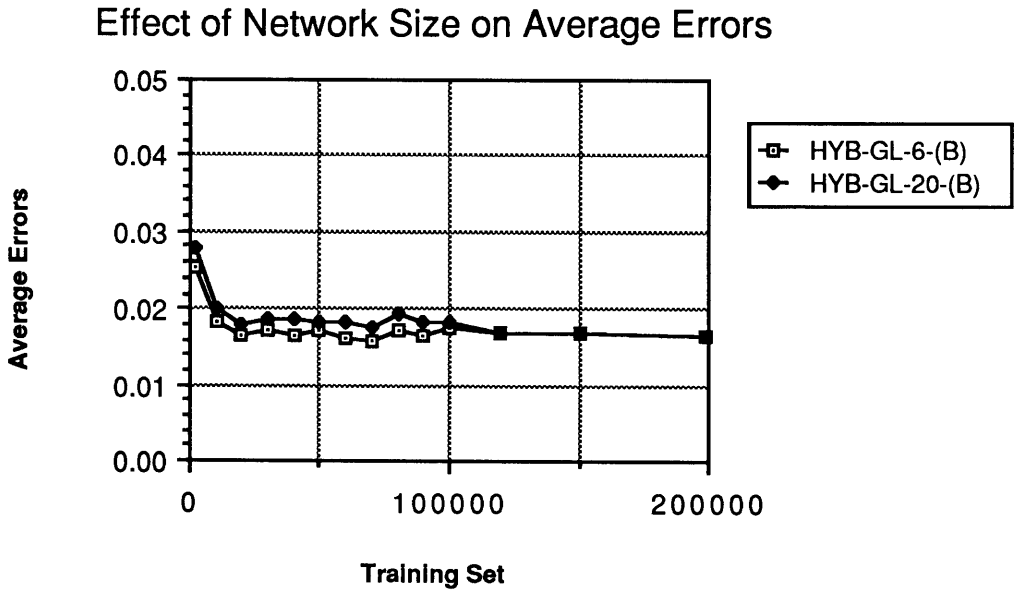
Number of Connections: $\mathbf{C(n)} = 0.25\mathbf{n}^2 + 3\mathbf{n}$
$\mathbf{C(n)} = 0.25\mathbf{n}^2 + 4\mathbf{n}$ for bias

STEADY-STATE RESULTS
(errors after 200,000 presentations)

	Average Error	Maximum Error
HYB-GL-6	0.0157	0.0927
HYB-GL-6-(B)	0.0167	0.0850
HYB-GL-20	0.0174	0.0917
HYB-GL-20-(B)	0.0166	0.0866

TRAINING RESULTS (HYB-GL-n)





Variant Architecture with Sigmoidal Hidden Neurons

LABEL: SIG-VAR-**n** ;No bias
SIG-VAR-**n**-(B) ; Bias attached to hidden cells

ARCHITECTURE: 3 Layers, 3 Inputs, 1 Output, **n** Hidden Cells

Connections: Layer 1 --> Layer 2
Layer 2 --> Layer 3

Inputs: **e**(n), **e**(n-1), **u**(n-1)

Cells: Layer 1 : Linear Input
Layer 2: Sigmoid Hidden
Layer 3: Sigmoid Output

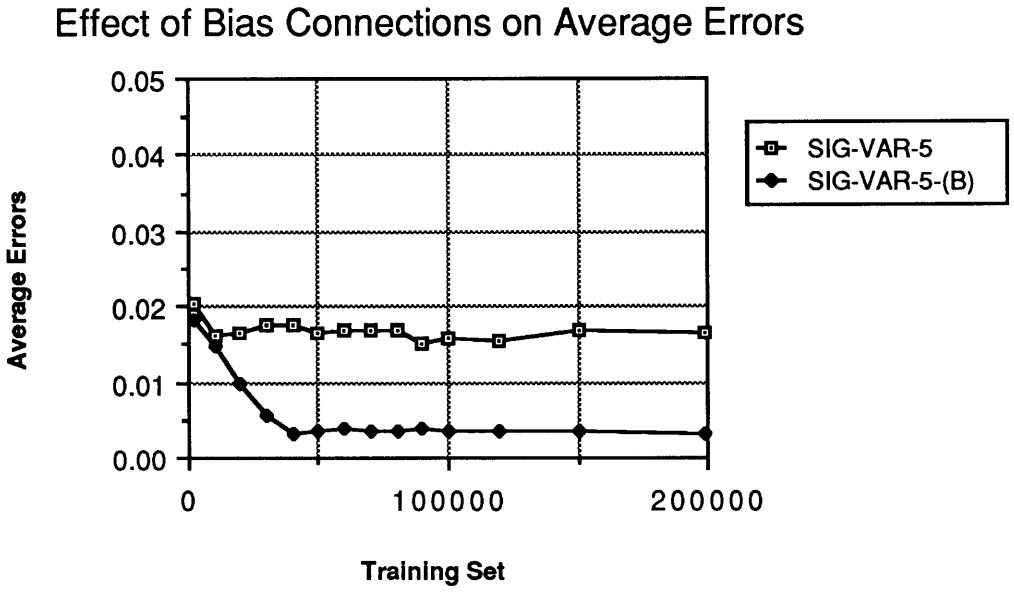
Number of Connections[1]: **C(n)** = 2**n** (3**n** for bias)

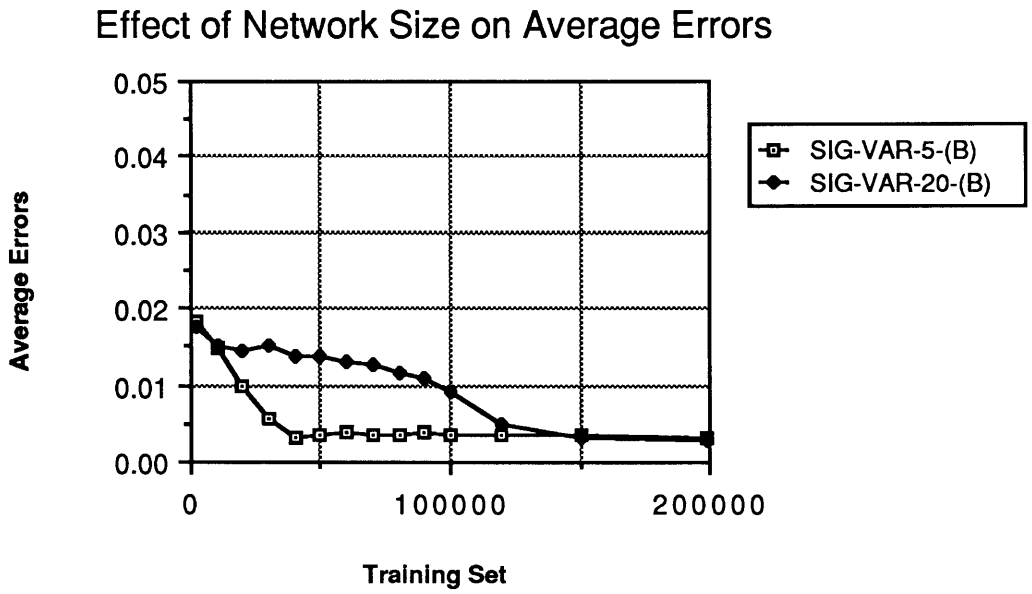
STEADY-STATE RESULTS
(errors after 200,000 presentations)

	Average Error	Maximum Error
SIG-VAR-5	0.0165	0.0949
SIG-VAR-5-(B)	0.0032	0.0346
SIG-VAR-20	0.0160	0.0893
SIG-VAR-20-(B)	0.0029	0.0345
SIG-VAR-40	0.0160	0.0897
SIG-VAR-40-(B)	0.0090	0.0680

[1] In addition to the neural connections, the variant architectures require connections to/and from the FIFO buffer that will allow storage and retrieval of **e**(n-1) and **u**(n-1).

TRAINING RESULTS





References

Crick, Francis H. and Asanuma, Chisato "Certain Aspects of the Anatomy and Physiology of the Cerebral Cortex", in *Parallel Distributed Processing*, vol. 2, pp 333-371, MIT Press, 1986.

Gallant, Stephen I. & King, Donna J. "Experiments with Sequential Associative Memories", Cognitive Science Society Conference, Montreal, August 17-19, 1988.

Hinton, G. E. & Sejnowski, T. J. "Learning and Relearning in Boltzmann Machines", in *Parallel Distributed Processing*, vol. 1, pp 282-317, MIT Press, 1986.

Kohonen, Teuvo "Self-Organization and Associative Memory", Springer-Verlag, Berlin, 1984.

Minsky, M. & Papert, S. "Perceptrons: An Introduction to Computational Geometry", MIT Press, 1969.

Rumelhart, D. E. & McClelland, J. L. (Eds) "Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol. 1.", MIT Press, 1986.

Sanner, Robert "Neuromorphic Regulation of Dynamic Systems Using Back Propagation Networks", MIT Space Systems Lab, 1989.

Werbos, P. J. "Beyond regression: New tools for prediction and analysis in the behavioral sciences", Masters thesis, Harvard University, 1974.

Other Sources

Akin, David "Microcomputer Laboratory", Lecture Notes, MIT Department of Aeronautics and Astronautics, Spring 1987.

Asado "Intelligent Control of Dynamic Systems", Lecture Notes, MIT Mechanical Engineering Department, Fall 1989

Franklin, Gene F. & Powell, J. David "Digital Control of Dynamic Systems", Addison Wesley Publishing, Stanford University, 1980.

Gallant, Stephen I. "Perceptron-Based Learning Algorithms", Technical Report NU-CCS-89-21, College of Computer Science, Northeastern University, August 1989.

Grossberg, S. "Studies of Mind and Brain", Reidel Press, Boston 1982

Grossberg, S. "Nonlinear Neural Networks: Principles, Mechanisms, and Architectures", *Neural Networks*, vol 1, pp 17-61, 1989.

Grossberg, S. "Neural Networks and Natural Intelligence", MIT Press, 1988.

Hall, Stephen "Digital Control of Dynamic Systems", Lecture Notes, MIT Department of Aeronautics and Astronautics, Fall 1988.

Hildebrand, F. B. "Advanced Calculus for Applications", Prentice-Hall, MIT, 1976.

Strang, Gilbert "Introduction to Applied Mathematics", Wellesley-Cambridge Press, MIT, 1986.

Wasserman, Phillip D. "Neural Computing: Theory and Practice", Von Nostrand Reinhold, New York, 1989.