This is to certify that the

dissertation entitled

Analog CMOS Implementation of Artificial
Neural Networks for Temporal Signal Learning

presented by

Hwa-Joon Oh

has been accepted towards fulfillment
of the requirements for

Ph. D. degree in Electrical Engineering

Major professor

Date July 3, '96

Analog CMOS Implementation of Artificial Neural Networks for Temporal Signal Learning

By

Hwa-Joon Oh

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Electrical Engineering

1996

ABSTRACT

Analog CMOS Implementation of Artificial Neural Networks for Temporal Signal Learning

By

Hwa-Joon Oh

A recurrent neural network with a recurrent learning rule is implemented using CMOS technology. We employ several building blocks for the implementation including a wide-range transconductance amplifier, a modified Gilbert multiplier, and a vector multiplier.

A sigmoid function generator is designed using the wide-range trans-conductance amplifier. The output of the wide-range transconductance amplifier is current. To convert the current output to voltage, we use active resistors. The modified Gilbert multiplier and the vector multiplier are implemented using current bus and active resistors. Their four-quadrant and dot-product multiplications are verified through the PSPICE circuit simulations.

We have developed a modified recurrent back-propagation learning rule for temporal learning. Its forward instantaneous update scheme is suitable for analog hardware implementations.

We have designed 4-neuron and 6-neuron recurrent neural network prototypes.

We have implemented the neural network using standard CMOS circuits and verified their performance using extensive PSPICE circuit simulations. We have trained the two prototype neural networks to learn different state trajectories and the PSPICE circuit simulation shows that the recurrent neural network learn the temporal signals for reproduction and classification successfully.

Finally, a two-dimensional scalable array configuration is designed for a large-scale implementation of fully connected recurrent neural network with learning. With the 2-D array configuration, the layout offers a simple and scalable VLSI architecture.

To my parents, wife Jeong-Hyeon, and son Eugene

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Dr. Fathi M. Salam for his able guidance and constructive criticism throughout the course of my research.

I would like to thank Dr. Hassan Khalil, Dr. Timothy Grotjohn, Dr. Gregory M. Wierzba, and Dr. Frank Hoppensteadt for being on my graduate thesis committee. Their critical comments and useful discussions aided in improving the quality of my thesis.

My colleagues at the Circuits, Systems and Artificial Neural Networks Laboratory, Department of Electrical Engineering, MSU, provided extensive help during the course of this research. I would like to sincerely thank Ammar Gharbi and Kay Hyounseok for their help, moral support and useful discussions.

I also record my appreciation of the help rendered by the staff and the faculty of the Department of Electrical Engineering, MSU.

I would like to express my sincere gratitude to my wife, Jeong-Hyeon and son, Eugene, for love, affection and patience.

I would like to express my sincere thanks and appreciation to my parents and parents-in-law for their support and sacrifice.

I also gratefully acknowledge the partial support during the research from ONR, the Michigan Research Excellence Fund (REF), and Innovating Computing Technology, Inc. (IC Tech).

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

The brain is a highly complex, nonlinear, and parallel computing structure. Neural networks have been motivated from its inception by recognizing that the brain computes in an entirely different way than the conventional digital computer. Artificial neural networks are machines that are designed to model the way in which the brain performs a particular task or function of interest [1][2][3][20].

An important feature of neural networks is that they perform useful computations through a process of learning. To achieve good performance, neural networks employ a massive interconnection of simple computing cells referred to as neurons. In neural networks, interneuron connection strengths known as synaptic weights are used to store the knowledge. The knowledge is acquired by the neural network through a learning process. The procedure used to perform the learning process is called a learning algorithm. A learning algorithm modifies the synaptic weights of the neural network in order to get a desired design objective.

Neural networks have some unique attributes [48]: nonlinearity, input-output mapping, adaptivity, the ability to learn from their environment, the ability to generalize from weak assumptions, fault tolerance, VLSI implementability. There have been a lot of architectures and learning algorithms in artificial neural networks [4]. One important artificial neural network is the feedforward neural network and the back-

propagation learning algorithm. Typically, the multilayer feedforward neural network consists of an input layer, one or more hidden layers of computation nodes, and an output layer of computation nodes. The error back-propagation process consists of two processes: a forward pass and a backward pass. In the forward pass, input patterns are applied to the network, and their effects propagate through the network. In the backward pass, the desired patterns are presented to the network and the outputs of the network are compared with the desired ones. The errors are calculated and propagated backward through the connection weights to minimize the errors. The synaptic weights are adjusted so as to make the actual responses of the network move closer to the desired responses. Multilayer feedforward networks have been applied successfully to solve some difficult and diverse problems by training them in a supervised manner [1][20][48].

The neural networks are usually implemented using electronic components or simulated in software on a conventional computer. In the forward pass and the backward pass of neural network, high computational requirement is needed. To meet the computational requirement, analog hardware implementation of the neural networks will be an ideal medium for real time learning and processing. The advantages of using analog VLSI as technology medium for special-purpose neural network implementations include the inherent parallelism of the operations, fast speed on learning and processing, the compact size, and low power consumption of the elements performing the computational functions [5].

Some of the traditional analog design requirements such as accurate absolute component values, device matching, precise time constants, etc., are not major concerns in neural networks. This is primarily because computation precision of individual neurons is not important [6]. For learning neural networks, the effects of mismatches and offsets in the analog components can be greatly compensated by the learning hardware parameters directly on the implemented neural networks. The learning

performance of the neural networks may still be affected by the analog precision of the implemented learning function themselves, depending on the nature of the algorithm used.

The list of general-purpose and special-purpose neural chips available presently is quite diverse, and still growing.

The direct implementation of biological circuitry has been done by Mead's group [7][38]. They have found parallels between the behavior of subthreshold analog CMOS VLSI and biological neural circuitry. They have implemented a wide variety of neural VLSI systems that have been successfully constructed, including retinas[8], cochleas[9], and other biological systems.

Intel's ETANN chip [10] is an analog model of 64 analog neurons and 10240 analog synapses. It uses EEPROM technology to store synaptic weights and the learning algorithm is performed at external general computers. The network can simultaneously compute the dot-product of a 64 element analog vector with a 64×64 synaptic array at a rate in excess of 1.3 billion interconnections per second. All elements of the computation are done in the analog domain and strictly in parallel.

AT&T has built an ANNA (Analog Neural Network Arithmetic and logic unit) chip [11] that is a hybrid analog-digital neural network chip. The chip implements 4096 physical synapses. The resolution of the synaptic weights is 6 bits, and that of the states (input and output of the neuron) is 3 bits. The chip uses analog computation internally, but all input/output is digital. The chip can be reconfigured for synaptic weight and input vectors of varying dimension, namely, 64, 128, and 256. The ANNA chip is implemented for the application of high-speed optical character recognition with a total of 136,000 connections on a single chip.

An analog VLSI neural network processor was designed and fabricated for communication receiver applications [12]. A channel equalizer was implemented with a neural chip configured as a three-layer perceptron network. The number of neurons

in the input layer, two hidden layer, and the output layer is 8, 12, 12, and 1, respectively. The whole network includes a total of 252 synapse cells, and the input layer consists of the switched-capacitor analog delay circuits. The synapse cell is realized with a wide-range Gilbert multiplier circuit. The neuron circuit consists of a linear current-to-voltage converter and a sigmoid function generator with a controllable voltage gain. Network training is performed by the modified Kalman filtering algorithm and the learning process is done in the companion DSP board which keeps the synaptic weight for the chip. The chip requires the refresh hardware to maintain the weight values.

Storage of adjustable analog weights is one of the most important problems faced in analog implementation of artificial neural networks. The storage form can be analog: it would thus have the properties of an analog memory cell. A commonly used storage technique is based on storing charge across a capacitor. The storage capacitor will slowly discharge. For today's conventional technologies, the storage time is of the order of several milliseconds. To store the weight control voltage for longer periods, the charge stored on the capacitance needs to be refreshed once every few milliseconds. The natural decay of a capacitor's charge is one of the inherent limitations of the analog storage of weights.

We have designed a neural network which has two feedforward neural networks in one chip [13]. The first feedforward neural network has its learning circuits. The second neural network does not have the learning circuit. The weight of the second feedforward neural network, is fed from the first neural network through voltage followers. If we keep applying the input-target pairs to the first feedforward neural network, there is no need to make an external interface circuit for weight refreshing in the second feedforward neural network.

A recent paper [14] which employs a stochastic perturbative algorithm uses a local analog memory technique which does not require external storages. The weight

refresh is performed in the background, and does not interfere with the continuous-time network operation.

The back propagation algorithm is one of the most popular methods for the design of neural networks. A major limitation of the standard back propagation algorithm is its focus on approximating static mappings. This static input-output mapping is well suited for static information processing problems, however, it is not suitable for dynamic temporal information processing.

Time-varying signals are important in many of the cognitive tasks encountered in practice, e.g., in vision, speech, control, and signal processing [15]. It is necessary to provide the neural network with dynamic properties that make it responsive to time-varying signals. For a neural network to be dynamic, it must be given memory [48]. One way in which a neural network can assume dynamic behavior is to make it *recurrent*, that is to build feedback into its design. In the recurrent neural network, connections are allowed in both ways between a pair of units, and even from a unit to itself.

Analog recurrent neural network learning on time-varying signals offers a wide range of attractive applications, e.g., for process control, identification of dynamic system, and adaptive signal processing.

Several versions of gradient descent algorithms for supervised learning in dynamic recurrent neural networks exist [16]. Pearlmutter [50] has derived learning procedure which can learn nonfixed point attractor. The technique is called the back-propagation through time. Pineda [54] has studied the fixed point learning procedure using the error back-propagation algorithm. It is called the recurrent back-propagation learning rule. An on-line [55], but computationally expensive, procedure for determining the derivatives of the states with respect to the weight parameters has been discovered and applied to the recurrent networks. It is called real time recurrent learning. The above mentioned algorithms are implemented in software.

Their analog hardware implementations for real-time operations have currently not been demonstrated.

We have deigned and fabricated feedforward neural networks with the modified update law [13][28][47]. The test results have demonstrated the successful operations of the feedforward neural networks [13][28][47][61][62]. The circuit designs of the neural network at this dissertation are based on the test results of previous fabrications and test results.

In this dissertation, recurrent artificial neural networks with on-chip learning circuit are implemented using standard CMOS technology. We have modified the learning rule from the time-dependent recurrent back-propagation learning rule. We avoid the backward integration and the memory requirement by modification. Its forward instantaneous update scheme is suitable for an analog hardware implementation. This is the first successful demonstration of the implementation of the gradient descent algorithm in the recurrent neural network. In order to implement the recurrent neural network and its learning algorithm, we employ a wide-range transconductance amplifier, an active resistor, a modified Gilbert multiplier, and a vector multiplier.

Contributions of this dissertation are as follows:

1. A simple sigmoid function generator is designed using a wide-range transconductance amplifier. Its current output is converted to voltage via active resistors in order to achieve voltage-to-voltage operations. The sigmoid function generator is simulated using the PSPICE circuit simulator. Its characteristics are shown in this dissertation.

2. The modified Gilbert multiplier is designed and implemented as a CMOS circuit. Its voltage-to-voltage operation is achieved through active resistors. Its four-quadrant multiplication is verified in this dissertation. The performance of the multiplier using the PSPICE circuit simulator is shown in this dissertation

3. The vector multiplier is designed using the modified Gilbert multiplier. High dimensional multiplier is implemented through simple current summing nodes. The performance of the vector multiplier is presented using the PSPICE circuit simulator.

4. The modified recurrent back-propagation rule is presented. Its continuous-time modified update law from the time-dependent recurrent back-propagation learning rule is shown in this dissertation. The modified algorithm is suitable for the analog CMOS implementation.

5. The MATLAB simulations show the conditions for the stable operation of the network. The damping factors in the modified equations are important for the stability of the recurrent neural network. We show the range of the parameters which ensure the successful learning in the learning phase.

6. The modified recurrent back-propagation rule is implemented with on-chip learning. Its learning rule is implemented in CMOS circuit and the PSPICE simulations demonstrate its learning capability. We have demonstrated the learning capability by training a circular state trajectory. In the circular trajectory generation experiment, the recurrent neural network can generate a limit cycle. We have performed experiments with different parameters for successful learning. At the trajectory recognition experiment, the simulation result shows that the recurrent neural network can distinguish different trajectories.

7. A two-dimensional scalable array configuration is designed for large-scale implementation. With the 2-D array configuration, the layout offers a simple and scalable VLSI architecture. We have designed 4-neuron recurrent neural network and its number of input and target is easily configurable. We show the subcell design, the floor plan, and its layout in this dissertation.

This dissertation is divided into 7 chapters. Chapter 2 reviews the model of neuron and three artificial neural network models. McCulloch and Pitts model, a feedforward neural network model, and Hopfield neural network model are investigated. In Chapter 3, basic CMOS circuits and analog neural subcircuits are discussed for the implementations of the artificial neural networks. Active resistors are designed to convert the current output to the voltage output and its characteristics are shown using the PSPICE circuit simulator. The designs of a transconductance amplifier, a modified Gilbert multiplier, and a vector multiplier are demonstrated. Chapter 4 reviews the algorithms for the temporal signal learning. We have reviewed the Pearlmutter's algorithm, the classification of temporal trajectories, the Pineda's algorithm, and the William and Zipper's algorithm. The hardware requirement for the William and Zipper's algorithm are investigated in this chapter. In Chapter 5, the modified back-propagation learning rule is presented. Its MATLAB simulations for stability are shown in this chapter and its implementation is explained. We have demonstrated that the learning scheme successfully learns the temporal signals by generating the circular trajectory and recognizing the different state trajectories in the recurrent neural networks. In Chapter 6, we present the 2-D array configuration of the modified back-propagation learning rule. With the 2-D array configuration, the layout offers a simple and scalable VLSI architecture. We show the layout and pin-configuration of the chip. Chapter 7 summarizes the conclusions of this research work.

CHAPTER 2

Neural Network Models

2.1 The model of a Neuron

The basic unit in the nervous system is specialized cell which is called neuron. A typical view of neuron is shown in Figure 2.1.

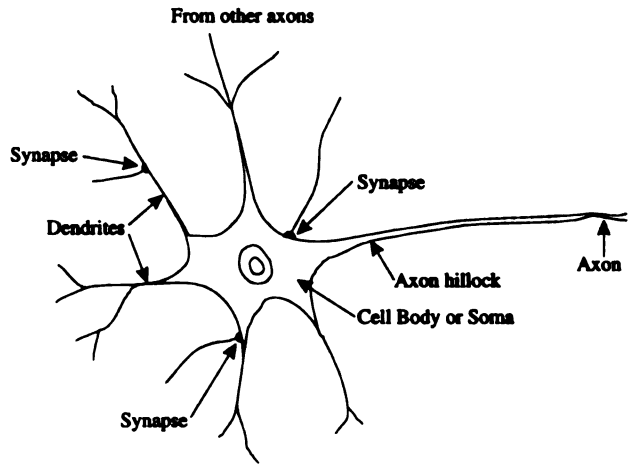

Figure 2.1. The structure of a classical neuron

Most neurons share certain structural features that make it possible to distinguish four regions of the cell: the cell body or soma, the dendrites, the axon, and synapse.

The individual nerve cell transmits nerve impulses over a single long fiber (the axon) and receives them over numerous short fibers (the dendrites) [17].

The functioning of the brain depends on the flow of information through elaborate circuits consisting of networks of neurons. Information is transferred from one cell to another at specialized points of contact: the synapses. A typical neuron may have anywhere from 1,000 to 10,000 synapses and may receive information from something like 1,000 other neurons. Synapses are most often made between the axon of one cell and the dendrite of another. These synaptic contacts are the primary information processing elements in neural systems.

The transmission of a signal from one cell to another cell at a synapse is a complex chemical process. At a synapse the axon usually enlarges to form a terminal button, which is the information-delivering part of the junction. The terminal button contains tiny spherical structures called synaptic vesicles, each of which can hold several thousand molecules of chemical transmitter. On the arrival of a nerve impulse at the terminal button, some of the vesicles discharge their contents into the narrow cleft that separate the button from the membrane of another cell's dendrite, which is designed to receive the chemical message. Hence the information is relayed from one neuron to another by means of a transmitter. The "firing" of a neuron-the generation of nerve impulses-reflects the activation of hundreds of synapses by impinging neurons. Some synapses are excitatory in that they tend to promote firing, whereas others are inhibitory and so are capable of canceling signals that otherwise would excite a neuron to fire.

Equipped with a tree of filamentary dendrites, the neuron body aggregates synaptic inputs from other neurons. The input currents are integrated by the capacitance of the cell body until a critical threshold potential is reached, at which point an output is generated in the form of a nerve pulse. This output pulse propagates down the axon, which ends in a tree of synaptic contacts to the dendrites of other neurons.

2.2 McCulloch and Pitts neural network model

Due to the complexity and diversity of the properties of biological neurons, the task of compressing their complicated characteristics into a model is extremely difficult. Neural computational elements are nonlinear, and typically these are analog in nature. Many researchers have tried to model neural network systems with current knowledge of biological neurons since it is still under research.

McCulloch and Pitts [18] proposed a simple model of a neuron as a binary threshold unit. Specifically, the model neuron computes a weighted sum of its inputs from other units, and outputs a one or a zero according to whether this sum is above or below a certain threshold:

$$n_j(t+1) = S(\sum_i w_{ji} n_i(t) + \theta_j) \tag{2.1}$$

The diagram is shown in Figure 2.2. Here n_j is either 1 or 0, and represents the state

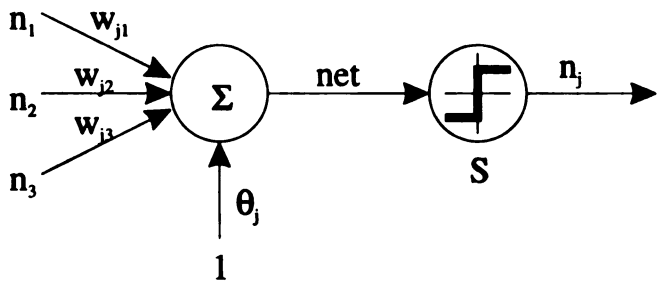

Figure 2.2. Schematic diagram of a McCulloch-Pitts neuron

of neuron j as *firing* or *notfiring* respectively. Time t is taken as discrete, with one time unit elapsing per processing step. S is the unit step function, or Heaviside

function:

$$S(x) = \begin{cases} 1 & \text{if } x \geq 0; \\ 0 & \text{otherwise.} \end{cases} \tag{2.2}$$

The weight w_{ji} represents the strength of the synapse connecting neuron i to neuron j. It can be positive or negative corresponding to an excitatory or inhibitory synapse respectively. It is zero if there is no synapse between j and i. The cell specific parameters θ_j is the threshold value for unit j; the weighted sum of inputs must reach or exceed the threshold for the neuron to fire.

Though it is a simple model, a McCulloch-Pitts neuron is computationally a powerful device. McCulloch and Pitts proved that a synchronous assembly of such neurons is capable in principle of universal computation for suitably chosen weights w_{ji}. This means that it can perform any computation that an ordinary computer can, though not necessarily so rapidly or conveniently.

Real neurons involve many complications omitted from this simple description. The most significant ones include [1]:

- Real neurons are often not even approximately threshold device as described above. Instead they respond to their input in a continuous way. However, the nonlinear relationship between the input and the output of a cell is a universal feature. The hypothesis is that it is the nonlinearity that is essential, not its specific form.

- Many real cells also perform a nonlinear summation of their inputs. There can even be significant logical processing (e.g., AND, OR, NOT) within the dendritic tree.

- A real neuron produces a sequence of pulses, not a simple output level. Representing the firing rate by a single number like n_i, ignores much information that might be carried by such a pulse sequence.

- Neurons do not all have the same fixed delay ($t \rightarrow t+1$). Nor are they updated synchronously by a central clock. In fact, Neurons are operating in asynchronous way.

2.3 Feedforward neural network model

In a feedforward neural network, information flows in the forward direction only, and there are no feedback loops. The network is always stable, and its state depends on the inputs in a simple manner. Suppose that N neurons form a feedforward neural network with M inputs. Its structure is shown in Figure 2.3.

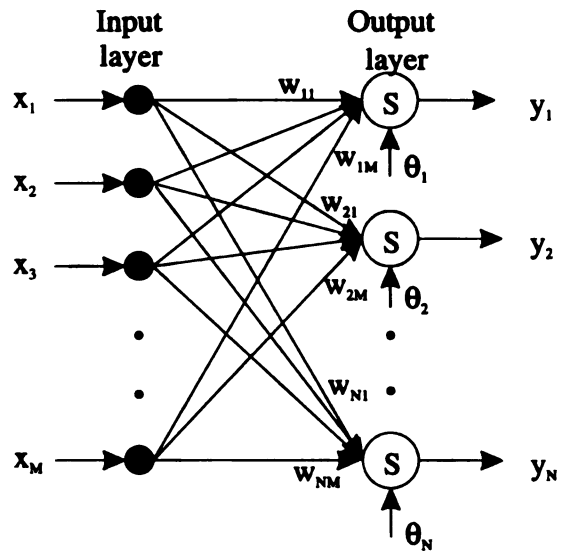

Figure 2.3. Single layer feedforward neural network. S means sigmoid function

For this network, the input-output relationship can be expressed as

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = S\left(\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1M} \\ w_{21} & w_{22} & \cdots & w_{2M} \\ \vdots & \vdots & & \vdots \\ w_{N1} & w_{N2} & \cdots & w_{NM} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix} + \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_N \end{bmatrix} \right) \tag{2.3}$$

where x_i, $1 \leq i \leq M$, is the input neuron, y_j, $1 \leq j \leq N$, is the output neuron, and w_{ji} is the connection weight from ith input neuron to jth output neuron. The above equation is represented by

$$\mathbf{Y} = \mathbf{S}(\mathbf{WX} + \mathbf{\Theta}) \tag{2.4}$$

where $\mathbf{X}$ is an M-component column vector of inputs, $\mathbf{W}$ is an $N \times M$ synaptic weight matrix, and $\mathbf{Y}$ is the resulting N-component column vector of outputs. $\mathbf{\Theta}$ is an N-component column vector of threshold and $\mathbf{S}$ is a differential, bounded, and strictly increasing monotone function. The threshold can be included into the weight matrix implicitly by using $w_{i0} = \theta_i$ and $x_0 = 1$. Thus the general equation is expressed as:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \end{bmatrix} = S\left(\begin{bmatrix} w_{10} & w_{11} & w_{12} & \cdots & w_{1M} \\ w_{20} & w_{21} & w_{22} & \cdots & w_{2M} \\ \vdots & \vdots & \vdots & & \vdots \\ w_{N0} & w_{N1} & w_{N2} & \cdots & w_{NM} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix} \right) \tag{2.5}$$

This type of network has only one layer of neurons and there is no connection between neurons themselves. It is called single-layered feedforward neural network or Perceptron [19]. Naturally, its practical applications are limited and interesting applications are emerged when many layers are interacted each other. This can be achieved through cascading two or more layers of such simple network, as shown in

Figure 2.4. Any layer between the input layer and output layer is called a hidden

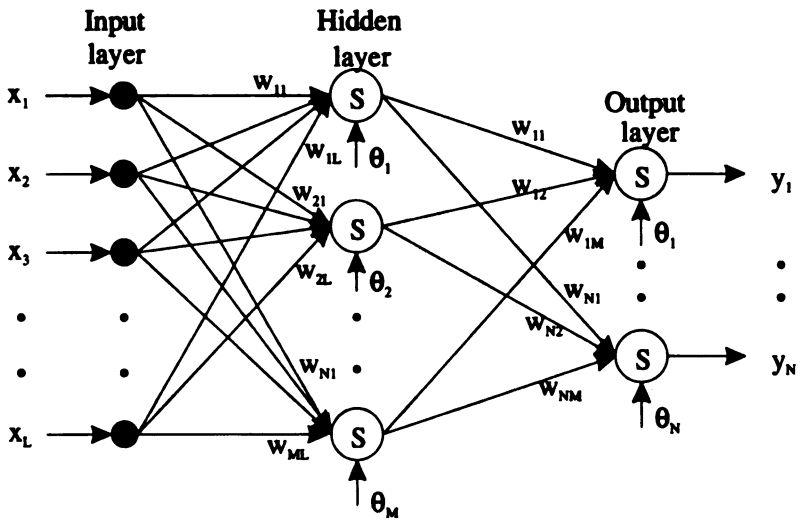

Figure 2.4. Two layer feedforward neural network

layer. Thus, all but the output layer are hidden layer in internal layers.

The operation of such networks duplicates that of Figure 2.3, that is, the outputs of each layer are produced from the weighted sum of the previous layer's output. As the number of layers increases, the usefulness of the network also increases. The output from a multi-layered neural network can be expressed as

$$\mathbf{Y} = \mathbf{S}(\mathbf{W}^1\mathbf{S}(\mathbf{W}^2 \cdots \mathbf{S}(\mathbf{W}^K\mathbf{X}))) \tag{2.6}$$

where $\mathbf{W}^k$ is a matrix of weights for kth layer, the size of this matrix depending on the number of neurons acting as source and destination for the layer.

With the development of the back-propagation training algorithm, multi-layered feedforward neural networks overcome many of the limitations of single-layered feed-forward neural network [20]. The back-propagation algorithm uses a gradient-descent

search technique to minimize a cost function which is difference between the desired output and the actual output. Also, this approach is called supervised learning. For training the desired output, the input vector is applied to the neural network and its output vector is produced. The actual output is compared to the desired output and the error is passed in a backward direction to adjust the connection weights. The modification of weights is carried out from layer to layer in a backward fashion and this process is iteratively continued until its error is minimized.

2.3.1 The standard back-propagation algorithm

The static equations of feedforward artificial neural networks [1][20][48][49] are given by

$$net_{p,j} = g(w_{ji}, x_{p,i}) \tag{2.7}$$

$$y_{p,j} = S(net_{p,j}) = S(g(w_{ji}, x_{p,i})) \tag{2.8}$$

where $g()$ denotes the general synapse function. Usually, it is given as a dot-product multiplication between input vector and weight matrix. Here, p represents the index of patterns and w_{ji} is a connection weight from the (output of the) ith node to the (input of the) jth node and $x_{p,i}$, which is an output of ith node at pth pattern. Observe that, in a feedforward network, $y_{p,j}$ is the output of the jth neuron in the present layer and $x_{p,i}$ is the output of the ith neuron from the previous layer. S is a differentiable, bounded, and strictly increasing monotone function. S is in fact a diffeomorphism usually referred to as a sigmoid function.

The desired patterns are presented to the neural network and the outputs of the networks are compared with the desired ones. The error signal at the output of neuron j is defined by

$$e_{p,j} = d_{p,j} - y_{p,j} \tag{2.9}$$

The errors between the desired output (d) and the actual outputs (y) are calculated and propagated backward through the connection weights to minimize the total error. The total squared error is given by

$$E = \sum_p E_p = \frac{1}{2}\sum_p \sum_j (d_{p,j} - y_{p,j})^2 = \frac{1}{2}\sum_p \sum_j e_{p,j}^2 \tag{2.10}$$

where p denotes the pth pattern and j denotes the jth output neuron.

The error is a function of connection weight w_{ji} and updating the weight in gradient system is defined as the delta rule:

$$w_{ji}^k = w_{ji}^{k-1} + \Delta w_{ji} \tag{2.11}$$

where Δw_{ji} denotes the changes in the weight w_{ji} at the kth iteration due to the input-target patterns. The delta rule defines the change due to applying p patterns such as

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} = -\eta \sum_p \frac{\partial E_p}{\partial w_{ji}} \tag{2.12}$$

where $\eta > 0$ is the (learning) rate. Observe that η is assumed to be sufficiently small in order for equation (2.12) to be truly gradient system. The method of gradient descent has the task of continually seeking the bottom point of the error measure.

According to the chain rule, we may express this gradient as follows:

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial e_{p,j}} \cdot \frac{\partial e_{p,j}}{\partial y_{p,j}} \cdot \frac{\partial y_{p,j}}{\partial net_{p,j}} \cdot \frac{\partial net_{p,j}}{\partial w_{ji}} \tag{2.13}$$

Differentiating E_p in equation (2.10) with respect to $e_{p,j}$, we get

$$\frac{\partial E_p}{\partial e_{p,j}} = e_{p,j} \tag{2.14}$$

Differentiating both sides of equation (2.9) with respect to $y_{p,j}$, we obtain

$$\frac{\partial e_{p,j}}{\partial y_{p,j}} = -1 \tag{2.15}$$

Next, differentiating equation (2.8) with respect to $net_{p,j}$, we get

$$\frac{\partial y_{p,j}}{\partial net_{p,j}} = S_j'(net_{p,j}) \tag{2.16}$$

Finally, differentiating equation (2.7) with respect to w_{ji} yields

$$\frac{\partial net_{p,j}}{\partial w_{ji}} = \frac{\partial g(w_{ji}, x_{p,i})}{\partial w_{ji}} \tag{2.17}$$

Hence, the use of equation (2.14) to equation (2.17) in equation (2.13) yields

$$\frac{\partial E_p}{\partial w_{ji}} = -e_{p,j} S_j'(net_{p,j}) \frac{\partial g(w_{ji}, x_{p,i})}{\partial w_{ji}} \tag{2.18}$$

Accordingly, the use of equation (2.18) in equation (2.12) yields

$$\Delta w_{ji} = -\eta \sum_p \frac{\partial E_p}{\partial w_{ji}} \tag{2.19}$$

$$= \eta \sum_p \delta_{p,j} \frac{\partial g(w_{ji}, x_{p,i})}{\partial w_{ji}} \tag{2.20}$$

where the local gradient $\delta_{p,j}$ is defined by

$$\delta_{p,j} = -\frac{\partial E_p}{\partial e_{p,j}} \cdot \frac{\partial e_{p,j}}{\partial y_{p,j}} \cdot \frac{\partial y_{p,j}}{\partial net_{p,j}} \tag{2.21}$$

$$= e_{p,j} S_j'(net_{p,j}) \tag{2.22}$$

The local gradient points to required changes in synaptic weights. We may identify the local gradient in two distinct cases, depending on where in the network neuron j

located.

For an output layer, we may use equation (2.9) to compute the error signal. Thus it is straightforward to compute the local gradient $\delta_{p,j}$ using equation (2.22).

$$\Delta w_{ji} = \eta \sum_{p} (d_{p,j} - y_{p,j}) S_j'(net_{p,j}) \frac{\partial g(w_{ji}, x_{p,i})}{\partial w_{ji}} \tag{2.23}$$

When neuron j is located in a hidden layer of the network, there is no specified desired response for that neuron. Accordingly, the error signal for a hidden neuron would have to be determined recursively in terms of the error signals of all the neurons to which that hidden neuron is directly connected. Suppose that neuron x_i is in the input layer, neuron y_j is in the hidden layer, and neuron z_k is in the output layer. For the input-to-hidden connection weight w_{ji}, we must differentiate E_p with respect to the w_{ji}'s, which more deeply embedded. Using the chain rule, we obtain

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial y_{p,j}} \cdot \frac{\partial y_{p,j}}{\partial w_{ji}} \tag{2.24}$$

$$= \frac{\partial E_p}{\partial y_{p,j}} \cdot \frac{\partial y_{p,j}}{\partial net_{p,j}} \cdot \frac{\partial net_{p,j}}{\partial w_{ji}} \tag{2.25}$$

$$= -\delta_{p,j} \frac{\partial g(w_{ji}, x_{p,i})}{\partial w_{ji}} \tag{2.26}$$

where the local gradient $\delta_{p,j}$ is given by

$$\delta_{p,j} = -\frac{\partial E_p}{\partial y_{p,j}} \cdot \frac{\partial y_{p,j}}{\partial net_{p,j}} \tag{2.27}$$

$$= -\frac{\partial E_p}{\partial y_{p,j}} \cdot S_j'(net_{p,j}) \tag{2.28}$$

Using the chain rule again, we may express the first term of equation (2.28) by

$$\frac{\partial E_p}{\partial y_{p,j}} = \sum_{k} \frac{\partial E_p}{\partial e_{p,k}} \frac{\partial e_{p,k}}{\partial y_{p,j}} \tag{2.29}$$

$$= -\sum_k e_{p,k} \frac{\partial z_{p,k}}{\partial y_{p,j}} \tag{2.30}$$

$$= -\sum_k e_{p,k} \frac{\partial z_{p,k}}{\partial net_{p,k}} \frac{\partial net_{p,k}}{\partial y_{p,j}} \tag{2.31}$$

$$= -\sum_k e_{p,k} S'_k(net_{p,k}) \frac{\partial g(w_{kj}, y_{p,j})}{\partial y_{p,j}} \tag{2.32}$$

$$= -\sum_k \delta_{p,k} \frac{\partial g(w_{kj}, y_{p,j})}{\partial y_{p,j}} \tag{2.33}$$

where , in the last line, we have used the definition of the local gradient $\delta_{p,k}$ given in equation (2.22) with the index k substituted for j.

Finally, using equation (2.33) in equation (2.28), we get the local gradient $\delta_{p,j}$ for hidden neuron j, after rearranging terms, as follows:

$$\delta_{p,j} = S'_j(net_{p,j}) \sum_k \delta_{p,k} \frac{\partial g(w_{kj}, y_{p,j})}{\partial y_{p,j}} \tag{2.34}$$

Thus, for a hidden layer, the delta rule is given by

$$\Delta w_{ji} = \eta \sum_p S'_j(net_{p,j}) \frac{\partial g(w_{ji}, x_{p,i})}{\partial w_{ji}} \sum_k \delta_{p,k} \frac{\partial g(w_{kj}, y_{p,j})}{\partial y_{p,j}} \tag{2.35}$$

where $\sum_k$ is the summation over the next layer and $\delta_{p,k}$ is a back-propagated error from the next layer.

2.3.2 The modified update law

Let's consider the new update law. Define the total energy by the following equation:

$$E_p = (\frac{1}{2} \sum_j (d_{p,j} - y_{p,j})^2 + \frac{1}{2} \sum_j \sum_i \alpha_{ji} w_{ji}^2) \tag{2.36}$$

where α_{ji} is a sufficiently small 'forgetting factor', signifying damping, which is also important for global stability. Update law in continuous time system is expressed as

$$\Delta w_{ji} \approx \dot{w}_{ji} dt = -\eta_{ji} \frac{\partial E}{\partial w_{ji}} dt \tag{2.37}$$

The differential dt is the time-step in the integration of the derivative quantity, $\dot{w}_{ji}$. If we let the differential dt be sufficiently small, the approximation becomes more accurate. η_{ji} is the learning rate of each weight.

After the chain rule, **the continuous time gradient descent dynamic update law** is given by the differential equation as [25][26].

$$\dot{w}_{ji} = \eta_{ji} \sum_{p} \delta_{p,j} S'_j(net_{p,j}) \frac{\partial g(w_{ji}, x_{p,i})}{\partial w_{ji}} - \eta_{ji} \alpha_{ji} w_{ji} \tag{2.38}$$

If the neuron j is in an output layer,

$$\delta_{p,j} = (d_{p,j} - y_{p,j}) \tag{2.39}$$

If neuron j is in any hidden layer,

$$\delta_{p,j} = \sum_{k} \delta_{p,k} \frac{\partial g(w_{kj}, y_{p,j})}{\partial y_{p,j}} \tag{2.40}$$

where k is the index for the elements in the immediate subsequent layer.

The modified continuous time update law is obtained by removing the sigmoid derivative terms, namely $S'_j(net_{p,j})$ [25][27]. Also, we can have w_{kj} instead of the partial derivative $\partial g(w_{kj}, y_{p,j})/\partial y_{p,j}$ and $x_{p,i}$ instead of $\partial g(w_{ji}, x_{p,i})/\partial w_{ji}$ in equation (2.38) and (2.40). It is shown that the derivative terms in the equation may be removed without loss in stability or convergence of the update law [25][26]. The

modified update law is thus given by

$$\dot{w}_{ji} = \eta_{ji} \sum_{p} \delta_{p,j} x_{p,i} - \eta_{ji} \alpha_{ji} w_{ji} \tag{2.41}$$

For an output layer, the modified update law is obtained as

$$\delta_{p,j} = (d_{p,j} - y_{p,j}) \tag{2.42}$$

For a hidden layer, the modified update law is given by

$$\delta_{p,j} = \sum_{k} \delta_{p,k} w_{kj} \tag{2.43}$$

where $\delta_{p,j}$ is given as

$$\delta_{p,j} = e_{p,k} \tag{2.44}$$

Observe that an essential difference between the modified (continuous time) update law and the conventional (discrete time) back-propagation is the absence of the derivatives of the sigmoid functions. This results in simplifying the learning rule with major payoff in implementations, both in software and hardware.

2.3.3 Learning methods

We have written the update rule as sums over p patterns. For given training patterns, back-propagation learning may proceed in one of two basic ways: **parallel learning** or **sequential learning** [27][28].

In parallel learning, weight updating is performed after the presentation of all the training patterns (i.e., **batch mode**). In hardware implementation of parallel learning, it requires excessive copies of the feedforward neural network equal to the number of patterns and additional terminals for each input-target pair. In parallel

learning, the number of input-target pairs is limited to the number of the copies in the neural network implementation.

Although we have written the modified update rule as sums over all patterns p, they can be used incrementally. A pth pattern is presented at the input and then all weights are updated before the next pattern is considered. This clearly decreases the error measure (for small enough η) at each pattern. We refer to this approach as **sequential learning**. In sequential learning, the network receives p input-target pairs periodically as time-varying signal. If we continue to feed the input-target pairs to the network, the network will learn the input-target pairs by converging to a set of equilibrium weights.

From an implementation point of view, the sequential learning is preferred over the parallel learning, because it requires less hardware component. Moreover, given that the patterns are presented to the network in a random manner, the use of sequential learning makes the search in weight space stochastic in nature. This randomization makes the search less likely for the back-propagation algorithm to be trapped in a local minimum. On the other hand, the use of parallel learning provides a more accurate estimate of the gradient vector, while sequential learning gets the time-averaged value in weight space. The relative effectiveness of the two learning modes depends on the problem [1][48].

2.4 Feedback neural network model

Biological systems have some sort of feedback among various neurons. A more realistic neural network model should have such feedback path.

In this section, We briefly introduce the Hopfield model [21][22][23]. Extensions to the Hopfield model have generated interest in a new class of dynamic network models called *recurrent neural networks* which are capable of performing a wide variety of

computational tasks including sequence recognition, trajectory following, nonlinear prediction, and system modeling [24].

In the Hopfield model, each neuron has a nondecreasing sigmoid nonlinearity. Its output is fed back to all other neurons via synaptic weights. Each synaptic weight is denoted by T_{ij}, which connects from the output of neuron j to the input of the neuron i. There is a symmetric requirement on the connections, namely, we must have $T_{ij} = T_{ji}$. The Hopfield model is shown in Figure 2.5.

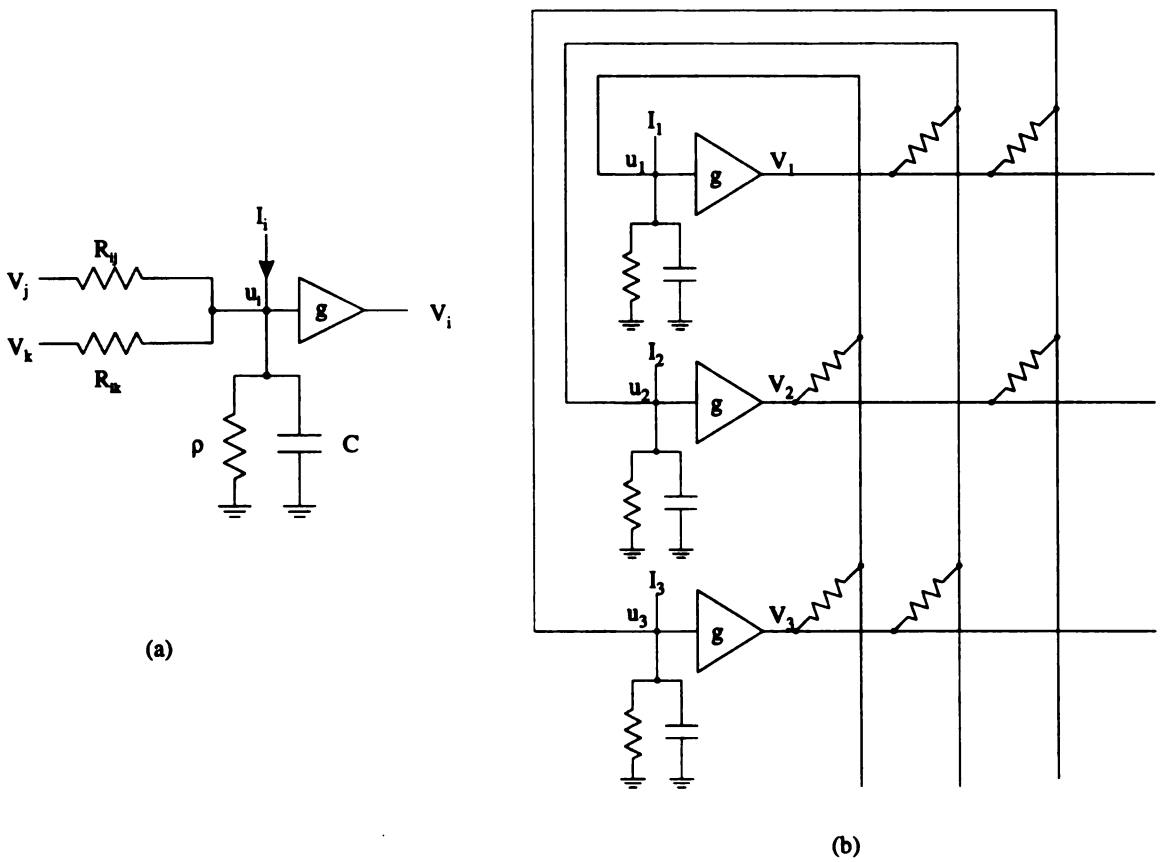

Figure 2.5. (a) The circuit of one unit in Hopfield model. (b)A three neuron Hopfield neural network.

Each unit i is consists of the circuit shown in Figure 2.5(a). u_i is the input voltage,

V_i is the output voltage, and the amplifier has the transfer function of $V_i = g(u_i)$. I_i is the external input to the node i. The input of each unit is connected to ground with a resistor R and a capacitor C. The output of unit j is connected to the input of unit i with a resistor R_{ij}.

The circuit equations are

$$C\frac{du_i}{dt} + \frac{u_i}{\rho} = \sum_j \frac{1}{R_{ij}}(V_j - u_i) + I_i \tag{2.45}$$

or, equivalently

$$C\frac{du_i}{dt} = \sum_j T_{ij}V_j - \frac{1}{R_i}u_i + I_i \tag{2.46}$$

where

$$T_{ij} = \frac{1}{R_{ij}} \tag{2.47}$$

$$\frac{1}{R_i} = \frac{1}{\rho} + \sum_j \frac{1}{R_{ij}} \tag{2.48}$$

The equilibrium points of the above neural system are the roots of the n simultaneous equations

$$0 = \sum_j T_{ij}V_j - \frac{1}{R_i}g_i^{-1}(V_i) + I_i \tag{2.49}$$

The dynamic behavior of the Hopfield model can be examined by considering the energy function of equation (2.46). Equation (2.46) can be rewritten as

$$C\frac{du_i}{dt} = -\frac{\partial E}{\partial V_i} \tag{2.50}$$

where the energy function, E, is given by

$$E = -\frac{1}{2}\sum_i \sum_j T_{ij}V_iV_j + \sum_i \frac{1}{R_i}\int_0^{V_i} g_i^{-1}(V_i)dV - \sum_i I_iV_i \tag{2.51}$$

The time derivative of the energy function along trajectories is

$$\begin{aligned}
\frac{dE}{dt} &= -\sum_i \frac{\partial E}{\partial u_i}\frac{du_i}{dt} \\
&= -\sum_i \frac{\partial E}{\partial V_i}\frac{dV_i}{du_i}\frac{du_i}{dt} \\
&= -\sum_i \frac{1}{C}\frac{dV_i}{du_i}\left(\frac{\partial E}{\partial V_i}\right)^2 \\
&= -\sum_i \frac{1}{C}\frac{dg_i(u_i)}{du_i}\left(\frac{\partial E}{\partial V_i}\right)^2 \\
&\leq 0
\end{aligned}$$

since $C \geq 0$ and $g_i(u_i)$ is a monotone nondecreasing sigmoid function. Therefore, this system is a gradient-descent system. That is, this energy function decreases along trajectories and its time-derivative equals zero at $\frac{\partial E}{\partial V_i} = 0$, which is an equilibrium point of this system.

The Hopfield model is proposed as an associative memory or to solve optimization problems.

CHAPTER 3

Analog CMOS Circuit Blocks

3.1 CMOS transistor

Research into some of the unusual electrical properties of semiconductors led to the development of the transistor, a device for controlling the flow of electrons in a solid crystal. Like a switch, a transistor can either allow or inhibit the flow of electric current in response to an external signal. Metal-Oxide-Silicon (MOS) Field Effect Transistors are commonly used in digital and analog electronics. A MOS transistor is formed by creating islands of semiconducting material, doped with either negative *N-type* or positive *P-type* charge carriers, in a substrate of the same material doped with charge carriers of the opposite type. The schematic drawing of an N-channel MOS transistor and P-channel MOS transistor is shown in Figure 3.1.

By alternating the voltage applied to the gate, charge carriers from the source are either attracted toward the channel or repelled from the channel. A channel under the gate is formed since enough attracted or repelled charges are accumulated. The channel allows current flows all the way across the gate region.

The first MOS electronic circuits employed p-channel (PMOS) transistors. As MOS technology advanced, n-channel (NMOS) transistors replaced PMOS transistors because they offered higher speed performance than PMOS. The need for reduced

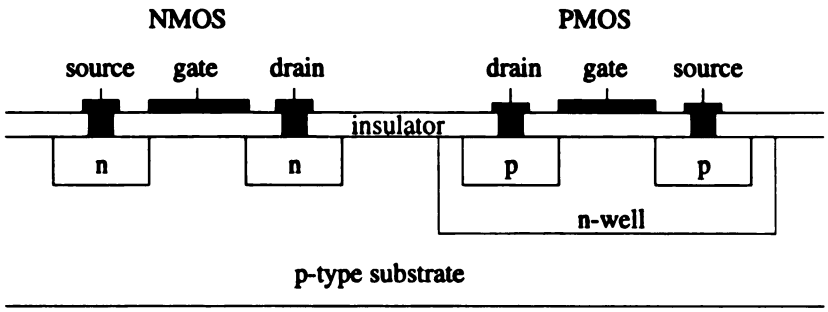

Figure 3.1. The schematic diagram of NMOS and PMOS

power consumption led to the development of the larger but more powerful efficient Complementary MOS (CMOS) transistors.

3.1.1 MOS Models

The n-channel and p-channel enhancement MOS devices along with the convention for the electrical variables are shown in Figure 3.2.

In Figure 3.2, (a) and (b) shows the convention of the four terminal device respectively. If the bulk terminal of NMOS is connected to the lowest circuit level, usually V_{SS} or GND, the bulk the convention of (c) is used. This is equivalent to (d). In (d), the bulk terminal of PMOS is connected to the highest voltage of the circuit, V_{DD}.

The following model equations is restricted to the n-channel transistor. The p-channel model equation is identical with the exception of sign changes in some of the equations. The same model is used for the PMOS if all the voltages and currents are multiplied by -1 and the absolute value of the p-channel threshold is used.

When the length or width of the MOS is greater than about 10μm, the substrate doping is low, and when a simple model is desired, the model suggested by Sah [29] and used in SPICE by Shichman and Hodges [30] is very appropriate. The dc model

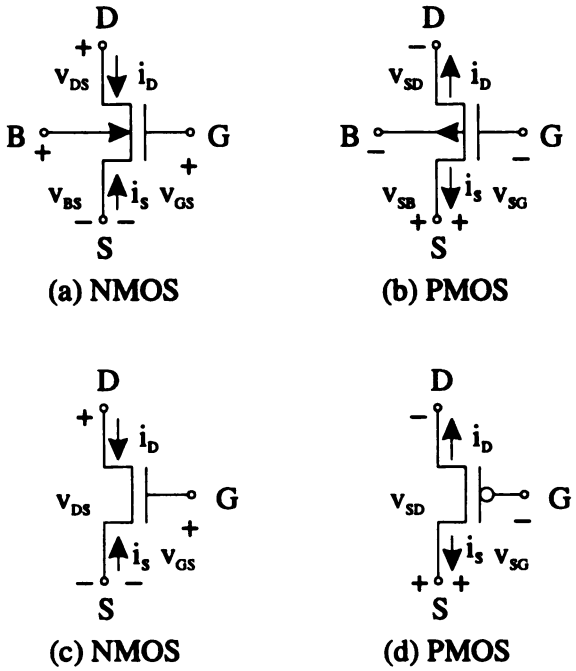

Figure 3.2. The convention of NMOS and PMOS, D:drain, S:source, G:gate, B:bulk

introduced by Sah is given by

$$I_D = \beta((V_{GS} - V_T) - \frac{V_{DS}}{2})V_{DS} \tag{3.1}$$

which was derived for small value of V_{DS}, $0 < V_{DS} \leq V_{GS} - V_T$. Small values of V_{DS} correspond to the ohmic region of operation. The region is termed the *ohmic, linear,* or *active region.* In this equation,

$$\begin{aligned}
\beta &= \textit{the transconductance parameter} \\
&= K\frac{W}{L} = (\mu_0 C_{ox})\frac{W}{L}(amps/volt_2), \\
K &= \mu_0 C_{ox} \\
\mu_0 &= \textit{surface mobility of the channel}, \\
C_{ox} &= \textit{capacitance per unit area of the gate oxide} = \frac{\epsilon_{ox}}{t_{ox}},
\end{aligned}$$

L = *channel length*,

W = *channel width*,

V_T = *threshold voltage*

For $V_{DS} \geq V_{GS} - V_T > 0$, the current remains practically constant(independent of V_{DS}) at the obtained when the channel is pinched off. The equation is obtained by

$$I_D = \frac{\beta}{2}(V_{GS} - V_T)^2 \tag{3.2}$$

which is good for $V_{DS} > V_{GS} - V_T$ and $V_{GS} > V_T$. The region of operation is termed the *saturation region*. When the MOSFET is operating in the saturation region, the MOSFET is inherently a transconductance-type device with the voltage input, V_{GS}, and the current output, I_D.

If $V_{GS} - V_T$ is zero or negative, then the MOS is in the *cutoff region* and the current becomes zero.

$$I_D = 0, \quad V_{GS} - V_T \leq 0 \tag{3.3}$$

The model based on the equations (3.1), (3.2) and, (3.3) is the simplest model. In many situations, this model is quite tractable for hand calculations and adequate for the analytical portions of the design.

It can be shown theoretically and experimentally that the drain current in the saturation region increases slightly in a linear manner with V_{DS}. Defining λ to be the coefficient that represents the linear dependence of I_D on V_{DS}, a more accurate expressions for the drain current in the saturation region is given by

$$I_D = \frac{\beta}{2}(V_{GS} - V_T)^2(1 + \lambda V_{DS}) \tag{3.4}$$

The coefficient λ is quite small for long devices but increases considerably for very

short transistors.

The W/L ratio is the only geometrical design parameter available to the design engineer that affects the performance of MOS transistor. Assuming the parameter K and V_T are constant, it can be shown that the device is electrically symmetric with respect to drain and source. The choice of which end of the channel to designate as source and drain is thus arbitrary. Since the MOS is a bi-directional device, the source for an n-channel transistor is always at the lower potential of the two nodes. For the p-channel transistor, the source is always at the higher potential.

The threshold voltage, V_T, is somewhat dependent upon the bulk-source voltage. The dependence can be approximated by

$$V_T = V_{T0} + \gamma(\sqrt{\phi - V_{BS}} - \sqrt{\phi}) \tag{3.5}$$

where V_{BS} is the bulk-source voltage and V_{T0}, γ, and ϕ are process parameters:

$$\begin{aligned} V_{T0} &= \textit{threshold voltage for } V_{BS} = 0 \\ \gamma &= \textit{bulk threshold parameter} \\ \phi &= \textit{strong inversion surface potential} \end{aligned}$$

Note that the change in V_T can be quite significant for large V_{BS}. The effect becomes even worse with larger λ.

This simple model has five electrical and process parameters that completely define it. These parameters are K, V_T, γ, λ, and ϕ. They constitute the Level 1 model parameters of SPICE (Simulation Program with Integrated Circuit Emphasis) circuit simulator [31][32]. In many situations, this model is quite tractable for hand calculations and adequate for the analytical portions of the design. With the Level 1 model, the simulation does not perform the short- and narrow-channel effects.

In the Level 2 model based on Meyer [33], the short- and narrow-channel effects are calculated. The Level 2 model differs from the Level 1 model both in its method of calculating the effective channel length (λ effects) and the transition between the saturation and ohmic region. The Level 2 model offers improvements in performance which are particularly significant for short channel devices.

The Level 3 model has been developed to simulate a semi-empirical model. Several empirical parameters (parameters not obviously related to or motivated by the device physics of the MOS transistor) are introduced in the Level 3 model. It simulates quite precisely the characteristics of MOS which have a channel length up to 2μm. The basic equations have been proposed by Dang [34].

It is useful to examine the differences among the three models [31]. The Level 1 model is elementary, the Level 2 model uses processing parameters and geometry, and the Level 3 model uses measured characteristics. Usually the Level 1 model is not sufficiently precise because the theory is too approximated and the number of fitting parameters too small; its usefulness is in a quick and rough estimate of circuit performances. The Level 2 model can be used with differing complexity by adding the parameters relating to the effects needed to simulate with this model. However, if all the parameters are used, i.e., the greatest possible complexity is obtained, this model requires a great amount of CPU time for the calculations, and it often causes problems with convergence [35]. The Level 3 model takes less time and less errors on simulation than the Level 2 model. The only disadvantage of the Level 3 model is the complexity in the calculation of some of its parameters.

3.2 Transconductance amplifier

The transconductance amplifier is a device that generates current as its output. If the output current is proportional to the difference between two input voltages, V_1

and V_2, the circuit is called a **differential transconductance amplifier.** Since the input terminal of the transconductance amplifier receives the voltage input, we usually need current-to-voltage conversion at the output stage of the amplifier. The current-to-voltage converter is implemented using operational amplifier and resistors. However, it is not practical to use operational amplifier inside the neural network implementation. Thus a simple and convenient current-to-voltage converter is needed.

3.2.1 Active resistors/Loads

The current-to-voltage converter is implemented using two transistors. The active resistor is used to produce a dc-voltage drop or provide a resistance which is linear over a small range. The active resistor is achieved by simply connecting the gate to the drain. The active resistor is shown in Figure 3.3.

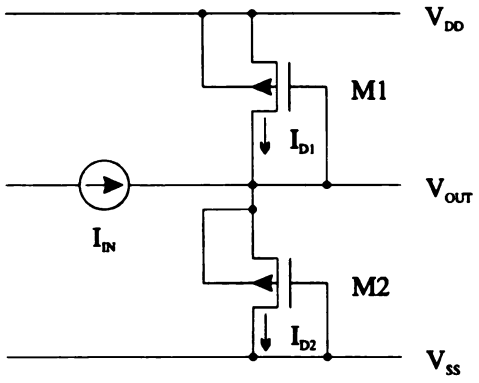

Figure 3.3. The circuit diagram of active resistors using PMOS

In this figure, there is a current source I_{IN} and two PMOS transistors. If we assume that M1 and M2 is a matched pair, i.e., they have the same conductance β, and work in saturation region, we have the circuit equation by applying Kirchhoff's

current law at the V_{OUT} node:

$$I_{IN} + I_{D1} = I_{D2} \tag{3.6}$$

$$I_{D1} = \frac{\beta}{2}(V_{DD} - V_{OUT} - V_{T1})^2 \tag{3.7}$$

$$I_{D2} = \frac{\beta}{2}(V_{OUT} - V_{SS} - V_{T2})^2 \tag{3.8}$$

After some algebraic manipulations with the assumption, $V_T = V_{T1} = V_{T2}$, V_{OUT} is given as

$$V_{OUT} = \frac{1}{(V_{DD} - V_{SS} - 2V_T)}\frac{1}{\beta}I_{IN} + \frac{(V_{DD} + V_{SS})}{2} \tag{3.9}$$

$$= \frac{1}{G}I_{IN} + \frac{(V_{DD} + V_{SS})}{2} \tag{3.10}$$

where G is the conductance of the active resister,

$$G = (V_{DD} - V_{SS} - 2V_T)\beta \tag{3.11}$$

If we assume that V_{DD} is 5V and V_{SS} is 0V, then V_{OUT} will be 2.5V when I_{IN} is zero.

In Figure 3.4, the characteristics of current-to-voltage relation are shown with different W/L ratios of the active resistor. In this graph, the W/L ratio of the upper PMOS is assumed to be same as the W/L ratio of the lower PMOS. We can see that the resistance increases as the W/L ratio of PMOS decreases. In Table 3.1, the calculated resistances at the $(0A, 2.5V)$ point from Figure 3.4 are shown with different W/L ratios. From the table, the value of resistance of active resistors ranges from $10K\Omega$ to $1M\Omega$.

If we need larger active resistors but we have a limitation of area in chip layout for active resistors, we can cascade active resistors. Two upper and two lower active

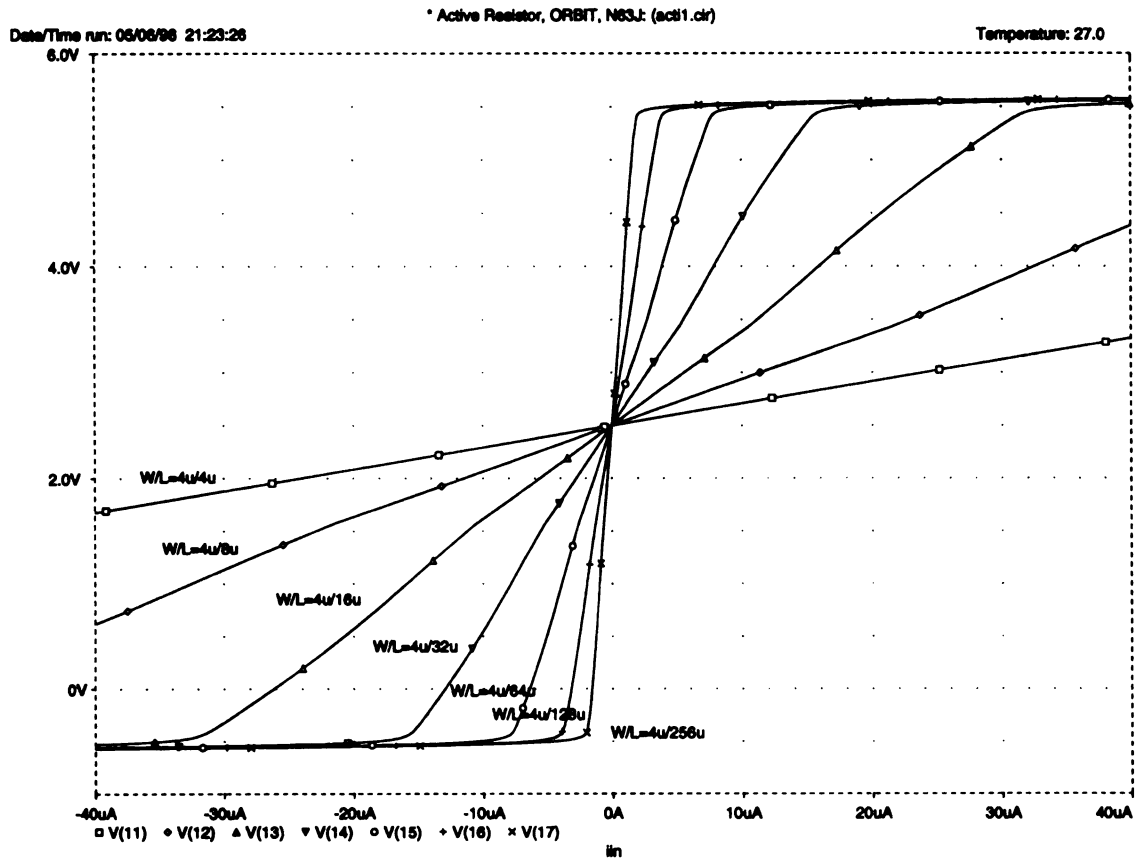

Figure 3.4. The PSPICE circuit simulation result: Active resistors with different W/L ratios

Table 3.1. Resistance with different W/L ratios for Active resistors

$(W/L)_{M1}$	$(W/L)_{M2}$	Resistance, $R = \frac{1}{G} = \frac{\Delta V}{\Delta I}$
$4\mu/4\mu$	$4\mu/4\mu$	$20.61K\Omega$
$4\mu/8\mu$	$4\mu/8\mu$	$47.03K\Omega$
$4\mu/16\mu$	$4\mu/16\mu$	$95.85K\Omega$
$4\mu/32\mu$	$4\mu/32\mu$	$192.94K\Omega$
$4\mu/64\mu$	$4\mu/64\mu$	$392.04K\Omega$
$4\mu/128\mu$	$4\mu/128\mu$	$786.05K\Omega$
$4\mu/256\mu$	$4\mu/256\mu$	$1573.1K\Omega$

resistors approximately have 9 times greater resistance than that of single upper and single lower active resistor with the same W/L ratio. The circuit diagram of the two cascaded active resistors and its PSPICE simulations are illustrated in Figure 3.5 and 3.6. In Table 3.2, the resistances of the cascaded active resistors are shown.

To make active resistors operate correctly, we have to consider the body effect of the transistor. To make two threshold voltages, V_{T1} and V_{T2}, the same, M1 and M2 should have the same bulk-source voltage, V_{BS}, according to equation (3.5).

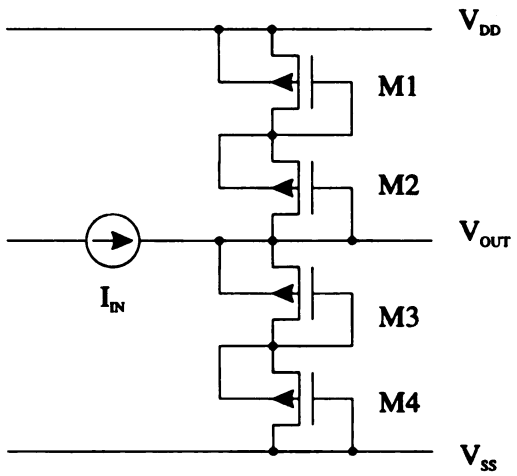

Figure 3.5. The circuit diagram of the cascaded active resistors using PMOS

That is why each transistor's bulk terminal is connected to its source. The selection of NMOS or PMOS as linear active resistors depends on the fabrication technology. If n-well is used for fabrication, PMOS is used as active resistors because we can isolate the bulk voltage of PMOS, i.e., the bulk voltage of NMOS is tied together to the p-type substrate. When p-well is used for fabrication, NMOS should be used as active resistors since we can isolate the p-well only.

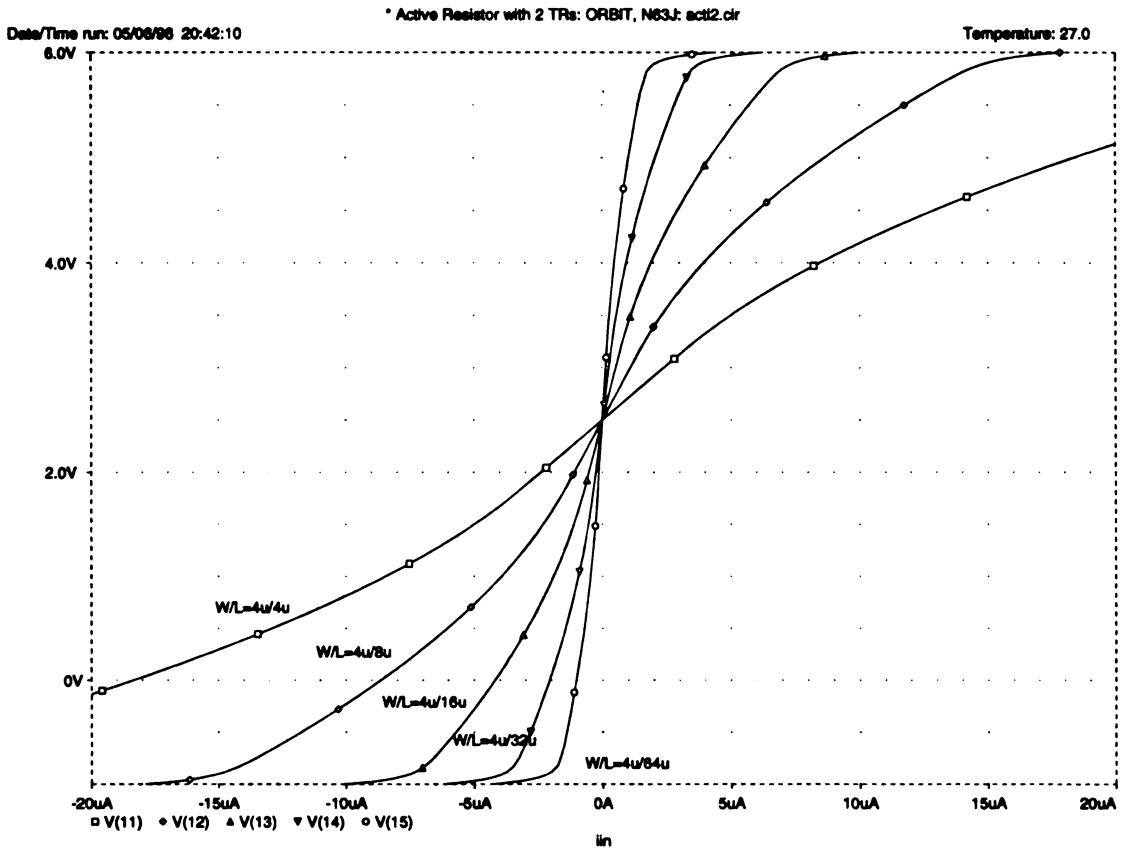

Figure 3.6. The PSPICE circuit simulation result: The cascaded active resistors with different W/L ratios

Table 3.2. Resistance with different W/L ratios for the cascaded active resistor

$(W/L)_{M1} = (W/L)_{M2}$	$(W/L)_{M3} = (W/L)_{M4}$	Resistance, $R = \frac{1}{G} = \frac{\Delta V}{\Delta I}$
$4\mu/4\mu$	$4\mu/4\mu$	$208.81K\Omega$
$4\mu/8\mu$	$4\mu/8\mu$	$455.80K\Omega$
$4\mu/16\mu$	$4\mu/16\mu$	$893.65K\Omega$
$4\mu/32\mu$	$4\mu/32\mu$	$1730.4K\Omega$
$4\mu/64\mu$	$4\mu/64\mu$	$3418.6K\Omega$

3.2.2 Simple transconductance amplifier

The differential transconductance amplifier is one of the most versatile circuits in analog circuit design [36][37]. The objective of the differential amplifier is to amplify the difference between two different voltages regardless of the common-mode value. The differential amplifier is characterized by its common-mode rejection ratio (CMRR) which is the ratio of the differential gain to the common-mode gain. Another characteristic affecting performance of the differential amplifier is voltage offset. If the terminals of the differential amplifier are connected together, the output offset voltage is the voltage which appears at the output of the differential amplifier.

Let us consider the large- and small-signal characteristic of the CMOS differential amplifier [36]. Figure 3.7 shows a CMOS differential amplifier that uses n-channel MOS devices, M1 and M2, as the differential pair. M3 is a current source and the loads for M1 and M2 are obtained from a simple p-channel current mirror (M4 and M5). If M4 and M5 is matched, then the current of M1 will determine the current in

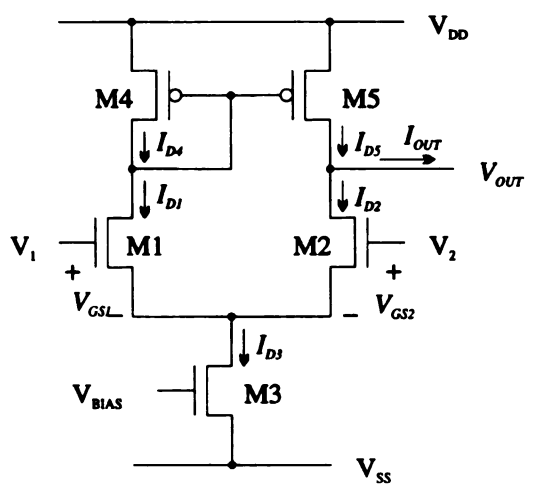

Figure 3.7. Simple transconductance amplifier

M4. This current will be mirrored in M5. If $V_1 = V_2$, then the currents in M1 and M2 are equal. Thus the current in M5 is equal to the current in M2, causing I_{out} to be zero. If $V_1 > V_2$, then I_{D1} increases with respect to I_{D2} since $I_{D3} = I_{D1} + I_{D2}$. This increase in I_{D1} implies an increase in I_{D4} and I_{D5}. However, I_{D2} is decreased with respect to I_{D1}, therefore, the only way to establish circuit equilibrium for I_{OUT} is to become positive. It can be seen that if $V_1 < V_2$ then I_{OUT} becomes negative.

The large-signal characteristics can be developed by assuming that M1 and M2, the differential pair, are always in saturation. The relationship describing large-signal behavior are given as

$$V_{ID} = V_{GS1} - V_{GS2} = \sqrt{\frac{2I_{D1}}{\beta}} - \sqrt{\frac{2I_{D2}}{\beta}} \tag{3.12}$$

$$I_{D3} = I_{D1} + I_{D2} \tag{3.13}$$

where it has been assumed that M1 and M2 are matched ($\beta = \beta_1 = \beta_2$). The solution for I_{D1} and I_{D2} is given by

$$I_{D1} = \frac{I_{D3}}{2} + \frac{I_{D3}V_{ID}}{2}\sqrt{\frac{\beta}{I_{D3}} - \frac{\beta^2 V_{ID}^2}{4I_{D3}^2}} \tag{3.14}$$

$$I_{D2} = \frac{I_{D3}}{2} - \frac{I_{D3}V_{ID}}{2}\sqrt{\frac{\beta}{I_{D3}} - \frac{\beta^2 V_{ID}^2}{4I_{D3}^2}} \tag{3.15}$$

where these relationships are valid only for $V_{ID} < \sqrt{\frac{2I_{D3}}{\beta}}$. Figure 3.8 shows a plot of the normalized drain current of M1 and M2 versus the normalized input voltage. If we assume that the currents in the current mirror are identical, then I_{OUT} can be found by subtracting I_{D2} from I_{D1}. The output voltage V_{OUT} of the differential transconductance amplifier can be found by assuming that a load resistance R_L is

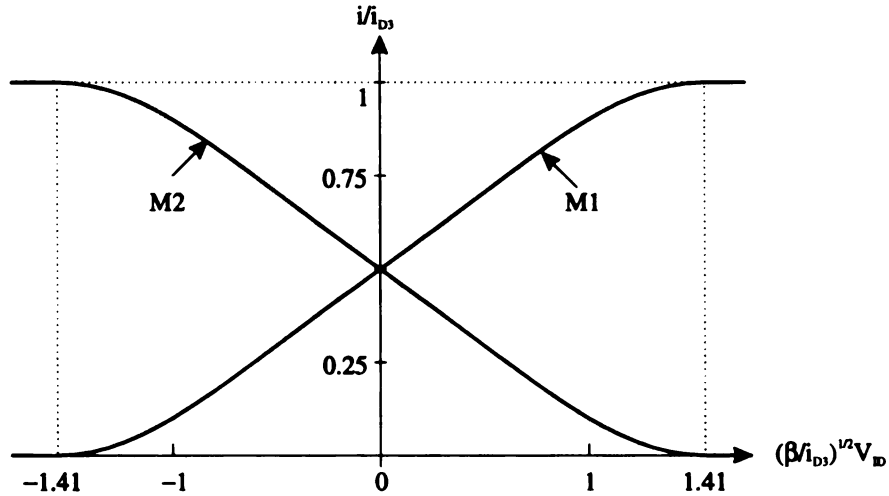

Figure 3.8. Current output of the simple transconductance amplifier

connected from the output of the amplifier to ground.

$$V_{OUT} = I_{OUT}R_L = (I_{D1} - I_{D2})R_L = I_{D3}V_{ID}\sqrt{\frac{\beta}{I_{D3}} - \frac{\beta^2 V_{ID}^2}{4I_{D3}^2}}R_L \tag{3.16}$$

The differential-in, differential-out transconductance is written as

$$g_{md} = \frac{\partial I_{OUT}}{\partial V_{ID}}\bigg|_{(V_{ID}=0)} = \sqrt{\beta I_{D3}} = \sqrt{\frac{K_1 I_{D3} W_1}{L_1}} \tag{3.17}$$

The small-signal voltage gain of the differential amplifier can be found by differentiating Eq.(3.16) with respect to V_{ID} and setting $V_{ID} = 0$, giving

$$A_V = \frac{\partial V_{OUT}}{\partial V_{ID}} = \sqrt{\beta I_{D3}}R_L = \sqrt{\frac{K_1 I_{D3} W_1}{L_1}}R_L \tag{3.18}$$

We note several important deviations from ideal behavior of the differential amplifier. The first is the mismatch between transistors. Not all transistors are created equal. Some are created with a higher transconductance property than are others. It causes current mismatch and shift in characteristic curve and results in voltage offset.

The designer must consider the worst case V_T spread (specified by the process) in each transistor and adjust current level and transconductance to meet the requirements. Typically, the voltage offset of a CMOS differential amplifier is 5 to 20 millivolts.

The common mode gain of the CMOS differential amplifier is ideally zero, because the current-mirror load rejects any common-mode signal. Due to the mismatches in the differential amplifier, a common-mode response might exist. This mismatches consist of a non-unity current gain in the current mirror and geometrical mismatches between M1 and M2.

Another important characteristic is the input common-mode voltage range. The input common-mode range is defined by the input voltage range over which both M1 and M2 remain in saturation. The highest input voltage at the gate of M1 (or M2) when $V_1 = V_2$ is found to be

$$V_1 = V_{DD} - V_{SG4} - V_{DS1} + V_{GS1} \tag{3.19}$$

For saturation, the minimum value of V_{DS1} is

$$V_{DS1} = V_{GS1} - V_{TO1} \tag{3.20}$$

Substituting and replacing the equations give the final result,

$$V_1 = V_{DD} - \sqrt{\frac{2I_{D4}}{\beta_4}} - |V_{TO4}| + V_{TO1} \tag{3.21}$$

The last two terms are determined by the process and the equations now becomes

$$V_1(max) = V_{DD} - \sqrt{\frac{I_{D3}}{\beta_4}} - |V_{TO4}|(max) + V_{TO1}(min) \tag{3.22}$$

As V_1 approaches V_{SS}, M1 will be in the saturation region and close to cutoff.

Therefore, it makes more sense to relate $V_1(min)$ to V_{BIAS} when M3 is no longer in saturation. The gate voltage on M1 can be shown to be

$$V_1 = V_{GS1} + V_{DS3} \tag{3.23}$$

Set $V_{DS3} = V_{GS3} - V_{TO3}$ to get

$$V_1 = V_{GS3} + V_{GS1} - V_{TO3} \tag{3.24}$$

$$V_1(min) = V_{BIAS} + \sqrt{\frac{2I_{D3}}{\beta_1}} + V_{TO1}(max) - V_{TO3}(min) \tag{3.25}$$

Third, the common-mode input voltage has a significant effect on the transfer function, particularly the output-signal swing. In this case, the swing limitation will be based on keeping both M2 and M5 in saturation. When V_1 is taken above V_2, the output voltage, V_{OUT}, increases. The output voltage is given as

$$V_{OUT} = V_{DD} - V_{SD5} \tag{3.26}$$

M5 is at the edge of saturation when $V_{SD5} = V_{SG5} - |V_{TO5}|$. Using this relationship, the maximum output voltage is given as

$$V_{OUT} = V_{DD} - V_{SG4} + |V_{TO5}| \tag{3.27}$$

$$V_{OUT}(max) = V_{DD} - \sqrt{\frac{2I_{D4}}{\beta_4}} - |V_{TO4}| + |V_{TO5}| \tag{3.28}$$

The minimum output voltage is found by determining when M2 is at the edge of saturation. The minimum output voltage is given by

$$V_{OUT}(min) = V_2 - V_{T2} \tag{3.29}$$

Table 3.3. The transistor sizes of the simple transconductance amplifier

Transistor	W/L ratio
M1	$4\mu/4\mu$
M2	$4\mu/4\mu$
M3	$12\mu/4\mu$
M4	$15\mu/4\mu$
M5	$15\mu/4\mu$

To verify the limitation on the output voltage range, a PSPICE simulation has been performed. To see the output current, we attached two resistors at the end of the output terminal. Since two resistors have the same value of $100K\Omega$, the output current is $I_{R1} - I_{R2}$. The circuit diagram is shown in Figure 3.9

The W/L ratios of the transistors are shown in Table 3.3. Figure 3.10 shows the PSPICE simulation result of the differential transconductance amplifier. The influence of V_2 upon I_{OUT} is illustrated in this figure. The PSPICE input file is shown in Appendix A.1.

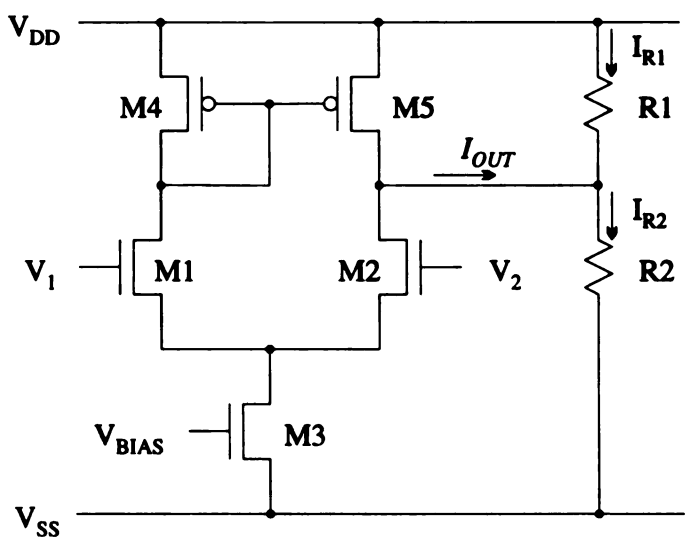

Figure 3.9. Simple transconductance amplifier with load resistors

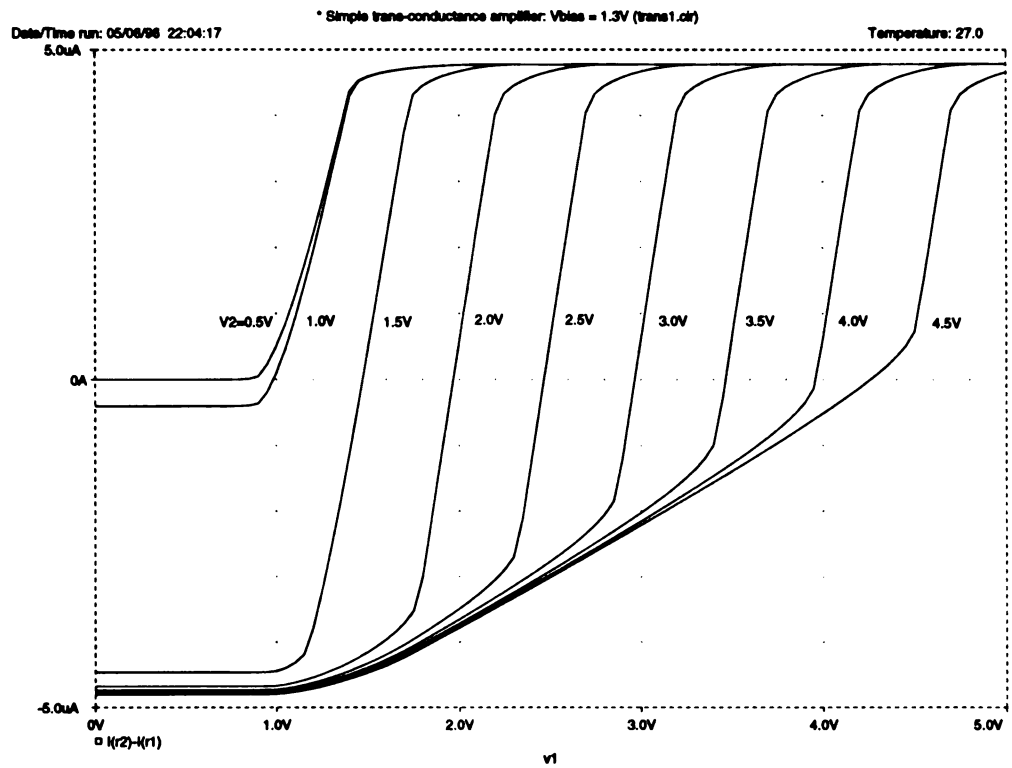

Figure 3.10. Simple transconductance amplifier: Output current ($I_{R1} - I_{R2}$) as function of V1 for different values of V2

3.2.3 Wide range transconductance amplifier

A simple transconductance amplifier will not generate output voltage below $V_{OUT}(min)$, which, in turn, is dependent on the input voltages. We can remove this restriction by a simple addition to the simple transconductance amplifier, as shown in Figure 3.11.

To overcome the problem, two extra current mirrors are usually added [38][39]. By reflecting the currents of M1 and M2 to upper current mirrors, the output current is just the difference between I_1 and I_2. The major advantage of the wide-range amplifier over the simple circuit is that both input and output voltages can run almost up to V_{DD} and almost down to V_{SS}, without affecting the operation of the circuit.

The output current, I_{OUT}, in the PSPICE simulation is shown in Figure 3.12 and Figure 3.13. $100K\Omega$ resistors are used in Figure 3.12, and $1M\Omega$ resistors are used in

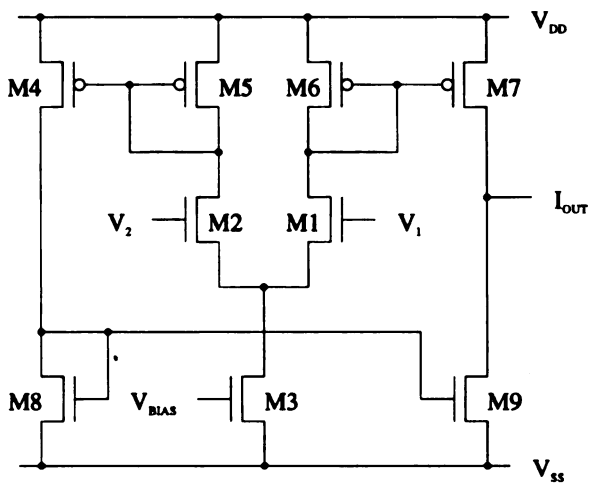

Figure 3.11. The circuit diagram of the wide range transconductance amplifier

Table 3.4. The transistor sizes of the wide range transconductance amplifier

Transistor	(W/L) ratio
M1	$4\mu/4\mu$
M2	$4\mu/4\mu$
M3	$12\mu/4\mu$
M4	$15\mu/4\mu$
M5	$15\mu/4\mu$
M6	$15\mu/4\mu$
M7	$15\mu/4\mu$
M8	$4\mu/4\mu$
M9	$4\mu/4\mu$

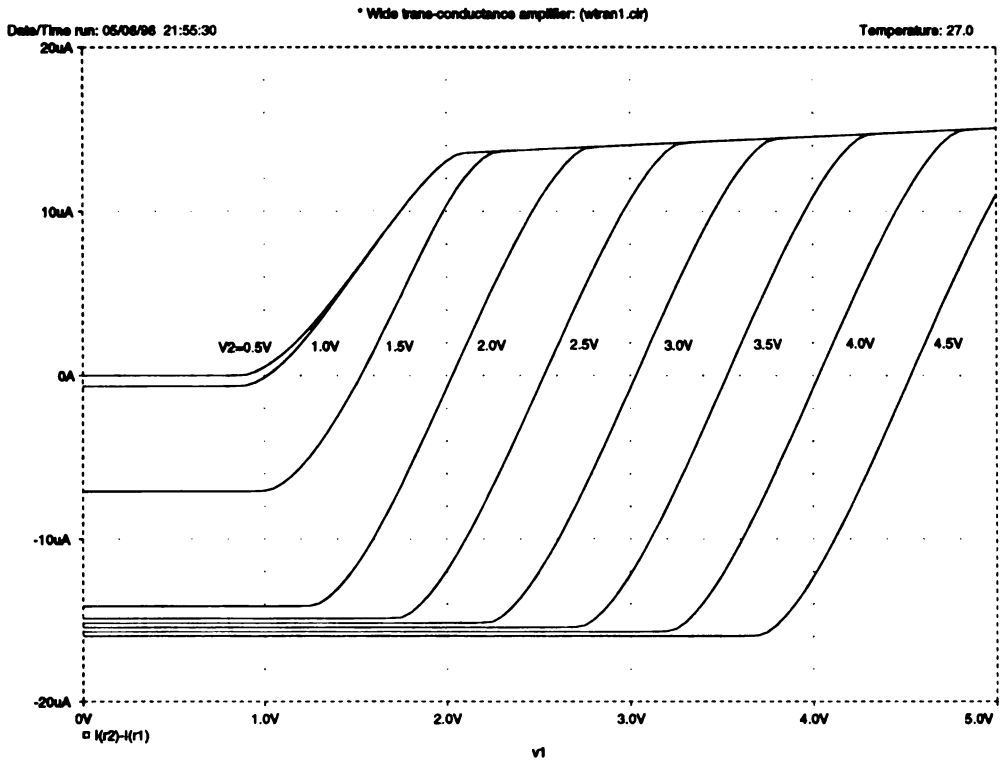

Figure 3.12. The PSPICE circuit simulation result: the wide range transconductance amplifier, output current as function of V1 for several values of V2, Resistor = $100K\Omega$

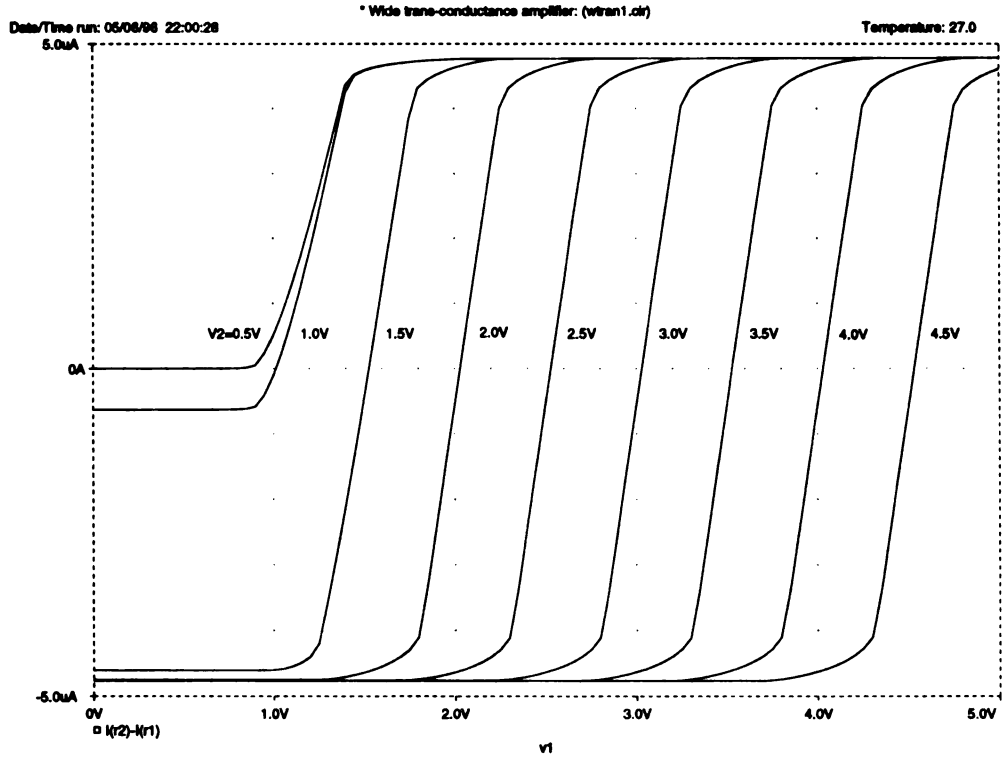

Figure 3.13. The PSPICE circuit simulation result: the wide range transconductance amplifier, output current as function of V1 for several values of V2, Resistor = $1M\Omega$

Figure 3.13. The output current represented in the difference current between R_1 and R_2. The PSPICE input file is shown in Appendix A.2 and the sizes of transistors are shown in Table 3.4. As the value of resistors increased, the saturated level of current is decreased and the current gain at the operating point is increased.

3.2.4 Sigmoid function generator

The sigmoid function can be obtained by using a wide range transconductance amplifier and active resistors. The output of the transconductance amplifier is in current. Thus we attached diode-connected MOS transistors as active resistors to convert current to voltage. Using active linear resistors, we can achieve voltage-to-voltage operations.

The complete circuit of sigmoid function generator is shown in Figure 3.14. The sizes of the transistors are shown in Table 3.5. The transconductance of the differential transconductance amplifier is proportional to the bias current. The (W/L) ratios of the M1 and M2 transistors also control the transconductance. We can achieve a larger transconductance when we increase the width of the transistor M1 and M2. The PSPICE simulation results are shown in Figure 3.15 and Figure 3.16. The PSPICE input files are shown in Appendix A.3 and A.4.

In Figure 3.15, V_{OUT} is shown with different W/L ratios of the active resistor. If the W/L ratio is decreased, the gain is that of the *tanh* like function and saturated level of high and low voltage is increased. In Figure 3.16, V_{OUT} is plotted according to the different V_{BIAS} voltages. From this figure, it is obvious that the gain of the sigmoid function generator and the level of the saturated voltage are dependent on the V_{BIAS} voltage.

From the PSPICE circuit simulation, we have two parameters to control the characteristics of the sigmoid function generator. The W/L ratio of the active resistor and the bias voltage are the controllable parameters.

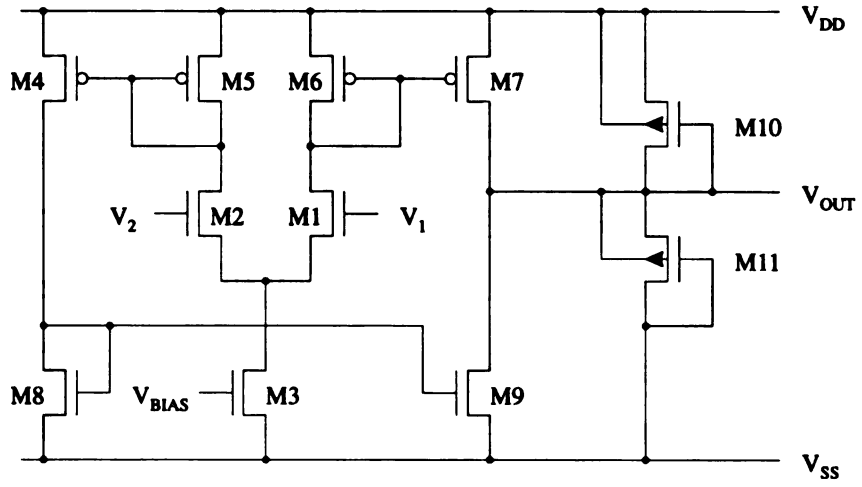

Figure 3.14. The circuit diagram of sigmoid function generator

Table 3.5. The transistor sizes of the sigmoid function generator

Transistor	(W/L) ratio
M1	$24\mu/4\mu$
M2	$24\mu/4\mu$
M3	$10\mu/4\mu$
M4	$15\mu/4\mu$
M5	$15\mu/4\mu$
M6	$15\mu/4\mu$
M7	$15\mu/4\mu$
M8	$16\mu/4\mu$
M9	$16\mu/4\mu$
M10	$4\mu/19\mu$
M11	$4\mu/21\mu$

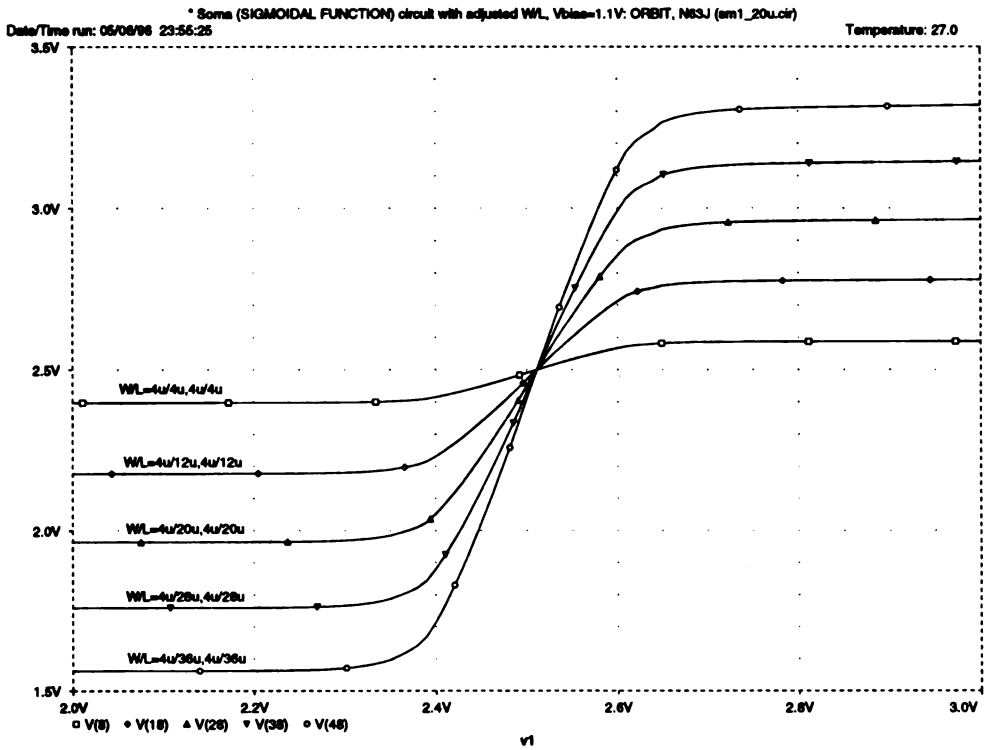

Figure 3.15. The PSPICE circuit simulation result: Sigmoid function generator with different W/L ratio of the active resistors (M10 and M11) in PMOS when the bias voltage is 1.1V

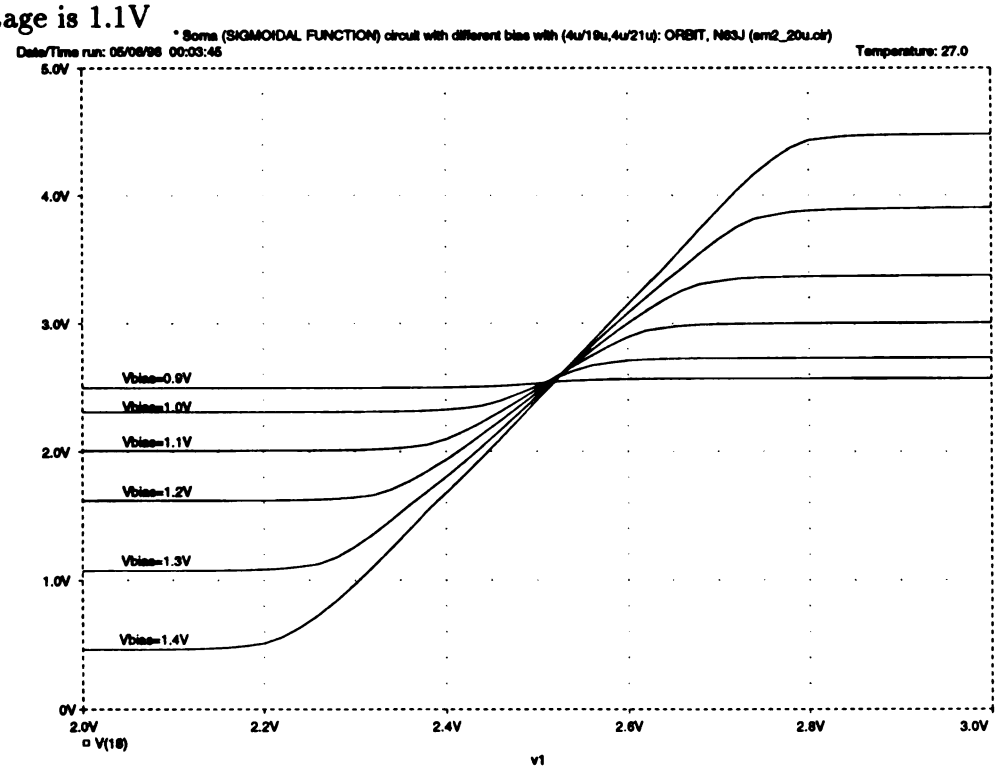

Figure 3.16. The PSPICE circuit simulation result: Sigmoid function generator with different bias voltages when (W/L) of the active resister (M10 and M11) is (4μm/19μm,4μm/21μm)

3.3 The Modified Gilbert Multiplier

To implement the multiplication of the neural networks, a multiplier is one of the most important component. Several different techniques are used in the multiplier design. Multipliers based on a modified Gilbert cell [40][41][42] are popular. The other technique is the use of the square-law characteristics in saturation region [43][44][45]. Others have implemented the multiplier based on the current-voltage characteristics in the non-saturation region [46].

We consider a modified Gilbert multiplier as a synapse circuit[38]. The circuit diagram of the modified Gilbert multiplier is shown in Figure 3.17.

Assume that all transistors in Figure 3.17 are in saturation region, and are matched so that the transconductance parameters satisfy the equations $\beta_N = \beta_{M1} = \beta_{M2}$ and $\beta_P = \beta_{M3} = \beta_{M4} = \beta_{M5} = \beta_{M6}$.

The output current is the difference between $I_{D(M12)}$ and $I_{D(M13)}$ since the current $I_{S(M21)}$ and $I_{S(M22)}$ are reflected by the current mirrors. Defining the output current $I_+ = I_{S(M4)} + I_{S(M5)}$ and $I_- = I_{S(M3)} + I_{S(M6)}$, it can be readily shown that the differential output current $I_{DIFF} = I_+ - I_-$ is given by

$$I_{DIFF} = \sqrt{2\beta_P}(V_3 - V_4)\left(\sqrt{I_{D(M1)}}\sqrt{1 - \frac{\beta_P(V_3 - V_4)^2}{2I_{D(M1)}}} - \sqrt{I_{D(M2)}}\sqrt{1 - \frac{\beta_P(V_3 - V_4)^2}{2I_{D(M2)}}}\right) \tag{3.30}$$

The above equation can be approximated into

$$I_{DIFF} = \sqrt{2\beta_P}(\sqrt{I_{D(M1)}} - \sqrt{I_{D(M2)}})(V_3 - V_4) \tag{3.31}$$

if the following condition is satisfied

$$\frac{\beta_P(V_3 - V_4)^2}{2I_{D(M1)}} \ll 1$$

$$\frac{\beta_P(V_3 - V_4)^2}{2I_{D(M2)}} \ll 1$$

Also, the current $I_{D(M1)}$ and $I_{D(M2)}$ are dependent on the voltage difference $(V_1 - V_2)$. Since $(V_1 - V_2)$ is given by

$$V_1 - V_2 = \sqrt{\frac{2I_{D(M1)}}{\beta_N}} - \sqrt{\frac{2I_{D(M2)}}{\beta_N}} \tag{3.32}$$

the equation (3.31) becomes

$$I_{DIFF} = \sqrt{\beta_P \beta_N}(V_3 - V_4)(V_1 - V_2) \tag{3.33}$$

This is the ideal characteristics of the approximated equation.

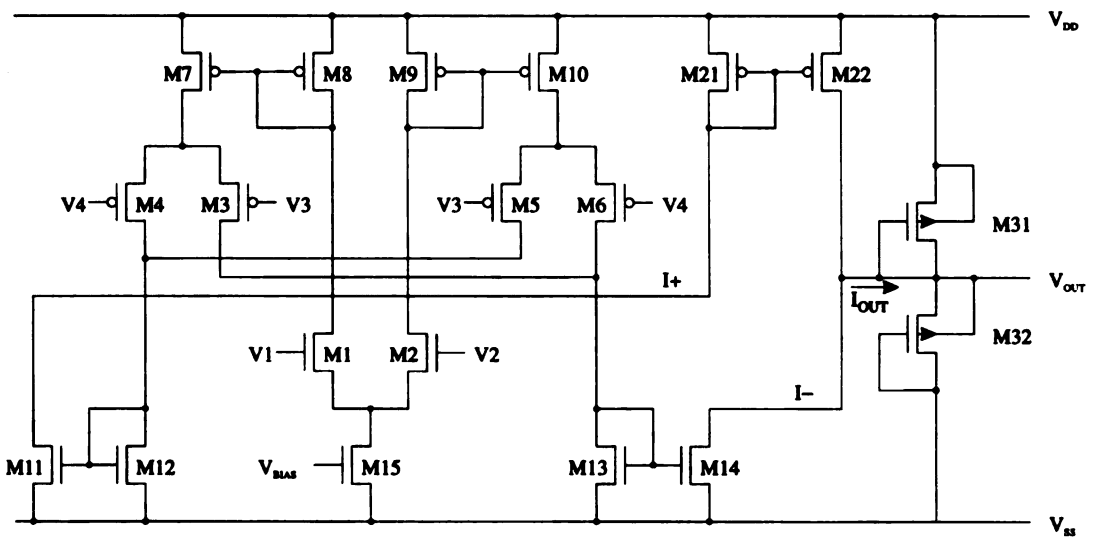

Figure 3.17. The circuit diagram of the modified Gilbert multiplier

The modified Gilbert multiplier takes the difference between two voltages $(V_3 - V_4)$ and multiply that difference by a difference two other voltages $(V_1 - V_2)$. In the small

Table 3.6. The transistor sizes of the modified Gilbert multiplier

Transistor	(W/L) ratio	Transistor	(W/L) ratio
M1	$4\mu/4\mu$	M11	$4\mu/4\mu$
M2	$4\mu/4\mu$	M12	$4\mu/4\mu$
M3	$15\mu/4\mu$	M13	$4\mu/4\mu$
M4	$15\mu/4\mu$	M14	$4\mu/4\mu$
M5	$15\mu/4\mu$	M15	$12\mu/4\mu$
M6	$15\mu/4\mu$	M21	$11\mu/14\mu$
M7	$15\mu/4\mu$	M22	$11\mu/14\mu$
M8	$15\mu/4\mu$	M31	$4\mu/12\mu$
M9	$15\mu/4\mu$	M32	$4\mu/12\mu$
M10	$15\mu/4\mu$		

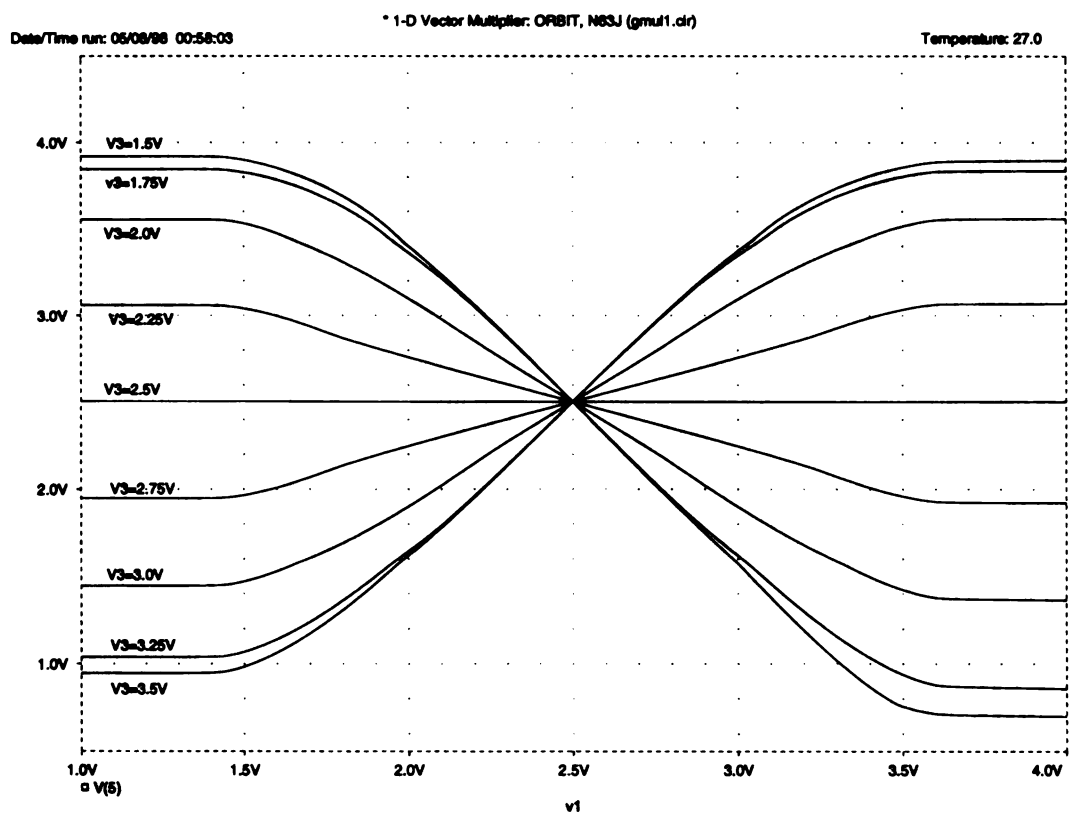

Figure 3.18. The PSPICE circuit simulation result: the modified Gilbert multiplier with V2 = V4 = 2.5V, V(5) is the output voltage

signal range, the characteristic curve is approximately linear, and all four inputs carry information about multiplication. For the large-signal range, the multiplier is nonlinear. However, the nonlinearity does not cause any instability. Since the modified Gilbert multiplier has current outputs, we employed PMOS active resistors at the output stage of the multiplier to convert current to voltage. Thus, the voltage-to-voltage multiplier is achieved.

The PSPICE circuit simulation result of the modified Gilbert multiplier is shown in Figure 3.18. In this figure, V_2 and V_4 are connected to Reference voltage, 2.5V. With different voltages of V_3, the output voltage is shown according to the input voltage, V_1. The output voltage shows four quadrant multiplication. The PSPICE input file is shown in Appendix A.5.

3.4 The Vector Multiplier

In a dot-product operation, two vectors are multiplied to generate a scalar quantity. Let $X = (x_1, x_2, \ldots, x_n)^T$ and $Y = (y_1, y_2, \ldots, y_n)^T$ be $N \times 1$ vectors. Their dot-product operation is expresses as

$$z = X^T \cdot Y = \sum_i x_i y_i \tag{3.34}$$

In a vector multiplier, the vector multiplication is given as

$$V_{out} = \sum_i (V_{i1} - V_{i2})(V_{i3} - V_{i4}) \tag{3.35}$$

To implement the dot-product operation, we use the modified Gilbert multiplier to obtain the vector multiplication. There are identical Gilbert multiplier subcircuits in the vector multiplier and they are connected together at the terminals of the current mirror. The differential output currents from the multiplier are summed on

two current buses and they are converted to voltage through active PMOS resistors. One thus can construct larger dimensional vector multipliers by simply adding the Gilbert multiplier subcircuits.

Figure 3.19 presents the 3-dimensional vector multiplier. In this figure, identical three Gilbert multiplier subcircuits are connected together via I_+ and I_- current bus.

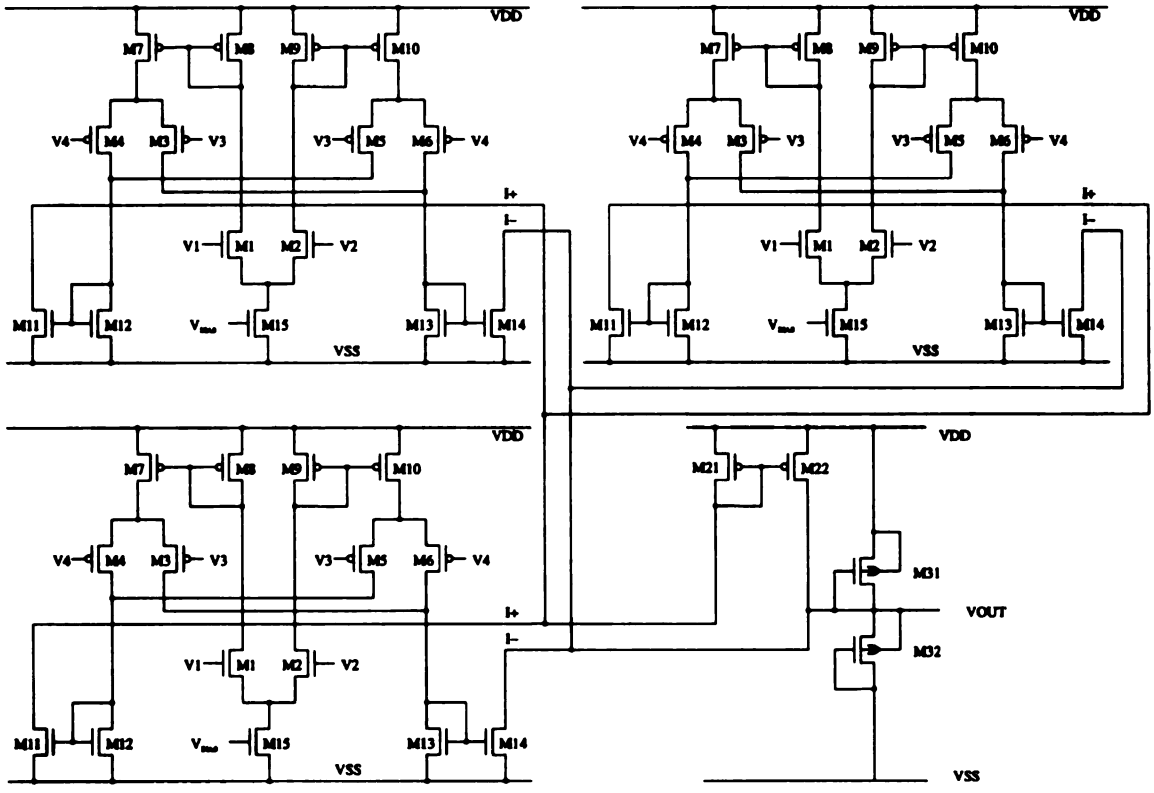

Figure 3.19. The circuit diagram of the 3-D Vector Multiplier

The PSPICE simulation result is shown in Figure 3.20. In this figure, all V1,V2,V3 and V4 of the subcircuits are connected together. V2 and V4 are connected to 2.5V and V3 is varied from 0.5V to 4.5V. The output voltage obtained by the different voltages of V1 and V3 are shown in this figure. The output voltage gain and the saturated voltage are controllable by the V_{BIAS} voltage and the (W/L) ratio of active

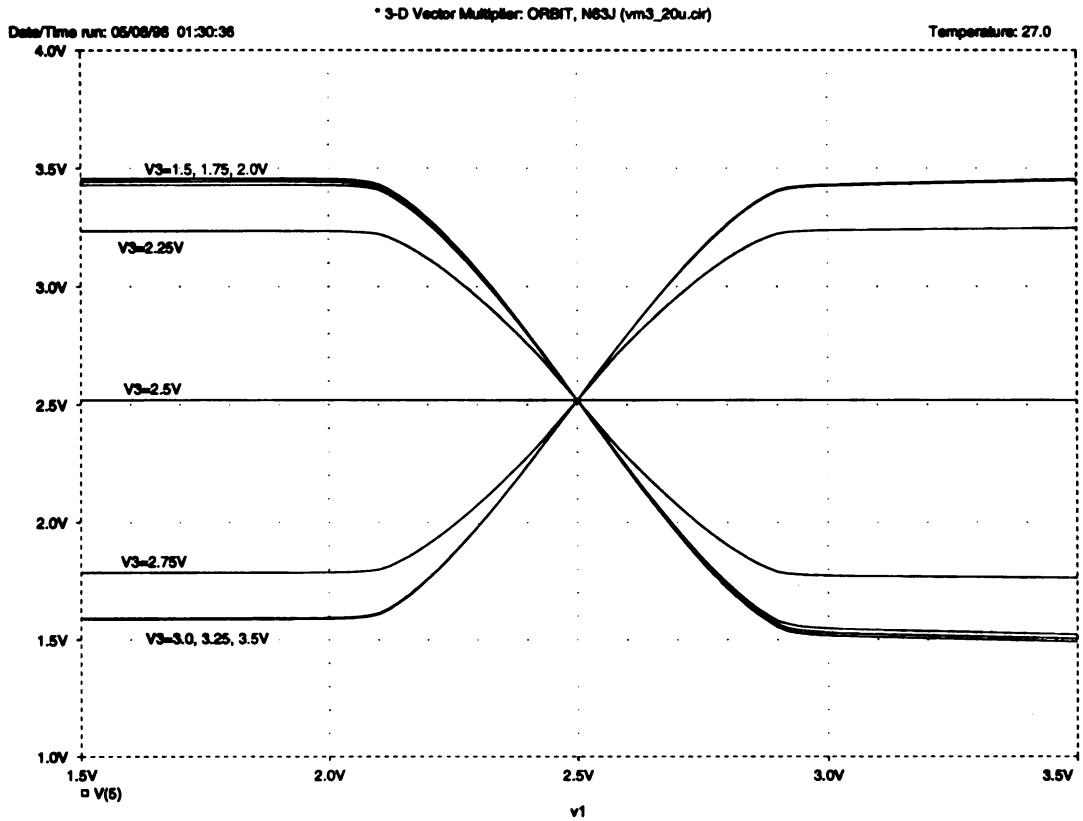

Figure 3.20. The PSPICE circuit simulation result: the 3-dimensional vector multiplier

resistors.

3.5 Implementation of the modified update law

With a view towards a circuit realization, we have proposed the circuit to implement the modified update law [13][28][47]. To approach the concept of implementing the differential equation, let us first review the circuit shown in Figure 3.21.

In this figure, there is a current source I_{IN} from the modified Gilbert multiplier and two PMOS transistors as active resistors. A capacitor is attached to the V_{OUT} node to generate the dynamic behavior of the circuit. We assume that M1 and M2 are a matched pair (i.e., they have the same conductance β), work in the saturation region, and have the same threshold voltage. We have the circuit equation by applying

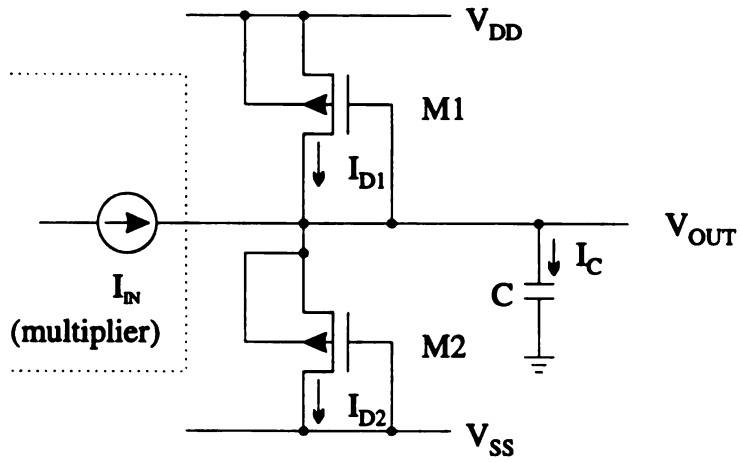

Figure 3.21. The implementation of the differential equation

Kirchhoff's current law at the V_{OUT} node,

$$I_{IN} + I_{D1} = I_{D2} + I_C \tag{3.36}$$

$$I_{D1} = \beta(V_{DD} - V_{OUT} - V_T)^2 \tag{3.37}$$

$$I_{D2} = \beta(V_{OUT} - V_{SS} - V_T)^2 \tag{3.38}$$

$$I_C = C\frac{dV_{OUT}}{dt} \tag{3.39}$$

After some algebraic manipulations, we have a solution for V_{OUT} as

$$\frac{dV_{OUT}}{dt} = \frac{1}{C}I_{IN} - \frac{2\beta}{C}(V_{DD} - V_{SS} - 2V_T)V_{OUT} + \frac{\beta}{C}(V_{DD} - V_{SS} - 2V_T)(V_{DD} + V_{SS}) \tag{3.40}$$

$$\frac{dV_{OUT}}{dt} = \frac{1}{C}I_{IN} - \frac{2G}{C}V_{OUT} + \frac{G}{C}(V_{DD} + V_{SS}) \tag{3.41}$$

where G is given as equation (3.11)

$$G = \beta(V_{DD} - V_{SS} - 2V_T)$$

The last term can be treated as an AC-ground term. If we rewrite the equation with the shifted variable $\overline{V_{OUT}}$,

$$\overline{V_{OUT}} = V_{OUT} - \frac{V_{DD} + V_{SS}}{2} \tag{3.42}$$

$$\frac{d\overline{V_{OUT}}}{dt} = \frac{1}{C} I_{IN} - \frac{2G}{C}\overline{V_{OUT}} \tag{3.43}$$

With a new variable $\overline{V_{OUT}}$, the reference voltage is shifted to 2.5V when V_{DD} is 5.0V and V_{SS} is 0V.

The equation (3.43) is qualitatively same as the following differential equation (3.44).

$$\frac{dw_{ij}}{dt} = \eta(d_i - y_i)x_j - \eta\alpha w_{ij} \tag{3.44}$$

Thus the learning parameter, η, is related with the capacitor value in the learning circuit and the α parameter is related with the conductance of the transistor.

CHAPTER 4

Learning Temporal Signals

The supervised learning algorithm will be extended to incorporate the use of temporal processing. The standard feedforward network makes its topology implausible in a neurobiological sense. On the other hand, by permitting arbitrary connections, including the use of feedback, the recurrent network assumes a more neurobiologically plausible topology [48].

In this chapter, we review several learning methods in literature for temporal signal. We consider a neural network consisting of the interconnection of N continuous-valued neurons, V_i, continuous in both time and amplitude, and with the synaptic weights from neuron j to neuron i denoted by w_{ij}. Neurons are assumed to have the nonlinear function, S, such as *tanh*. Unlike the standard feedforward network, the network will be permitted to have *feedback* connections among the neurons; that is, the network is recurrent.

Three subsets of neurons are identified in the neural networks:

1. *Input neurons*: they receive stimuli directly from the externally applied pattern.

2. *Output neurons*: they supply the overall response learned by the network.

3. *Hidden neurons*: which are neither input nor output neurons.

Of particular interest is the ability of the recurrent network to deal with time-varying input or output through its own temporal operation.

4.1 Time-dependent recurrent back-propagation

Pearlmutter [50] has developed an algorithm for training a general continuous-time recurrent network. The units evolve according to

$$\tau \frac{dy_i}{dt} = -y_i + S(\sum_j w_{ij} y_j) + I_i \tag{4.1}$$

$$= -y_i + S(net_i) + I_i \tag{4.2}$$

Some of the units are designed as input units and have input values I_i. Similarly, some units are used as output units with desired training values d_i.

The objective is to minimize the difference between the outputs of the neural networks and the desired outputs:

$$E = \frac{1}{2} \int_{t_0}^{t_1} \sum_{k \in O} [y_k - d_k]^2 dt \tag{4.3}$$

where the sum is only over the units that are output units.

We must minimize equation (4.3), subject constraints equation (4.1). For this purpose, we introduce the functional:

$$J = \int_{t_0}^{t_1} L dt \tag{4.4}$$

$$= \int_{t_0}^{t_1} \frac{1}{2} \sum_{k \in O} [y_k - d_k]^2 + \sum_i \lambda_i(t) \left[\tau \frac{dy_i}{dt} + y_i - S(\sum_i w_{ij} y_j) - I_i \right] dt \tag{4.5}$$

where the constraints equation (4.1) is multiplied by Lagrange multipliers, $\lambda_i(t)$.

The variation of the integral[51][52] leads to the well-known Euler-Lagrange equa-

tions for the functions $y_i(t), \lambda_i(t)$ depending on time. For $y_i(t)$, we have

$$\frac{\partial L}{\partial y_i} - \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{y}_i}\right) = 0 \tag{4.6}$$

and

$$\frac{\partial L}{\partial \lambda_i} = 0 \tag{4.7}$$

with the end-point conditions:

$$\frac{\partial L}{\partial \dot{y}_i}(t = t_1) = 0 \tag{4.8}$$

Notice that the equation (4.7) is in fact equivalent to equation (4.1).

The equation (4.6) can be expressed as

$$\frac{\partial L}{\partial y_i} = \delta_k(y_k - d_k) + \lambda_i - \sum_j \lambda_j S'_j w_{ji} \tag{4.9}$$

$$\frac{\partial L}{\partial \dot{y}_i} = \tau \lambda_i \tag{4.10}$$

where δ_k is 1 if $k \in O$ and 0 if otherwise. S'_j is the derivative of $S(net_j)$ in terms of net_j. Finally, we obtain

$$\tau \dot{\lambda}_i = \delta_k(y_k - d_k) + \lambda_i - \sum_j \lambda_j S'_j w_{ji} \tag{4.11}$$

The boundary conditions are given by

$$\frac{\partial L}{\partial \dot{y}_i}(t = t_1) = 0 = \lambda_i(t = t_1) \tag{4.12}$$

As for the back-propagation algorithm, we adapt the weights by gradient descent

method such as

$$\Delta w_{ij} = -\eta \frac{\partial J}{\partial w_{ij}} = \eta \int_{t_0}^{t_1} y_j S_i' \lambda_i dt \tag{4.13}$$

where η is a small positive constant.

Equations (4.1), (4.11), and (4.13) specify the learning rule for the recurrent network. Equation (4.11) is a backward integration in time and has a final value in equation (4.12). The learning algorithm proceeds as follows:

1. Starting from $t = t_0$ to $t = t_1$, integrate the activation equation (4.1) in forward time with initial condition. Record every state, y_i, and the derivative of S_i, $S'(net_i)$.

2. Make input signal halt. With the final value of λ, in equation (4.12), start integrating the equation (4.11) in backward time. Record every co-state, λ_i.

3. With the data which are obtained in step 1 and step 2, update the corresponding weight change using equation (4.13), and repeat the computation.

This algorithm is successfully used by Pearlmutter to learn temporal trajectories [50]. He has trained a recurrent neural network with no inputs, four hidden units, and two output units to follow the circular trajectory. In addition to the circular trajectory, he has trained a network with ten hidden units to follow the figure eight trajectory. Since this algorithm requires analog memories and backward integration, it has limitation for analog circuit implementation.

4.2 Classification of temporal trajectories

The previous Pearlmutter's algorithm is used for the identification of temporal trajectories. For the classification of temporal trajectories, Sotelino et al. [53] has modified the Pearlmutter's algorithm. In Sotelino's algorithm, the decision is taken immediately at the end of the input signal. The output units play the same role as the

hidden units before the end of the input signal, and the task of the network is easier: the activation of the output units just has to pass through a point; they do not have to follow a trajectory.

As in the Pearlmutter's algorithm, the activation rule for V_i is given by

$$\tau \frac{dV_i}{dt} = -V_i + S(\sum_i w_{ij} V_j) + I_i \tag{4.14}$$

We have to minimize the functional

$$J = \frac{1}{2} \sum_{k \in O} [V_k - D_k]^2 |_{t=t_1} + \int_{t_0}^{t_1} L dt \tag{4.15}$$

where L is the Lagrange function

$$L = \sum_i \lambda_i(t) \left[\tau \frac{dV_i}{dt} + V_i - S(\sum_i w_{ij} V_j) - I_i \right] dt \tag{4.16}$$

The output units do not have to follow a desired time trajectory any more. They have to classify the signal by clamping output units on at time t_1. The end-point term at time t_1 is exactly the same as the one used in standard back-propagation algorithm for the output layer. The difference is that the dynamic is continuous.

By the Euler-Lagrange equations,

$$\tau \dot{\lambda_i} = \lambda_i - \sum_j \lambda_j S_j' w_{ji} \tag{4.17}$$

The end-point conditions at $t = t_1$ are now

$$0 = \frac{\partial L}{\partial \dot{V_i}} + \frac{\partial}{\partial V_i} \left[\frac{1}{2} \sum_{j \in O} (V_j - D_j)^2 \right]_{t=t_1} \tag{4.18}$$

and we obtain

$$\lambda_i(t = t_1) = -\delta_j(V_j(t = t_1) - D_j) \tag{4.19}$$

The gradient descent rule is obtained as same as the Pearlmutter's algorithm.

$$\Delta w_{ij} = -\eta \frac{\partial J}{\partial w_{ij}} = \eta \int_{t_0}^{t_1} V_j S_i' \lambda_i dt \tag{4.20}$$

The procedure of updating the weight is the same way in the previous section. The fundamental difference with Pearlmutter's algorithm is the end-point condition, equation (4.19). The final value of the Lagrange parameters is no longer zero, but the difference to the target value at this time. This can be considered as the error signal that has to be back propagated in time.

4.3 Recurrent Back-propagation

Pineda [54] has developed the back-propagation algorithm that can be extended to arbitrary networks. Let us consider N continuous-valued units, V_i, with connection weight, w_{ij}, and activation function, S. Some of these units are used as input neurons and have input values of I_i. We define $I_i = 0$ if the neuron i is not the input neuron. Similarly some may be output neurons with desired values of D_i.

The dynamics of the network is based on the following differential equations

$$\tau \frac{dV_i}{dt} = -V_i + S(\sum_j w_{ij} V_j) + I_i \tag{4.21}$$

$$= -V_i + S(net_i) + I_i \tag{4.22}$$

where w_{ij} is a connection weight from jth neuron to the ith neuron. It is easily seen

that this dynamic rule leads to the right fixed point, where $dV_i/dt = 0$, given by

$$V_i = S(\sum_j w_{ij} V_j) + I_i \tag{4.23}$$

We assume that at least one such fixed point exists and is a stable attractor.

An error measure for the fixed point is the quadratic one

$$E = \frac{1}{2} \sum_k E_k^2 \tag{4.24}$$

where

$$E_k = \begin{cases} D_k - V_k & if\ K\ is\ an\ output\ unit; \\ 0 & otherwise \end{cases} \tag{4.25}$$

Gradient descent method gives

$$\Delta w_{rs} = -\eta \frac{\partial E}{\partial w_{rs}} = \eta \sum_k E_k \frac{\partial V_k}{\partial w_{rs}} \tag{4.26}$$

On performing the differentiation in equation (4.23), one immediately obtains

$$\frac{\partial V_i}{\partial w_{rs}} = S'(net_i) \left[\delta_{ir} V_s + \sum_j w_{ij} \frac{\partial V_j}{\partial w_{rs}} \right] \tag{4.27}$$

where δ_{ir} is the Kronecker δ symbol. With solving for the derivatives, the result is

$$\sum_j L_{ij} \frac{\partial V_j}{\partial w_{rs}} = \delta_{ir} S'(net_i) V_s \tag{4.28}$$

where

$$L_{ij} = \delta_{ij} - S'(net_i) w_{ij} \tag{4.29}$$

Inverting the linear equations (4.28) gives

$$\frac{\partial V_k}{\partial w_{rs}} = (L^{-1})_{kr} S'(net_r) V_s \tag{4.30}$$

On substituting equation (4.30) into (4.26) one immediately obtains

$$\Delta w_{rs} = \eta \delta_r V_s \tag{4.31}$$

where

$$\delta_r = S'(net_r) \sum_k E_k (L^{-1})_{kr} \tag{4.32}$$

Equations (4.31) and (4.32) specify a formal learning rule. Equation (4.32) requires a matrix inversion to calculate the error signal δ_r. Direct matrix inversion is not suitable for implementation of the neural networks. A local method for calculation of δ_r is obtained by the introduction of an associated dynamic system. Consider the vector z whose components are defined in terms of the components of δ according to

$$\delta_r = S'(net_r) z_r \tag{4.33}$$

so that

$$z_r = \sum_k E_k (L^{-1})_{kr} \tag{4.34}$$

Equations (4.32) and (4.33) imply that z_r satisfies

$$\sum_r L_{ri} z_r = E_i \tag{4.35}$$

Now observe that the solutions of equation (4.35) are the steady-state solution of

$$\tau \frac{dz_i}{dt} = -\sum_r L_{ri} z_r + E_i \tag{4.36}$$

Using equation (4.29),

$$\tau \frac{dz_i}{dt} = -z_i + \sum_r S'(net_r) w_{ri} z_r + E_i \tag{4.37}$$

Equations (4.31) and (4.33) lead to a learning rule of the form

$$\Delta w_{rs} = \eta S'(net_r) z_r V_s \tag{4.38}$$

Equations (4.22), (4.37), and (4.38) completely specify the dynamics for an adaptive neural network, provided that equations (4.22) and (4.37) are convergent.

The whole procedure is

1. Use the activation equation (4.22) to find $V_i's$.
2. Compare with the targets to find the $E_i's$ from equation (4.25).
3. Relax the network equation (4.37) to find $z_i's$.
4. Update the weights using equation (4.38).

4.4 Real-time recurrent learning

Williams and Zipser [55] showed how to construct a learning rule for general recurrent networks that runs continuously. The network so trained is called a *real-time recurrent network*. The network operates at discrete time and the rule can be run *on-line*, learning while sequences are being presented, rather than the whole sequences are shown.

Consider a network consisting of a total of N neurons and M input connections. Let $x(n)$ denotes the $M \times 1$ input vector at discrete time n, and let $y(n+1)$ be the corresponding output vector produced one step later at time $n+1$. The input

vector $x(n)$ and one-step delayed output vector $y(n)$ are concatenated to form the $(M+N) \times 1$ vector $z(n)$. Let A denote the set of indices i for which $x_i(n)$ is an external input, and let B denote the set of indices i for which $z_i(n)$ is the output of a neuron. We have

$$z_i(n) = \begin{cases} x_i(n) & if\ i \in A \\ y_i(n) & if\ i \in B \end{cases} \tag{4.39}$$

Let W be the $N \times (M+N)$ weight matrix of the network. The net internal activation of neuron i is given by

$$net_i(n) = \sum_j w_{ij} z_j(n) \tag{4.40}$$

$$y_i(n+1) = S(net_i(n)) \tag{4.41}$$

Let $d_i(n)$ denotes the desired (target) response of neuron i at time step n. Let C denote the set of neurons that chosen to be visible neurons externally. The remaining neurons are hidden neurons. An appropriate error measure $e(n)$ on neuron i at time step n is given by

$$e_i(n) = \begin{cases} d_i(n) - y_i(n) & if\ i \in C \\ 0 & \text{otherwise} \end{cases} \tag{4.42}$$

The instantaneous error measure at time step n is

$$E(n) = \frac{1}{2} \sum_{i \in C} e_i^2(n) \tag{4.43}$$

To minimize the error, we use the gradient descent method. For a particular weight $w_{pq}(n)$, we may define the incremental change $\Delta w_{pq}(n)$ at time step n as follows:

$$\Delta w_{pq}(n) = -\eta \frac{\partial E(n)}{\partial w_{pq}} \tag{4.44}$$

$$= \eta \sum_k e_k(n) \frac{\partial y_k(n)}{\partial w_{pq}} \tag{4.45}$$

where η is the learning-rate parameters. The last derivative in (4.45) can now be found by differentiating the activation equation (4.41),

$$\frac{\partial y_i(n)}{\partial w_{pq}} = S'(net_i(n-1)) \left[\delta_{ip} z_q(n-1) + \sum_j w_{ij} \frac{\partial y_j(n-1)}{\partial w_{pq}} \right] \tag{4.46}$$

where δ_{ip} denotes that $\delta_{ip} = 1$ when $i = p$, otherwise it is zero.

It is natural to assume that the initial state of the network at time step $n = 0$, say, has no functional dependence on the synaptic weights; this assumption implies that

$$\frac{\partial y_i(0)}{\partial w_{pq}} = 0 \tag{4.47}$$

We now may define a dynamic system by a triply indexed set of variables, π^i_{pq}, where

$$\pi^i_{pq} = \frac{\partial y_i(n)}{\partial w_{pq}} \tag{4.48}$$

For every step n and all appropriate i, p, q the dynamics of the system are governed by:

$$\pi^i_{pq}(n) = S'(net_i) \left[\delta_{ip} z_q(n-1) + \sum_j w_{ij} \pi^j_{pq} \right] \tag{4.49}$$

with initial conditions

$$\pi^i_{pq} = 0 \tag{4.50}$$

The real-time recurrent learning algorithm for training the recurrent neural network proceeds as follows:

1. For every time step n, starting from $n = 0$, use the activation equation (4.41) of the network to compute the output value of the N neurons. For the initial values of the weights, choose them from a set of uniformly distributed random numbers.

2. Use equation (4.49) and (4.50) to compute the variables $\pi^i_{pq}(n)$.

3. Use the value of $\pi^i_{pq}(n)$ obtained in step 2, and the error signal $e_i(n)$ expressed in equation (4.42), to compute the corresponding weight changes

$$\Delta w_{pq}(n) = \eta \sum_i e_i(n)\pi^i_{pq}(n) \tag{4.51}$$

4. Update the weight w_{pq} in accordance with

$$w_{pq}(n+1) = w_{pq} + \Delta w_{pq}(n) \tag{4.52}$$

and repeat the computation.

The use of the instantaneous gradient rather than the true gradient over the whole interval is analogous to that encountered in the standard back-propagation algorithm used to train a multilayer feedforward neural network, where weight changes are made after each pattern presentation. The practical differences between the real-time and non real-time versions are often slight. These two versions become more nearly identical as the learning rate η is sufficiently small [55]. The real-time version avoids any storage requirements and is especially simple to implement.

4.5 Hardware limitation of the real-time recurrent learning rule

At the Williams and Zipser's algorithm, the learning rule runs on-line. The on-line version of the Williams and Zipser's algorithm is appropriate for analog circuit implementation since analog hardware usually doesn't have memory capability. Though the learning equations of the real-time recurrent learning rule are simple and easy to implement with analog hardware, the Williams and Zipser's algorithm requires a large number of π equations.

Table 4.1. Hardware requirements of the real-time recurrent learning rule

Number of neurons, N	Number of inputs, M	Number of weights $W = N(M+N)$	Number of π equations $N^2M + N^3$
2	2	8	16
3	2	15	45
4	2	24	96
5	2	35	175
6	2	48	288
7	2	63	441
8	2	80	640
9	2	99	891
10	2	120	1200
2	3	10	20
3	3	18	54
4	3	28	112
5	3	40	200
6	3	54	324
7	3	70	490
8	3	88	704
9	3	108	972
10	3	130	1300
2	4	12	24
5	4	45	225
10	4	140	1400
20	4	480	9600
50	4	2700	135000
100	4	10400	1040000

Consider a recurrent neural network with N neurons and M inputs. Under these conditions, the dimension of the weight matrix becomes $N \times (M + N)$. The implementation of the real-time recurrent learning rule uses π equations (4.49) and weight update equations (4.45). The π equations take total $N \times (N \times (M + N))$ equations. Thus, the required number of π equations becomes $N^2M + N^3$.

In table 4.1, the required number of weight equations and π equations for different number of neurons and inputs are shown. Nevertheless, because of its ease of implementation, the real-time recurrent learning rule is used by many researchers working

with small networks.

CHAPTER 5

Implementation of the Modified Recurrent Back-propagation

5.1 The modified recurrent back-propagation rule

Let us consider N neuron recurrent neural network which is fully connected among neurons. Let y_i be the output of the neuron, w_{ij} be a connection weight from the jth neuron to the ith neuron, and x_i be the external inputs. Some of the neurons can be externally seen as output neurons, and some neurons receive the external inputs.

The dynamics of the network in the Pearlmutter's algorithm is expressed as equation (4.1)

$$\begin{aligned} \tau \frac{dy_i}{dt} &= -y_i + S(\sum_j w_{ij} y_j) + I_i \\ &= -y_i + S(net_i) + I_i \end{aligned}$$

We propose the equivalent discrete model to the recurrent neural network as

$$y_i(t + \Delta t) = S(net_i(t)) \tag{5.1}$$

$$= S(\sum_j w_{ij} y_j(t) + x_i(t)) \tag{5.2}$$

where S means the sigmoid function, and $x_i(t)$ represents the input value which may be zero if $y_i(t)$ is not used as an input neuron in the network. The activation equation (5.2) is analogous to the equations in the feedforward neural network.

In the Pearlmutter's algorithm, the error is defined in the integral form as equation (4.3)

$$E = \frac{1}{2} \int_{t_0}^{t_1} \sum_{k \in O} [y_k - d_k]^2 dt$$

The error measure of the proposed network is defined at instantaneous time t such as

$$E(t) = \frac{1}{2} \sum_k e_k^2(t) + \frac{1}{2} \sum_i \sum_j \alpha_{ij} w_{ij}^2 \tag{5.3}$$

$$= \frac{1}{2} \sum_k (d_k(t) - y_k(t))^2 + \frac{1}{2} \sum_i \sum_j \alpha_{ij} w_{ij}^2 \tag{5.4}$$

where k denotes the output neuron, $d_k(t)$ is the desired value of the output neuron k, and α_{ij} is a small positive constant, which is damping factor.

In the Pearlmutter's algorithm, the following two equations (4.11) and (4.13) specify the learning rule.

$$\tau \dot{\lambda}_i = \delta_k (y_k - d_k) + \lambda_i - \sum_j \lambda_j S_j' w_{ji} \tag{5.5}$$

$$\Delta w_{ij} = -\eta \frac{\partial J}{\partial w_{ij}} = \eta \int_{t_0}^{t_1} y_j S_i' \lambda_i dt \tag{5.6}$$

We propose the **modified recurrent back-propagation update law** with equa-

tions (5.7) and (5.8).

$$\tau_z \frac{dz_i(t)}{dt} = -z_i(t) + \sum_j w_{ji} z_j(t) + e_k(t) \tag{5.7}$$

$$\frac{dw_{ij}(t)}{dt} = \eta z_i(t) y_j(t) - \eta \alpha_{ij} w_{ij}(t) \tag{5.8}$$

In the Pearlmutter's algorithm, the forward integration from t_0 to t_1 is used in the weight update equation (5.6). If we consider that the weight update law changes the weights only at instantaneous time t, not at the interval from t_0 to t_1, the weight update equation becomes equation (5.8) with damping factor α. The modification from equation (5.6) to equation (5.8) is based on the derivation in Section 2.3.2. We use the continuous-time gradient descent method for modification and we remove the sigmoid derivative term. In the implementation of the modified recurrent back-propagation update law, we set the η parameter as small as possible and the integration is obtained by averaging of the weight value through time. Since we start the learning with arbitrary initial conditions and continue to learn the target with periodical waveform, the λ equation (5.5) is modified to the z equation (5.7). For analog hardware implementation, the backward integration is impossible. We employ only forward instantaneous update in the z equation.

Also, this updating rule is the special case of the Pineda's algorithm. The Pineda's algorithm assumes the activation equation is convergent and the error measure is calculated in the fixed point. The update rule at the modified recurrent back-propagation is the on-line version of the Pineda's algorithm. It updates the weights at instantaneous time t. The activation equation at neurons gives instantaneous response to the network at time t. Instantaneous error is measured at time t and used to update the weights, while the time-varying inputs and time-varying targets are changing with time.

Since we have showed the successful operations of the feedforward neural network with the modified algorithm that does not have the derivative of non-linearity term [13][28][47], the derivative of the non-linearity term is omitted. The omission of the derivative of the non-linearity term makes the analog hardware implementation simple.

5.2 Stability of the modified recurrent back-propagation

If we do not guarantee that the network will be stable, it may not perform the task we wish it to learn. We need to investigate the stability of the learning rule.

To consider the stability of the recurrent neural network with fixed point learning, we must consider the stability of the dynamic equation of the system and the z equation. It has been shown previously by Almeida [56] that if the dynamic equation of the system is stable, then the z equation is stable as well.

Other stability requirements for the dynamic equation of the neural network deal with constraints on the weight matrix. Hopfield [21] has guaranteed that if the weight matrix is symmetric, he could find a Lyapunov function and global asymptotic stability. We can not ensure that our weight matrix will be symmetric. Other type of constraints [57][58] deal with the size of the weights in the network relative to the maximum gain of the sigmoid function. However, they consider only static patterns. The key pattern is given as a constant input to the network. There is no report on the stability analysis of the recurrent neural network when the pattern is a dynamic (temporal) pattern.

In the modified recurrent back-propagation learning rule, we have two basic equa-

tions for the learning rule such as

$$\dot{w}_{ij}(t) = \eta z_i(t) y_j(t) - \eta \alpha_{ij} w_{ij}(t) \tag{5.9}$$

$$\dot{z}_i(t) = \sum_j w_{ji}(t) z_j(t) + e_k(t) - \beta z_i(t) \tag{5.10}$$

Note that we have inserted the β parameter to investigate the stability of the z equation. In equations (5.9) and (5.10), the damping factor α and β control the stability of the equations.

We have simulated the learning rule in MATLAB. The network is given as 4-neuron neural network with threshold weights. Thus there are total 20 weights, 16 connection weights and 4 threshold weights.

The experiment has two inputs and one target. In this experiment, two input values are given as

$$x_1(t) = sin(0.5 \times \pi \times t) \tag{5.11}$$

$$x_2(t) = sin(0.25 \times \pi \times t) \tag{5.12}$$

The target value is given as

$$d_1(t) = \frac{1}{2}(x_1(t) + x_2(t)) \tag{5.13}$$

With input and target values, the differential equations (5.9) and (5.10) are solved with the activation equation,

$$y_i(t + \Delta t) = tanh(\sum_j w_{ij}(t) y_j(t))$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = tanh \left(\begin{bmatrix} w_{10} & w_{11} & w_{12} & w_{13} & w_{14} \\ w_{20} & w_{21} & w_{22} & w_{23} & w_{24} \\ w_{30} & w_{31} & w_{32} & w_{33} & w_{34} \\ w_{40} & w_{41} & w_{42} & w_{43} & w_{44} \end{bmatrix} \begin{bmatrix} 1 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ x_1 \\ x_2 \end{bmatrix} \right) \tag{5.14}$$

where *tanh* is a hyperbolic-tangent function. There are 16 differential equations for the weight update equation,

$$\frac{dw_{ij}(t)}{dt} = \eta z_i(t) y_j(t) - \eta \alpha w_{ij}(t)$$

We assume all α_{ij} has the same quantity α.

$$\begin{bmatrix} \dot{w}_{10} & \dot{w}_{11} & \dot{w}_{12} & \dot{w}_{13} & \dot{w}_{14} \\ \dot{w}_{20} & \dot{w}_{21} & \dot{w}_{22} & \dot{w}_{23} & \dot{w}_{24} \\ \dot{w}_{30} & \dot{w}_{31} & \dot{w}_{32} & \dot{w}_{33} & \dot{w}_{34} \\ \dot{w}_{40} & \dot{w}_{41} & \dot{w}_{42} & \dot{w}_{43} & \dot{w}_{44} \end{bmatrix} = \eta \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} \begin{bmatrix} 1 & y_1 & y_2 & y_3 & y_4 \end{bmatrix}$$

$$-\eta\alpha \begin{bmatrix} w_{10} & w_{11} & w_{12} & w_{13} & w_{14} \\ w_{20} & w_{21} & w_{22} & w_{23} & w_{24} \\ w_{30} & w_{31} & w_{32} & w_{33} & w_{34} \\ w_{40} & w_{41} & w_{42} & w_{43} & w_{44} \end{bmatrix} \tag{5.15}$$

and 4 differential equations for the z equation,

$$\tau_z \frac{dz_i(t)}{dt} = \sum_j w_{ji} z_j(t) + e_k(t) - \beta z_i(t)$$

Table 5.1. The MATLAB simulation results

η(learning rate)	α(weight update)	β(z equation)	Results
0.50	0.050	0.20	infinity
0.50	0.050	0.30	infinity
0.50	0.050	0.35	infinity
0.50	0.050	0.40	converge
0.50	0.050	0.50	converge
0.05	0.050	1.00	converge
0.10	0.050	1.00	converge
0.30	0.050	1.00	converge
0.50	0.050	1.00	converge
0.70	0.050	1.00	converge
0.90	0.050	1.00	converge
1.10	0.050	1.00	converge
1.20	0.050	1.00	infinity
1.30	0.050	1.00	infinity
0.05	0.025	1.00	converge
0.10	0.025	1.00	converge
0.30	0.025	1.00	converge
0.50	0.025	1.00	converge
0.70	0.025	1.00	converge
0.80	0.025	1.00	infinity
0.90	0.025	1.00	infinity
0.10	0.010	1.00	infinity
0.20	0.010	1.00	infinity
0.30	0.010	1.00	infinity
0.40	0.010	1.00	infinity
0.50	0.010	1.00	infinity

$$\tau_z \begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \dot{z}_3 \\ \dot{z}_4 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} \\ w_{12} & w_{22} & w_{32} & w_{42} \\ w_{13} & w_{23} & w_{33} & w_{43} \\ w_{14} & w_{24} & w_{34} & w_{44} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} + \begin{bmatrix} e_1 \\ 0 \\ 0 \\ 0 \end{bmatrix} - \beta \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} \tag{5.16}$$

In MATLAB simulation, we have changed the parameters η, α, and β to investigate the stability. The simulation results are summarized in Table 5.1.

From the simulation results, the stable operation of the network is determined by the damping factors. If the α parameter and the β parameter are not big enough to

make the equations stable, the w values and the z values increase to infinity.

The first set of data in Table 5.1 shows the stability dependence on the β parameter. With the fixed value of $\eta = 0.5$ and $\alpha = 0.05$, the small β value drives the neural network into infinity values of weights and z values. The weights have converged to the constant values with ripples with $\beta = 0.40$, however, the weights have diverged to the infinity value when $\beta = 0.35$.

The rest of the results in the table are obtained with the fixed value of β since equation (5.8) implies the β parameter is 1. From the results in the table, we can see that the small value of α, such as $\alpha = 0.01$, makes the unstable operation in the neural network. Another observation on the α parameter suggests that a large α parameter is not desirable for successful learning. The large α parameter ensures the convergence of the weights. However, the large α parameter tends to drive the weights to near zero since the damping factor dominates the weight update equation.

The η parameter usually deals with the speed of the learning equation. The large value of η causes the divergence of the weights and the z values. Once the differential equations are stabilized, the η parameter determines the success or failure of the learning. If the learning rate (η) is not sufficient small, the ripple in the converged weights becomes significant since we are using the instantaneous learning in continuous time. To get the constant weight in learning phase, we need to have a very small η value. In the learning phase, we usually take the averaged value of the oscillating weights. If the weights are converged with large oscillation, we can not decide the constant weight values.

From Figure 5.1 to Figure 5.4, we display four simulation results in Table 5.1. In Figure 5.1 and Figure 5.2, the effect of the α parameter is shown. The large α parameter causes the large magnitude oscillation of the converged weight. In Figure 5.3, it is shown that the constant converged weight can be obtained with the small η parameter. Figure 5.4 shows the diverged weights to the infinity when η has a large

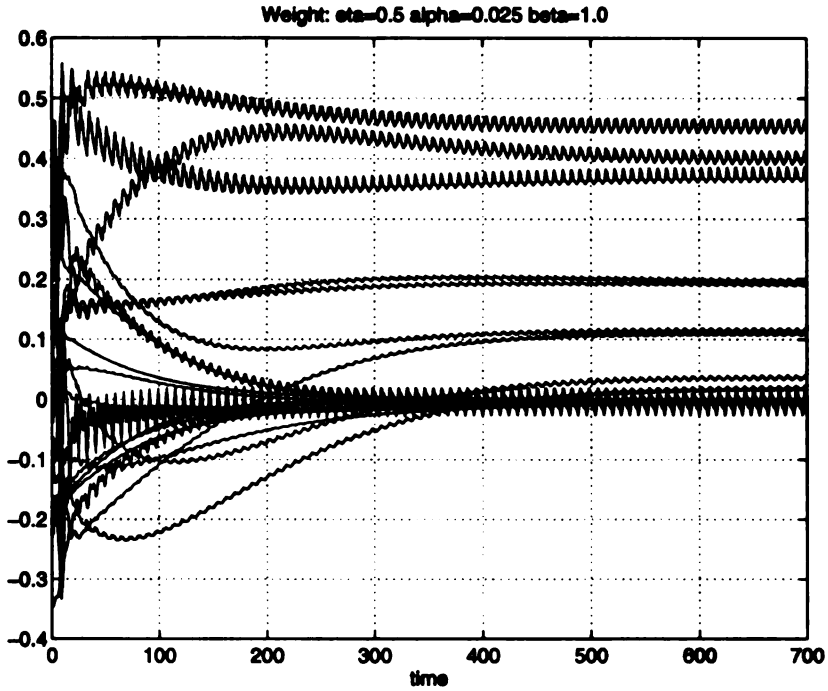

Figure 5.1. The MATLAB simulation result: the converged weights example

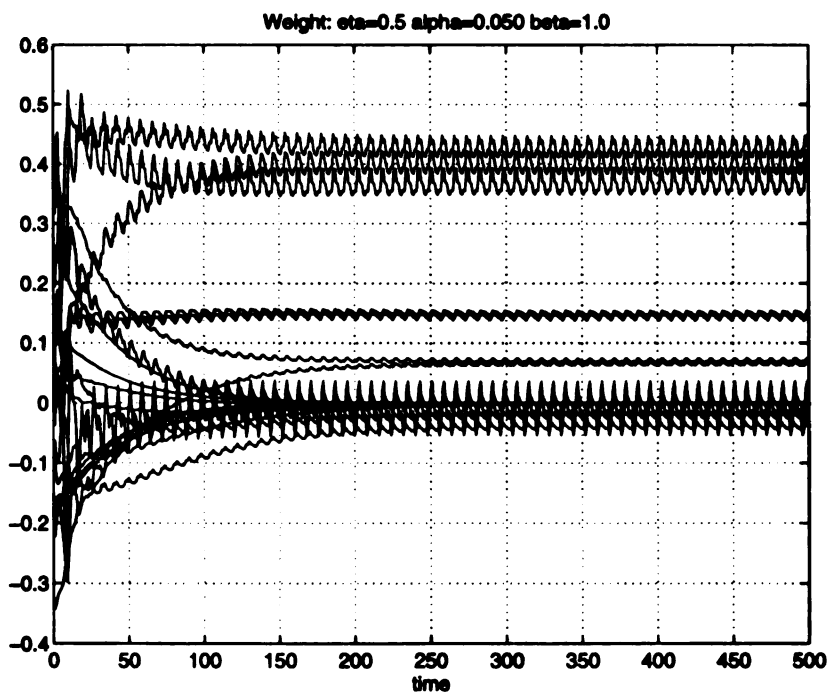

Figure 5.2. The MATLAB simulation result: the converged weights example

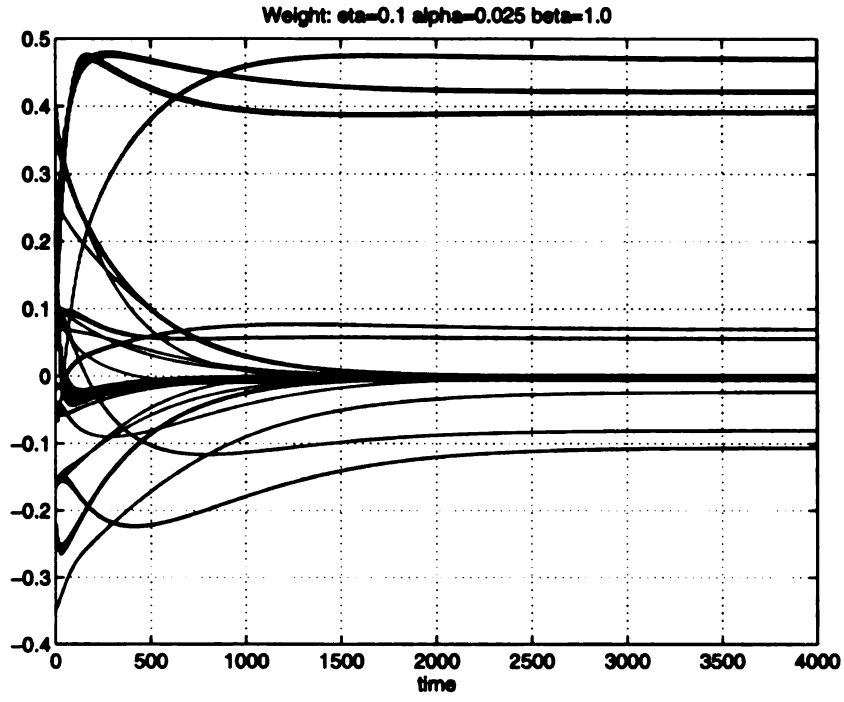

Figure 5.3. The MATLAB simulation result: the converged weights example

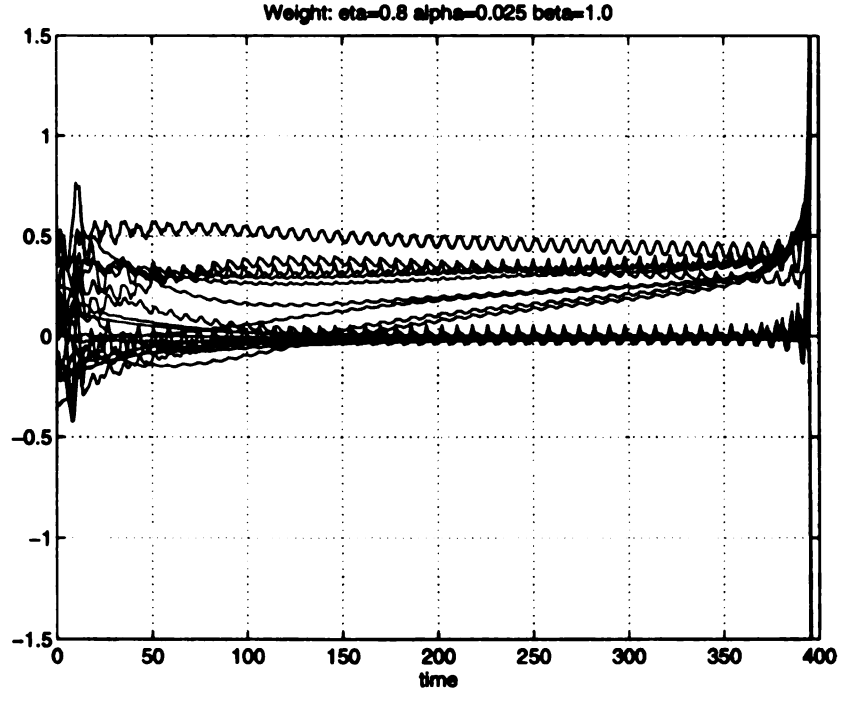

Figure 5.4. The MATLAB simulation result: the diverged weights example

value.

5.3 Implementation of the modified recurrent back-propagation

The recurrent neural networks with learning capability are designed using the modified recurrent back-propagation learning rule. This implemented recurrent network can be divided into two parts: a recurrent neural network and a learning network.

Suppose that the recurrent neural network has four neurons. It is fully connected including self-feedback. There are 16 connection weights and 4 threshold weights in this recurrent neural network. The recurrent neural network is shown in Figure 5.5.

Let us suppose that the recurrent neural network has two inputs, x_1 and x_2. These inputs are connected through the neuron, y_3 and y_4. Two neurons, y_1 and y_2, will be visible with the target signal, d_1 and d_2. w_{i0} denotes the threshold weight and w_{ij} is the connection weight from the jth neuron to the ith neuron.

The activation equation is given as

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = S\left(\begin{bmatrix} w_{10} & w_{11} & w_{12} & w_{13} & w_{14} \\ w_{20} & w_{21} & w_{22} & w_{23} & w_{24} \\ w_{30} & w_{31} & w_{32} & w_{33} & w_{34} \\ w_{40} & w_{41} & w_{42} & w_{43} & w_{44} \end{bmatrix} \begin{bmatrix} Vth \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} + Vlogic1 \begin{bmatrix} 0 \\ 0 \\ x_1 \\ x_2 \end{bmatrix} \right) \tag{5.17}$$

In circuit implementation, activation equations (5.17) are implemented using the 6-dimensional vector multiplier. There are 5 multiplications between the weight matrix and the input vector and a multiplication for input. If there is an input signal to the neuron, the activation equation is implemented using whole 6 modified Gilbert

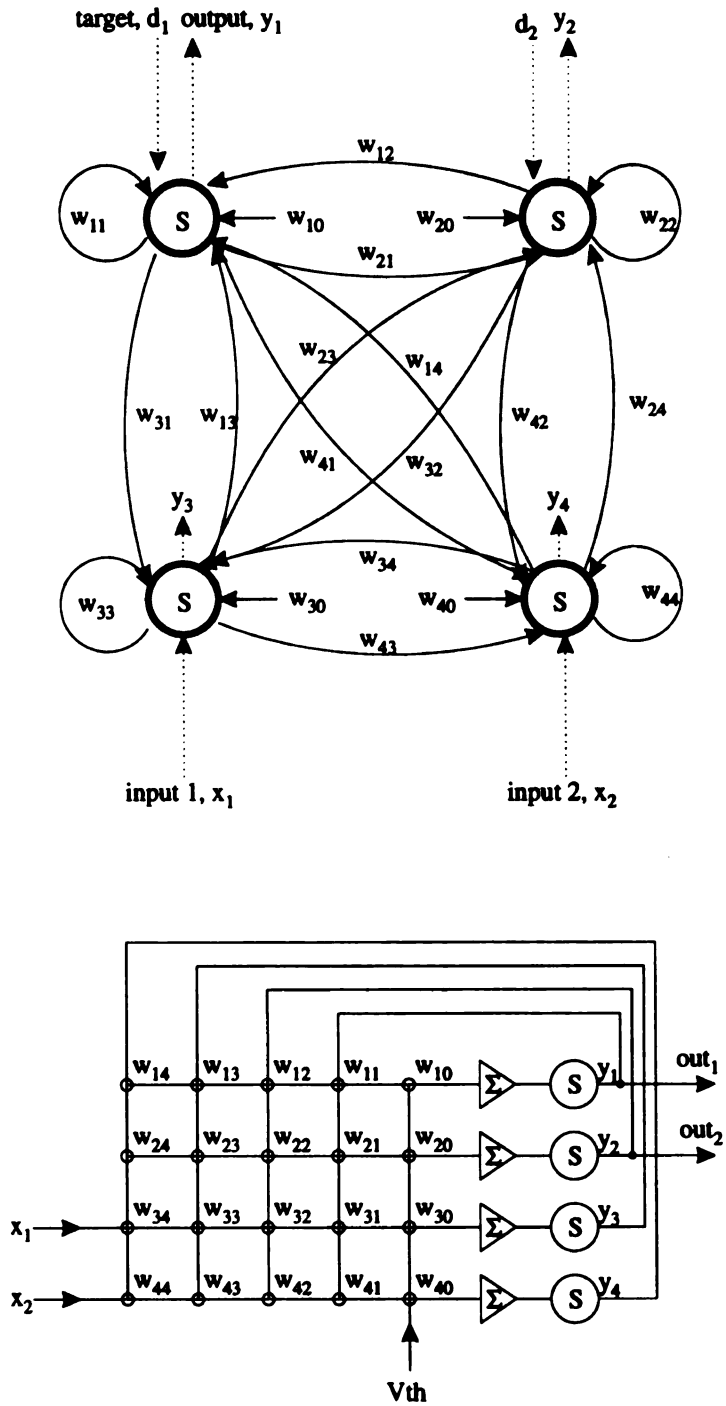

Figure 5.5. The recurrent neural network with four neurons and two inputs, S means the sigmoid function generator

multiplier sub-circuits. If there is no input to the neuron, the 6-dimensional multiplier uses only 5 modified Gilbert multiplier sub-circuits by connecting the input terminal to the reference voltage, 2.5V. If there is no + or - sign in the dotted pair, $V2$ and $V4$ are 2.5V. Vth denotes the threshold voltage which is chosen to be 3.5V and $Vlogic1$ has been chosen as the role of identity 1 in multiplication. Since the high saturated voltage of the sigmoid function generator is 3V, $Vlogic1$ is selected to be 3V.

The sigmoid function generator is designed to have the saturated voltage as 2V and 3V. The gain and the voltage characteristics are shown in Section 3.2.4. The bias voltage is selected as 1.1V.

The learning network is implemented using weight update equations (5.8) and z equations (5.7). There are 20 differential equations for the weight update equation,

$$\begin{bmatrix} \dot{w}_{10} & \dot{w}_{11} & \dot{w}_{12} & \dot{w}_{13} & \dot{w}_{14} \\ \dot{w}_{20} & \dot{w}_{21} & \dot{w}_{22} & \dot{w}_{23} & \dot{w}_{24} \\ \dot{w}_{30} & \dot{w}_{31} & \dot{w}_{32} & \dot{w}_{33} & \dot{w}_{34} \\ \dot{w}_{40} & \dot{w}_{41} & \dot{w}_{42} & \dot{w}_{43} & \dot{w}_{44} \end{bmatrix} = \eta \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} \begin{bmatrix} Vth & y_1 & y_2 & y_3 & y_4 \end{bmatrix} - \eta\alpha \begin{bmatrix} w_{10} & w_{11} & w_{12} & w_{13} & w_{14} \\ w_{20} & w_{21} & w_{22} & w_{23} & w_{24} \\ w_{30} & w_{31} & w_{32} & w_{33} & w_{34} \\ w_{40} & w_{41} & w_{42} & w_{43} & w_{44} \end{bmatrix} \quad (5.18)$$

Weight update equations (5.18) are implemented using 1-dimensional multiplier since the term $\eta\alpha$ is implemented using active resistors and capacitors. The learning rate, η is related with the capacitor value and the damping factor, α is related with the conductance of active resistors. Active resistors of the 1-dimensional multiplier uses cascaded PMOS as shown in 3.5 since the small α parameter is required for successful learning.

There are 4 differential equations for the z equation,

$$\tau_z \begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \\ \dot{z}_3 \\ \dot{z}_4 \end{bmatrix} = \begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} \\ w_{12} & w_{22} & w_{32} & w_{42} \\ w_{13} & w_{23} & w_{33} & w_{43} \\ w_{14} & w_{24} & w_{34} & w_{44} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} + Vlogic1 \begin{bmatrix} e_1 \\ e_2 \\ 0 \\ 0 \end{bmatrix} - Vbeta \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} \quad (5.19)$$

The differential equations for the z equation (5.19) are implemented using 6-dimensional vector multiplier. The 6-dimensional vector multiplier can be used as a 6-dimensional vector multiplier or a 5-dimensional vector multiplier depending on the existence of the target signal. If the neuron is not used as an output neuron, there is no target signal to receive. In the case of non-output neuron, the e_i is set to zero by connecting $V1 = V2$ together to prevent producing the multiplication since $V1$ is connected to the target signal and $V2$ is connected to the output signal.

We have tried to implement the damping factor (the last term, -z) in the z equation using the active resistors. We have found that the use of the active resistors can not guarantee the stable operation in the PSPICE circuit simulation. We realized the damping factor using the negative self-feedback. Using the negative self-feedback, we can control the stability of the neural network with the associated voltage to the negative self-feedback. The associated voltage to the negative self-voltage is labeled *Vbeta*. *Vbeta* is selected to 3V. If we reduce this voltage to 2.7V or 2.6V, then the circuit operation turns to be unstable. When *Vbeta* is 2.7V, the weight values and the z values usually hit the rail voltage (5V or GND). *Vlogic*1 has been chosen as the same role of identity 1 in multiplication. It is selected to be 3V.

In an analog circuit implementation, different time constants are used for equations (5.17), (5.18), and (5.19). The time y and z spend settling is negligible compared to the rate of weight change η. The learning rate (η) is designed to be slow compared to the speed of presentation of new training samples.

The circuit block diagram of the recurrent neural network with modified recurrent back-propagation learning circuit is shown in Figure 5.6.

In this diagram, 6D denotes 6-dimensional vector multiplier. S means the sigmoid function generator. y_i represents the state of neuron, z_i is the state of z equations, and w_{ij} is the connection weight from the jth neuron to the ith neuron. In the vector multiplier, the usual multiplication occurs in the form of $\sum_i (V1 - V2) \times (V3 - V4)$. The dotted lines inside the vector multiplier show the multiplication pair. If $V2$ and $V4$ are $Vref = 2.5V$, $(x_1 - 2.5V) \times (y_1 - 2.5V)$ becomes $x_1 \times y_1$ since 2.5V is the virtual ground term. $Vlogic1$ has been chosen as 3V and $Vref$ is the reference voltage which is 2.5V. Vth supplies the threshold voltage which is chosen to be 3.5V. $Vbeta$ in the implementation of z equation is selected to be 3V.

5.4 Simulation results of 4 neuron recurrent neural network

5.4.1 A circular trajectory generation

We have trained a recurrent neural network with no input units, two hidden neurons and two output neurons. The neural network has fully connected weights. It has 20 weights, 16 connection weights and 4 threshold weights. Since the neural network has no inputs, all input terminals are connected to the reference voltage, 2.5V.

It is trained to follow a circular trajectory. To learn the circular trajectory is the well-known problem [14][50]. The desired state d_1 and d_2 are plotted against each other in Figure 5.7. The target trajectory consists of a sine waveform and a cosine waveform. The trajectory is given as the continuous waveform to the neural network.

We have run the PSPICE transient analysis. We performed the PSPICE circuit simulation using the MOSIS 2.0μm CMOS parameters. The PSPICE parameters are

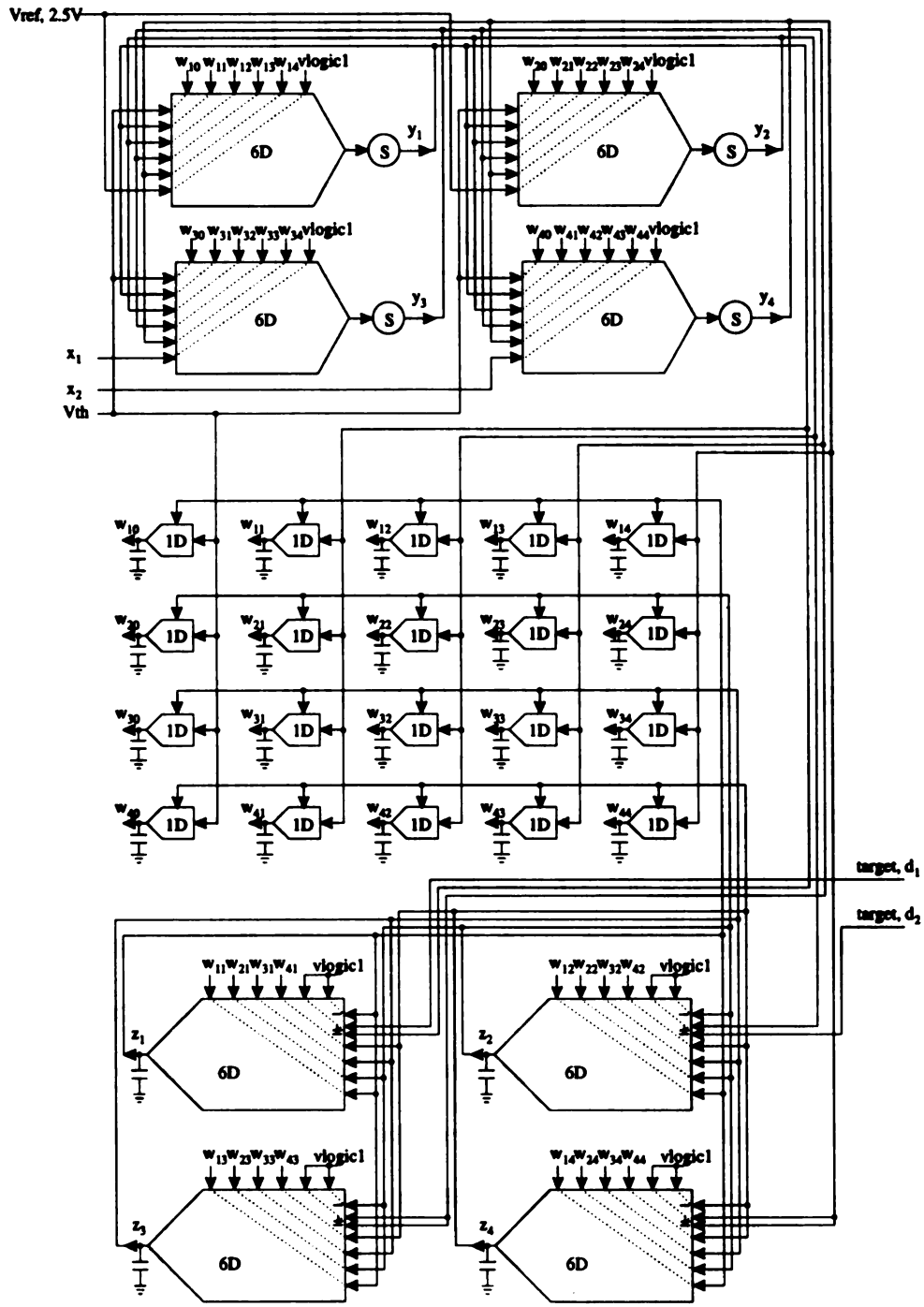

Figure 5.6. The block diagram of recurrent neural network with the modified recurrent back-propagation: four neurons and two inputs

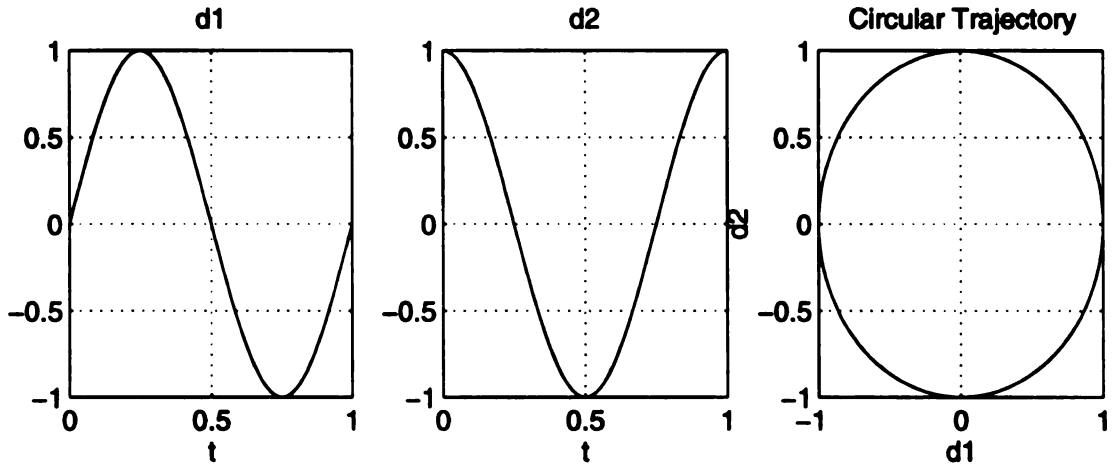

Figure 5.7. The circular trajectory

shown in Appendix A.1.

We apply the target signal to the neural network and observe the output waveform and the weight waveforms. If the output is following the target closely and weights have converged to the constant values, we take the averaged value of the weights. The obtained weight values are used as a verification of learning. After learning, we set the weight values to the neural network and see the generation of the trajectory by the neural network. If the neural network generates the circular trajectory, the learning is successful.

We found that the randomly initialized weight values didn't fail to learn the trajectory. We have performed several experiments with different frequencies of waveforms, different values of capacitance, and different gains of active resistors in the weight update equation. The results of the experiment are summarized in Table 5.2.

In Table 5.2, $(W/L)^{Active}_{Resistor}$ denotes that (W/L) ratios of 4 PMOS transistors in cascaded active resistors of 1-D multiplier for weigh update equation. The multiplier with (W/L)=4/64 has higher gain than that of (W/L)=4/16. From the table, high gain of active resistor will be desirable for successful learning. High gain of the active

Table 5.2. The simulation results of the circular trajectory experiment

$(W/L)^{Active}_{Resistor}$	*Target*	C_W	C_Z	*Result*	Remark
4/64	1MHz	20pF	1pF	fail	
4/64	1MHz	50pF	1pF	fail	
4/64	1MHz	100pF	1pF	fail	
4/64	500KHz	20pF	1pF	success	
4/64	500KHz	50pF	1pF	success	example 1
4/64	500KHz	100pF	1pF	success	
4/64	250KHz	20pF	1pF	small amplitude	
4/64	250KHz	50pF	1pF	success	
4/64	250KHz	100pF	1pF	success	
4/64	100KHz	20pF	1pF	fail	
4/64	100KHz	50pF	1pF	small amplitude	example 3
4/64	100KHz	100pF	1pF	success	
4/16	1MHz	20pF	1pF	fail	
4/16	1MHz	50pF	1pF	fail	
4/16	1MHz	100pF	1pF	fail	
4/16	500KHz	20pF	1pF	small amplitude	
4/16	500KHz	50pF	1pF	success	example 2
4/16	500KHz	100pF	1pF	success	
4/16	250KHz	20pF	1pF	fail	example 4
4/16	250KHz	50pF	1pF	small amplitude	
4/16	250KHz	100pF	1pF	success	
4/16	100KHz	20pF	1pF	fail	
4/16	100KHz	50pF	1pF	fail	
4/16	100KHz	100pF	1pF	small amplitude	

resistors implies the small value of the α parameter.

From the table, Cw means the capacitor value of the weight update equation. In the weight update equation, a larger value of the capacitor implies a small learning parameter η. When the learning parameter is not small enough, the weight has a ripple in its converged value. Small capacitance causes oscillatory behavior on the converged weight values and makes the determination of the final constant weight difficult. With the small capacitance, it is observed that the output waveforms usually follow the target waveforms. However, the weight is oscillating and not approaching the constant value when the capacitance is small. A large value of capacitance (small learning rate) is needed for near constant value of the weights.

When the target signal is very fast trajectory such as $1MHz$, the learning is not successful. The reason is that the circuit can not respond to the high frequency target. In this experiment, a target around $500KHz$ has the best results. With $250KHz$ target signal, the weight has a small oscillation in its converged value. Usually, we take the averaged value in the learning phase when the weights have oscillation. If the peak-to-peak value of the oscillated value is not small enough, the taken averaged value can not generate the exact trajectory. It results in a small amplitude trajectory generation. In Table 5.2, the *Result* has three cases in the testing phase such as *success*, *small amplitude*, and *fail*. The *success* case generates the circular trajectory closely, the *small amplitude* case generates the circle in small magnitude. The *fail* case can not generate the circular trajectory in the testing phase.

Both successful and failed cases of Table 5.2 are illustrated from Figure 5.8 to 5.23. We show four examples that are marked in Table 5.2. First two examples are successful cases, the third example is the case of *small amplitude*, and the last example is the failed case.

The obtained weights of each example are shown from Table 5.3 to 5.6. In these tables, w_{ij} denotes the connection weight from jth neuron to ith neuron and w_{i0}

denotes the threshold neuron at the neuron i.

Each example consists of four figures. In the first figure, three sub-plots are shown. V(7) is the first target signal, d_1, and V(8) is the second target signal, d_2. V(1) represents the actual output signal, y_1, and V(2) represents y_2. The second sub-plot shows d_1 and y_1, and the third sub-plot shows d_2 and y_2. The trajectory shape is illustrated in the first sub-plot in the figure. The transient trajectory is shown in V(1) versus V(2) plot (y_1 versus y_2 plot).

In the second figure, the error measure is shown in the first sub-plot. The error measure is given as

$$E_{rms} = \frac{1}{2}\sqrt{(d_1 - y_1)^2 + (d_2 - y_2)^2} \tag{5.20}$$

The value E_{rms} has the sense of a *root-mean-square normalized error.* The second and the third sub-plot show the weight values. The weights are labeled as $w_{ij} = V(ij)$ and $w_{i0} = V(6i)$. The final weight in the learning phase is obtained by averaging each weight at the end of simulation.

The third figure and the fourth figure show the results in the testing phase with the weights that are obtained in the learning phase. Two experiments with different initial conditions are shown in the third figure and in the fourth figure. The third figure shows the test result when the initial is inside the circle and the fourth figure shows the test result when the initial condition is given at the outside of the circle. The first sub-plot in each figure shows the state trajectory.

Except the failed case, all test results (two successful cases and a small amplitude case) show that the circular trajectory is a limit cycle. The neural network has learned the stable oscillation and generated the circular trajectory irrespective of the initial state.

Table 5.3. The averaged weights in the learning phase from the PSPICE transient analysis: example 1

j	1	2	3	4	0
w_{1j}	2.7844	2.4111	2.5234	2.5264	2.5276
w_{2j}	2.5706	2.7708	2.5674	2.5619	2.5262
w_{3j}	2.5279	2.5338	2.5253	2.5254	2.5339
w_{4j}	2.5285	2.5313	2.5248	2.5252	2.5338

Table 5.4. The averaged weights in the learning phase from the PSPICE transient analysis: example 2

j	1	2	3	4	0
w_{1j}	2.7751	2.4051	2.5147	2.5164	2.5286
w_{2j}	2.5901	2.7702	2.5427	2.5401	2.5309
w_{3j}	2.5188	2.5223	2.5152	2.5152	2.5225
w_{4j}	2.5190	2.5208	2.5150	2.5150	2.5226

Table 5.5. The averaged weights in the learning phase from the PSPICE transient analysis: example 3

j	1	2	3	4	0
w_{1j}	2.7452	2.4099	2.5296	2.5305	2.5215
w_{2j}	2.5772	2.7536	2.5614	2.5587	2.5276
w_{3j}	2.5292	2.5306	2.5250	2.5252	2.5344
w_{4j}	2.5294	2.5289	2.5248	2.5249	2.5342

Table 5.6. The averaged weights in the learning phase from the PSPICE transient analysis: example 4

j	1	2	3	4	0
w_{1j}	2.7452	2.4099	2.5296	2.5305	2.5215
w_{2j}	2.5772	2.7536	2.5614	2.5587	2.5276
w_{3j}	2.5292	2.5306	2.5250	2.5252	2.5344
w_{4j}	2.5294	2.5289	2.5248	2.5249	2.5342

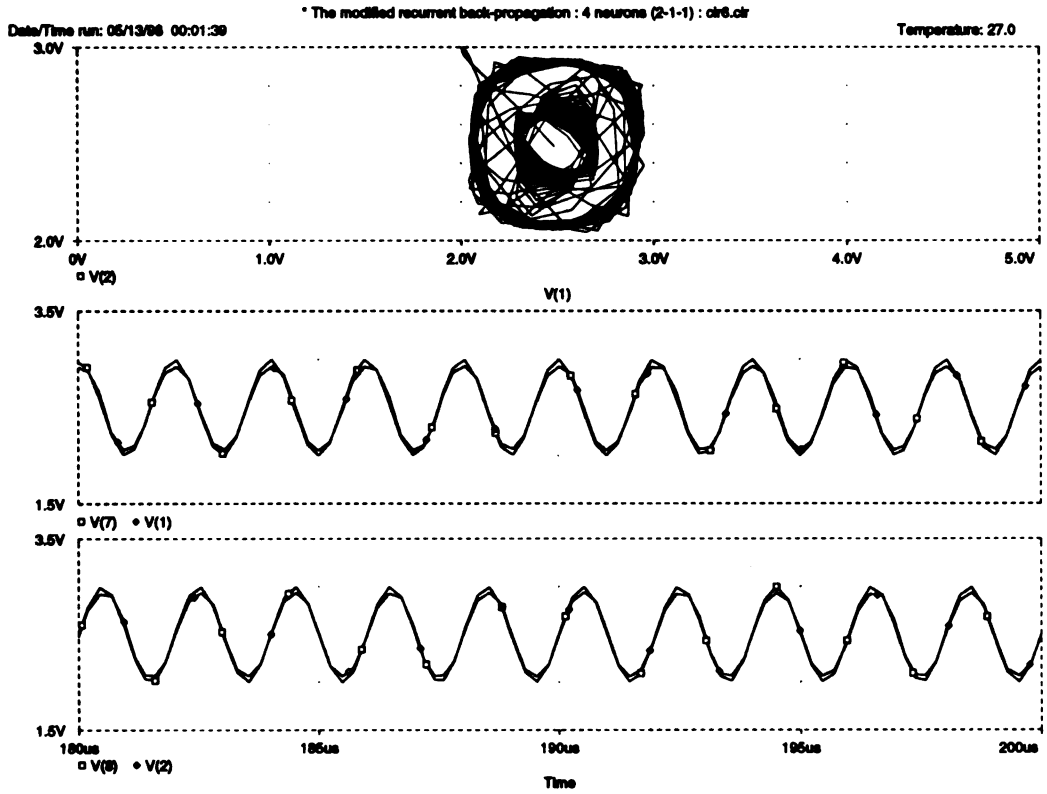

Figure 5.8. Example 1 (the first figure): Input and target signals

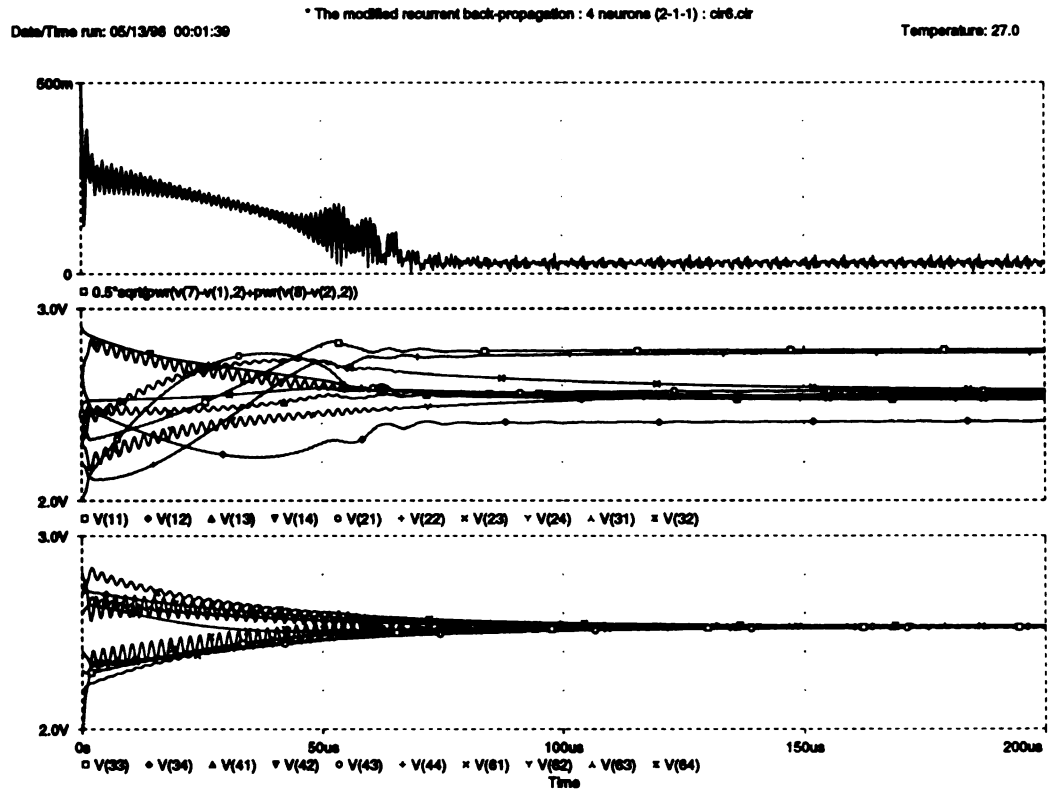

Figure 5.9. Example 1 (the second figure): E_{rms} and the weights

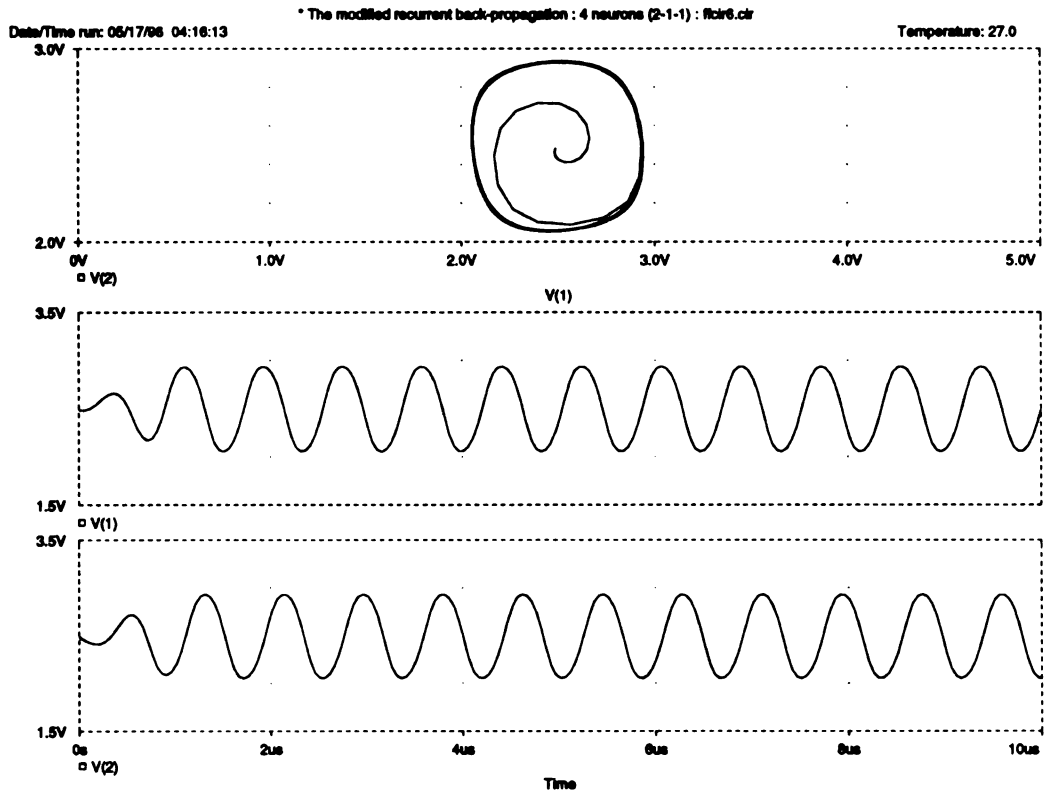

Figure 5.10. Example 1 (the third figure): Test phase with the initial condition inside the circle

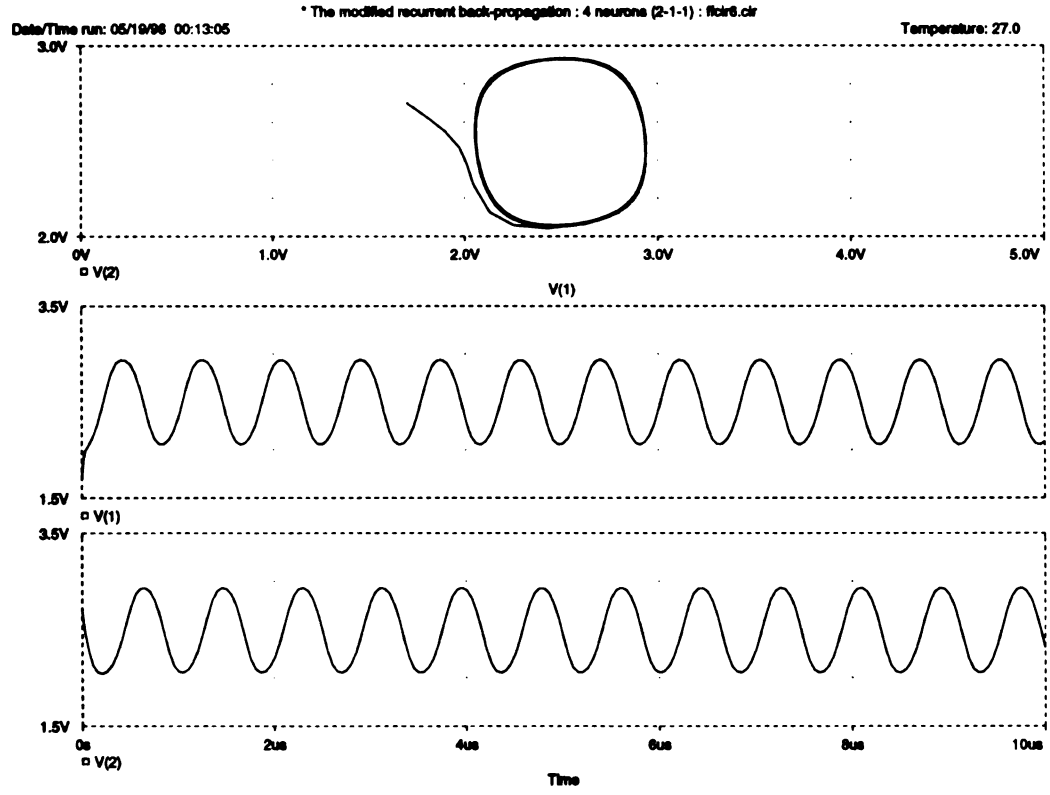

Figure 5.11. Example 1 (the fourth figure): Test phase with the initial condition outside the circle

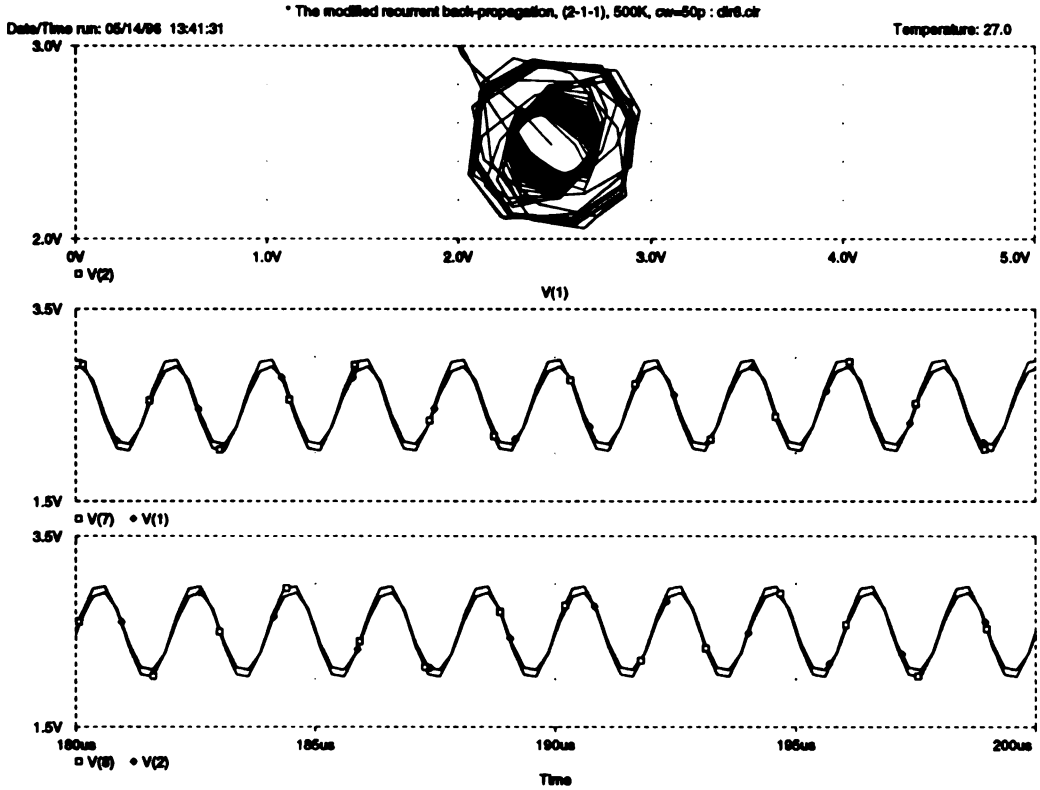

Figure 5.12. Example 2 (the first figure): Input and target signals

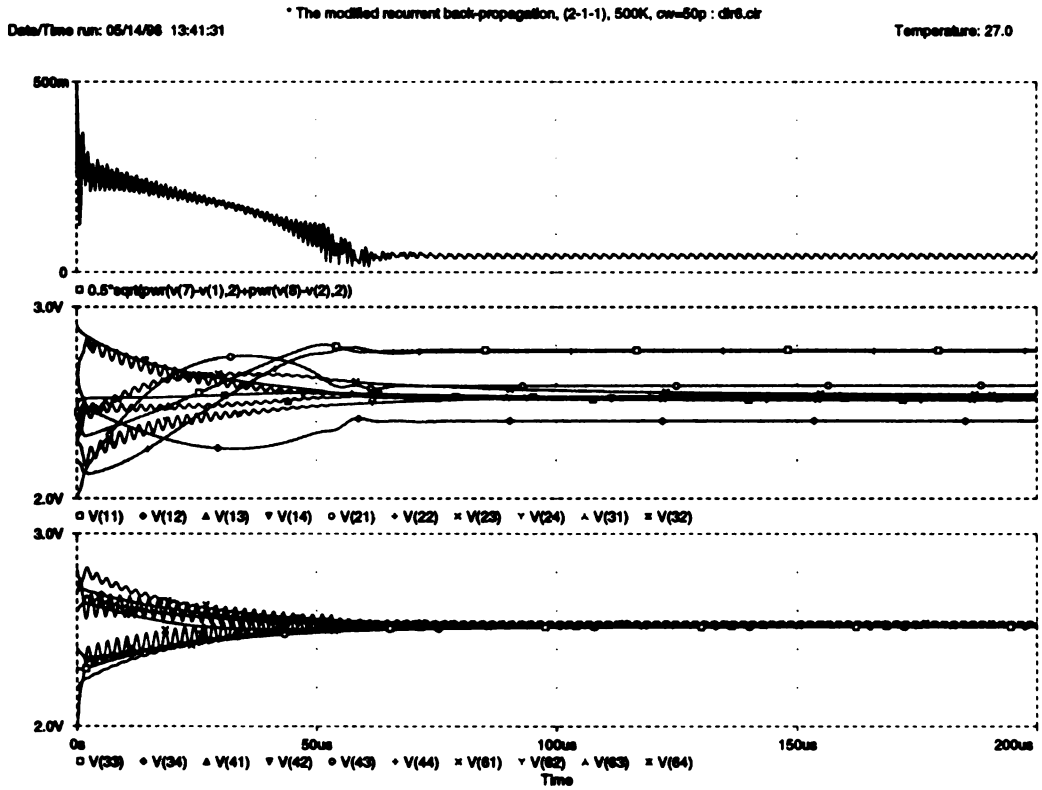

Figure 5.13. Example 2 (the second figure): E_{rms} and the weights

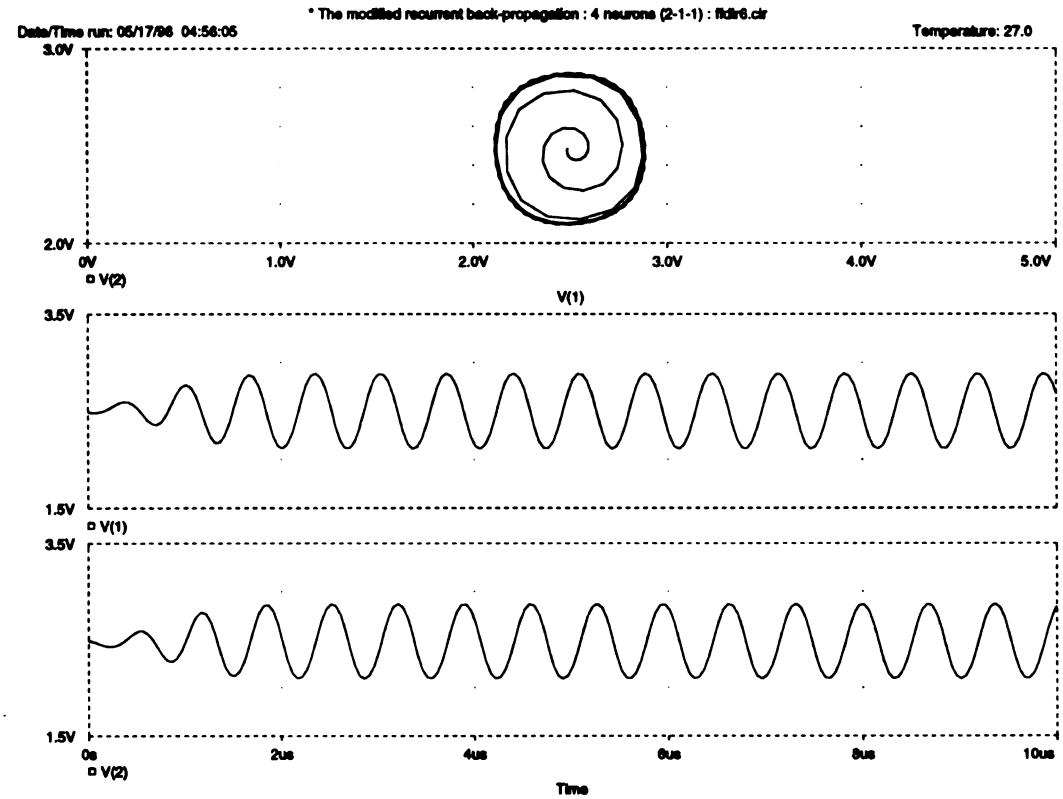

Figure 5.14. Example 2 (the third figure): Test phase with the initial condition inside the circle

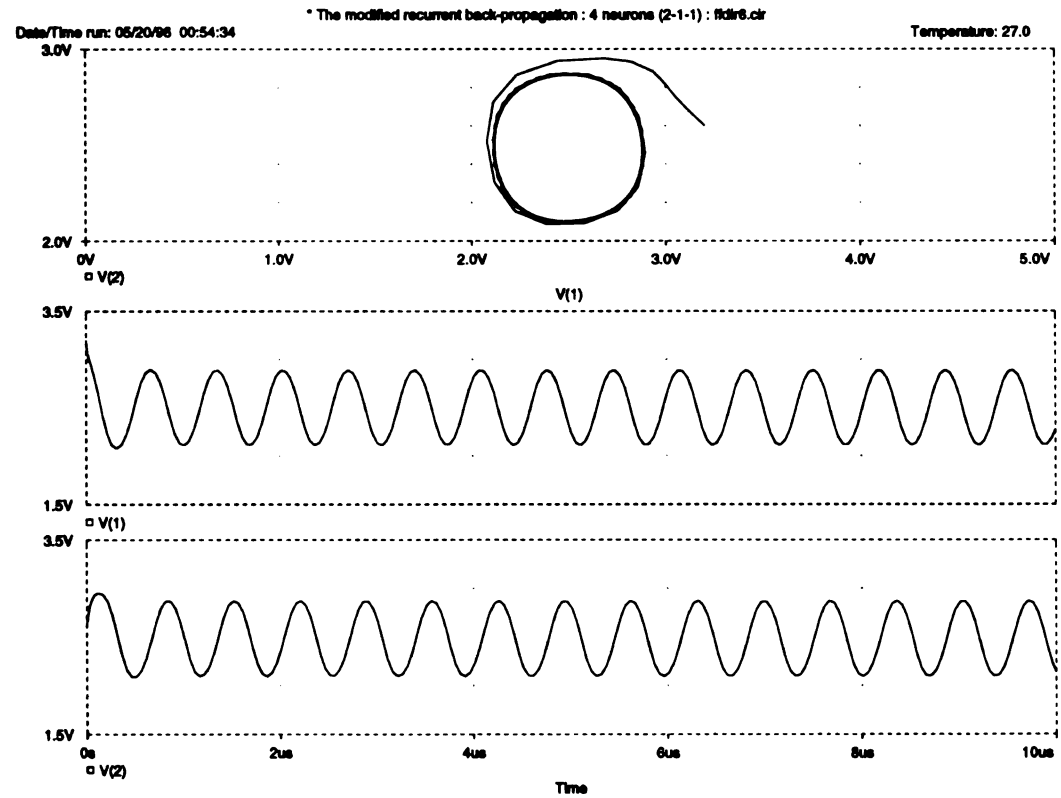

Figure 5.15. Example 2 (the fourth figure): Test phase with the initial condition outside the circle

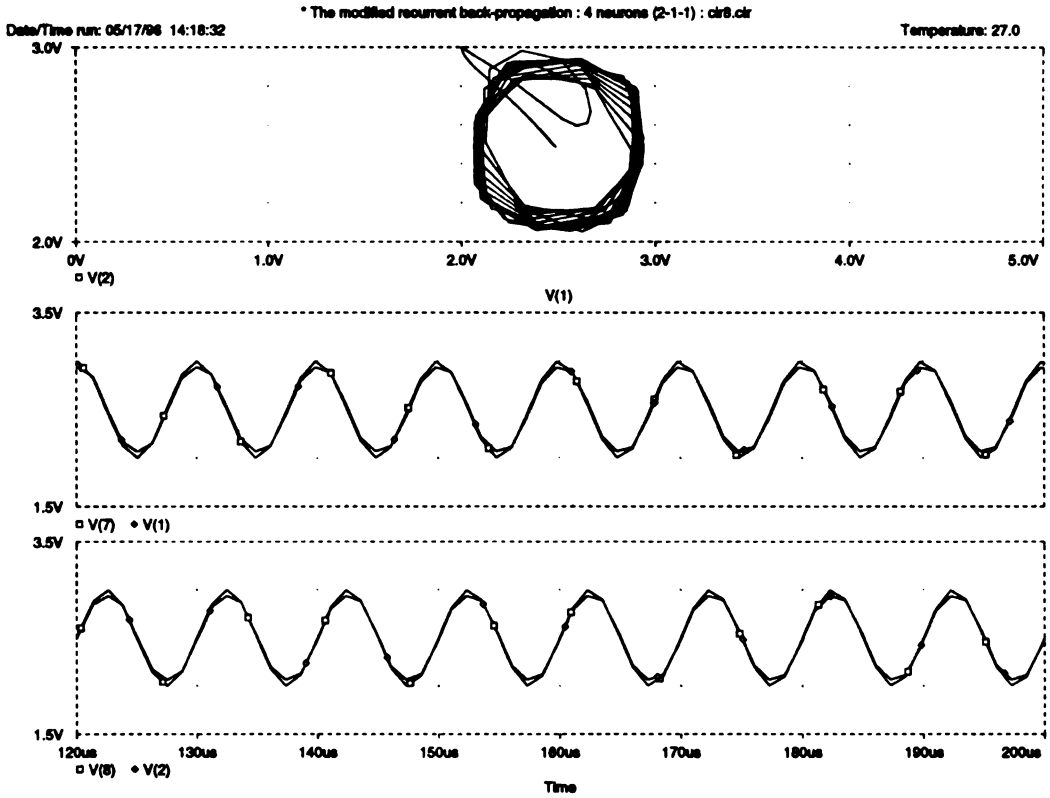

Figure 5.16. Example 3 (the first figure): Input and target signals

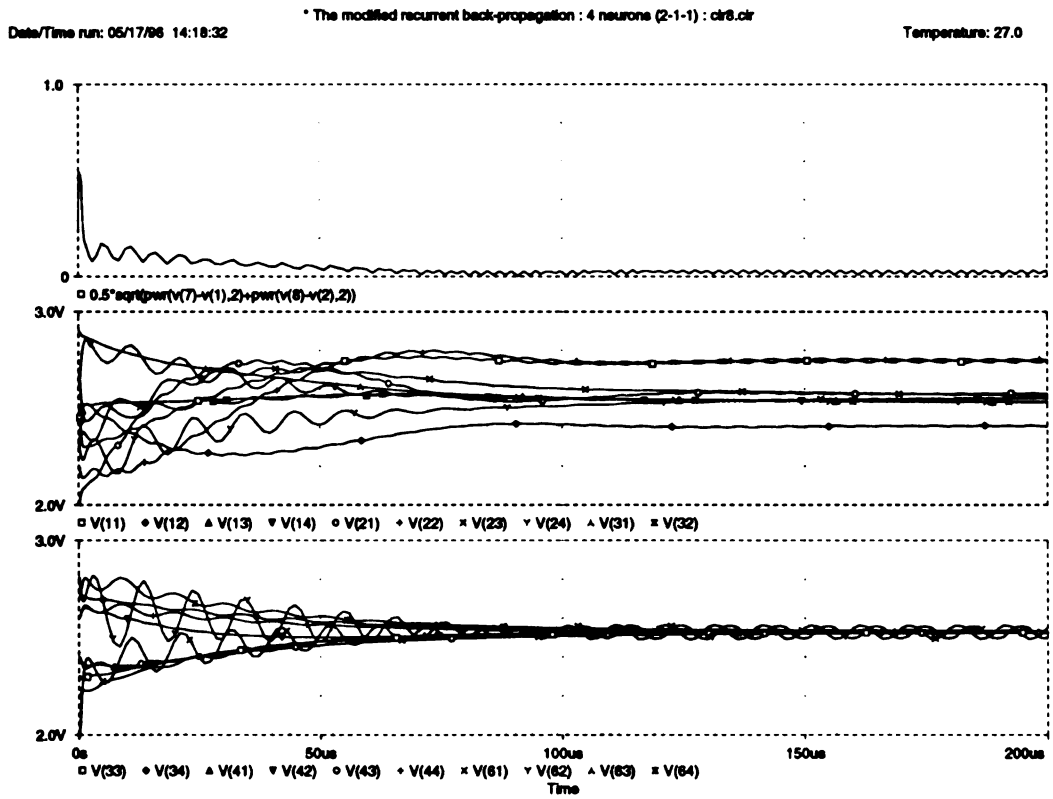

Figure 5.17. Example 3 (the second figure): E_{rms} and the weights

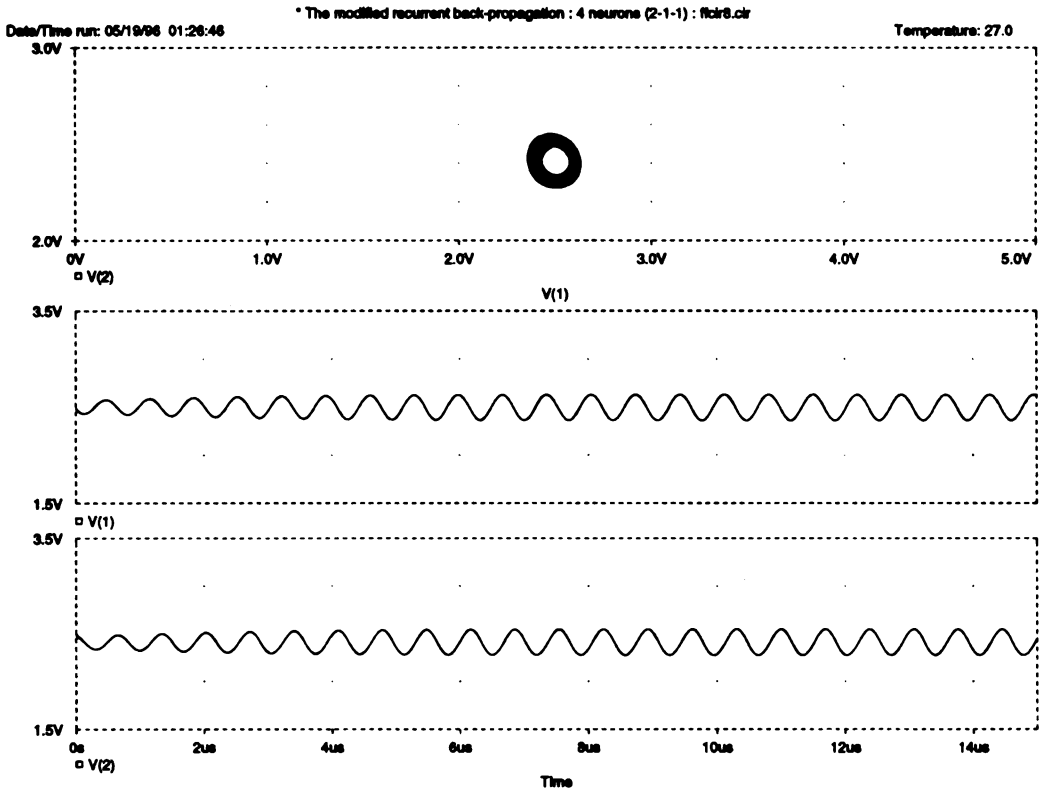

Figure 5.18. Example 3 (the third figure): Test phase with the initial condition inside the circle

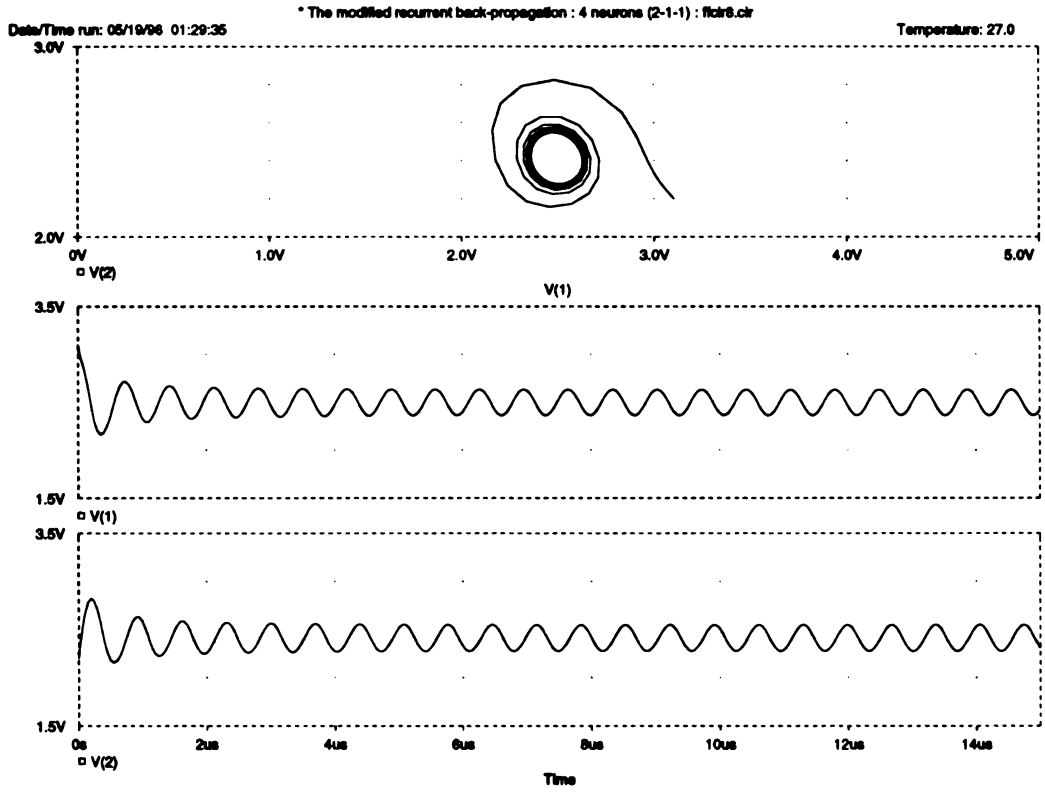

Figure 5.19. Example 3 (the fourth figure): Test phase with the initial condition outside the circle

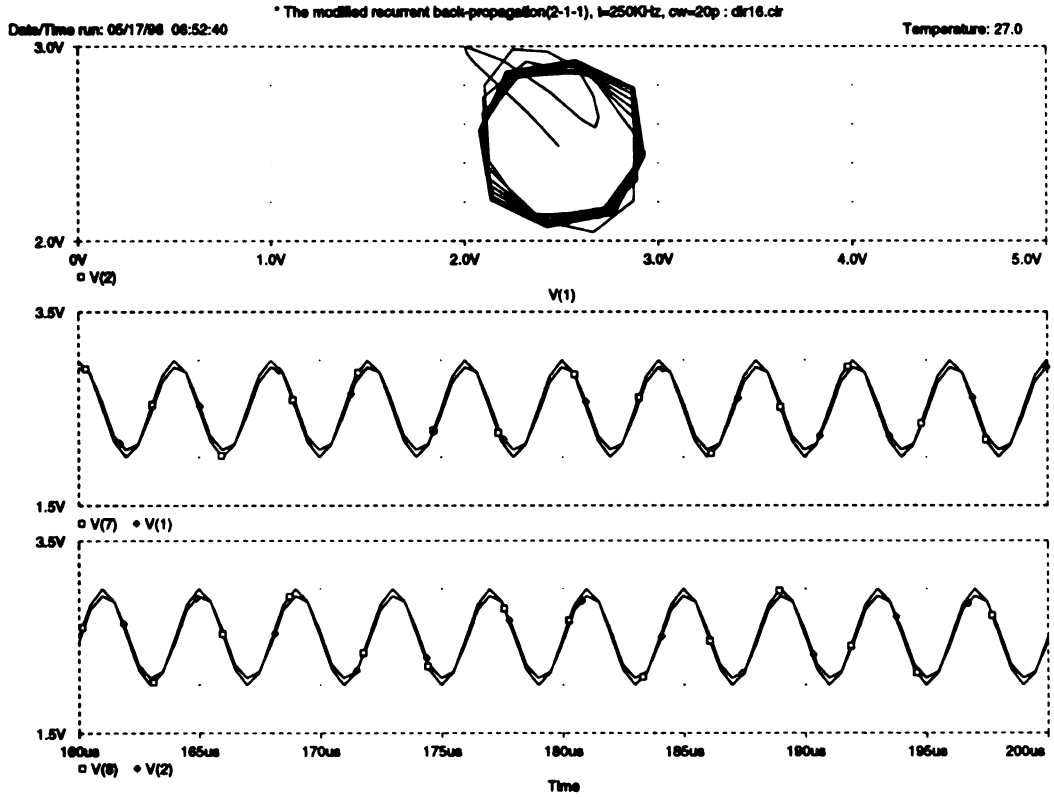

Figure 5.20. Example 4 (the first figure): Input and target signals

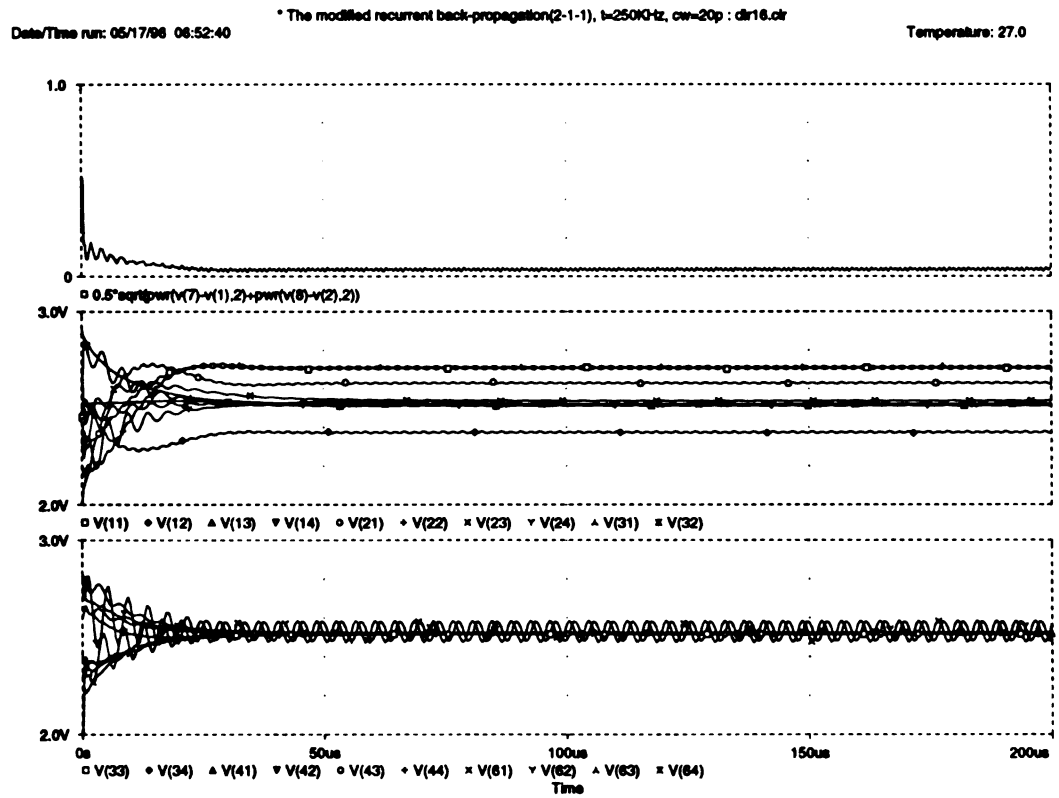

Figure 5.21. Example 4 (the second figure): E_{rms} and the weights

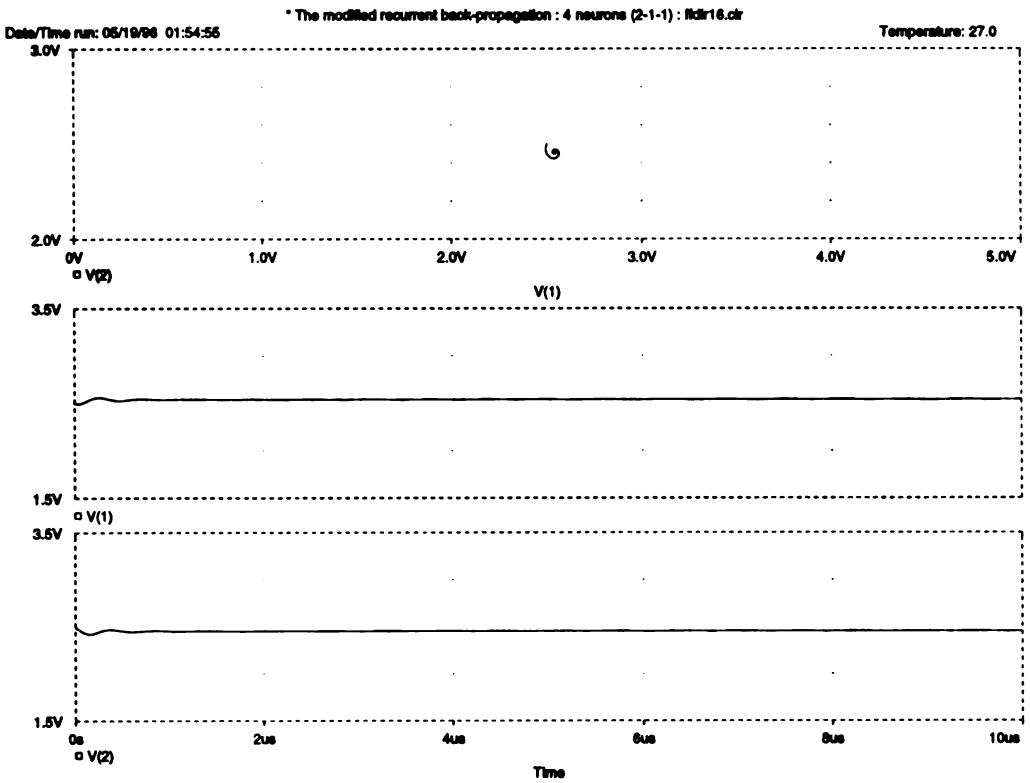

Figure 5.22. Example 4 (the third figure): Test phase with the initial condition inside the circle

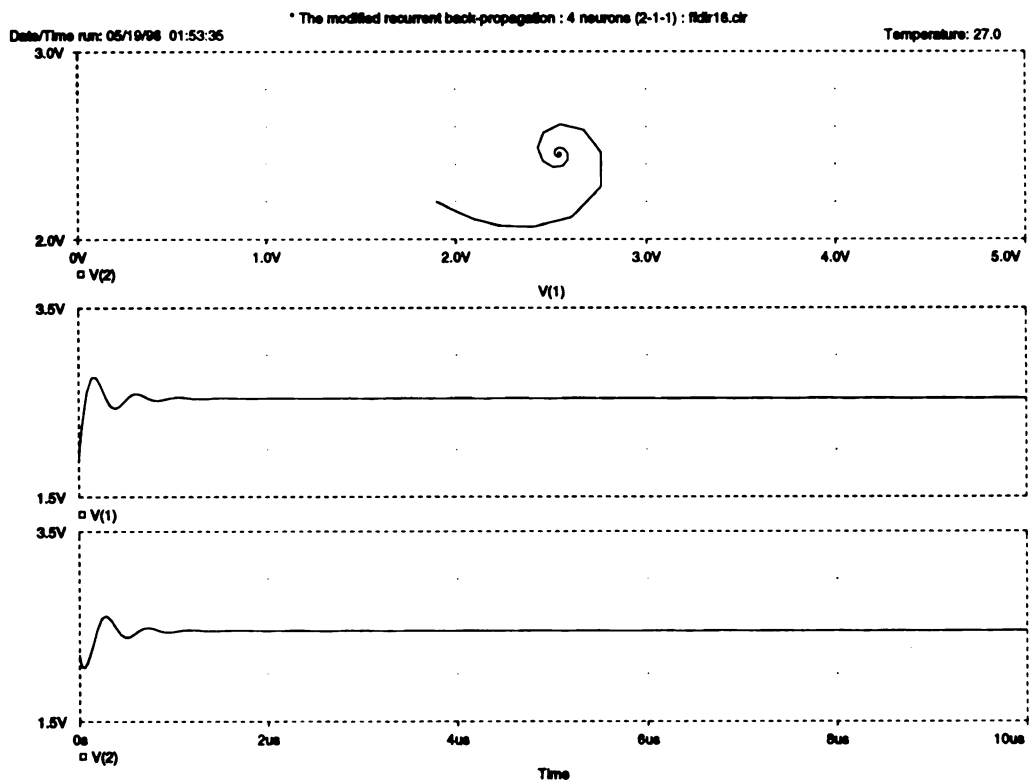

Figure 5.23. Example 4 (the fourth figure): Test phase with the initial condition outside the circle

5.4.2 Trajectory recognition

The 4-neuron recurrent neural network is trained to learn two trajectories. There are two inputs and one output in the 4-neuron recurrent neural network. The learning circuit employs the modified recurrent back-propagation learning rule. This circuit does not include the threshold parameters.

The general procedures for learning and testing are the follows:

1. In the learning phase, the first trajectory is applied to the inputs of the neural network and an associated target is also applied to the neural network. After one period of the first trajectory waveform, the second trajectory is applied to the recurrent neural network with its target waveform. We keep supplying the input-target pair to the neural network.

2. We can distinguish two trajectories by assigning the different states of the output waveform. the first trajectory is assigned to the high (or low) state of the target and the second trajectory is assigned to the low (or high) state of the target.

3. Each input-target pair is applied to the neural network as continuous waveforms and the PSPICE transient analysis is performed. After the transient analysis, we measure the output waveform and the target waveform. If the output waveform is following close to the target waveform, it is considered as successful learning.

4. We measure the weight values. If the learning parameter (η) is small, the weight values are converged to the near constant value. However, the weights have some ripples in their waveform since the capacitors on the learning circuit have limited value. We measure the averaged value of the weights.

5. After the learning phase, we perform the testing phase in the recurrent neural network. We set the weights of the neural network to the obtained weight values

at the learning phase. We verify the learning by applying the input waveforms to the recurrent neural network.

In Figure 5.24, two state space trajectories are shown. The trajectory 1 consists of $V1$ and $V2$ waveform. $V1$ is applied to the input 1 of the neural network and $V2$ is applied to the input 2. The trajectory 2 consists of $W1$ and $W2$ waveform. $W1$ and $W2$ are applied to the input 1 and input 2 of the recurrent neural network, respectively.

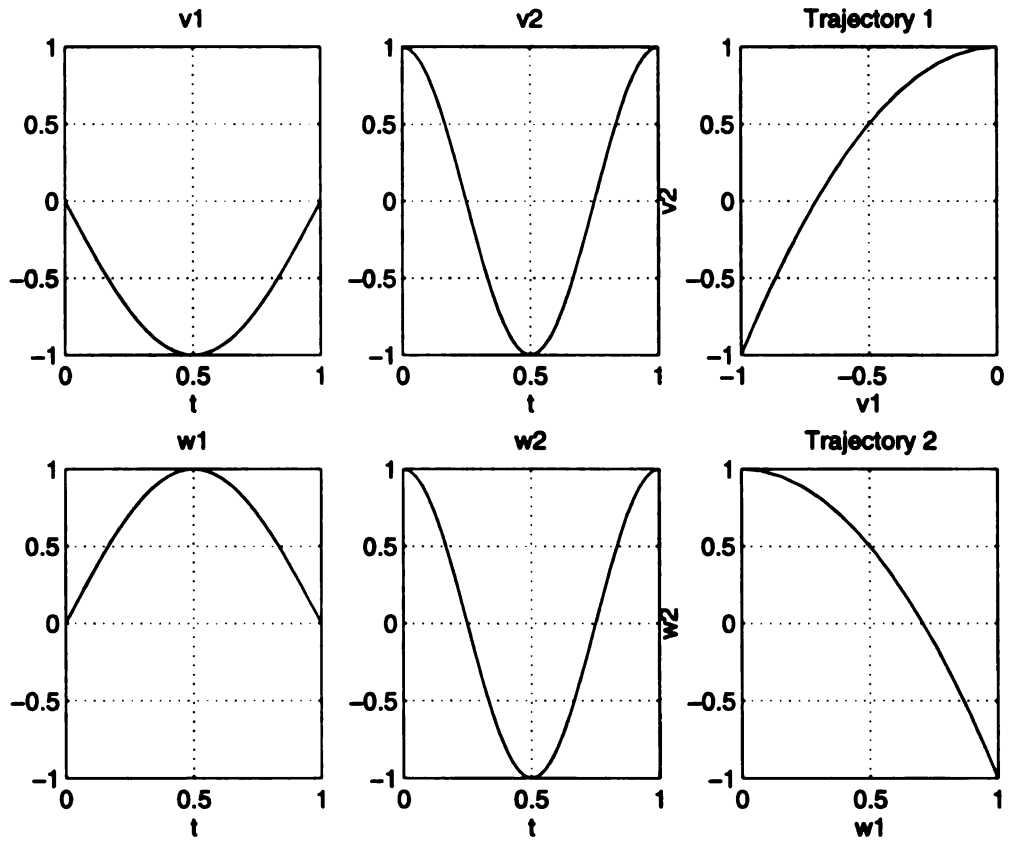

Figure 5.24. Two state trajectories

We have performed the PSPICE transient analysis with different parameters. The results of the transient analysis in PSPICE simulation are shown in Figure 5.25 and

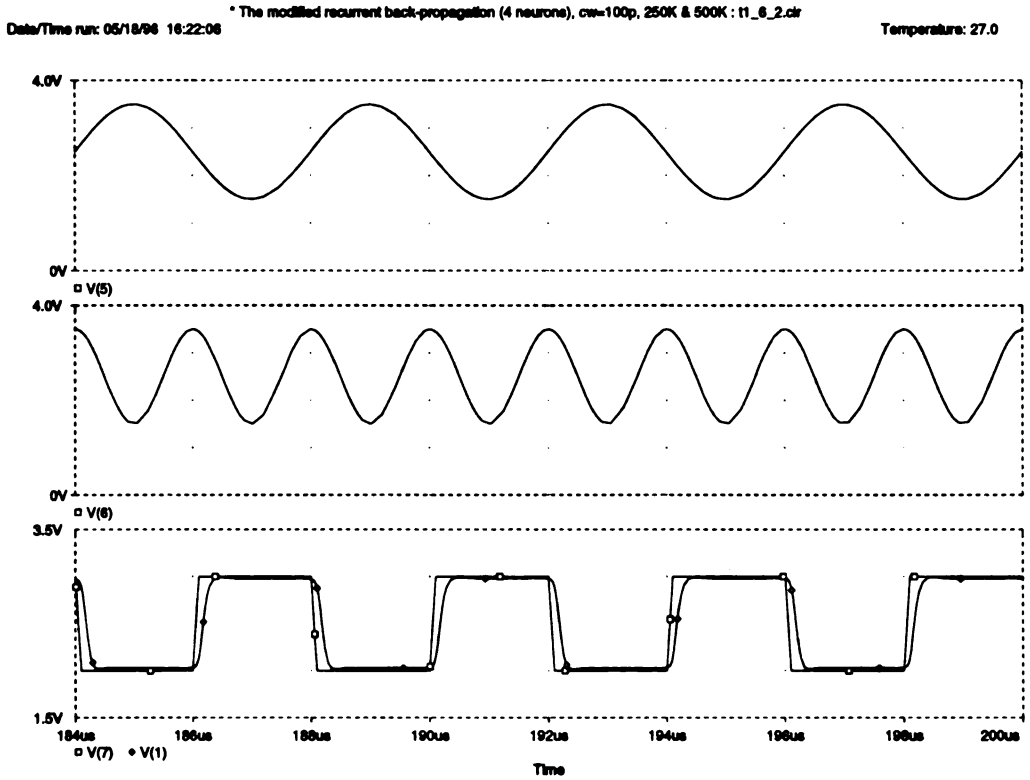

Figure 5.25. The PSPICE transient analysis: V(5), V(6), V(7), and V(1) are x_1, x_2, d_1, and y_1 of the recurrent neural network, respectively

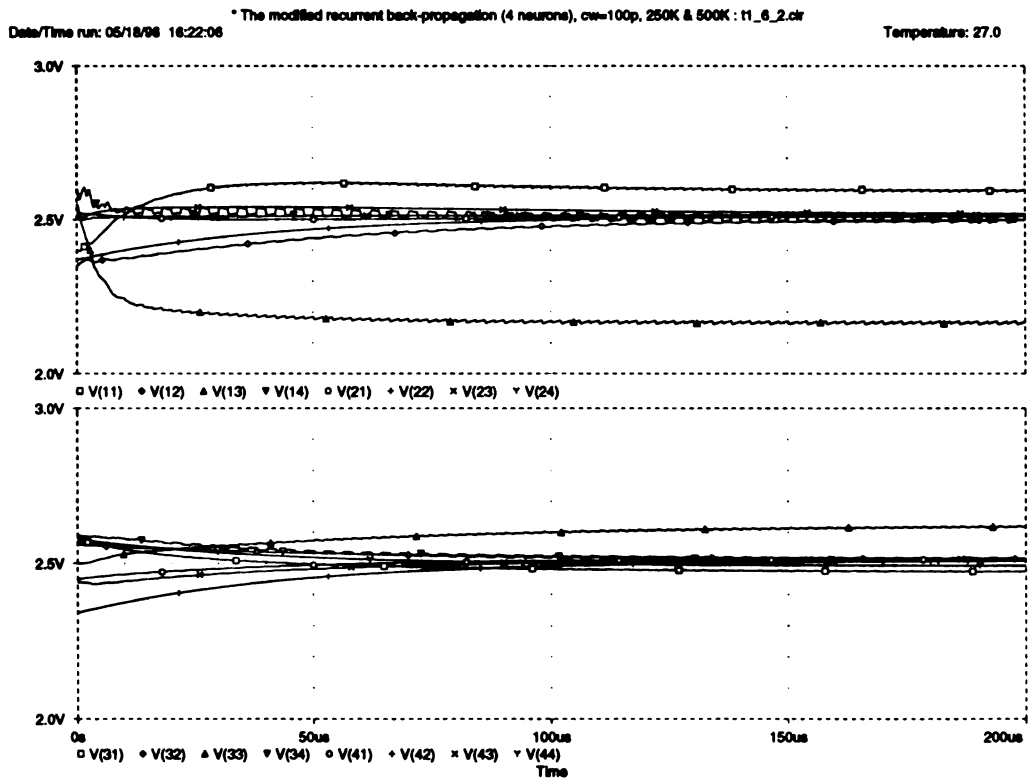

Figure 5.26. PSPICE transient analysis: weight waveforms, $w_{ij} = V(ij)$

Table 5.7. The averaged values of the weight from the PSPICE transient analysis

j	1	2	3	4
w_{1j}	2.5883	2.5042	2.1633	2.5008
w_{2j}	2.5132	2.5131	2.5148	2.5136
w_{3j}	2.4733	2.5140	2.6207	2.5158
w_{4j}	2.5125	2.5092	2.5151	2.4912

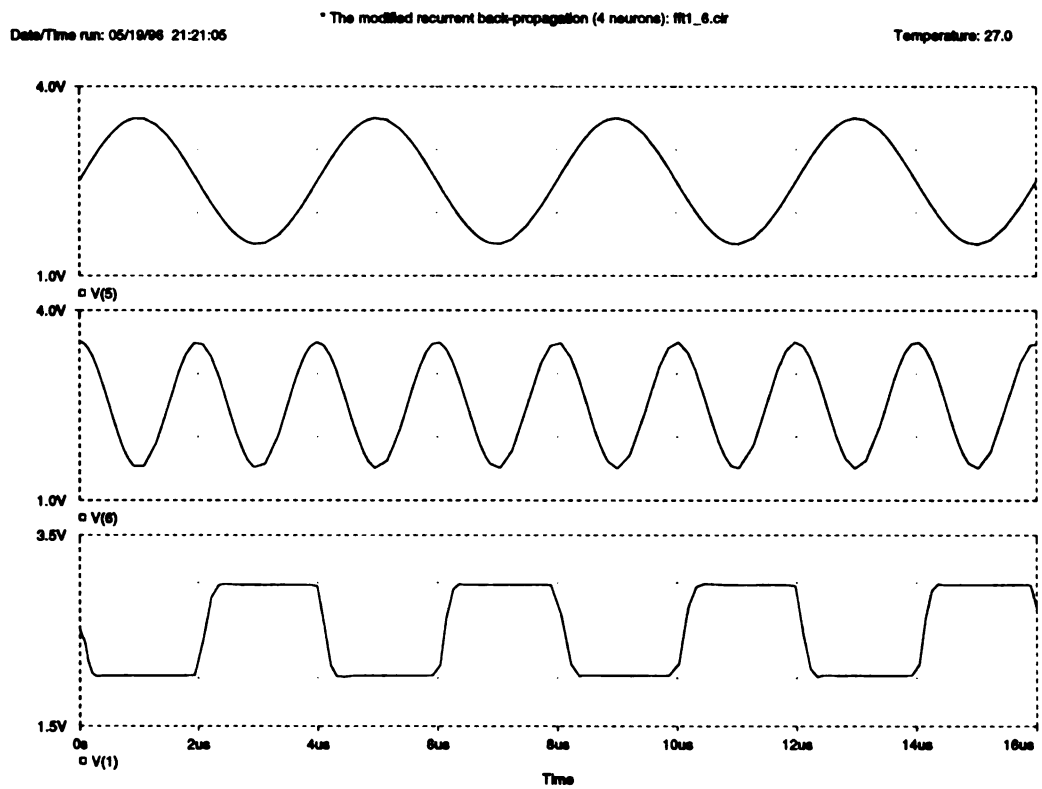

Figure 5.27. PSPICE transient analysis: the test result, V(5), v(6), and V(1) are the input 1, the input 2, and the actual output

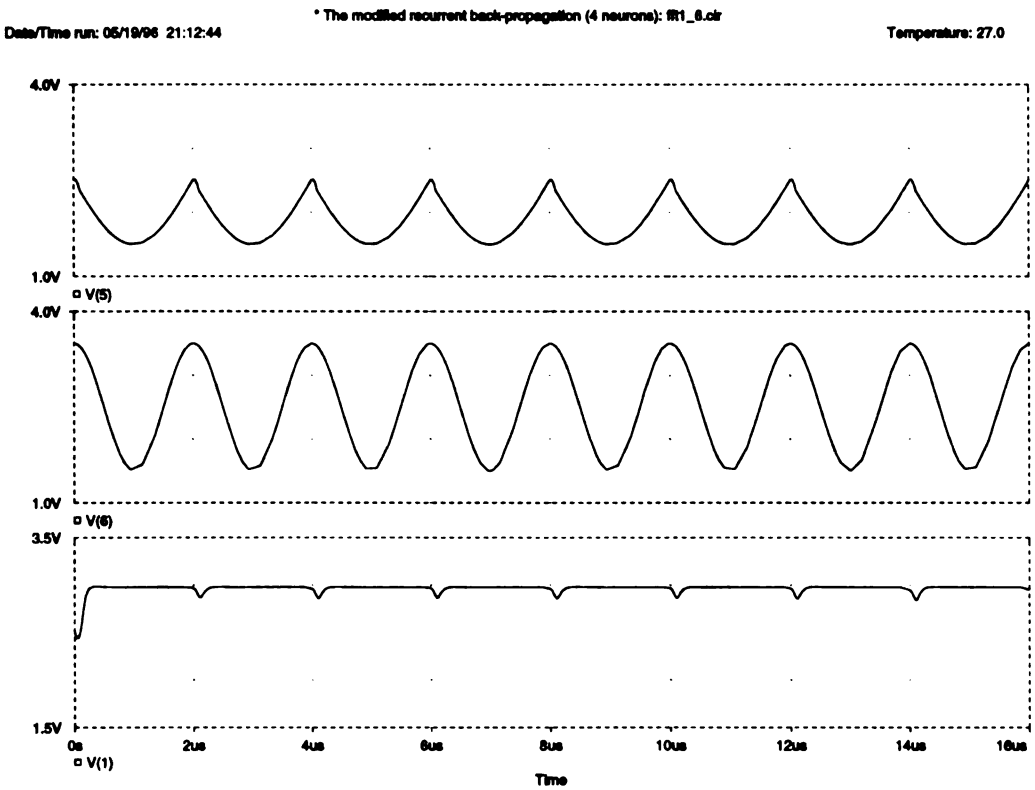

Figure 5.28. PSPICE transient analysis: the test result of the trajectory 1, V(5), V(6), and V(1) is the input 1, the input 2, and the actual output

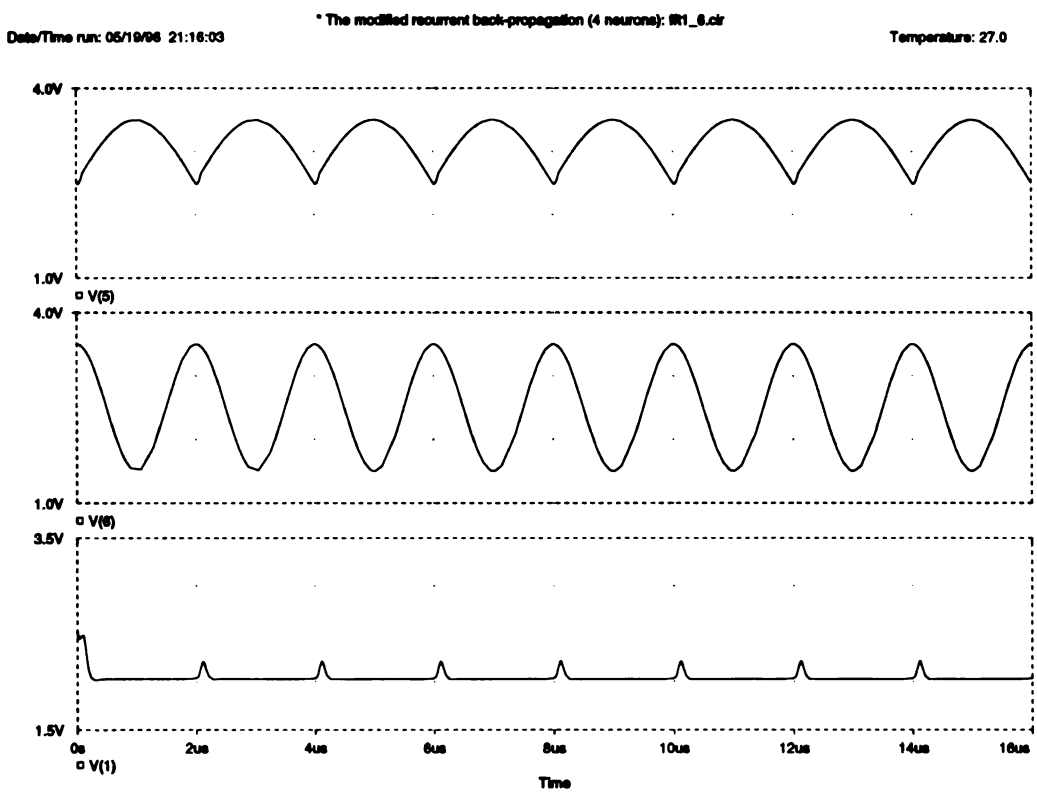

Figure 5.29. PSPICE transient analysis: the test result of the trajectory 2, V(5), V(6), and V(1) is the input 1, the input 2, and the actual output

Figure 5.26. In Figure 5.25, input waveforms, the target waveform of the trajectory, and the actual output waveform is shown. The weight values in the transient analysis are shown in Figure 5.26.

The averaged weight values which are obtained in the learning phase are summarized in Table 5.7. After setting these weight values to the recurrent neural network, the testing phase is performed. The result is shown in Figure 5.27, 5.28, and 5.29. In Figure 5.27, the waveform which is used at the learning phase is used as a test waveform. In Figure 5.28 and Figure 5.29, the trajectory 1 and the trajectory 2 is applied to the neural network to test the learning, respectively.

In this simulation, we use $250KHz$ signal for x_1 and $500KHz$ signal for x_2. The (W/L) ratio of the active resistor in 1-D multiplier of the weight update circuit is given as 4/64. Other simulations with (W/L)=4/16 have the same results as those of (W/L)=4/64. The capacitor of the weight update equation is given as a 100pF capacitor and the capacitor of the z equation is given as a 1pF capacitor in figures. We have changed the capacitor value of the weight update equation from 20pF to 200pF in other simulations. The test results are almost same as the case of 100pF capacitor. The simulation results show that this recurrent neural network succeeds to learn the different trajectories.

5.5 Simulation results of 6 neuron recurrent neural network

The 6-neuron recurrent neural network with learning capability is implemented using the modified recurrent back-propagation learning rule. We have performed the PSPICE circuit simulation with MOSIS $0.5\mu m$ technology for future development. The SPICE parameters for MOSIS $0.5\mu m$ HP process are shown in Appendix A.2.

The recurrent neural network has 6 neurons. It is fully connected including self-feedback and there are 36 connection weights in this recurrent neural network. The recurrent neural network is shown in Figure 5.30.

The recurrent neural network receives two inputs, x_1 and x_2. These inputs are connected through the neurons, y_5 and y_6. Two neurons, y_1 and y_2, are visible to outside with the target signals, d_1 and d_2, respectively.

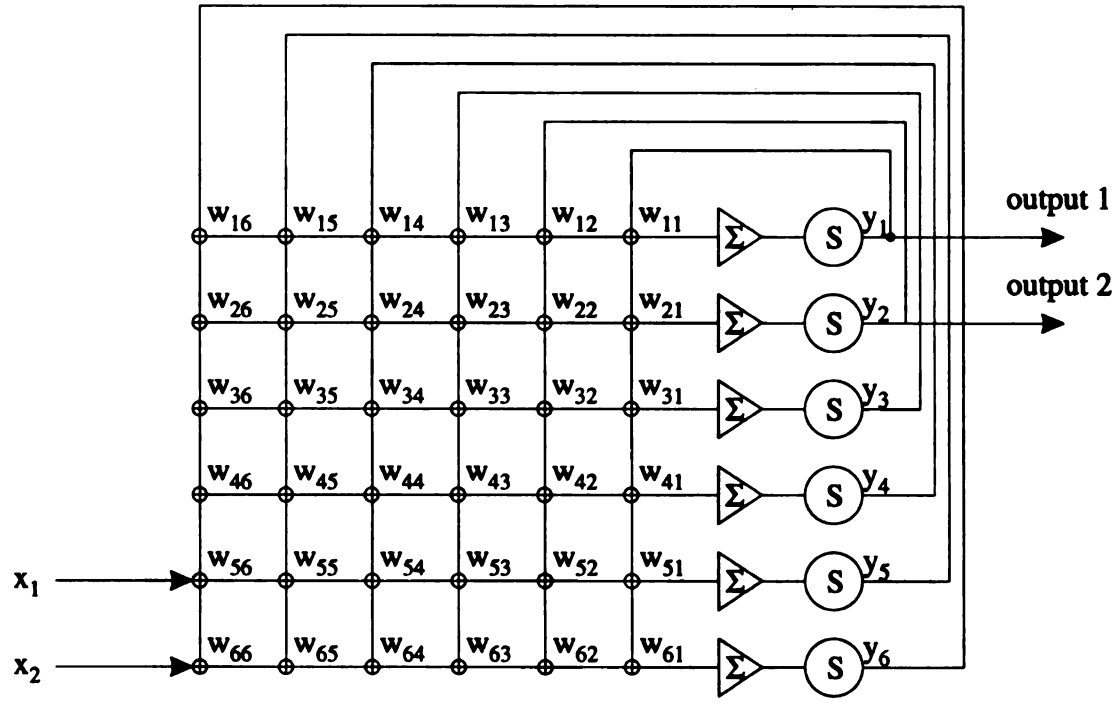

Figure 5.30. The recurrent neural network with six neurons, two inputs, and one output, S means the sigmoid function generator

We follow the general procedure for learning and testing as shown in the 4-neuron recurrent neural network. The 6-neuron recurrent neural network is trained to learn two trajectories. In Figure 5.31, two state space trajectories are shown. The trajectory 1 consists of $V1$ and $V2$ waveform. The trajectory 2 consists of $W1$ and $W2$ waveform. $V1$ and $W1$ are applied to x_1 and $V2$ and $W2$ are applied to x_2 of the

Table 5.8. The averaged values of the weight in the 6-neuron recurrent neural network with two output neurons

j	1	2	3	4	5	6
w_{1j}	2.5481	2.4543	2.5318	2.5326	2.1609	2.7469
w_{2j}	2.4893	2.5948	2.5277	2.5280	2.7394	2.1480
w_{3j}	2.5217	2.5201	2.5210	2.5204	2.5253	2.5213
w_{4j}	2.5210	2.5196	2.5207	2.5204	2.5250	2.5191
w_{5j}	2.4830	2.5547	2.5200	2.5199	2.6287	2.3962
w_{6j}	2.5542	2.4545	2.5123	2.5152	2.3975	2.6496

recurrent neural network.

In this simulation, we distinguish two trajectories by assigning two different states (ON/OFF) at two output neurons. If the recurrent neural network receives the trajectory 1, one of the output neurons will be ON state. If the trajectory 2 is applied to the input of the neural network, the other output neuron will be ON.

We have performed the PSPICE transient analysis. The results of the transient analysis in PSPICE simulation are shown in Figure 5.32 and Figure 5.33. In Figure 5.32, input waveforms, the target waveform, and the actual output of the neural network are shown. The actual output waveform tries to match the target waveform. The weight values at the end of simulation are shown in Figure 5.33. We measure the averaged values of the weights. The averaged weight values are summarized in Table 5.8.

After the learning phase, we perform the testing phase. We set the weights of the neural network to the obtained weight values at the learning phase. We verify the learning by applying the input waveforms to the neural network. The test result is shown in Figure 5.34, 5.35, and 5.36. In Figure 5.34, the waveform which is used at the learning phase is used as a test waveform. In Figure 5.35 and Figure 5.36, the trajectory 1 and the trajectory 2 is applied to the neural network to test the learning, respectively.

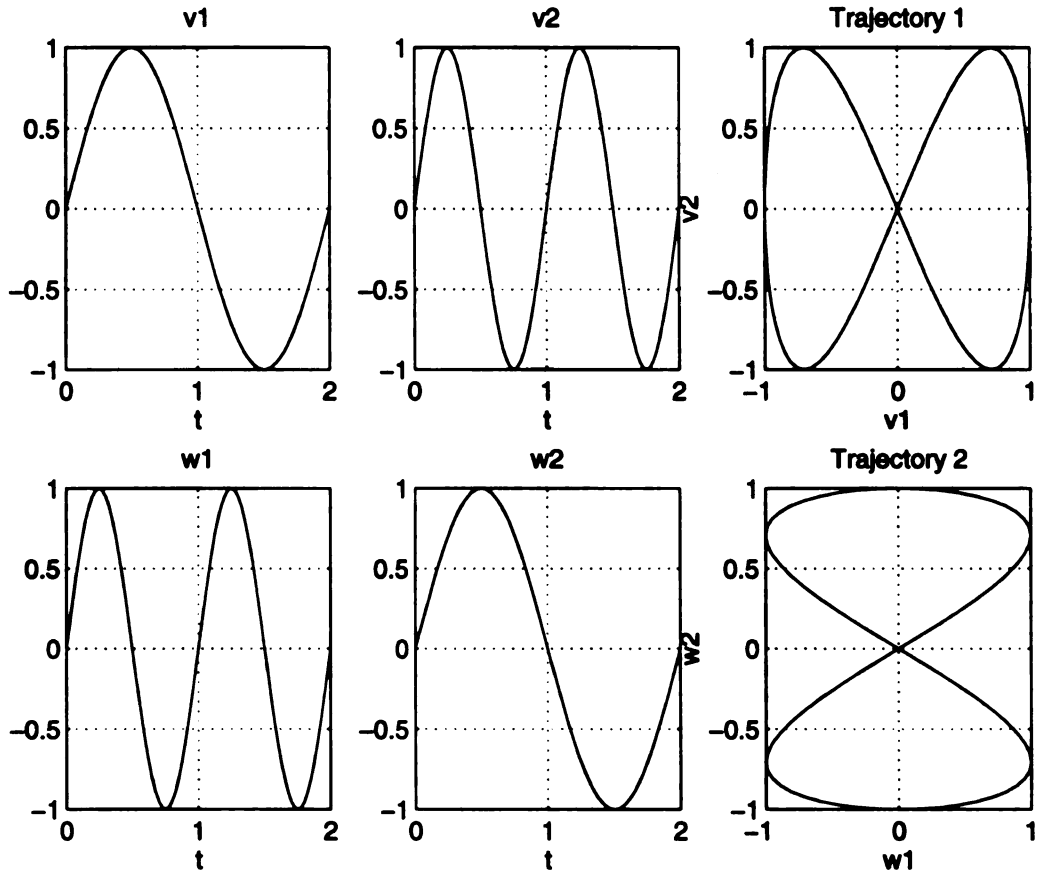

Figure 5.31. Two state trajectories: the two output neuron case on 6-neuron recurrent neural network

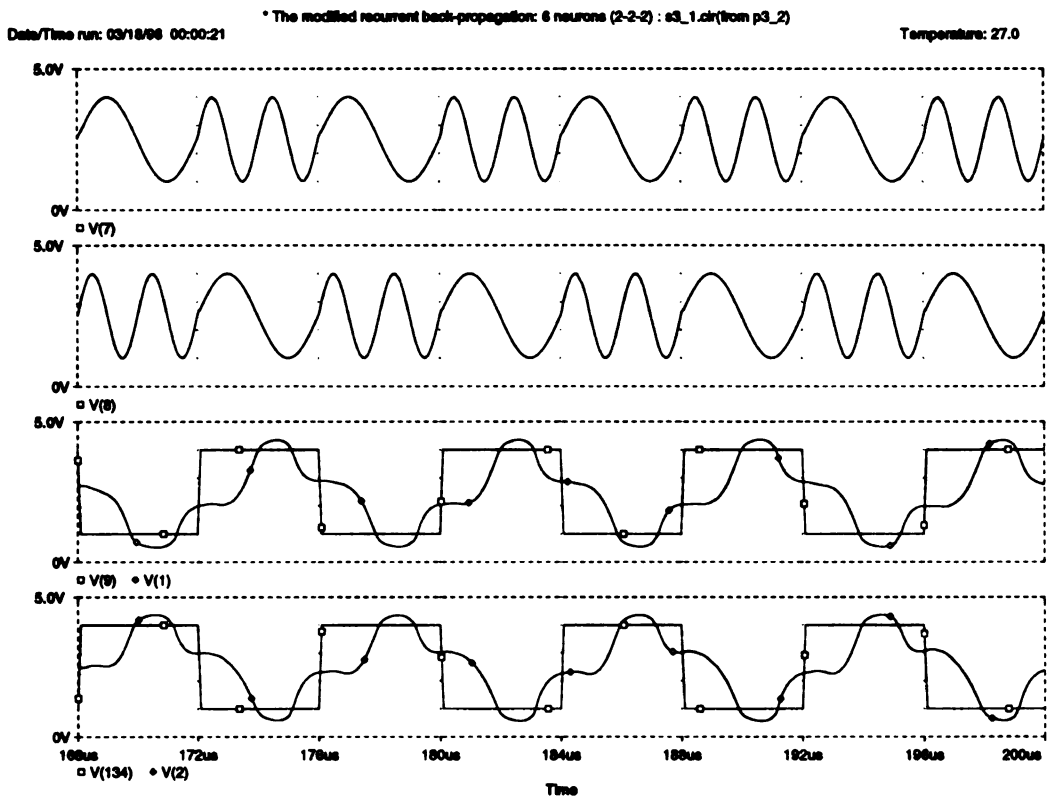

Figure 5.32. The PSPICE transient analysis: V(7) and V(8) are x_1 and x_2, V(9) and V(1) are d_1 and y_1, V(134) and V(2) are d_2 and y_2

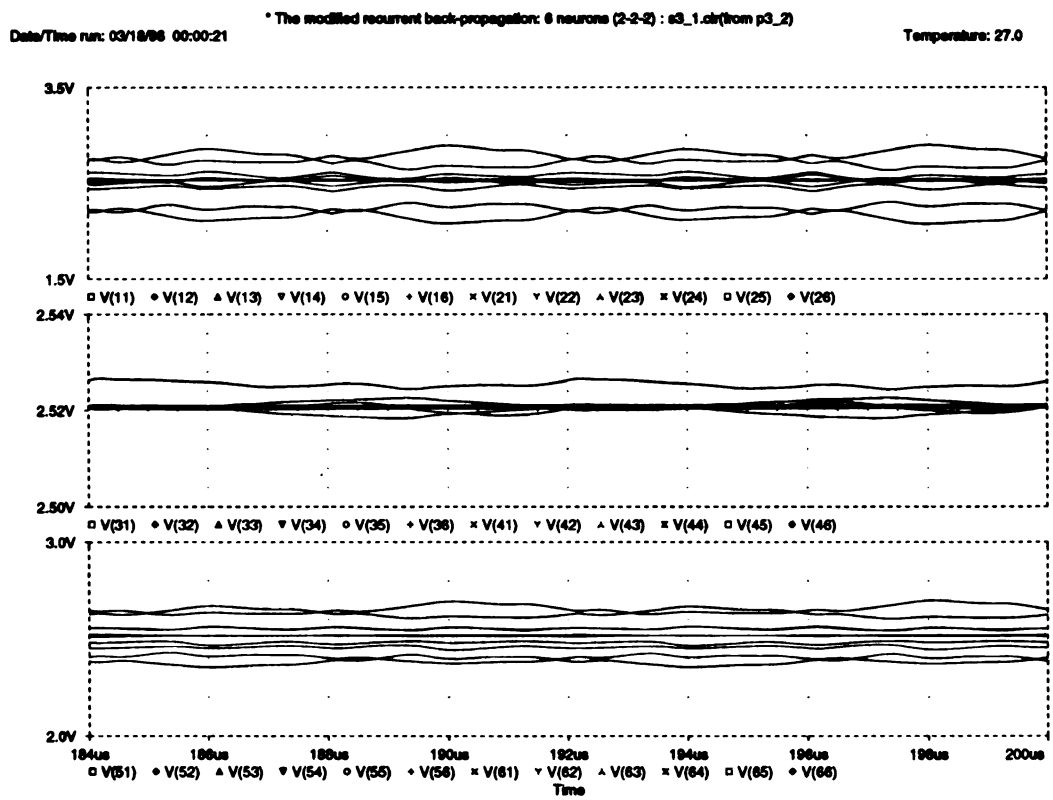

Figure 5.33. PSPICE transient analysis: weight waveforms of the 6-neuron recurrent neural network with two output neurons, $w_{ij} = V(ij)$

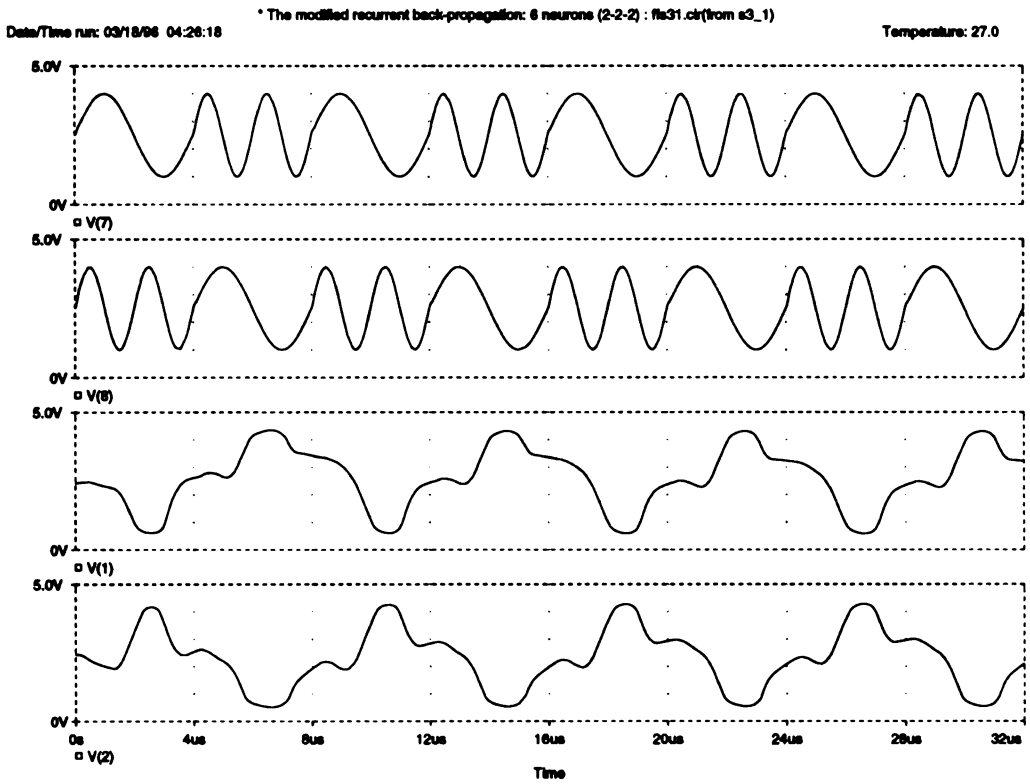

Figure 5.34. PSPICE transient analysis: the test result of the 6-neuron recurrent neural network, V(7), V(8), V(1), and V(2) are x_1, x_2, y_1, and y_2, respectively

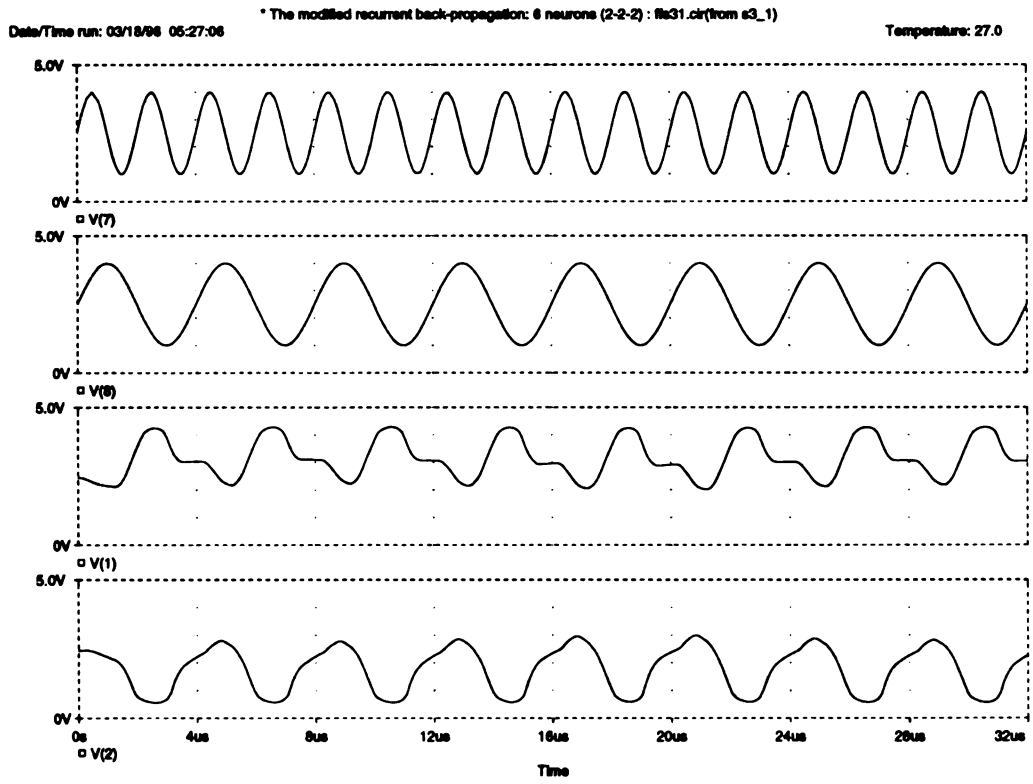

Figure 5.35. PSPICE transient analysis: the test result of the 6-neuron recurrent neural network, V(7), V(8), V(1), and V(2) are x_1, x_2, y_1, and y_2, respectively

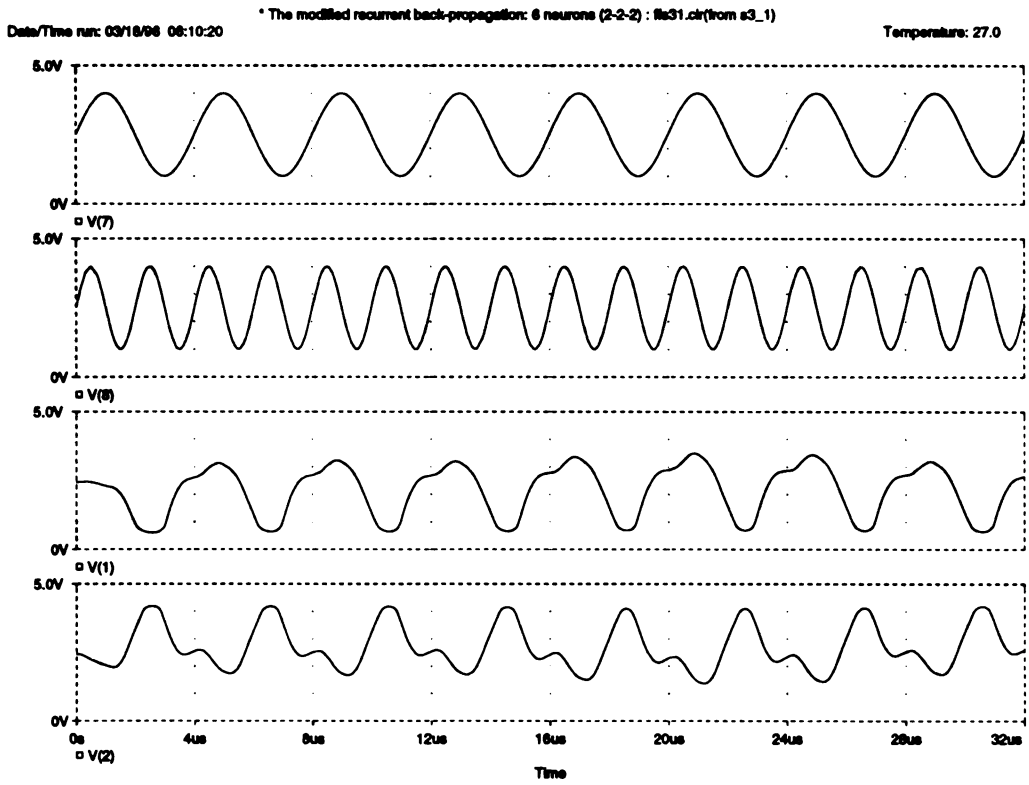

Figure 5.36. PSPICE transient analysis: the test result of the 6-neuron recurrent neural network, V(7), V(8), V(1), and V(2) are x_1, x_2, y_1, and y_2, respectively

As shown in the figures, the output of the neuron y_1 always goes to high state and the output of the neuron y_2 goes to low state when the trajectory 1 is applied to the neural network.

Since we take the averaged value of the weights, the actual output waveform in the testing phase is a little different from the output waveform in the learning phase. However, it is very close to the output waveform in the learning phase. The result shows that the approximation of the averaged weight value is quite acceptable in the test results.

If the output waveform of the testing phase is not saturated enough to classify the input waveforms, we can attach the buffer amplifier at the output of the recurrent neural network. The buffer amplifier can be a comparator or a double inverter with the threshold voltage of 2.5V.

In this simulation, $250KHz$ and $500KHz$ sine waveforms represent the state tra-

jectory. The capacitor of the weight update equation is implemented using a 200pF capacitor and the capacitor of the z equation is assigned to 10pF.

In the testing phase, we have applied the input trajectory that is not one of the learned trajectories. In this testing simulations, we applied the circular trajectory with different frequencies. The circular trajectory is shown in Figure 5.7. It consists of the sine waveform and the cosine waveform with same frequency.

The test results are shown in Figure 5.37 and Figure 5.38. With the circular trajectory of $500KHz$, the neural network classifies it as a trajectory 1. The simulation result is shown in Figure 5.37. With the circular trajectory of $250KHz$, the neural network can not classify the input trajectory. The output states of the neural network is changed as the input trajectory is applied. The simulation result is shown in Figure 5.38. Futher exhaustive testing is needed to characterize the behavior for other arbitrary signals.

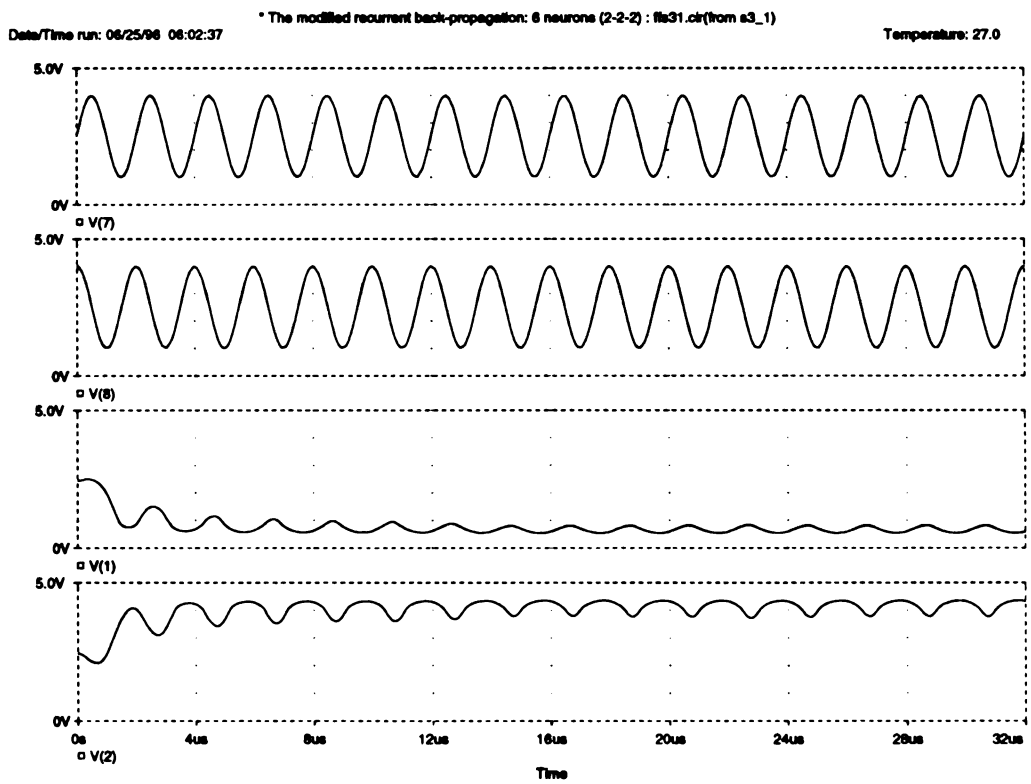

Figure 5.37. Testing phase with the circular trajectory: when the circular trajecroty is $500KHz$

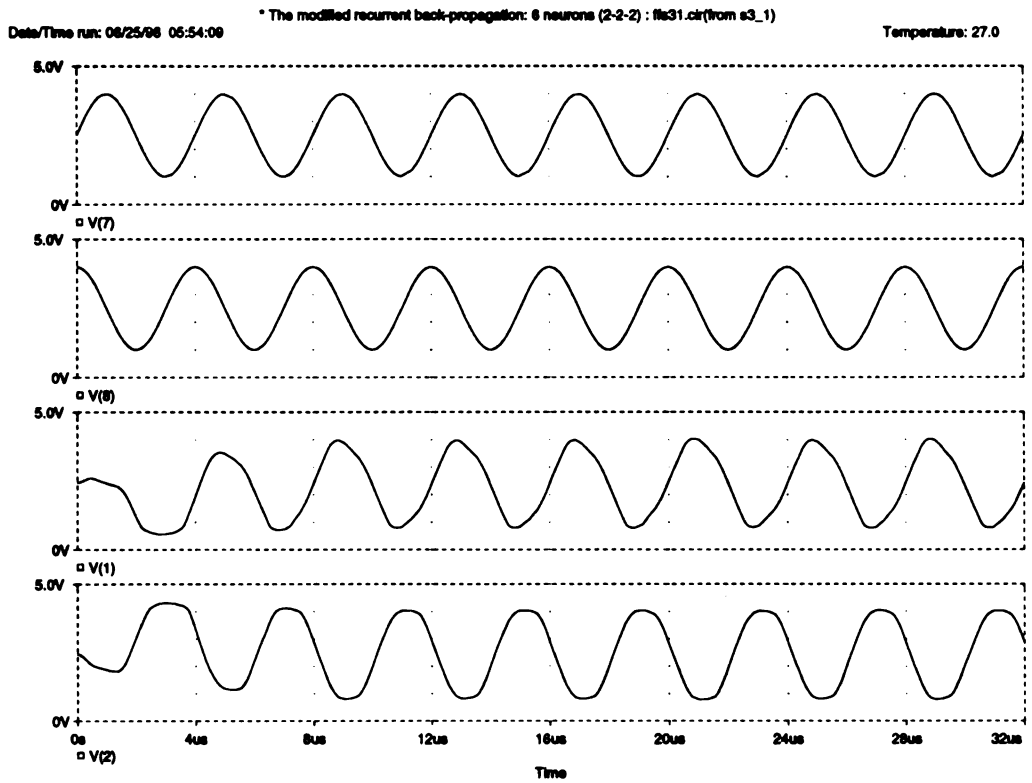

Figure 5.38. Testing phase with the circular trajectory: when the circular trajecroty is $250KHz$

5.6 Hardware implementation considerations

5.6.1 Hardware requirements of the modified recurrent back-propagation learning rule

The advantage of the implementation of the modified recurrent back-propagation learning rule over the modified recurrent real-time learning rule is that the modified recurrent back-propagation learning rule requires less learning circuitry. In section 4.5, we have investigated the hardware requirement of the real-time recurrent learning rule. The number of the learning circuitry increases in order of N^3 as the number of neuron is N. However, in the modified recurrent back-propagation learning rule, it needs only N equations for the z equation when the number of neuron is N. The

modified recurrent back-propagation learning rule is an efficient and economical way to implement the recurrent neural network.

5.6.2 Offset voltage adjustment

In the realization of the multiplier circuit, there is an offset current about $1\mu A$. If active resistors have larger resistance, this offset current has become a significant offset voltage. To compensate the offset voltage, we usually change the W/L ratio of active resistors. The W/L ratios of the upper PMOS are adjusted with the W/L ratios of the lower PMOS. If the offset voltage is lower than 2.5V, the resistance of the lower PMOS is increased by decreasing its W/L ratio, or the resistance of the upper PMOS is decreased by increasing its W/L ratio. If the offset voltage is higher than 2.5V, the opposite way is performed to get the near zero offset voltages.

5.6.3 The learning rate

The learning parameter η is related with the capacitor value in the learning circuit. In the PSPICE circuit simulation, the value of the capacitor is given from $10p$ to $200p$. With this value, on-chip implementation of the capacitor will not be appropriate. If the network is small and the chip has enough pins to connect each weight value to outside, the external capacitors will be used. If the capacitor value is the main concern for design, we need to develop the multiplier as a low current multiplier. In PSPICE circuit simulation, the output current of the multiplier ranges from $0\mu A$ to $50\mu A$. If we reduce the current level to the nano-A range, the capacitor value of several pico-F will be enough. One way of achieving the low-power multiplier is to use the sub-threshold design [38]. However, the design of a low-power multiplier includes several factors to be considered. The matching problem of the transistors, the poor linearity of the multiplication, and the slow speed of the circuit need to be overcome

in the sub-threshold design.

5.6.4 Weight refresh

In order to provide the two phases (the learning phase and the testing phase), the neural chip has two modes, the learning mode and the test mode [27][47]. The neural chip executes the learning mode with its input-target pair and the weight values are taken by the Analog-to-Digital (A/D) converter with external interface. In the testing phase, the obtained weights are written onto the capacitor which holds analog weight voltage using Digital-to-Analog (D/A) converter. The weight values have to be refreshed since the capacitor always discharges. The interface circuit for A/D converter, D/A converter, and refresh circuit is costly.

If we employ the second recurrent neural network (the slave neural network) whose weight values are taken from the first neural network (the master neural network), we don't need the refresh interfaces and the operations of two modes. The weight voltages are transferred through the voltage follower. The weight averaging also is achieved through the low-pass filtering with voltage follower. The slave neural network can be used for any testing works or any applications as long as the master neural network executing the learning tasks with its continuous input-target pair.

5.6.5 Temperature effects

The temperature dependence of CMOS components is an important performance characteristic in analog circuit design. The PSPICE simulations of the recurrent neural networks are valid only for limited ranges about room temperature. In the PSPICE simulation of the modified Gilbert multiplier and the sigmoid function generator with different temperatures, the change of the characteristics due to the change of temperature is much larger in the simulation of the sigmoid function generator. If

we tune the bias voltage of the sigmoid function generator via external pin in the neural chip, we can achieve proper operations in the wide range of temperature. In the PSPICE simulations of the trajectory generation and the trajectory recognition, the neural chip shows the successful operations over the temperature range of 0 to $50°C$. Other modifications are necessary for extreme temperature ranges. In Appendix C, the PSPICE simulation results of the trajectory generation with different temperature are shown. With the temperature range of 0 to $50°C$, the weights have converged as shown in Appendix C.1 and C.2. However, the weights hit the rail voltage of the circuit with extreme temperature such as $125°C$ or -$50°C$ as shown in Appendix C.3.

5.6.6 Future work

In this implementation, we don't include the update for time constant parameters. The time constant parameters give another degree of the freedom to the solution space of the neural network. Time constants of the equations are controlled by the capacitor value and the amount of current to the capacitor. The current can be controlled by the bias voltage. We need to develop the modified back-propagation learning rule to the time constant parameters.

We need an efficient architecture for practical applications. Experiments with small-scale problem have proved as fruitful in many areas of science and engineering. However, not every phenomenon encountered in dealing with small models can be usefully scaled up [19]. We have seen many interesting demonstrations of neural networks solving problems of very small scale but not doing so well when those problems were scaled up.

Control domains are the most natural application for continuous-time recurrent neural networks. Signal processing and speech recognition and generation are also domains to which the recurrent neural network might be naturally applied. Certainly there is no reason to use a recurrent network when a feedforward layered neural

network suffices. Almeida [59] pointed out that one should not expect a major increase in the performance of a perceptron in every situation with feedback. In most cases, the best network structure will probably turn out to have feedback only in a small group of units. Ljung [60] also mentioned that, for system identification, the identifier must be chosen to have a small number of parameters, i.e., fewer parameters for a neural network. This is because the more parameters we use, the higher is the random influence on the model. We need to investigate the characteristics of new architectures such as partially connected recurrent neural network not the fully connected neural network.

CHAPTER 6

2-Dimensional Scalable Array Configuration

The recurrent neural network and its learning algorithm is implemented on a single analog CMOS chip. The floor plan of the recurrent neural network is organized in the 2-D array configuration. With the 2-D array configuration, the layout offers a simple and scalable VLSI architecture for implementing a fully interconnected recurrent neural network.

6.1 Subcell design

The multiplication between two quantities is the basic circuit of the recurrent neural network. Since the modified Gilbert multiplier generates the current output, we employ the current bus to collect the analog current outputs. The collected currents are converted to the voltage output through active resistors.

To support the current bus and active resistors, we divide the modified Gilbert multiplier into two parts. The first component is the multiplier subcircuit which generates the current output, I+ and I-. The second component consists of a current mirror and active resistors. The modified Gilbert multiplier cell and the current

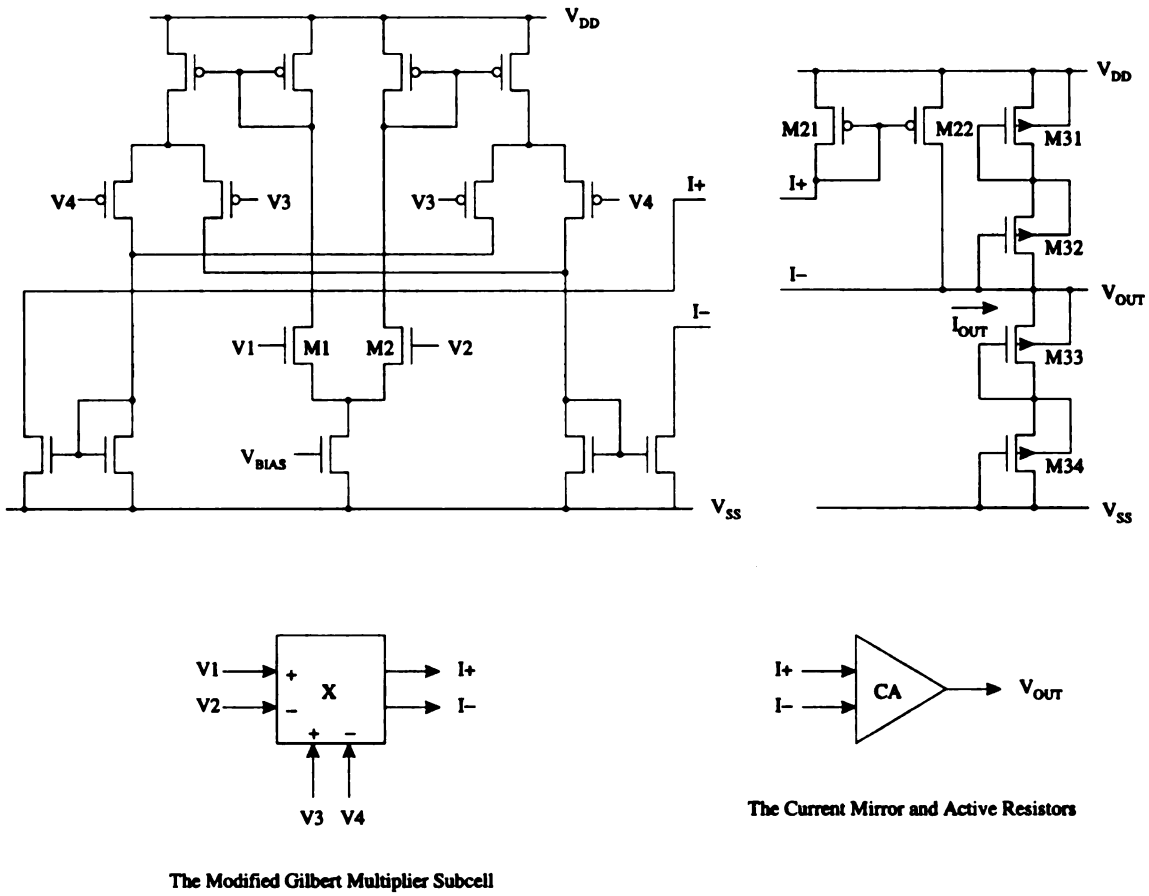

Figure 6.1. 2-D array configuration elements

mirror and active resistors are shown in Figure 6.1.

In this figure, we label the modified Gilbert multiplier subcell as a X component. The current mirror and active resistors are labeled as a CA component. The X component receives four voltage inputs and generates current outputs, I+ and I-. The current output is proportional to the multiplication between $(V1 - V2)$ and $(V3 - V4)$. Since the neural network chip operates from ground to VDD of 5V, the virtual ground voltage becomes 2.5V. We label this virtual ground as reference voltage, Vref, 2.5V. If $V2$ and $V4$ terminals of the X component are connected to the reference voltage, the multiplication between $V1$ and $V3$ occurs. If $V1$ and $V2$ are connected together, then there is no multiplication on the X component since

$(V1 - V2)$ produce the zero term. These current outputs are applied to the I+ and I- of the CA component. The CA component converts the current inputs to the voltage output.

There are two variations in the X component, $X1$ and $X2$. The $X2$ component has higher conductance than that of the $X1$ component. We use large (W/L) ratio of the transistor M1 and M2 in the $X2$ component. The $X2$ component is used at the 1-dimensional multiplier for weight update equation.

The CA component has three variations, $CA1$, $CA2$, and $CA3$. We have designed the voltage gain of the learning circuit is a little higher than that of the recurrent neural activation circuit. The $CA1$ component is used at the recurrent learning circuit and the $CA2$ component is used at the z equation generator. The $CA3$ component is used at the weight update equation and has the highest voltage gain overall.

6.2 Implementation of Floor Plan

The floor plan of the VLSI recurrent neural network is organized in the 2-D array of weight interconnections. The block diagram of a fully connected 4-neuron recurrent neural network is shown in Figure 6.2. The 2-D array of weights and two boundary cells are shown in this figure.

In Figure 6.2, we implement 4-neuron recurrent neural networks with maximum two input neurons and two output neurons. The number of inputs or outputs can be reduced by connecting the input terminal to the Vref or connecting the target terminal with the output terminal.

The main function of the weight cell, w_{ij} in Figure 6.2, is generating the activation term, $w_{ij}y_j$, of the neural network and the z term, $w_{ij}z_i$, in the learning equation. The upper $X1$ component generates the $w_{ij}z_i$ current output and the lower right $X1$ component generates the $w_{ij}y_j$ current output. These current outputs are connected

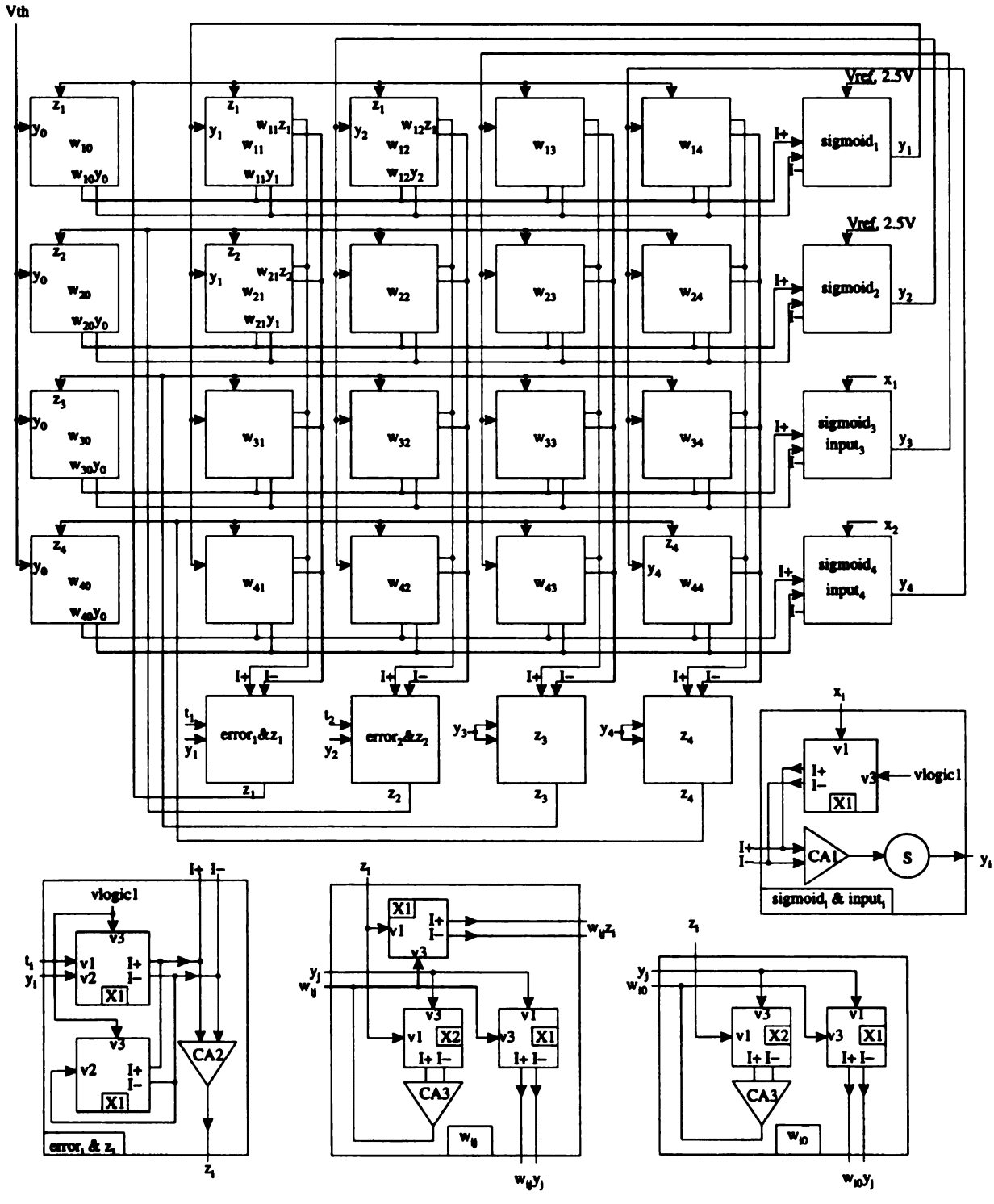

Figure 6.2. The array structure of the recurrent neural network

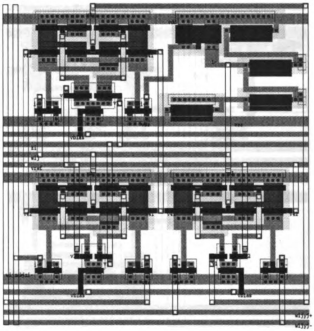

Figure 6.3. The MAGIC layout of the weight cell, w_{ij}

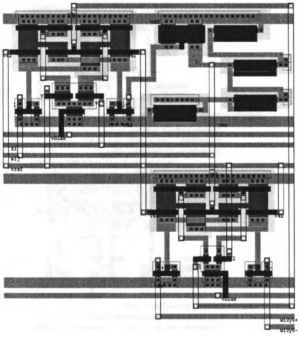

Figure 6.4. The MAGIC layout of the weight cell, w_{i0}

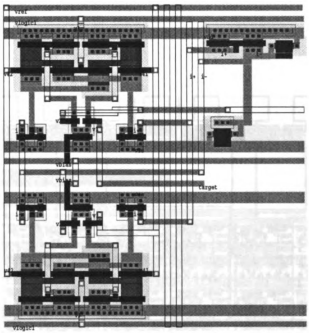

Figure 6.5. The MAGIC layout of the $error_i$ & z_i cell

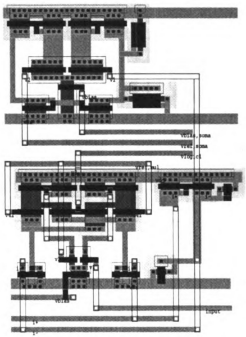

Figure 6.6. The MAGIC layout of the $sigmoid_i$ & $input_i$ cell

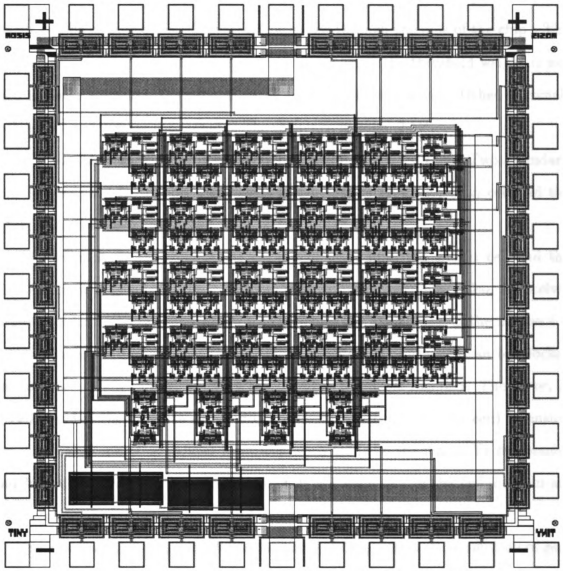

Figure 6.7. The MAGIC layout of the recurrent neural network

to the current buses and are collected in the $error_i$ & z_i cell and in the $sigmoid_i$ & $input_i$ cell. Also the update rule for weight itself is performed with z_i and y_j terms by the lower left $X2$ component and the $CA3$ component. The capacitor is not included in this cell. The capacitor is located in the outside area of the array structure or weight can be connected to the external capacitor through the pins of the chip . The threshold weight cell, w_{i0}, is shown in Figure 6.2. Since the threshold weight is not used at the z equation, the generation of $w_{ij}z_i$ circuit is removed. Other terminals such as $V2$ and $V4$ are connected to the reference voltage, $2.5V$.

The 2-D array of weights is connected with the boundary cells. Two boundary cells are designed to support the weight array. One is an $error_i$ & z_i cell and the other is a $sigmoid_i$ & $input_i$ cell.

The $error_i$ & z_i cell collects the current of $w_{ij}z_i$ from the weight cell and the $CA2$ component generates the z_i voltage with collected currents. Also, it receives target waveform and output waveform to generate the error term. If the neuron i is not used as an output neuron, the upper $X1$ component in the cell can be blocked not to generate any output by tying the t_i and y_i terminals together. The $error_i$ & z_i cell also has the negative self-feedback component (the lower $X1$ cell) to ensure the convergent operation. Other terminals, such as $V4$ in the upper $X1$ component and $V1$ in the lower $X1$ component, which is not displayed in the cell diagram are connected to the reference voltage, $2.5V$.

The $sigmoid_i$ & $input_i$ cell collects the current of $w_{ij}y_j$ from the weight cell to generate the net_i of the neuron i. These currents are converted to the voltage through the $CA1$ component and the output of the $CA1$ is applied to the sigmoid function generator. The sigmoid function generator is implemented using the wide-range transconductance amplifier and its output becomes the state of the recurrent neural network. Also, this cell can receive the input waveform via the $X1$ component. If the neuron is used as an input neuron, the $X1$ component is activated to get the input

waveform. If the neuron is not used as an input neuron, the $V1$ terminal of the $X1$ component is connected to the reference voltage, $2.5V$. The $V2$ and $V4$ terminals of the $X1$ cell are connected to the reference voltage.

This current-bus output arrangement combined with the boundary cells offers a simple and scalable VLSI architecture for implementing a fully interconnected recurrent neural network with learning circuit.

Figure 6.3 shows the MAGIC layout of the weight cell, w_{ij}. It has two $X1$ components, one $X3$ component, and one $CA3$ cell. It has been drawn using Scalable CMOS (SCMOS) technology and its size is $252\lambda \times 240\lambda$. If $1\lambda = 1\mu$, the actual size becomes $252\mu m \times 240\mu m$. The actual size is dependent on the fabrication technology. In Figure 6.4, the layout of the w_{i0} cell is shown. The size of the w_{i0} cell is $252\lambda \times 224\lambda$.

The other cells such as $error_i$ & z_i and $sigmoid_i$ & $input_i$ has smaller size than the weight cell since they use fewer components. In Figure 6.5, the layout of the $error_i$ & z_i cell is displayed. Figure 6.6 shows the layout of the $sigmoid_i$ & $input_i$ cell. Each size is $220\lambda \times 214\lambda$ and $247\lambda \times 178\lambda$, respectively.

In Figure 6.7, the whole chip layout is displayed. 4×5 weight array is located in the center of the chip and the lower and right boundary cells are surround the weigh array. The lower boundary cells are $error_i$ & z_i cells and right boundary cells are $sigmoid_i$ & $input_i$ cells. The chip has 40 pins, 34 analog pads and 6 VDD and GND pads. It is designed to fabricate via MOSIS Tiny chip. In Table 6.1, the pin assignment of 40-pin tiny chip is shown.

Table 6.1. The pin assignment of the 4 neuron recurrent neural network chip

Pin	1	2	3	4	5
Signal	x_1	x_2	y_1	y_2	VDD
Pin	6	7	8	9	10
Signal	w_{11}	w_{12}	w_{13}	w_{14}	GND
Pin	11	12	13	14	15
Signal	w_{10}	w_{21}	w_{22}	w_{23}	VDD
Pin	16	17	18	19	20
Signal	w_{24}	w_{20}	Vbias,mul	Vref,mul	Vth
Pin	21	22	23	24	25
Signal	unused	Vlogic1	w_{31}	w_{32}	GND
Pin	26	27	28	29	30
Signal	w_{33}	w_{34}	w_{30}	w_{41}	VDD
Pin	31	32	33	34	35
Signal	w_{42}	w_{43}	w_{44}	w_{40}	GND
Pin	36	37	38	39	40
Signal	t_1	t_2	unused	Vref,sig	Vbias,sig

CHAPTER 7

Conclusion

A recurrent neural network with a modified recurrent back-propagation learning rule is implemented using analog CMOS technology. In order to implement the recurrent neural network and its learning algorithm, we employ a modified Gilbert multiplier, an active resistor, and a wide-range transconductance amplifier.

The sigmoid function generator is designed using the transconductance amplifier. The limitation of the output voltage is resolved by using the wide-range transconductance amplifier. The output of the transconductance amplifier is current.

To convert the current output to the voltage output, we use an active resistor. To get appropriate ranges of resistance of the active resistor, we performed the PSPICE circuit simulation with different W/L ratios. By adjusting the W/L ratio of the transistors, we can get a proper resistance value for converting the current output to the voltage output.

The modified Gilbert multiplier uses voltage signals for its inputs where its output is current. We attached active resistors to get voltage-to-voltage operations. In the small-signal range, the characteristic curve is approximately linear, and its four-quadrant multiplication is verified through the PSPICE circuit simulation.

Since the modified Gilbert multiplier cell generates the current output, the vector multiplier is designed on the current bus to collect the currents from the modified

Gilbert multiplier cells. The dimension of the vector multiplier can be increased by simply placing the modified Gilbert multiplier cell on the current bus. The adjusted active resistor converts this current to the voltage for voltage-to-voltage operations.

We have reviewed four learning algorithms for temporal signal learning. The Pearlmutter's algorithm converts a network evolving through time into a network whose activation is flowing through a number of layers (space). The requirements of the forward and backward integration make analog hardware implementation difficult. For the classification of temporal trajectories, Sotelino et al. has developed the modified version of the Pearlmutter's algorithm. This algorithm has the same problems as the Pearlmutter's algorithm. The recurrent back-propagation algorithm by Pineda assumes that the activation equation of the neural network is convergent and the error is measured in the fixed point. The real-time recurrent learning by William and Zipser has on-line updating rule. However, its hardware requirements are so massive that we can not build a large network economically.

We have modified the Pearlmutter's algorithm and the Pineda's algorithm for the modified recurrent back-propagation. Its forward instantaneous update scheme is suitable for an analog hardware implementation.

We have built a 4-neuron recurrent neural network and a 6-neuron recurrent neural network. We have implemented the modified recurrent back-propagation learning rule using standard CMOS circuit and performed the PSPICE circuit simulations.

In the 4-neuron recurrent neural network simulations, we have verified its functions by generating a circular trajectory. Simulation results show that the output signal is following the target signal and weights are convergent. The circular trajectory is generated by the recurrent neural network as a limit cycle. In the trajectory recognition experiment, we trained the neural network to recognize different trajectories. Its learning phase and test phase results show that the modified recurrent back-propagation learning rule is successful in learning and in testing the temporal

signals.

In the 6-neuron recurrent neural network simulations, we have built two-neuron output neural network. We trained the neural network to learn different state trajectories and the PSPICE circuit simulations show the recurrent neural network has learned the temporal signals for classification.

A two-dimensional scalable array configuration is designed for large-scale implementation of fully connected recurrent neural network with learning. With the 2-D array configuration, the layout offers a simple and scalable VLSI architecture. We have built a 40-pin tiny chip using MOSIS's SCMOS technology.

APPENDICES

APPENDIX A

SPICE parameters

A.1 The SPICE parameters: MOSIS 2.0 μm ORBIT ANALOG process

	N MOS	P MOS
LEVEL	2	2
TPG		-1
LD	133.300000E-09	102.100000E-09
VTO	.8577	-.8721
KP	59.272000E-06	16.129000E-06
GAMMA	.5361	.5972
PHI	.7	.7
LAMBDA	.03084	.03942
RSH	12.93	.1019
IS	10.000000E-15	10.000000E-15
PB	.4	.9
PBSW	.4	.9
CJ	134.000000E-06	334.000000E-06
CJSW	611.000000E-12	397.000000E-12
MJ	.535	.585
MJSW	.2	.127
CGSO	174.800000E-12	133.890000E-12
CGDO	174.800000E-12	133.890000E-12
CGBO	345.820000E-12	401.740000E-12
NSUB	6.617000E+15	8.212000E+15
NFS	93.830000E+09	607.200000E+09
TOX	39.500000E-09	39.500000E-09
XJ	200.000000E-09	200.000000E-09
UO	678	184.5
UCRIT	6.778000E+03	207.200000E+03
UEXP	.0875	.4362
VMAX	48.300000E+03	999.900000E+03
DELTA	2.779	3.097

A.2 The SPICE parameters: MOSIS 0.5 μm HP process

	N MOS	P MOS
LEVEL	3	3
TPG		-1
LD	47.290000E-09	35.070000E-09
VTO	.6566	-.9213
KP	196.470000E-06	48.740000E-06
GAMMA	.5976	.4673
PHI	.7	.7
RSH	35.12	.11
IS	10.000000E-15	10.000000E-15
PB	.99	.99
PBSW	.99	.99
CJ	562.000000E-06	935.000000E-06
CJSW	50.000000E-12	289.000000E-12
MJ	.559	.468
MJSW	.521	.505
CGSO	305.150000E-12	239.220000E-12
CGDO	305.150000E-12	239.220000E-12
CGBO	402.390000E-12	375.790000E-12
NSUB	139.200000E+15	85.120000E+15
NFS	590.900000E+09	650.000000E+09
TOX	9.600000E-09	9.600000E-09
XJ	200.000000E-09	200.000000E-09
UO	546.2	135.5
VMAX	200.800000E+03	254.200000E+03
DELTA	.691	.2875
THETA	.2684	.1807
ETA	.03718	.0245
KAPPA	.02898	7.958

APPENDIX B

PSPICE Input Files

B.1 Simple transconductance amplifier

```
* Simple trans-conductance amplifier: Vbias = 1.3V (trans1.cir)
vdd 80 0 5.0
vss 90 0 0.0
v1 1 0
v2 2 0
vbias 3 0 1.3
vdum 6 66 0.0

m1 5 1 4 90 n w=4u l=4u
m2 6 2 4 90 n w=4u l=4u
m3 4 3 90 90 n w=12u l=4u
m4 80 5 5 80 p w=15u l=4u
m5 80 5 6 80 p w=15u l=4u

r1 80 66 1000K
r2 66 0 1000K

N63J SPICE LEVEL2 PARAMETERS
.MODEL N NMOS LEVEL=2 PHI=0.700000 TOX=3.9500E-08 XJ=0.200000U TPG=1
+ VTO=0.8577 DELTA=2.7790E+00 LD=1.3330E-07 KP=5.9272E-05
+ UO=678.0 UEXP=8.7500E-02 UCRIT=6.7780E+03 RSH=1.2930E+01
+ GAMMA=0.5361 NSUB=6.6170E+15 NFS=9.3830E+10 VMAX=4.8300E+04
+ LAMBDA=3.0840E-02 CGDO=1.7480E-10 CGSO=1.7480E-10
+ CGBO=3.4582E-10 CJ=1.34E-04 MJ=0.535 CJSW=6.11E-10
+ MJSW=0.200 PB=0.40
.MODEL P PMOS LEVEL=2 PHI=0.700000 TOX=3.9500E-08 XJ=0.200000U TPG=-1
+ VTO=-0.8721 DELTA=3.0970E+00 LD=1.0210E-07 KP=1.6129E-05
+ UO=184.5 UEXP=4.3620E-01 UCRIT=2.0720E+05 RSH=1.0190E-01
+ GAMMA=0.5972 NSUB=8.2120E+15 NFS=6.0720E+11 VMAX=9.9990E+05
+ LAMBDA=3.9420E-02 CGDO=1.3389E-10 CGSO=1.3389E-10
+ CGBO=4.0174E-10 CJ=3.34E-04 MJ=0.585 CJSW=3.97E-10
+ MJSW=0.127 PB=0.90

.probe v(6) v(66) i(r1) i(r2) i(vdum)
.dc v1 0.0 5.0 0.05 v2 0.5 4.5 0.5
.end
```

B.2 Wide range transconductance amplifier

```
* Wide trans-conductance amplifier: (wtran1.cir)
vdd 80 0 5.0
vss 90 0 0.0
v1 1 0
v2 2 0
vbias 3 0 1.3

m1 5 1 4 90 n w=4u l=4u
m2 7 2 4 90 n w=4u l=4u
m6 80 5 5 80 p w=15u l=4u
m7 80 5 6 80 p w=15u l=4u
m5 80 7 7 80 p w=15u l=4u
m4 80 7 8 80 p w=15u l=4u
m8 8 8 90 90 n w=4u l=4u
m9 6 8 90 90 n w=4u l=4u
m3 4 3 90 90 n w=12u l=4u

r1 80 6 1000K
r2 6 0 1000K

N63J SPICE LEVEL2 PARAMETERS
.MODEL N NMOS LEVEL=2 PHI=0.700000 TOX=3.9500E-08 XJ=0.200000U TPG=1
+ VTO=0.8577 DELTA=2.7790E+00 LD=1.3330E-07 KP=5.9272E-05
+ UO=678.0 UEXP=8.7500E-02 UCRIT=6.7780E+03 RSH=1.2930E+01
+ GAMMA=0.5361 NSUB=6.6170E+15 NFS=9.3830E+10 VMAX=4.8300E+04
+ LAMBDA=3.0840E-02 CGDO=1.7480E-10 CGSO=1.7480E-10
+ CGBO=3.4582E-10 CJ=1.34E-04 MJ=0.535 CJSW=6.11E-10
+ MJSW=0.200 PB=0.40
.MODEL P PMOS LEVEL=2 PHI=0.700000 TOX=3.9500E-08 XJ=0.200000U TPG=-1
+ VTO=-0.8721 DELTA=3.0970E+00 LD=1.0210E-07 KP=1.6129E-05
+ UO=184.5 UEXP=4.3620E-01 UCRIT=2.0720E+05 RSH=1.0190E-01
+ GAMMA=0.5972 NSUB=8.2120E+15 NFS=6.0720E+11 VMAX=9.9990E+05
+ LAMBDA=3.9420E-02 CGDO=1.3389E-10 CGSO=1.3389E-10
+ CGBO=4.0174E-10 CJ=3.34E-04 MJ=0.585 CJSW=3.97E-10
+ MJSW=0.127 PB=0.90

.probe v(6) i(r1) i(r2)
.dc v1 0.0 5.0 0.05 v2 0.5 4.5 0.5
.end
```

B.3 Sigmoid function generator 1

```
* Soma (SIGMOIDAL FUNCTION) circuit with adjusted W/L, Vbias=1.1V: ORBIT, N63J
(sm1-20u.cir)

vdd 80 0 5.0
vss 90 0 0.0

vbias 3 0 1.1
v2 2 0 2.5
v1 1 0

xmain1 80 90 1 2 3 8 main
xact1 80 90 8 ar1

xmain2 80 90 1 2 3 18 main
xact2 80 90 18 ar2

xmain3 80 90 1 2 3 28 main
xact3 80 90 28 ar3

xmain4 80 90 1 2 3 38 main
xact4 80 90 38 ar4

xmain5 80 90 1 2 3 48 main
xact5 80 90 48 ar5

.subckt main 80 90 1 2 3 8
m1 5 1 4 90 n w=24u l=4u
m2 6 2 4 90 n w=24u l=4u
m3 4 3 90 90 n w=10u l=4u
m4 80 6 7 80 p w=15u l=4u
m5 80 6 6 80 p w=15u l=4u
m6 80 5 5 80 p w=15u l=4u
m7 80 5 8 80 p w=15u l=4u
m8 7 7 90 90 n w=16u l=4u
m9 8 7 90 90 n w=16u l=4u
.ends

.subckt ar1 80 90 8
m10 80 8 8 80 p w=4u l=4u
m11 8 90 90 8 p w=4u l=4u
.ends

.subckt ar2 80 90 8
m10 80 8 8 80 p w=4u l=12u
m11 8 90 90 8 p w=4u l=12u
.ends
```

```
.subckt ar3 80 90 8
m10 80 8 8 80 p w=4u l=20u
m11 8 90 90 8 p w=4u l=20u
.ends

.subckt ar4 80 90 8
m10 80 8 8 80 p w=4u l=28u
m11 8 90 90 8 p w=4u l=28u
.ends

.subckt ar5 80 90 8
m10 80 8 8 80 p w=4u l=36u
m11 8 90 90 8 p w=4u l=36u
.ends

N63J SPICE LEVEL2 PARAMETERS
.MODEL N NMOS LEVEL=2 PHI=0.700000 TOX=3.9500E-08 XJ=0.200000U TPG=1
+ VTO=0.8577 DELTA=2.7790E+00 LD=1.3330E-07 KP=5.9272E-05
+ UO=678.0 UEXP=8.7500E-02 UCRIT=6.7780E+03 RSH=1.2930E+01
+ GAMMA=0.5361 NSUB=6.6170E+15 NFS=9.3830E+10 VMAX=4.8300E+04
+ LAMBDA=3.0840E-02 CGDO=1.7480E-10 CGSO=1.7480E-10
+ CGBO=3.4582E-10 CJ=1.34E-04 MJ=0.535 CJSW=6.11E-10
+ MJSW=0.200 PB=0.40
.MODEL P PMOS LEVEL=2 PHI=0.700000 TOX=3.9500E-08 XJ=0.200000U TPG=-1
+ VTO=-0.8721 DELTA=3.0970E+00 LD=1.0210E-07 KP=1.6129E-05
+ UO=184.5 UEXP=4.3620E-01 UCRIT=2.0720E+05 RSH=1.0190E-01
+ GAMMA=0.5972 NSUB=8.2120E+15 NFS=6.0720E+11 VMAX=9.9990E+05
+ LAMBDA=3.9420E-02 CGDO=1.3389E-10 CGSO=1.3389E-10
+ CGBO=4.0174E-10 CJ=3.34E-04 MJ=0.585 CJSW=3.97E-10
+ MJSW=0.127 PB=0.90

.probe v(8) v(18) v(28) v(38) v(48)
.dc v1 1.5 3.5 0.01
.end
```

B.4 Sigmoid function generator 2

```
* Soma (SIGMOIDAL FUNCTION) circuit with different bias with (4u/19u,4u/21u): ORBIT,
N63J (sm2-20u.cir)

vdd 80 0 5.0
vss 90 0 0.0

vbias 3 0
v2 2 0 2.5
v1 1 0

xmain2 80 90 1 2 3 18 main
xact2 80 90 18 ar2

.subckt main 80 90 1 2 3 8
m1 5 1 4 90 n w=24u l=4u
m2 6 2 4 90 n w=24u l=4u
m3 4 3 90 90 n w=10u l=4u
m4 80 6 7 80 p w=15u l=4u
m5 80 6 6 80 p w=15u l=4u
m6 80 5 5 80 p w=15u l=4u
m7 80 5 8 80 p w=15u l=4u
m8 7 7 90 90 n w=16u l=4u
m9 8 7 90 90 n w=16u l=4u
.ends

.subckt ar2 80 90 8
m10 80 8 8 80 p w=4u l=19u
m11 8 90 90 8 p w=4u l=21u
.ends

N63J SPICE LEVEL2 PARAMETERS
.MODEL N NMOS LEVEL=2 PHI=0.700000 TOX=3.9500E-08 XJ=0.200000U TPG=1
+ VTO=0.8577 DELTA=2.7790E+00 LD=1.3330E-07 KP=5.9272E-05
+ UO=678.0 UEXP=8.7500E-02 UCRIT=6.7780E+03 RSH=1.2930E+01
+ GAMMA=0.5361 NSUB=6.6170E+15 NFS=9.3830E+10 VMAX=4.8300E+04
+ LAMBDA=3.0840E-02 CGDO=1.7480E-10 CGSO=1.7480E-10
+ CGBO=3.4582E-10 CJ=1.34E-04 MJ=0.535 CJSW=6.11E-10
+ MJSW=0.200 PB=0.40
.MODEL P PMOS LEVEL=2 PHI=0.700000 TOX=3.9500E-08 XJ=0.200000U TPG=-1
+ VTO=-0.8721 DELTA=3.0970E+00 LD=1.0210E-07 KP=1.6129E-05
+ UO=184.5 UEXP=4.3620E-01 UCRIT=2.0720E+05 RSH=1.0190E-01
+ GAMMA=0.5972 NSUB=8.2120E+15 NFS=6.0720E+11 VMAX=9.9990E+05
+ LAMBDA=3.9420E-02 CGDO=1.3389E-10 CGSO=1.3389E-10
+ CGBO=4.0174E-10 CJ=3.34E-04 MJ=0.585 CJSW=3.97E-10
+ MJSW=0.127 PB=0.90
```

```
.probe v(18)
.dc v1 1.5 3.5 0.02 vbias 0.9 1.4 0.1
.end
```

B.5 Modified Gilbert Multiplier

```
* 1-D Vector Multiplier: ORBIT, N63J (gmul1.cir)
vdd 80 0 5.0
vss 90 0 0.0

v1 1 0
v2 2 0

vref 10 0 2.5
vbias1 11 0 1.5

xg1 80 90 1 10 2 10 11 5 gil1

.subckt gil1 80 90 1 2 3 4 5 7
xsg1 80 90 1 2 3 4 5 6 7 sgil
xre 80 90 6 7 linre1
.ends

.subckt sgil 80 90 1 2 3 4 5 13 14
m1 7 1 6 90 n w=4u l=4u
m2 8 2 6 90 n w=4u l=4u
m3 9 3 12 80 p w=15u l=4u
m4 9 4 11 80 p w=15u l=4u
m5 10 3 11 80 p w=15u l=4u
m6 10 4 12 80 p w=15u l=4u
m7 80 7 9 80 p w=15u l=4u
m8 80 7 7 80 p w=15u l=4u
m9 80 8 8 80 p w=15u l=4u
m10 80 8 10 80 p w=15u l=4u
m11 13 11 90 90 n w=4u l=4u
m12 11 11 90 90 n w=4u l=4u
m13 12 12 90 90 n w=4u l=4u
m14 14 12 90 90 n w=4u l=4u
m15 6 5 90 90 n w=12u l=4u
.ends

.subckt linre1 80 90 6 77
vdum 77 7 0.0
m31 80 7 7 80 p w=4u l=12u
m32 7 90 90 7 p w=4u l=12u

m21 80 6 6 80 p w=11u l=14u
m22 80 6 77 80 p w=11u l=14u
.ends

N63J SPICE LEVEL2 PARAMETERS
.MODEL N NMOS LEVEL=2 PHI=0.700000 TOX=3.9500E-08 XJ=0.200000U TPG=1
```

```
+ VTO=0.8577 DELTA=2.7790E+00 LD=1.3330E-07 KP=5.9272E-05
+ UO=678.0 UEXP=8.7500E-02 UCRIT=6.7780E+03 RSH=1.2930E+01
+ GAMMA=0.5361 NSUB=6.6170E+15 NFS=9.3830E+10 VMAX=4.8300E+04
+ LAMBDA=3.0840E-02 CGDO=1.7480E-10 CGSO=1.7480E-10
+ CGBO=3.4582E-10 CJ=1.34E-04 MJ=0.535 CJSW=6.11E-10
+ MJSW=0.200 PB=0.40
.MODEL P PMOS LEVEL=2 PHI=0.700000 TOX=3.9500E-08 XJ=0.200000U TPG=-1
+ VTO=-0.8721 DELTA=3.0970E+00 LD=1.0210E-07 KP=1.6129E-05
+ UO=184.5 UEXP=4.3620E-01 UCRIT=2.0720E+05 RSH=1.0190E-01
+ GAMMA=0.5972 NSUB=8.2120E+15 NFS=6.0720E+11 VMAX=9.9990E+05
+ LAMBDA=3.9420E-02 CGDO=1.3389E-10 CGSO=1.3389E-10
+ CGBO=4.0174E-10 CJ=3.34E-04 MJ=0.585 CJSW=3.97E-10
+ MJSW=0.127 PB=0.90

.probe v(5) i(xg1.xre.vdum) id(xg1.xre.m31) id(xg1.xre.m32)
.dc v1 1 4 0.02 v2 1.5 3.5 0.25
.end
```

B.6 3-D Vector Multiplier

```
* 3-D Vector Multiplier: ORBIT, N63J (vm3-20u.cir)
vdd 80 0 5.0
vss 90 0 0.0

v1 1 0
v2 2 0

vref 10 0 2.5
vbias1 11 0 1.1

xg1 80 90 1 10 2 10 11 1 10 2 10 11 1 10 2 10 11 5 gil3

.subckt gil3 80 90 1 2 3 4 5 11 12 13 14 15 21 22 23 24 25 7
xsg1 80 90 1 2 3 4 5 6 7 sgil
xsg2 80 90 11 12 13 14 15 6 7 sgil
xsg3 80 90 21 22 23 24 25 6 7 sgil
xre 80 90 6 7 linre3
.ends

.subckt sgil 80 90 1 2 3 4 5 13 14
m1 7 1 6 90 n w=4u l=4u
m2 8 2 6 90 n w=4u l=4u
m3 9 3 12 80 p w=15u l=4u
m4 9 4 11 80 p w=15u l=4u
m5 10 3 11 80 p w=15u l=4u
m6 10 4 12 80 p w=15u l=4u
m7 80 7 9 80 p w=15u l=4u
m8 80 7 7 80 p w=15u l=4u
m9 80 8 8 80 p w=15u l=4u
m10 80 8 10 80 p w=15u l=4u
m11 13 11 90 90 n w=4u l=4u
m12 11 11 90 90 n w=4u l=4u
m13 12 12 90 90 n w=4u l=4u
m14 14 12 90 90 n w=4u l=4u
m15 6 5 90 90 n w=12u l=4u
.ends

.subckt linre3 80 90 6 77
vdum 7 77 0.0
m31 80 7 7 80 p w=4u l=12u
m32 7 90 90 7 p w=4u l=12u

m21 80 6 6 80 p w=11u l=14u
m22 80 6 77 80 p w=11u l=14u
.ends
```

```
N63J SPICE LEVEL2 PARAMETERS
.MODEL N NMOS LEVEL=2 PHI=0.700000 TOX=3.9500E-08 XJ=0.200000U TPG=1
+ VTO=0.8577 DELTA=2.7790E+00 LD=1.3330E-07 KP=5.9272E-05
+ UO=678.0 UEXP=8.7500E-02 UCRIT=6.7780E+03 RSH=1.2930E+01
+ GAMMA=0.5361 NSUB=6.6170E+15 NFS=9.3830E+10 VMAX=4.8300E+04
+ LAMBDA=3.0840E-02 CGDO=1.7480E-10 CGSO=1.7480E-10
+ CGBO=3.4582E-10 CJ=1.34E-04 MJ=0.535 CJSW=6.11E-10
+ MJSW=0.200 PB=0.40
.MODEL P PMOS LEVEL=2 PHI=0.700000 TOX=3.9500E-08 XJ=0.200000U TPG=-1
+ VTO=-0.8721 DELTA=3.0970E+00 LD=1.0210E-07 KP=1.6129E-05
+ UO=184.5 UEXP=4.3620E-01 UCRIT=2.0720E+05 RSH=1.0190E-01
+ GAMMA=0.5972 NSUB=8.2120E+15 NFS=6.0720E+11 VMAX=9.9990E+05
+ LAMBDA=3.9420E-02 CGDO=1.3389E-10 CGSO=1.3389E-10
+ CGBO=4.0174E-10 CJ=3.34E-04 MJ=0.585 CJSW=3.97E-10
+ MJSW=0.127 PB=0.90

.probe
.dc v1 1.5 3.5 0.02 v2 1.5 3.5 0.25
.end
```

APPENDIX C

PSPICE simulation results with different temperature

C.1 Trajectory generation with 50^oC

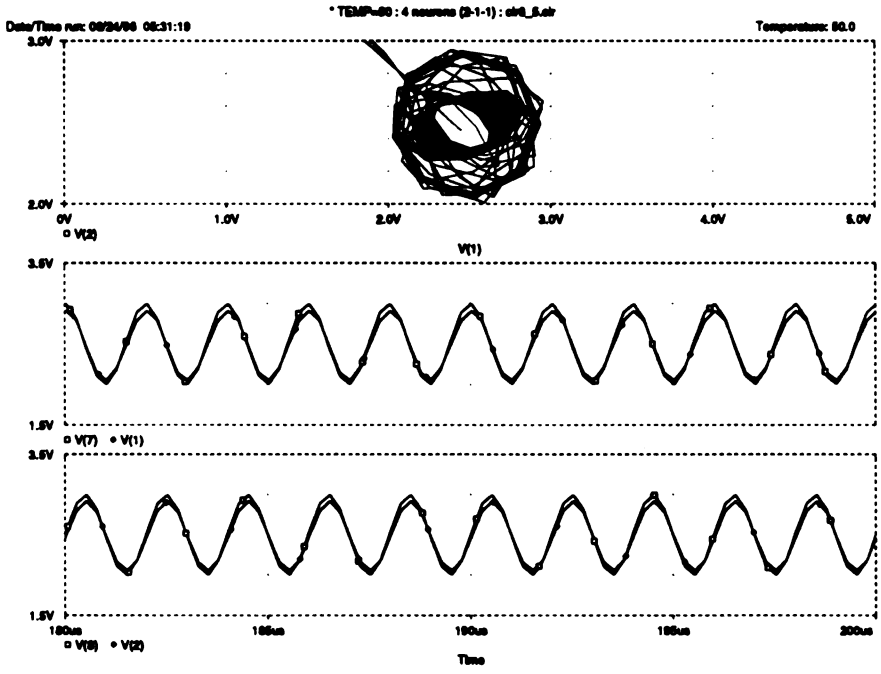

Figure C.1. Input, output and target signals

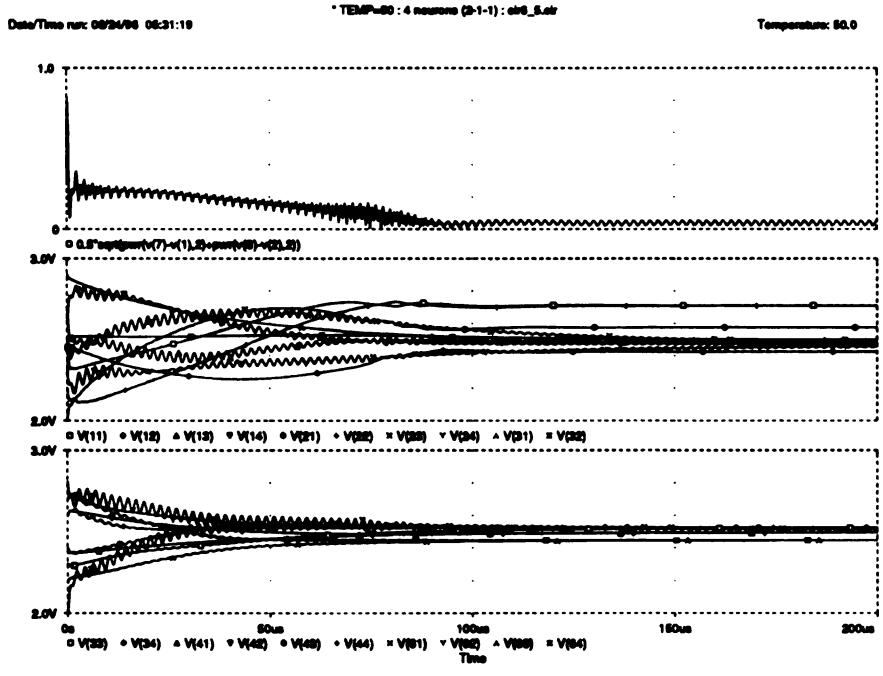

Figure C.2. E_{rms} and the weights

C.2 Trajectory generation with 0^oC

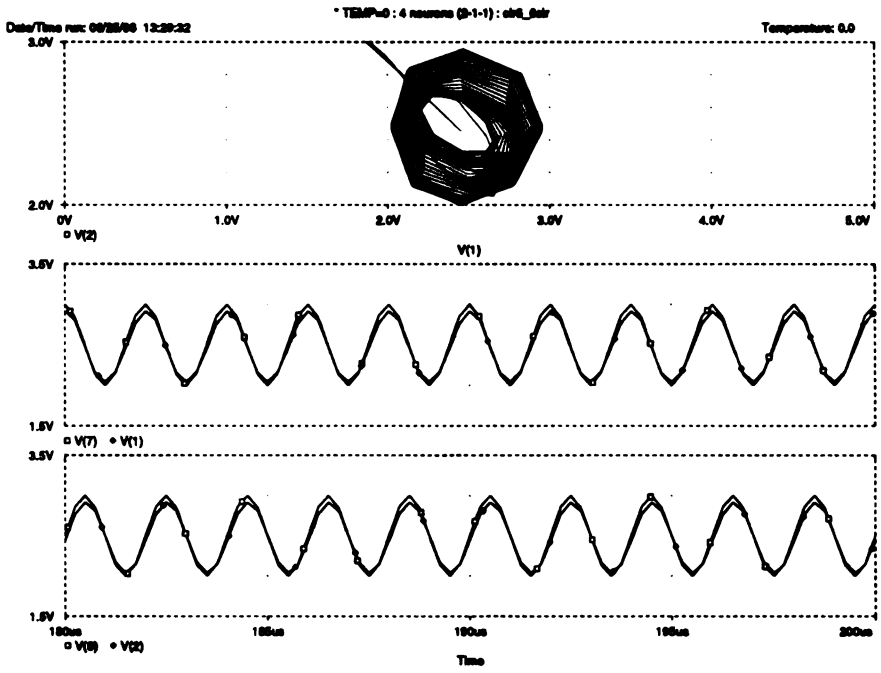

Figure C.3. Input, output and target signals

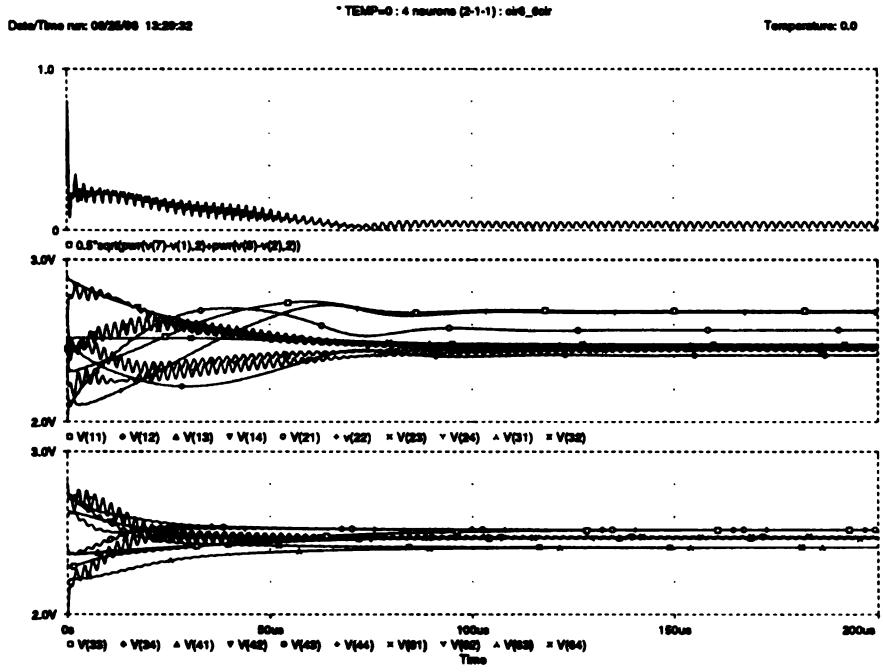

Figure C.4. E_{rms} and the weights

C.3 Trajectory generation with 125^oC and -50^oC

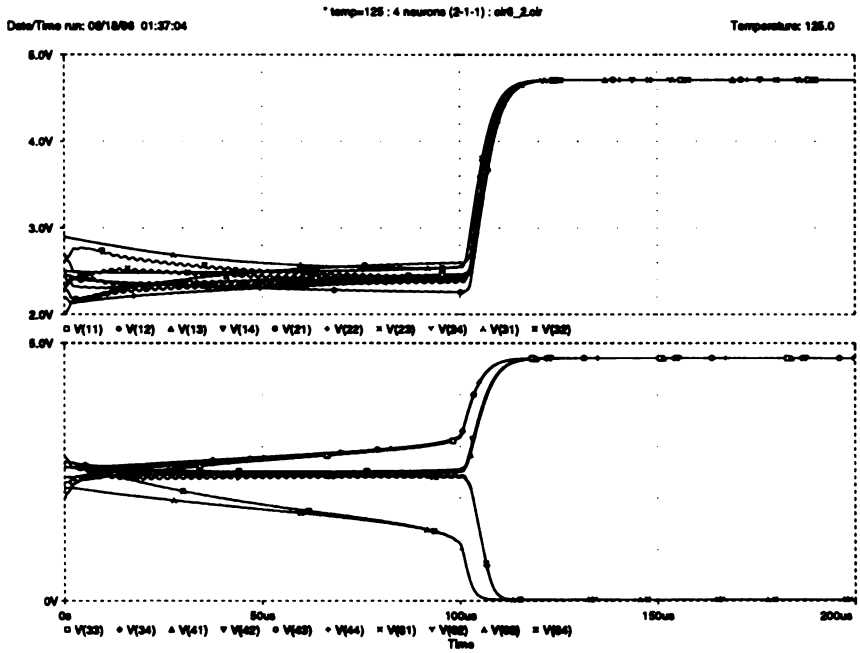

Figure C.5. Weights hit the rail voltage

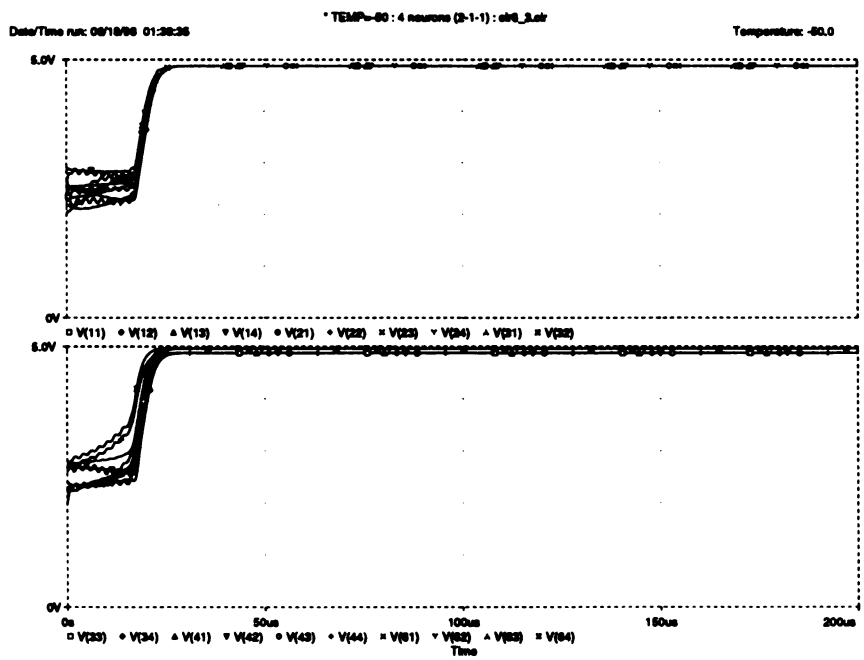

Figure C.6. Weghts hit the rail voltage

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] John Hertz, Anders Krogh, and Richard G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley Publishing Co., 1991.

[2] Judith E. Dayhoff, *Neural Network Architecture*, Van Nostrand Reinhold, New York, 1990.

[3] Patrick K. Simpson, *Artificial Neural Systems: Foundations, Paradigms, Applications, and Implementations*, Pergamon Press, New York, 1990.

[4] R. P. Lippmann, "An Introduction to Computing with Neural Nets", IEEE Acoustics, Speech and Signal Processing Magazine, 4(2), April 1987, pp. 4-22.

[5] S. Eberhardt, R. Tawel, T. Brown, T. Daud, and A. Thakoor, "Analog VLSI Neural Networks: Implementation Issues and Examples in Optimization and Supervised Learning", IEEE Transaction on Industrial Electronics, Vol. 39, No. 6, December 1992, pp. 552-564.

[6] Eric A. Vittoz, "Analog VLSI Signal Processing: Why, Where and How?", Analog Integrated Circuits and Signal Processing, July 1994, pp. 27-44.

[7] M. Maher, S. DeWeerth, M. Mahowald, and C. Mead, "Implementing Neural Architectures using Analog VLSI Circuits", IEEE Transaction on Circuits and Systems, Vol. 36, No. 5, May 1989, pp. 643-652.

[8] Caver Mead, "Adaptive Retina", in *Analog VLSI Implementation of Neural Networks*, C. Mead and M. Ismail, Eds. Boston: Kluwer Academic Publishers, 1989.

[9] R. F. Lyon and C. Mead, "An Analog Electronic Cochlea", IEEE Trans. Acoust. Speech Signal Proc., Vol. 36, No. 7, July 1988, pp.1119-1134.

[10] M. A. Holler, S. Tam, H. Castro, and R. Benson, "An Electrically Trainable Artificial Neural Network (ETANN) with 10240 Floating Gates Synapses", International Joint Conference on Neural Networks, 1989, Vol. 2, pp.191-196, Washington D.C.

[11] E. Sackinger, B. E. Boser, J. Bromley, Y. Le Cun, and L. D. Jackel, "Application of the ANNA Neural Network Chip to High-speed Character Recognition", IEEE Transactions on Neural Networks, Vol. 3, May 1992, pp.498-505.

[12] J. Choi, S. H. Bang, and B. J. Sheu, "A Programmable Analog VLSI Neural Network Processor for Communication Receiver", IEEE Transactions on Neural Networks, Vol. 4, May 1993, pp.484-495.

[13] Hwa-Joon Oh and Fathi M. Salam, "Analog CMOS Implementation of Neural Network for Adaptive Signal Processing", Proc. of IEEE International Symposium on Circuit and Systems 1994, London, England, May 30 - June 2, 1994, pp.503-506.

[14] Gert Cauwenberghs, "An Analog VLSI Recurrent Neural Network Learning a Continuous-Time Trajectory", IEEE Transactions on Neural Networks, Vol. 7, No. 2, March 1996, pp.346-361.

[15] Bart Kosko, Editor, *Neural Networks for Signal Processing*, Prentice-Hall, Inc., New Jersey, 1992.

[16] Barak A. Pearlmutter, "Gradient Calculations for Dynamic Recurrent Neural Network: A Survey", IEEE Trans. on Neural Networks, Vol. 6, No. 5, September 1995, pp.1212-1228.

[17] Charles F. Stevens, "The neuron", Scientific American, September 1979.

[18] Warren S. McCulloch and Walter Pitts, "A Logical Calculus of Ideas Immanent in Nervous Activity", Bulletin of Mathematical Biophysics 5, 1943, pp.115-133.

[19] Marvin L. Minsky and Seymour A. Papert, *Perceptrons*, Expanded edition, The MIT Press, 1990.

[20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, MIT Press, 1986, Vol. I.

[21] J. J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", Proc. Natl. Acad. Sci. U.S.A. Vol. 79, pp. 2554-2556, 1982.

[22] J. J. Hopfield, "Neurons with Graded Responses Have Collective Computational Properties Like Those of Two-State Neurons", Proceedings of the National Academy of Sciences, USA 81, pp.3088-3092.

[23] J. J. Hopfield and D. W. Tank, "Computing with Neural Circuits: A Model", Science 233, pp.625-633.

[24] Don R. Hush and Bill Horne, "Progress in Supervised Neural Networks: What's new since Lippman?", IEEE Signal Processing Magazine 10, 1993, pp. 8-39.

[25] Fathi M. Salam, "Learning Algorithms for Artificial Neural Nets for Analog Circuit Implementation", Computing Science and Statistics, Proc. of the 22nd Symposium on the Interface, May 16-19, 1990, pp.169-177.

[26] Fathi M. Salam, "A Modified Learning Rule for Feedforward Artificial Neural Nets for Analog Implementation", Memorandum No. MSU/EE/S 90/02, Department of Electrical Engineering, Michigan State University, Jan. 26 1990.

[27] Myung-Ryul Choi, *Implementation of Feedforward Artificial Neural Networks with Learning using Standard CMOS Technology*, Ph. D. Dissertation, Dept. of Electrical Engineering, Michigan State University, 1991.

[28] Hwa-Joon Oh and Fathi M. Salam, "A Modular Analog Chip for Feedforward Networks with On-Chip Learning", Proc. of IEEE 36th Midwest Symposium on Circuit and System, Detroit, Aug.1993. pp.766-769

[29] C. T. Sah, "Characteristics of the Metal-Oxide-Semiconductor Transistor", IEEE Trans. on Electron Devices, vol. ED-11, July 1964, pp.324-345.

[30] H. Shichman and D. Hodges, "Modeling and Simulation of Insulated-Gate Field Effect Transistor Switching Circuits", IEEE Journal of Solid State Circuits, Vol. SC-3, No. 3, Sep. 1968, pp.285-289.

[31] A. Paolo and M. Giuseppe, *Semiconductor Device Modeling with SPICE*, McGraw-Hill, Inc. 1988.

[32] Paul W. Tuinenga, *SPICE: A guide to circuit simulation and analysis using PSpice*, Prentice-Hall, New Jersey, 1988.

[33] J. E. Meyer, "MOS Models and Circuit Simulation", RCA Review, Vol. 32, 1971.

[34] L. M. Dang, "A Simple Current Model for Short Channel IGFET and Its Application to Circuit Simulation", IEEE Journal of Solid-State Circuits, Vol. 14(2), 1979.

[35] R. Kielkowski, *Inside SPICE: Overcoming the Obstacles of Circuit Simulation*, McGraw-Hill, 1994.

[36] Phillip E. Allen and Douglas R. Holberg, *CMOS Analog Circuit Design*, Holt, Rinehart and Winston (HRW), 1987.

[37] Randall L. Geiger, Phillip E. Allen, and Noel R. Strader, *VLSI Design Techniques for Analog and Digital Circuits*, McGraw-Hill Publishing Company, 1990.

[38] Caver Mead, *Analog VLSI and Neural Systems*, Addison-Wesley Publishing Company, 1989.

[39] M. Steyaert and W. Sansen, "High Performance Operational Amplifiers and Comparators", in *Analogue-Digital ASICs: Circuit techniques, design tools and applications*, edited by R. Soin, F. Maloberti and J. Franca, Chapter 3, pp.41-64., Peter Peregrinus Ltd. 1991.

[40] Barrie Gilbert, "A Precise Four-quadrant Multiplier with Subnanosecond Response", IEEE Journal of Solid-State Circuits, Vol. SC-3:365, 1968.

[41] S. Qin and R. Geiger, "A ±5-V CMOS Analog Multiplier", IEEE Journal of Solid-State Circuits, Vol. SC-22, No.6, December 1987, pp.1143-1146.

[42] J. N. Babanezhad and G. C. Temes, "A 20-V Four-quadrant CMOS Analog Multiplier", IEEE Journal of Solid-State Circuits, Vol. SC-20, No. 6, Dec. 1985, pp. 1158-1168

[43] K. Bult and H. Wallinga, "A CMOS Four-quadrant Analog multiplier", IEEE Journal of Solid-State Circuits, Vol. SC-21, No. 3, June 1986, pp. 430-435.

[44] J. Pena-Finol and J. A. Connelly, "A MOS Four-quadrant CMOS analog Multiplier using the Quarter-square Technique", IEEE Journal of Solid-State Circuits, Vol. SC-22, No. 6, Dec. 1987, pp.1064-1073.

[45] D. C. Soo and R. G. Meyer, "A Four-quadrant NMOS Analog Multiplier", IEEE Journal of Solid-State Circuits, Vol. SC-17, No. 6, Dec. 1982, pp.1174-1178.

[46] C. W. Kim and S. B. Park, "New Four-quadrant CMOS Analog Multiplier", Electron Letters, Vol. 23, No. 24, Nov. 1987, pp. 1268-1270.

[47] Hwa-Joon Oh and Fathi M. Salam, "4x4x2 Neural network Design Using Modular Neural Chips with On-chip Learning", Proc. of IEEE International Conference on Neural Networks (ICNN), Orlando, Jun.28-Jul.2, 1994, pp.2070-2073.

[48] Simon Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan College Publishing Company, Inc., 1994.

[49] Jacek M. Zurada, *Introduction to Artificial Neural Systems*, West Publishing Company, 1992.

[50] Barak A. Pearlmutter, "Learning State Space Trajectories in Recurrent Neural Networks", Proc. of International Joint Conference on Neural Networks (IJCNN) 1989, June 18-22, 1989, Vol. II, pp.365-372.

[51] D. Kirk, *Optimal control theory: An introduction*, Englewood Cliffs, NJ: Prentice-Hall, 1970.

[52] Frank L. Lewis, *Optimal Control*, John Wiley & Sons, NY, 1986.

[53] Luis G. Sotelino, Marco Saerens, and Hugues Bersini, "Classification of Temporal Trajectories by Continuous-Time Recurrent Nets", Neural Networks, Vol. 7, No. 5, pp.767-776, 1994.

[54] Fernando J. Pineda, "Generalization of Back-Propagation to Recurrent Neural Networks", Physical Review Letters, Vol. 59, Number 19, 9 Nov., 1987, pp.2229-2232.

[55] R. J. Williams and D. Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Neural Networks", Neural Computation 1, 1989, pp.270-280.

[56] L. B. Almeida, "A Learning Rule for Asynchronous Perceptrons with Feedback in a Combinatorial Environment", In IEEE First International Conference on Neural Networks (San Diego 1987), Eds. M. Caudill and C. Butler, Vol. II, pp.609-618.

[57] Kiyotoshi Matsuoka, "Stability Conditions for Nonlinear Continuous Neural Networks with Asymmetric Connection Weights", Neural Networks, Vol. 5, pp. 495-500, 1992.

[58] L. Jin, P. N. Nikiforuk, and M. M. Gupta, "Absolute Stability Conditions for Discrete-time Recurrent Neural Networks", IEEE Trans. on Neural Networks, Vol. 5, No. 6, November 1994, pp.954-964.

[59] L. B. Almeida, "Backpropagation in Non-Feedforward Networks", in I. Aleksander (Ed.), *Neural Computing Architectures*, Cambridge, MA: MIT Press, 1989, pp.75-91.

[60] L. Ljung, "Issue in system identification", IEEE Control System Magazine, Vol. 11, Jan. 1991, pp.25-29.

[61] Hwa-Joon Oh and Fathi Salam, "Analog CMOS Feedforward Artificial Neural Network with On-Chip Learning: Test Results", Proc. of 1993 International Symposium on Nonlinear Theory and Its Applications, Hawaii, Dec. 5 - 10, 1993.

[62] Fathi Salam and Hwa-Joon Oh, "Real-Time Tracking Control Using Modular Neural Chips with On-Chip Learning", Proc. of IEEE International Conference on Neural Networks (ICNN), 1996, June 2-6, Washington D.C.