Analysis and Implementation of Continuous-Time Feedback Neural Networks with Multiple-State Neurons

by

Bo Ling

A DISSERTATION

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

DOCTOR OF PHILOSOPHY

Department of Electrical Engineering
1993

ABSTRACT

Analysis and Implementation of Continuous-Time Feedback Neural Networks with Multiple-State Neurons

by

Bo Ling

This work has focused on the design and analysis of two types of continuous-time feedback neural networks, namely, the Hopfield-type feedback neural network and the dendro-dendritic artificial neural network (DANN). The Hopfield-type feedback neural networks with two-state neurons are widely used in various applications, such as pattern recognition, optimization, etc. The more recent DANN architecture has been used in pattern association and as an edge detector. All of the networks considered in this dissertation have multiple-state neurons.

For the Hopfield-type continuous-time feedback neural network with multiple-state neurons, we have proposed an analytical recursive algorithm to find a set of weights which will exactly store the desired gray-level patterns as asymptotically stable equilibria. We introduce a criterion which will ensure that the designed Hopfield-type neural network to be *implementable* as an electronic circuit. The criterion, moreover, allows the resistances in the network to be made as large as desired. This flexibility has the practically appealing consequence of using very large resistances and hence considerably reducing the power consumption in the circuit implementation. We also give explicit bounds on the variation of equilibria due to the variation in weights.

A cellular structure of the Hopfield-type feedback neural networks is also considered. We propose a large network structure which connects two or more smaller identical networks (or cells) together. A parallel recursive algorithm for this type of locally connected neural network is introduced. This parallel algorithm finds the weights for all cells simultaneously.

For the DANN architecture, we show that multiple patterns can be stored as asymptotically stable equilibria with only positive weights. We further formulate the problem of finding all positive weights as a standard linear programming problem. Thus, based on the given patterns, all positive weights can be easily determined. We also point out that the all positive weights can be solved via a nonlinear (e.g. quadratic) programming approach.

We have designed a CMOS circuit for a multiple-state neuron which operates in subthreshold. With this neuron, the network power dissipation is considerably reduced. As a prototype, a DANN architecture using 6 neurons has been designed. Each neuron has four states. Numerous PSpice simulations have shown that this prototype DANN network works quite well. A Tiny-Chip using 6-neuron DANN architecture has been designed and will be fabricated via MOSIS. All internal signals can be accessed through the pins of this chip for thorough laboratory testing.

ACKNOWLEDGMENTS

The author would like to express sincere thanks to Professor Fathi M.A. Salam for his inspiration and guidance. Thanks to Professor H.K. Khalil, Professor J. Deller, and Professor C.R. MacCluer for their comments and suggestions.

Table of Contents

List of Figures

CHAPTER 1

Introduction

Many models for the neural networks have been proposed since the work of McCulloch and Pitts [37] in 1943. Those models are often referred to as artificial neural networks. The Hopfield-type continuous-time feedback neural network [21] is one of several popular neural networks. In this network, the input of each neuron is the sum of the output signals of other neurons processed through proper weights. As proposed in [21], there is no self-feedback of the output of each individual neuron. One key issue in the design of the Hopfield-type feedback neural network is to find the symmetric weight matrix based on desired multiple patterns. The symmetry of the weight matrix is important in the design of the usual Hopfield-type feedback continuous-time neural network. Because of its structure simplicity, the Hopfield-type continuous-time feedback neural network has been the object of numerous proposed implemented using VLSI technology.

Another interesting type of continuous-time feedback neural network is the Dendrodendritic Artificial Neural Network (DANN) [48, 49]. In this network, all neurons are connected by either resistors or nMOS transistors which are controlled by their gate voltages [49]. The input and output of each individual neuron is also connected by a feedback nMOS transistor. The gate voltages of all those transistors can be set globally. It has been shown [30, 48, 49] that multiple patterns can be stored as asymptotically stable equilibria of a DANN with only positive weights, which is a very promising feature of DANNs. In the Hopfield-type feedback neural network, weights could be negative in order to store certain given patterns. Thus extra inverters are usually needed to implement those negative weights [21], which in turn would greatly increase the number of transistors in the network. In DANN, because of the positive weights, all weights can be realized simply by

resistors or MOS transistors. Consequently, the network implementation is more compact [47]. Moreover, compared with the Hopfield-type feedback neural network, the DANN network has two other major advantages: (1) It requires potentially fewer weights; (2) It has symmetric weights; (3) All weights can be nonlinear resistive elements. The realization of symmetric weights is a bottleneck in the implementation of Hopfield-type feedback neural network.

Many theoretical results on the Hopfield-type feedback neural network have been reported in the literature (e.g. [46], [26], [14]). Although those results have established the foundation of the Hopfield-type feedback neural network, there are still important unsolved problems in the design of this type of neural network. In parts of this thesis, we will propose solutions to some of those fundamental problems.

One important problem is the analytical algorithm for finding symmetric weights. Michel and his co-workers have reported several results on this problem. In [26], a design method is proposed for finding a symmetric weight matrix given analog (i.e. real number) patterns, but this method does not ensure the existence of the weight matrix. In [14], another method is proposed for finding a weight matrix which is not symmetric. The symmetry of the weight matrix, however, is important in the design of Hopfield-type feedback continuous-time neural network. It ensures the global stability of the designed network and the convergence to equilibria only.

For a given structure of a Hopfield-type feedback neural network, there is a capacity associated with it which limits the number of desired patterns that can possibly be stored. Yet, even though the network capacity may not be reached, some patterns may not be successfully stored. These "unstorable" patterns may cause all other patterns not to be stored since there may not exist a symmetric weight matrix which makes the given set of patterns

stored together. Thus, it appears to be practically useful to separate those "unstorable" patterns from the given set of patterns. In this case, one can still find a symmetric weight matrix which will make part of a set of desired patterns to be stored as equilibria.

A stored pattern will be retrieved by other key patterns only if it is an asymptotically stable equilibrium of the network. It is known [46] that a pattern, once stored, is asymptotically stable if it corresponds to values within very "flat" regions of the sigmoidal function. In such a "flat" region, the slope of the sigmoidal function is very small, but we don't know quantitatively how small the slope should be. It would be helpful if we knew the upper bound on those small slopes. With this upper bound, we would be able to design multiple-state neurons which will make the network capable of processing multiple-level patterns. This upper bound would be a useful guide in designing a (simple) CMOS multiple-state neuron circuit [32].

The analog VLSI design of the multiple-state neuron circuit is an interesting problem. With multiple-state neurons, the neural network may process gray-level images instead of binary-value images. In [60], a multiple-state neuron circuit is designed. This circuit operates above threshold which would cause a relatively large power dissipation in the network. To reduce the power dissipation to the micro watt level, it became a goal of this research to design a simple neuron circuit operating in sub-threshold. Moreover, because of the limited chip area, a simple multiple-state neuron circuit is desired.

Not all the (theoretically) designed Hopfield-type feedback neural networks can be implemented as electronic circuits. One has to ensure that the theoretical design maps the resistance to nonnegative values. Therefore, one needs to ensure a designed network to be implementable in the form of a circuit. In [14], a successful synthesis is proposed to resolve this problem, but, based on this approach, the resistance in the corresponding neu-

ral circuits may be small. This in turn would cause large network power dissipation. We must find a new method which ensures that the designed Hopfield-type feedback neural network is implementable and the resistors in the network are sufficiently very large.

In the Hopfield-type feedback neural network, the designed weights are realized by hardware devices. Due to the limited hardware precision, the implemented weights will not match the designed weights exactly. Thus, the stored pattern in the neural circuit will not be the desired pattern. Since the weights are realized by certain circuit devices, we can estimate their variation. Thus, it would be helpful to estimate the corresponding variation of equilibria due to the variation of weights. That is, we desire to quantify the sensitivity of the equilibria to the variations in weights. With the information on the variation of equilibria, we are able to predict the variation of the stored patterns of the implemented neural networks. This would enable us to quantify the robustness of the equilibria to variations in weights.

In real applications such as pattern recognition, a large number of neurons may be required. However, due to the limited chip area, the number of neurons implemented on a single chip can be quite limited. Therefore, we need to find a way to connect small networks (or chips) together to form a larger network. We may label such networks cellular neural chips (CNC). In this larger network, each small network (chip) performs a certain task. To build this larger network, we must find the connection structure of small cells, and a parallel algorithm which makes all cells find their own weights simultaneously. This parallel algorithm is important because it ensures the "programming" time of the large network will not increase significantly as its size becomes larger.

In this dissertation, based on multiple desired patterns, we introduce a new recursive algorithm to find weights for the Hopfield-type continuous-time feedback neural networks

with two-state or multiple-state neurons [27]. We propose a criterion which will ensure the designed Hopfield-type continuous-time feedback neural network with two-state neurons to be implementable [29]. With our approach, the resistance in the network can be made as large as necessary. We also estimate the variation of the equilibria based on the variation of weights [28]. The cellular Hopfield-type feedback neural network chips is introduced in [27]. A parallel algorithm for all cells to find their respective weights simultaneously is also provided, too. We design a CMOS VLSI multiple-state neuron circuit operating in sub-threshold [32].

For the DANN network, we formulate the problem of finding positive weights as a standard linear programming problem [30]. As a prototype, a DANN with six neurons has been designed and implemented . Each neuron has four states. PSpice simulation of this prototype DANN network has shown it works well. We also design the circuit layout of this six-neuron DANN network. A Tiny-Chip of this six-neuron DANN network has been designed and will be fabricated via MOSIS.

This dissertation contains six chapters. In Chapter 1, we have briefly described the research problems addressed. In Chapter 2, we focus on the Hopfield-type continuous-time feedback neural network. The analytical recursive algorithm for finding weights that store a given set of patterns is proposed. A cellular Hopfield-type feedback neural network is discussed in Chapter 3. In Chapter 4, we formulate the problem of finding positive weights for a DANN as a standard linear programming problem. In Chapter 5, A Tiny-Chip of a 6-neuron DANN network has been designed. Finally, in Chapter 6, summary and some final remarks are presented.

CHAPTER 2

Continuous-Time Feedback Neural Networks with Multiple-State Neurons

2.1 Background

The mathematical model for the Hopfield-type feedback neural network [21] is

$$c_i \frac{du_i}{dt} = \sum_{j=1}^{n} w_{ij} v_j + I_i - \frac{1}{r_i} u_i \tag{2.1.1i}$$

$$v_i = s_i(u_i) \tag{2.1.1ii}$$

for i = 1, 2, ..., n, where $c_i, r_i > 0$, $u_i \in R$ and $v_i \in (-\Delta, \Delta)$ with $0 < \Delta \in R$. $s_i : R \rightarrow (-\Delta, \Delta)$ is a monotone increasing C^1-function with $s_i(0) = 0$, $s_i(x) = \Delta$ (or $-\Delta$) if and only if $x \rightarrow +\infty$ (or $-\infty$). n denotes the number of neurons in the network (2.1.1). I_i is the external input. For simplicity, assume $I_i = 0$ for all i. w_{ij} are the weights, for $i, j = 1, ..., n$.

In compact matrix form, (2.1.1) can be written as

$$\boldsymbol{C}\frac{d\boldsymbol{u}}{dt} = \boldsymbol{W}\boldsymbol{v} - \boldsymbol{R}^{-1}\boldsymbol{u} \tag{2.1.2i}$$

$$\boldsymbol{v} = \boldsymbol{s}(\boldsymbol{u}) \tag{2.1.2ii}$$

where $\boldsymbol{u} = [u_1, ..., u_n]^T \in R^n$, $\boldsymbol{v} = [v_1, ..., v_n]^T \in R^n$, $\boldsymbol{C} = \text{diag}(c_1, ..., c_n) \in R^{n \times n}$, $\boldsymbol{R} = \text{diag}(r_1, ..., r_n) \in R^{n \times n}$, and $\boldsymbol{s}(\boldsymbol{u}) = [s_1(u_1), ..., s_n(u_n)]^T \in R^n$, and T denotes the transpose of a matrix.

The Hopfield energy function [21] is given as follows:

$$E(v) = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}v_i v_j + \sum_{i=1}^{n}\frac{1}{r_i}\int_0^{v_i} s_i^{-1}(x)\,dx \tag{2.1.3}$$

Many theoretical results on the Hopfield-type neural network have been reported in the literature (e.g. [46], [26], [14]). Those results have established the foundation of the Hopfield-type neural network. Some of the most important properties are:

(1) For the energy function defined in (2.1.3), for all $v \in (-\Delta, \Delta)^n$, $dE(v)/dt \le 0$, and $dE(v)/dt = 0$ if and only if v is an equilibrium point of (2.1.2).

(2) For any $v \in (-\Delta, \Delta)^n$, there exists a unique solution of (2.1.2) for $t \ge 0$.

(3) All solutions are bounded.

(4) An isolated local minimum of $E(v)$ is an asymptotically stable equilibrium of (2.1.2).

(5) There are only finite number of equilibria of (2.1.2).

(6) Let v^0 be an equilibrium point of (2.1.2). v^0 is a local minimum of $E(v)$ if and only if the Hessian matrix $J_{\nabla E}(v^0)$ is positive definite.

(7) Let v^0 be an equilibrium point of (2.1.2). v^0 is isolated if $\det(J_{\nabla E}(v^0)) \neq 0$.

(8) No limit cycles exit.

Those properties are related to the existence of the solutions and the stability of equilibria of (2.1.2). Since any desired patterns will be stored as stable equilibria, it is essential for them to be asymptotically stable.

2.2 Stability Analysis

Suppose $v^1, v^2, ..., v^m$ are m given patterns. Here, we may assume the weight matrix W is known. In the next two sections, we will introduce an analytical method to find this weight matrix. In order to retrieve v^i, $i = 1, 2, ..., m$, we must make v^i asymptotically stable.

For the Hopfield-type neural network described in (2.1.2), it is known that the isolated local minimum points of the energy function (2.1.3) are asymptotically stable. It can be observed that a stored pattern can be either local maximum or local minimum. In other words, an equilibrium might not be stable.

It is also known that a pattern, v, is a local minimum of energy function (2.1.3) if and only if the Hessian matrix $J_{\nabla E}(v)$ is positive definite. And, v is an isolated equilibrium if $\det(J_{\nabla E}(v)) \neq 0$. Thus, if we can find a condition which guarantees $J_{\nabla E}(v) > 0$, we know that v is asymptotically stable. It can be easily shown that

$$J_{\nabla E}(v) = - W + R^{-1}J_h(v) \tag{2.2.1}$$

where $h = s^{-1}$, $J_h(v) = \mathrm{diag}(h_1'(v_1), ..., h_n'(v_n))$, and W is the weight matrix for all of those m patterns, $v^1, v^2, ..., v^m$. Define $B = W - R^{-1}J_h(v)$. Then $J_{\nabla E}(v)$ is positive definite if and only if B is negative definite. The elements of B satisfy:

$$b_{ij} = w_{ij} \qquad \text{for } i, j = 1, 2, ..., n, i \neq j; \tag{2.2.2i}$$

$$b_{ii} = -r_i^{-1}h_i'(v_i) \qquad \text{for } i = 1, 2, ..., n. \tag{2.2.2ii}$$

Stability Criterion ([14], [46]):

Suppose $v = [v_1, ..., v_n]^T$ is an equilibrium point of network (2.1.2). Then v is

asymptotically stable if

$$h'_i(v_i) > r_i \sum_{j=1, j\neq i}^{n} |w_{ij}| \quad , \qquad \text{for } i = 1, ..., n. \tag{2.2.3}$$

Proof: Suppose (2.2.3) is true. Then

$$|b_{ii}| > \sum_{j=1, j\neq i}^{n} |b_{ij}| \quad , \qquad \text{for } i = 1, ..., n.$$

By *Gershgorin*'s eigenvalue estimation theorem, the matrix $\boldsymbol{B}$ has all negative eigenvalues. $\boldsymbol{B}$ is negative definite because of its symmetry. This completes the proof. ■

Let $u_i = h_i(v_i)$. Since $h_i = s_i^{-1}$, the inequality (2.2.3) is equivalent to

$$s'_i(u_i) < \frac{1}{r_i \sum_{j=1, j\neq i}^{n} |w_{ij}|} \quad , \qquad \text{for } i = 1, ..., n. \tag{2.2.4}$$

For given m patterns, $v^1, v^2, ..., v^m$, we have a recursive algorithm (introduced in Section 2.4) to find the corresponding weight matrix $\boldsymbol{W}$ which only depends on the values of the sigmoidal functions evaluated at those m patterns, and the values of those m patterns themselves. The values of the derivatives of sigmoidal functions at those patterns do not contribute to the calculation of weight matrix $\boldsymbol{W}$. In other words, the weight matrix $\boldsymbol{W}$ will not be changed if we vary the derivatives of the sigmoidal functions at those m patterns, v^1, $v^2, ..., v^m$. The stability criterion stated above actually put the requirement only on the derivatives of sigmoidal functions at the m patterns. This makes it possible to design a Hopfield-

type neural network in two steps:

Step 1: Find the weight matrix W for m patterns, $v^1, v^2, ..., v^m$;

Step 2: Adjust the derivatives of sigmoidal functions based on (2.2.4) to ensure the stability of those m patterns.

In reality, v is continuously evaluated in $(-\Delta, \Delta)^n$. The stability criterion requires that $s_i'(u_i)$ be less than some number. This is usually impossible. For example, $s_i'(u_i)$ would be very large if u_i is close to the origin. But, in most neural applications, the given patterns are binary, i.e., their elements take two different values, such as +1 and -1. And the sigmoidal functions of all neurons are chosen to be identical, i.e., $s_1 = ... = s_n = s_0$. Thus, for m patterns, $v^k = [v^k_1, ..., v^k_n]^T$ for $k = 1, 2, ..., m$, $v^k_i = +\upsilon$ or $-\upsilon$ for some $\upsilon > 0$, i = 1, ..., n. Define $\mu = s_0^{-1}(\upsilon)$. Thus, we have $u^k_i = +\mu$ or $-\mu$. For the binary input patterns, we have the following simple sufficient stability criterion:

Stability Criterion for Binary Pattern:

Suppose $v = [v_1, ..., v_n]^T$ with $v_i = +\upsilon$ or $-\upsilon$ is an equilibrium of network (2.1.2). Define $\mu = s_0^{-1}(\upsilon)$. Then v is asymptotically stable if

$$s_0'(\mu) < \frac{1}{\underset{1 \le i \le n}{\text{Max}} \{ r_i \sum_{j=1, j \neq i}^{n} |w_{ij}| \}} \tag{2.2.5}$$

In general, the upper bound of (2.2.5) is quite small. The above criterion actually requires the derivative of s_0 at $+\mu$ or $-\mu$ be bounded above by some positive number, denoted by ω, which depends on the given pattern, v. Since ω is pattern-dependent and the pattern length is n, there are only 2^n possible binary patterns. It is very easy to find a ω_0

such that any possible binary pattern, once stored, is asymptotically stable if $s_0'(\mu) < \omega_0$.

For m patterns, $v^1, v^2, ..., v^m$, we can find a weight matrix $\boldsymbol{W}$ such that these m patterns can be stored. Let SV_i, $i = 1, 2, ..., C(2^n,m)$, be a set of m possible binary patterns. For each pattern set, SV_i, we can exactly evaluate the upper bound ω_i. Define $\omega_0 = \text{Min}\{\omega_i, i = 1, 2, ..., C(2^n,m)\}$. If we design the sigmoidal function s_0 such that $s_0'(\mu) < \omega_0$, then any pattern in any possible pattern set, SV_i, $i = 1, 2, ..., C(2^n,m)$, will be stored as a locally asymptotically stable equilibrium of the Hopfield-type neural network (2.1.2).

If we further assume all elements of $\boldsymbol{R}$ are identical, i.e., $r_i = r$ for $i = 1, 2, ..., n$. Then (2.2.1) can be simplified as

$$\boldsymbol{J}_{\nabla E}(v) = - \boldsymbol{W} + r^{-1}h'(\mu)\boldsymbol{I} \tag{2.2.6}$$

Since $\boldsymbol{W}$ is symmetric, there exists orthogonal matrix $\boldsymbol{P}$ such that $\boldsymbol{W} = \boldsymbol{P}\Lambda\boldsymbol{P}^T$, where $\Lambda = \text{diag}(\lambda_1, \lambda_2, ..., \lambda_n)$, and λ_i for $i = 1, 2, ..., n$ are the eigenvalues of $\boldsymbol{W}$.

Theorem 2.2.1: ([46])

Suppose $v = [v_1, ..., v_n]^T$ with $v_i = +\upsilon$ or $-\upsilon$ is an equilibrium of network (2.1.2), and all elements of $\boldsymbol{R}$ are identical. Define $\mu = s_0^{-1}(\upsilon)$. Then $\boldsymbol{J}_{\nabla E}(v)$ is positive definite if $s_0'(\mu) < 1/(r\lambda_{max})$, where λ_{max} is the maximal modulus eigenvalue of $\boldsymbol{W}$.

2.3 An Analytical Method for Finding Weights

First, let us consider the single pattern case. Let $v = [v_1, ..., v_n]^T$ be a given pattern, where n denotes the length of the pattern, superscript T indicating the transpose of a matrix. In the associative memory network design, we wish to find the weight matrix W in network (2.2) such that the given pattern v will be stored as an equilibrium. Moreover, we desire v to be asymptotically stable such that it can be retrieved by some patterns other than v.

Mathematically, in order to be an equilibrium of network (2.1.2), v must satisfy

$$Wv = R^{-1}u \tag{2.3.1}$$

where W is the weight matrix which is symmetric and whose diagonal elements are all zeros; $R = \text{diag}(r_1, ..., r_n)$ with the resistor r_i connected to the *ith* neuron; $v = s(u)$, s is the sigmoidal function which is a strictly increasing C^1 function. Let $h = s^{-1}$. Obviously, h is also a strictly increasing function. Since $u = h(v)$, equation (2.3.1) can be rewritten as follows:

$$Wv = R^{-1}h(v). \tag{2.3.2}$$

In equation (2.3.2), only the elements of W are unknown. Recall that the elements of W satisfy [21]:

(1) $w_{ij} = w_{ji}$ for $i, j = 1, ..., n, i \neq j$ (*symmetry*);

(2) $w_{ii} = 0$ for $i = 1, ..., n$ (*zero diagonal elements*).

Therefore, among n^2 elements of W, there are only $(n^2 - n)/2$ elements needed to be deter-

mined. Let us denote this number by n_w:

$$n_w = \frac{1}{2}(n^2 - n) \tag{2.3.3}$$

This number, n_w, represents the total number of unknowns in (2.3.2).

The diagonal elements of the weight matrix can be non-zero. If we use non-zero diagonal elements, then the number of unknowns would be $(n^2 + n)/2$, instead of $(n^2 - n)/2$. If the number of neurons, n, is small, the non-zero diagonal elements will help to increase the capacity of network which limits the number of desired patterns to be potentially stored. If n is large, the non-zero diagonal elements will not significantly affect the neural network performance since $n_w >> n$. In this case, we would rather set them to be zero, which, in turn, reduces the number of transistors in the neural circuit implementation.

By proper algebraic operations, we can transfer the matrix equation (2.3.2) to the following equivalent matrix equation:

$$\boldsymbol{A}x = \boldsymbol{R}^{-1}h(v) \tag{2.3.4}$$

where

$$\boldsymbol{A} = \left[\begin{array}{c|c|c|c|c} \begin{matrix} v_2 & \cdots & v_n \\ \hline \multicolumn{3}{c}{v_1 \boldsymbol{I}_{n-1}} \end{matrix} & \begin{matrix} \boldsymbol{O}_{1\times(n-2)} \\ v_3 \quad \cdots \quad v_n \\ \hline \multicolumn{3}{c}{v_2 \boldsymbol{I}_{n-2}} \end{matrix} & \begin{matrix} \boldsymbol{O}_{2\times(n-3)} \\ v_4 \quad \cdots \quad v_n \\ \hline \multicolumn{3}{c}{v_3 \boldsymbol{I}_{n-3}} \end{matrix} & \cdots\cdots & \begin{matrix} \boldsymbol{O}_{(n-2)\times 1} \\ v_n \\ v_{n-1} \end{matrix} \end{array}\right]_{n\times n_w}$$

$$x = [x_1, ..., x_{nw}]^T$$

with

$$x_i = \begin{cases} w_{1,i+1} & 1 \le i \le n\text{-}1 \\ w_{2,i+2} & n \le i \le 2\,n\text{-}2 \\ \vdots & \vdots \\ w_{n-2,i+n-2} & n_w\text{-}2 \le i \le n_w\text{-}1 \\ w_{n-1,n} & i = n_w\,. \end{cases}$$

Notice that $n_w > n$ if $n > 3$. So, in general, matrix A has more columns than rows. It can also be seen that A is entirely determined by the given pattern v, i.e., A can be completely constructed once a pattern is provided. If we add non-zero diagonal elements of the weight matrix, the expression of the matrix A and vector x would be different, but their expression can be easily derived.

Let us denote $b = R^{-1}h(v)$ which is a constant vector for the given pattern, v. Thus, equation (2.3.4) can be rewritten as

$$Ax = b \tag{2.3.5}$$

In this matrix equation, there are n_w unknowns in n equations. Thus, we expect the difficulties in finding its solutions. Before continuing, let us investigate some properties of the matrix A which will be useful for helping us find solutions of (2.3.5).

Lemma 2.3.1:

Suppose $v = [v_1, ..., v_n]^T$ is a given pattern. A is defined in (2.3.4). If $v_i \neq 0$ for $i = 1$,

..., n, rank(A) = n.

Proof: Suppose $v_i \neq 0$ for all i. From the structure of A, we see that each column of A has only two non-zero elements because $v_i \neq 0$ for all i. Let a_i be the *ith* column of A, $i = 1, ..., n_w$. Then $a_1, a_2, ..., a_n$ are n linearly independent vectors, which implies rank(A) = n. This completes our proof. ■

Notice that AA^T is non-singular; in other words, A is of full row rank. In general, (2.3.5) may have more then one solution. If there exist solutions, they can be expressed in terms of the *Moore-Penrose* pseudo-inverse of a matrix [6]. The following theorem deals with the existence of solutions of (2.3.5):

Theorem 2.3.2 (Existence Theorem):

Suppose $v = [v_1, ..., v_n]^T$ is a given pattern. A is defined in (2.3.4). Then equation (2.3.5) has solutions if $v_i \neq 0$ for $i = 1, ..., n$.

Proof: Suppose $v_i \neq 0$ for all i. By Lemma 2.3.1, rank(A) = n, which implies $b \in R(A)$ where $R(A)$ denotes the range of A. Therefore, $Ax = b$ is consistent. It follows that (2.3.5) has solution(s). This completes the proof. ■

Theorem 3.2 actually guarantees that any continuous-valued single pattern with non-zero elements will be certainly stored, but, as we will show later, multiple patterns might not be stored all together in sequence.

We must note that Theorem 2.3.2 only states the existence of solutions of (2.3.5). But, in general, the solution is not unique. As a matter of fact, the general solution set of (2.3.5) can be expressed as $\{A^+b + N(A)\}$ ($N(A)$ represents the null space of A) which is a linear

manifold in R^{nw}. Therefore,

$$x = A^{+}b + y \tag{2.3.6}$$

is a solution of (2.3.5), where $A^{+} = A^{T}(AA^{T})^{-1}$, $y \in N(A)$. In this work, we choose

$$x = A^{+}b \tag{2.3.7}$$

as the solution of (2.3.5), which is the minimum-norm solution. Once x is solved in (2.3.7), the weight matrix W can be easily reconstructed as follows:

$$W = \begin{bmatrix} 0 & x_1 & x_2 & \cdots\cdots & x_{n-2} & x_{n-1} \\ x_1 & 0 & x_n & \cdots\cdots & x_{2n-3} & x_{2n-2} \\ & & \cdots\cdots & & & \\ x_{n-1} & x_{2n-2} & x_{3n-3} & \cdots\cdots & x_{nw} & 0 \end{bmatrix} \tag{2.3.8}$$

This W is the weight matrix of network (2.1.2) for the given pattern v which will make v exactly stored as an equilibrium of (2.1.2).

Now, let us consider the multiple-pattern case. Suppose we are given m patterns, v^1, v^2, ..., v^m. Our goal here is to find weight matrix W such that those m desired patterns will be stored (if possible) as equilibria of the Hopfield-type neural network (2.1.2).

For each v^i, $i = 1, 2, ..., m$, there exists A_i, b_i and x, as we shown early, such that $A_i x = b_i$. To find weight matrix W for storing v^1, v^2, ..., v^m as equilibria of (2.1.2), we need to find the common solutions of $A_i x = b_i$ for $i = 1, 2, ..., m$. In other words, we need to find if the following matrix equation

$$\begin{bmatrix} A_1 \\ A_2 \\ \cdots \\ A_m \end{bmatrix} x = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_m \end{bmatrix} \tag{2.3.9}$$

has solutions. Before proceeding, it is necessary to check whether (2.3.9) has solutions.

Recall that the solution set of $A_i x = b_i$ is given by $\{A_i^+ b_i + N(A_i)\}$ which is an affine linear space in R^{nw}. Thus, (2.3.9) has solutions if and only if

$$\{A_1^+ b_1 + N(A_1)\} \cap \{A_2^+ b_2 + N(A_2)\} \cap \ldots \cap \{A_m^+ b_m + N(A_m)\} \neq \varnothing$$

in other words, the intersection of the m affine manifolds, $\{A_i^+ b_i + N(A_i)\}$ for $i = 1, 2, \ldots, m$, is non-empty.

There are some other criteria used to check the existence of the common solutions of $A_i x = b_i$ for $i = 1, 2, \ldots, m$. For example, the common solutions exist if and only if $[b_1^T, b_2^T, \ldots, b_m^T]^T \in R([A_1^T, A_2^T, \ldots, A_m^T]^T)$, or the linear mapping of $[A_1^T, A_2^T, \ldots, A_m^T]^T$ is onto.

The major drawback of solving for the common solution of x from (2.3.9) is the large computation time. In fact, the dimension of $[A_1^T, A_2^T, \ldots, A_m^T]^T$ is $mn \times n_w$ which can be very large for large values of m and n. In the next section, we will introduce the so-called *Morris-Odell*'s Theorem to overcome this computational problem.

2.4 A Recursive Algorithm for Finding Weights Based on Multiple Patterns

Our goal in this section is to check for the existence of solutions and solve for the common solutions for the following linear equations with reasonable computation time:

$$\begin{bmatrix} A_1 \\ A_2 \\ \cdots \\ A_m \end{bmatrix} x = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_m \end{bmatrix} \tag{2.4.1}$$

Morris and Odell [41] propose a criterion to check the existence of common solutions of $A_i x = b_i$ for $i = 1, 2, ..., m$. An excellent algorithm for finding the common solutions (if they exist) is also given. In the following, we sketch their result.

Let m be a positive integer. Suppose A_i is a $n \times n_w$ matrix and b_i is a $n \times 1$ vector for $i = 1, 2, ..., m$. Define

$$\begin{aligned} C_k &= A_k F_{k-1} \\ d_k &= b_k - A_k e_{k-1} \\ e_k &= e_{k-1} + F_{k-1} C_k^{+} d_k \\ F_k &= F_{k-1}(I - C_k^{+} C_k) \end{aligned} \tag{2.4.2}$$

for $k = 2, 3, ..., m$ with $C_1 = A_1$, $d_1 = b_1$, $e_1 = A_1^{+} b_1$ and $F_1 = I - A_1^{+} A_1$. The *Morris-Odell*'s Theorem is stated as follows:

Theorem 2.4.1 (Morris and Odell [41])

$A_i x = b_i$ for $i = 1, 2, ..., m$ have a common solution if and only if $C_i C_i^+ d_i = d_i$ for $i = 1, 2, ..., m$. The general common solution is $x = e_m + F_m z$ where z is arbitrary.

The original theorem given by Morris and Odell in [41] was shown for general matrix equations. There are some useful corollaries of this theorem in their paper.

Let m be a positive real number, and $v^1, v^2, ..., v^m$ be m desired patterns. For those m patterns, we have corresponding m matrix equations: $A_i x = b_i$ for $i = 1, 2, ..., m$. Suppose all of those m patterns can be stored as equilibria of the Hopfield-type neural network (2.1.2); in other words, $A_i x = b_i$ for $i = 1, 2, ..., m$, have common solutions. Consequently, we have the symmetric weight matrix associated with those m patterns. Denote this weight matrix as $W^{1\text{-}m}$, where the superscript, 1-m, simply indicates the weight matrix for those m patterns, $v^1, v^2, ..., v^m$. We will use this notation even for the case that not all of those m patterns can be stored.

Now, r new patterns, $v^{m+1}, v^{m+2}, ..., v^{m+r}$, are provided. We wish for them, together with those old m patterns, $v^1, v^2, ..., v^m$, to be stored as equilibria of the Hopfield-type neural network (2.1.2). By applying *Morris-Odell*'s Theorem, we first check to see if v^{m+1} can be stored, i.e., check $C_{m+1} C^+_{m+1} d_{m+1} = d_{m+1}$, where $C_{m+1} = A_{m+1} F_m$, $d_{m+1} = b_{m+1} - A_{m+1} e_m$, and all A_{m+1}, F_m, b_{m+1}, e_m are all known. This evaluation involves only simple matrix operations. If equality holds, v^{m+1} can be stored and the common solution is given by $e_{m+1} = e_m + F_m C^+_{m+1} d_{m+1}$; if not, v^{m+1} can not be stored. Remove v^{m+1} from pattern list and proceed to v^{m+2}.

This algorithm is summarized as follows:

Step 1: For the pattern v^1, calculate C_1, d_1, e_1, and F_1.

Step 2: For a new pattern (say v^2), find C_2, d_2, e_2, and F_2, based on (2.4.2). Check if

$C_2C_2^+d_2 = d_2$ is satisfied. If yes, go to **Step 3**. If not, remove v^2 form the given set of patterns. Then go to **Step 3**.

Step 3: Check if all given patterns have been proposed. If not, choose a new pattern from the given pattern list. Go to **Step 2**. If all patterns have been processed, the solution vector is given by e_n.

2.5 Network Capacity, "Storable" and "Unstorable" Patterns

The network capacity deals with the maximum number of arbitrary patterns which can be stored. In other words, the network capacity, denoted as NC, is the maximum number of m (the number of given arbitrary patterns) such that the set of equations (2.4.1) will have at least one solution.

Let $A = [A_1^T, A_2^T, ..., A_m^T]^T$, $b = [b_1^T, b_2^T, ..., b_m^T]^T$. Since A_i and b_i, $i = 1, 2, ..., m$, are of dimension $n \times n_w$ and $n \times 1$, respectively, the matrix A and vector b are of dimension $mn \times n_w$ and $mn \times 1$. By Linear Algebra, equation (2.4.1) always has solution(s) if $mn \leq n_w$. This is because, in equation (2.4.1), the number of unknowns is greater than or equal to the number of equations if $mn \leq n_w$. Thus, the capacity NC of the Hopfield-type feedback neural network based on our design approach is at least $\lfloor 0.5(n-1) \rfloor$ where $\lfloor . \rfloor$ is defined by

$$\lfloor x \rfloor = \text{Max}\{n \in Z: n \leq x\} \tag{2.5.1}$$

For large n, the network capacity NC can reach $0.5n$, which is greater than $0.15n$. As shown in [14], the capacity can reach $n+1$. But, the designed network in [14] is not symmetric. Since our synthesis results in a symmetric network, the capacity is expected to be less than $n + 1$.

As we stated in the Section 2.4, Morris-Odell's Theorem can be viewed in a recursive way. Suppose we are given m patterns, $v^1, v^2, ..., v^m$. We want to find a symmetric matrix W such that these m patterns are stored as equilibria. For each v^i, we can form A_i and b_i.

For the matrix equations, $A_i x = b_i$ for $i = 1, 2, ..., m$, we first check $C_2 C_2^+ d_2 = d_2$ (since $C_1 C_1^+ d_1 = d_1$ is always true). This process is repeated. Suppose $C_k C_k^+ d_k = d_k$ for $k = 2, 3, ..., p$, where $2 \le p \le m$ and $C_{p+1} C^+_{p+1} d_{p+1} \ne d_{p+1}$. By *Morris-Odell*'s Theorem, $A_i x = b_i$ for $i = 1, 2, ..., p$ have common solutions given by e_p (assume $z = 0$), and $A_{p+1} x = b_{p+1}$ will not have solutions shared by $A_i x = b_i$ for $i = 1, 2, ..., p$ (although $A_{p+1} x = b_{p+1}$ always has solutions). In terms of neural network, the desired patterns, $v^1, v^2, ..., v^p$, will be exactly stored as equilibria of the Hopfield-type feedback neural network, but v^{p+1} cannot. Thus, we remove A_{p+1} from the list, $A_1, A_2, ..., A_m$ and rearrange $A_{p+2}, A_{p+3}, ..., A_m$ as B_{p+1}, $B_{p+2}, ..., B_{m-1}$, where $B_{p+1} = A_{p+2}, B_{p+2} = A_{p+3}, ..., B_{m-1} = A_m$. Now, check $C_i C_i^+ d_i = d_i$ for $i = p+1, p+2, ..., m-1$ on $B_{p+1}, B_{p+2}, ..., B_{m-1}$. This whole process is repeated until all A_i's have been checked.

When this checking process is completed, we will get a subset of matrix equations, $A_j x = b_j$ for $r \le j \le s$, where $1 \le r, s \le m$. Those matrix equations have common solution which is given by e_s (assume $z = 0$). We must note that this subset of matrix equations depends on the order of checking these equations. The algorithm stated above is also summarized in Section 2.4.

In terms of neural network, we deduce a subset of the desired patterns, $\{v^j, r \le j \le s, 1 \le r, s \le m\}$, which will all be exactly stored as equilibria of the Hopfield-type neural network (21..2). We call these patterns "storable" patterns, and the rest of patterns "unstorable" patterns. Once the common solution x is found, the weight matrix W can be easily reconstructed in the same way as we show in Section 2.3.

2.6 Parameter Determination for Implementable Networks

In this section, we consider the implementable problem (defined later) of the Hopfield-type feedback neural network with two-state neurons as defined in [21]. For this type of network, one problem in the implementation is to determine whether a designed network is implementable as an electronic circuit or not. Mathematically, we can find all weights for any values of the network parameters if those given patterns can be stored. The designed network based on this approach can be simulated by a digital computer, but may not be readily implementable via a circuit since the resistance in the corresponding neural circuit may be negative.

The mathematical model for the feedback continuous-time neural network [21] can be expressed as

$$c_i \frac{du_i}{dt} = \sum_{i=1}^{n} w_{ij} v_j + I_i - \frac{1}{r_i} u_i \tag{2.6.1a}$$

$$v_i = s_i(u_i) \tag{2.6.1b}$$

for $i = 1, ..., n$, where $c_i, r_i > 0$, $u_i \in R$ and $v_i \in (-\Delta, \Delta)$ with $0 < \Delta \in R$, $s_i: R \rightarrow (-\Delta, \Delta)$ is the sigmoidal function, n is the number of neurons in (2.7). The schematic diagram of a circuit realization of (2.7) is shown in Figure 2.6.1.

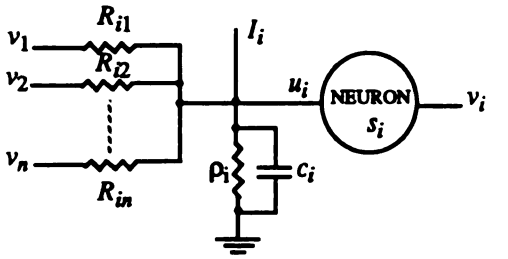


Figure 2.6.1: The schematic circuit diagram of a continuous-time neural network.

Note that R_{ij} and ρ_i, for $i, j = 1, ..., n$, are passive linear resistive elements, and C_i, $i = 1, ..., n$, are capacitors. Applying KCL, the equation of the circuit in Figure 2.6.1 can be easily derived as follows:

$$c_i \frac{du_i}{dt} + \frac{u_i}{\rho_i} = \sum_{i=1}^{n} \frac{1}{R_{ij}} (v_j - u_i) + I_i \tag{2.6.2}$$

which is equivalent to

$$c_i \frac{du_i}{dt} = \sum_{i=1}^{n} \frac{1}{R_{ij}} v_j + I_i - \left(\frac{1}{\rho_i} + \sum_{i=1}^{n} \frac{1}{R_{ij}} \right) u_i \tag{2.6.3}$$

for $i = 1, ..., n$. Comparing (2.6.3) with (2.6.1a), we obtain the following identities:

$$w_{ij} = \frac{1}{R_{ij}} \tag{2.6.4a}$$

$$\frac{1}{r_i} = \frac{1}{\rho_i} + \sum_{i=1}^{n} \frac{1}{R_{ij}} \tag{2.6.4b}$$

for $i = 1, ..., n$.

The network (2.6.1) can also be expressed in a matrix form:

$$\boldsymbol{C}d\boldsymbol{u}/dt = \boldsymbol{W}\boldsymbol{v} + \boldsymbol{I} - \boldsymbol{R}^{-1}\boldsymbol{u} \tag{2.6.5a}$$

$$\boldsymbol{v} = \boldsymbol{S}(\boldsymbol{u}) \tag{2.6.5b}$$

where $u = [u_1, ..., u_n]^T$, $v = [v_1, ..., v_n]^T$, $W = [w_{ij}]$, $C = \text{diag}(c_1, ..., c_n)$, $R = \text{diag}(r_1, ..., r_n)$, $I = [I_1, ..., I_n]^T$, $S(u) = [s_1(u_1), ..., s_n(u_n)]^T$.

For given desired patterns, $v^1, ..., v^m$, we expected to find weights, w_{ij}, $i, j = 1, ..., n$, such that those m patterns are stored as asymptotically stable equilibria of (2.6.1). From a mathematical point of view, we can find all weights for any value of r_i, $i = 1, ..., n$, provided that all these m patterns, $v^1, ..., v^m$, can be stored. This approach will not cause any problems in computer simulation as pointed out in [14], but, the designed system may not be implementable as in the circuit of Figure 1 if $\rho_i < 0$ for some $i \in \{1, ..., n\}$. In this case, ρ_i must be realized with a negative resistance. Therefore, for a designed system to be implementable as the circuit in Figure 1, $\rho_i > 0$ for $i = 1, ..., n$.

For simplicity, we assume $s_i(.) = s_0(.)$, and $r_i = r_0$ for all i. This assumption is reasonable for the circuit implementation. Consider a desired binary pattern, $v = [v_1, ..., v_n]^T$ with $v_i = \upsilon$ or $-\upsilon$ for $0 < \upsilon < \Delta$. Define $\mu = s_0^{-1}(\upsilon)$. $s_0^{-1}(.)$ is an odd function, $-\mu = s_0^{-1}(-\upsilon)$. The relation between μ and υ is depicted in Figure 2.6.2.

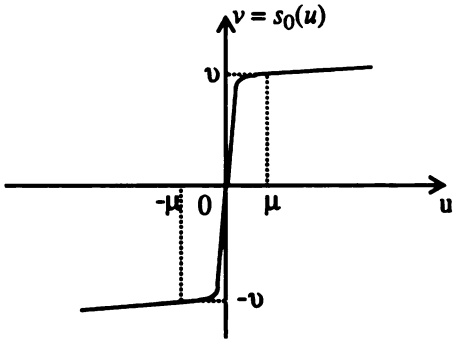


Figure 2.6.2: Relation between μ and υ.

Assume $I_i = 0$ for all i. To be an equilibrium of (2.6.1), v must satisfy

$$\boldsymbol{W}v = \boldsymbol{R}^{-1}S^{-1}(v) \tag{2.6.6}$$

where $\boldsymbol{W} = [w_{ij}]$ is the weight matrix, $\boldsymbol{R} = r_0\boldsymbol{I}$ which is a constant matrix, and $S^{-1}(v) = [s_0^{-1}(v_1), ..., s_0^{-1}(v_n)]^T$. In Section 2.3, we have shown that the weight matrix $\boldsymbol{W}$ can be constructed as

$$\boldsymbol{W} = \begin{bmatrix} 0 & x_1 & x_2 & \cdots\cdots & x_{n-2} & x_{n-1} \\ x_1 & 0 & x_n & \cdots\cdots & x_{2n-3} & x_{2n-2} \\ & & \cdots\cdots & & & \\ x_{n-1} & x_{2n-3} & x_{3n-3} & \cdots\cdots & x_{nw} & 0 \end{bmatrix} \tag{2.6.7}$$

where $n_w = (n^2 - n)/2$, and $x = [x_1, ..., x_{nw}]^T$ is given as

$$x = \boldsymbol{A}^{+}\boldsymbol{R}^{-1}S^{-1}(v) \tag{2.6.8}$$

where $\boldsymbol{A}^{+} = \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^{-1}$. From (2.6.8), we see that

$$\begin{aligned} \|x\|_\infty &\le \|\boldsymbol{A}^{+}\|_\infty \|\boldsymbol{R}^{-1}\|_\infty \|S^{-1}(v)\|_\infty \\ &\le \|\boldsymbol{A}^T\|_\infty \|(\boldsymbol{A}\boldsymbol{A}^T)^{-1}\|_\infty \|\boldsymbol{R}^{-1}\|_\infty \|S^{-1}(v)\|_\infty \end{aligned} \tag{2.6.9}$$

For the desired binary pattern v, $\|S^{-1}(v)\|_\infty = \mu$, $\|\boldsymbol{A}^T\|_\infty = 2\upsilon$. And $\|\boldsymbol{R}^{-1}\|_\infty = r_0^{-1}$ since $\boldsymbol{R} = r_0\boldsymbol{I}$. It is easy to verify that

$$AA^T = \upsilon^2 \begin{bmatrix} n\text{-}1 & \omega & \cdots\cdots & \omega \\ \omega & n\text{-}1 & \cdots\cdots & \omega \\ \cdots & \cdots & \cdots & \cdots \\ \omega & \omega & \cdots\cdots & n\text{-}1 \end{bmatrix} \tag{2.6.10}$$

where $\omega = 1$ or -1, n is the number of neurons in (2.6.1). Denote

$$\Theta = \begin{bmatrix} n\text{-}1 & \omega & \cdots\cdots & \omega \\ \omega & n\text{-}1 & \cdots\cdots & \omega \\ \cdots & \cdots & \cdots & \cdots \\ \omega & \omega & \cdots\cdots & n\text{-}1 \end{bmatrix}^{-1} \tag{2.6.11}$$

Thus, $\|(AA^T)^{-1}\|_\infty = \upsilon^{-2}\|\Theta\|_\infty$. It follows that

$$\|x\|_\infty \le \frac{2\mu}{r_0 \upsilon}\|\Theta\|_\infty \tag{2.6.12}$$

As we stated earlier, a designed system is implementable if $\rho_i > 0$ for $i = 1, ..., n$. From (2.6.4), it is observed that $\rho_i > 0$ if

$$\frac{1}{r_0} > \sum_{j=1}^{n} |w_{ij}| \tag{2.6.13}$$

for $i = 1, ..., n$. Since

$$\|W\|_\infty \ge \sum_{j=1}^{n} |w_{ij}| \tag{2.6.14}$$

for $i = 1, ..., n$, we see that (2.6.13) is satisfied if

$$\frac{1}{r_0} > \|W\|_\infty \tag{2.6.15}$$

for $i = 1, ..., n$. From (2.6.7), we know that

$$\|W\|_\infty \le n\|x\|_\infty \tag{2.6.16}$$

Combining (2.6.16) with (2.6.12), we obtain

$$\|W\|_\infty \le \frac{1}{r_0}\left(\frac{2n\mu}{\upsilon}\|\Theta\|_\infty\right) \tag{2.6.17}$$

Therefore, (2.6.15) will be satisfied if

$$\frac{2n\mu}{\upsilon}\|\Theta\|_\infty < 1 \tag{2.6.18}$$

which is equivalent to

$$\frac{\upsilon}{\mu} > 2n\|\Theta\|_\infty \tag{2.6.19}$$

The inequality (2.6.19) is an explicit constraint on the network parameters for the implementable circuit realization.

For a given binary pattern, v, the value of $\|\Theta\|_\infty$ is constant. The inequality (2.6.19) gives a sufficient condition for a designed system to be implementable, namely that υ/μ is greater than some known constant. Some values of $\|\Theta\|_\infty$ and $2n\|\Theta\|_\infty$ for different n are listed in Table 2.6.1. From the data in Table 2.6.1, we observe that the value of $2n\|\Theta\|_\infty$

decreases as n increases. In the circuit implementation, the ratio of υ/μ can be changed by varying $s_0'(0)$.

Table 1:

n	$\|\Theta\|_\infty$	$2n\|\|\Theta\|\|_\infty$
3	1.2500	7.5000
4	0.7116	5.6928
5	0.4585	4.5850
6	0.3500	4.2000
7	0.2835	3.9690
8	0.2381	3.8096
9	0.2051	3.6918
10	0.1802	3.6040

From (2.6.4), ρ_i can be expressed as

$$\rho_i = \frac{1}{\frac{1}{r_0} - \sum_{j=1}^{n} w_{ij}} \tag{2.6.20}$$

By applying (2.6.12), (2.6.14), and (2.6.16), it is easy to verify that

$$\rho_i \geq \frac{1}{\frac{1}{r_0} + \sum_{j=1}^{n} |w_{ij}|} \geq \frac{1}{\frac{1}{r_0} + \|W\|_\infty}$$

$$\geq \frac{1}{\frac{1}{r_0} + n\|x\|_\infty} \geq \frac{r_0}{1 + \frac{2n\mu}{\upsilon}\|\Theta\|_\infty} \tag{2.6.21}$$

Thus, by choosing a sufficiently large value of r_0, ρ_i will be very large. Consequently, the power dissipation of the entire network can be greatly reduced, which is one of the major

advantages of our approach.

Example: Let us look at an example. Suppose we want to store a pattern, $v = [2.5, 2.5, 2.5, 2.5]^T$. Thus $\upsilon = 2.5$. The matrix A can be constructed. It is easy to verify that

$$\Theta = \begin{bmatrix} 3 & 1 & 1 & 1 \\ 1 & 3 & 1 & 1 \\ 1 & 1 & 3 & 1 \\ 1 & 1 & 1 & 3 \end{bmatrix}^{-1} = \begin{bmatrix} 0.41677 & -0.0833 & -0.0833 & -0.0833 \\ -0.0833 & 0.41677 & -0.0833 & -0.0833 \\ -0.0833 & -0.0833 & 0.41677 & -0.0833 \\ -0.0833 & -0.0833 & -0.0833 & 0.41677 \end{bmatrix}$$

and $\|\Theta\|_\infty = 0.6667$. Thus, $2n\|\Theta\|_\infty = 5.3336$. Let $r_0 = 10K\Omega$. First, suppose the sigmoidal function $s_0(.)$ is given such that $\mu = 3$. Since $\upsilon/\mu = 0.8333$, the inequality of (2.6.19) is not satisfied. Since $b = 10^{-4}[3, 3, 3, 3]^T$, by (3.3), $x = 4\times10^{-5}[1, 1, 1, 1, 1, 1]^T$. The weight matrix W can be constructed from (2.6.7) as

$$W = 4 \times 10^{-5} \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

By (2.6.20), $\rho_1 = \rho_2 = \rho_3 = \rho_4 = -50K\Omega$. Therefore, those resistors must be realized with negative resistance. Now, increase the slope of the sigmoidal function $s_0(.)$ such that $\mu = 0.4$. In this case, (2.6.19) is satisfied. By applying the same procedure, we find that $\rho_1 = \rho_2 = \rho_3 = \rho_4 = 12.7K\Omega$ which is positive as we expect. Therefore, they can be easily implemented.

Now, consider the multiple-pattern case. Suppose we are given m binary patterns, v^1,

..., v^m. As shown in the Section 2.3, in order to find the weight matrix W, the following m matrix equations, $A_i x = r_0^{-1} S^{-1}(v^i)$ for $i = 1, ..., n$, must have (at least) one common solution. The existence of such common solution can be checked by the Morris-Odell's Theorem. Without loss of generality, assume this common solution exists. Thus,

$$x = \begin{bmatrix} A_1 \\ \vdots \\ A_m \end{bmatrix}^{+} \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix} \tag{2.6.22}$$

where $b_i = r_0^{-1} S^{-1}(v^i)$. It is easy to see that

$$\left\| \left[b_1^T \cdots b_m^T \right]^T \right\|_\infty = \frac{\mu}{r_0} \tag{2.6.23}$$

and

$$\left\| \left[A_1^T \cdots A_m^T \right] \right\|_\infty \le \sum_{k=1}^{m} \left\| A_k^T \right\|_\infty = 2m\upsilon \tag{2.6.24}$$

Moreover,

$$\begin{bmatrix} A_1 \\ \\ A_m \end{bmatrix} \left[A_1^T \quad A_m^T \right] = \upsilon^2 \psi^{-1} \tag{2.6.25}$$

where ψ has the same format as Θ in (2.6.11). Following the same argument stated earlier in this section, we see that

$$\|x\|_\infty \le \frac{2m\mu}{r_0 \upsilon}\|\psi\|_\infty \tag{2.6.26}$$

It follows that $\rho_i > 0$ if

$$\frac{\upsilon}{\mu} > 2mn\|\psi\|_\infty \tag{2.6.27}$$

where m is the number of given patterns, n is the number of neurons. The inequality (2.6.27) shows that a designed network is implementable if the ratio of υ/μ is larger than some known constant.

Now, let us consider the circuit implementation of this kind of neuron which will make the inequality (2.6.27) true. As we have shown, the designed system will be implementable if the ratio of the network parameters, υ/μ, is larger than a specific quantity. This ratio can be changed by varying the slope of the sigmoidal function around the origin. Since it may not be practical to adjust the slope of the sigmoidal function on-line, it may be desired to design a neuron circuit which has a very sharp slope around the origin. This will guarantee the ratio of υ/μ to be very large. In this case, we expect our designed system to be implementable.

As we stated early, the network power dissipation will be reduced if we choose large values of r_i, $i = 1, ..., n$, in the network model. Another way to minimize the network power dissipation is to design the neuron circuit such that it operates in sub-threshold [38].

A two-state neuron circuit is shown in Figure 2.6.3 which is built using inverters in series. The input - output characteristic with sharp slope at the origin is shown in Figure

2.6.4. But, this circuit is not operating in sub-threshold. The network power dissipation may not be sufficiently small.

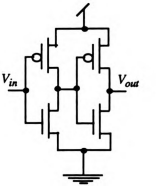


Figure 2.6.3: A simple 2-state neuron circuit.

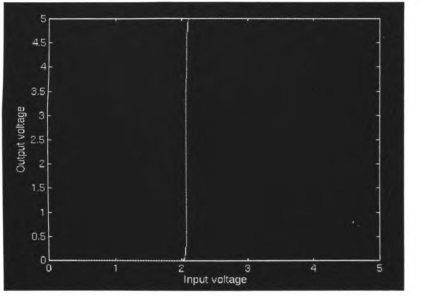


Figure 2.6.4: The input-output characteristic of the circuit shown in Figure 2.6.3.

For sub-threshold operation, we choose the neuron circuit as the transconductance amplifier [38] shown in Figure 2.6.5. In this circuit, there is one current mirror and one bias transistor with its gate voltage controlled by the bias voltage V_{bias}. The voltage V_1 is the input voltage, V_2 is an internal voltage which is 0.7V in our simulation. Since the bias voltage is set at 0.7V. We can reduce one power source if we let V_2 and V_{bias} are equal.

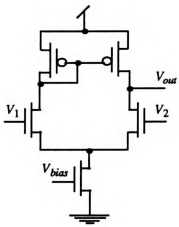


Figure 2.6.5: A 2-state neuron operating in sub-threshold.

Figure 2.6.6 shows the characteristic of the input - output. It is observed that the transition voltage occurs at 0.7V which is the internal voltage V_2.

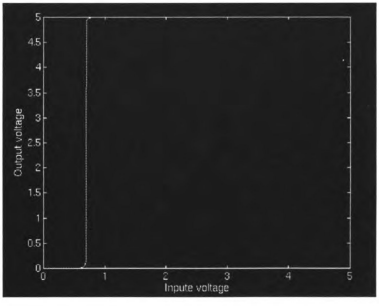


Figure 2.6.6: The input-output characteristic of the circuit shown in Figure 2.6.5.

The bias voltage, V_{bias}, controls the amount of current flowing through the neuron circuit. In our simulation, we set V_{bias} = 0.7V. Thus, the circuit operates in sub-threshold, which minimizes the network power dissipation. The characteristic of bias current and input voltage is shown in Figure 2.6.7.

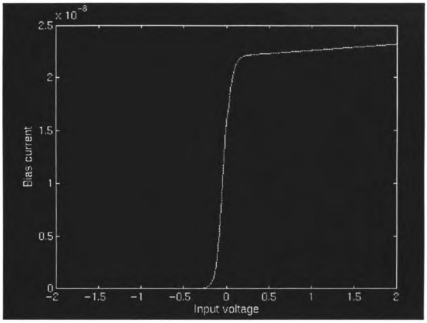


Figure 2.6.7: The characteristic of bias current and input voltage V_1 of the circuit shown in Figure 2.6.5.

Comparing the circuits in Figure 2.6.3 and Figure 2.6.5, the circuit in Figure 2.6.5 requires more voltage sources and one more transistor, but it operates below threshold.

2.7 Persistence of Equilibria under Weight Variations

In the Hopfield-type feedback continuous-time neural network [21], patterns are stored as asymptotically stable equilibria of the network by choosing proper weights. As we show in the Section 2.3 and Section 2.4, the weight matrix can be found by an analytical algorithm so that all patterns are exactly stored as equilibria. The Hopfield-type feedback neural network can be implemented via VLSI technology due to its structure simplicity, but, in hardware implementation, there is inevitably a quantitative difference between the calculated and the implemented weights. This variation in weights has consequent variation in equilibria. Indeed, possible bifurcations may arise, e.g. saddle-node bifurcations ([9], [17]). Thus, equilibria may lose stability or it may disappear.

The feedback continuous-time neural network [21] is expressed as

$$\boldsymbol{C}du/dt = \boldsymbol{T}x - \boldsymbol{R}^{-1}S^{-1}(x) \tag{2.7.1}$$

where $\boldsymbol{C}$ and $\boldsymbol{R}$ are both constant matrices and $S(.)$ is a bounded C^1 "sigmoidal" function, $x = S(u) = [s_0(u_1), ..., s_0(u_n)]^T$. $\boldsymbol{T}$ is the weight matrix which represents the calculated weight matrix. Let $\boldsymbol{T}'$ be the implemented weight matrix. Thus, $\delta\boldsymbol{T} = \boldsymbol{T}' - \boldsymbol{T}$ is the matrix of perturbation on $\boldsymbol{T}$. The perturbed feedback continuous-time neural network can be described as

$$\boldsymbol{C}dz/dt = \boldsymbol{T}'y - \boldsymbol{R}^{-1}S^{-1}(y) \tag{2.7.2}$$

where $y = S(z)$.

Let $f(.) = \boldsymbol{T} - \boldsymbol{R}^{-1}S^{-1}(.)$. Then (2.7.1) can be rewritten as

$$C du/dt = f(x) \tag{2.7.3}$$

And its perturbed system can be described as

$$C dz/dt = g(y) \tag{2.7.4}$$

where $g(.) = f(.) + \delta f(.)$, and $\delta f(.) = \delta T$ which is a linear operator. Let x^0 be an equilibrium of (2.7.3). And let y^0 be the corresponding equilibrium of the perturbed system (2.7.4). Let $L(x^0, y^0)$ be the line segment connecting x^0 and y^0.

In general, the system is designed such that $|D_x f(x^0)| \neq 0$ which makes x^0 an isolated equilibrium by the *Inverse Function Theorem* [52]. For the perturbed system (2.7.4), $g(y) = f(y) + \delta T y$. For the system (2.7.3) to have the property of persistence of equilibria, δT has to be small enough in the sense of a norm. This means that $|D_y g(x)| \neq 0$ for all x within a neighborhood of x^0, which implies that all the eigenvalues of $D_x f(x^0)$ must have non-zero real parts.

Let us define the set

$$\mathcal{D} = \{x \in (-\Delta, \Delta)^n \mid D_x f(x) \text{ is non-singular}\}$$

Obviously, $x^0 \in \mathcal{D}$ since $|D_x f(x^0)| \neq 0$. By the continuity of $D_x f(x)$, there exists a neighborhood $B(x^0)$ of x^0 such that $B(x^0) \subset \mathcal{D}$.

Lemma 2.7.1:

Let x^0 be an equilibrium of (2.7.3). There exists $0 < K < \infty$ such that

$\|(D_x f(z))^{-1}\|_p \leq K$ for all $z \in B(x^0) \subset \mathcal{D}$, where $\|.\|_p$ is some norm in R^n.

Proof: Since the mapping φ: $z \rightarrow D_x f(z)$ is continuous and $W = (-\Delta, \Delta)^n$ is bounded, there exists $0 < K_1 < \infty$ such that

$$\|D_x f(z) - D_x f(x^0)\|_p \leq K_1$$

for all $z \in B(x^0)$. And since the mapping ψ: $D_x f(z) \rightarrow (D_x f(z))^{-1}$ is continuous, the range is also bounded. Thus, there exists $0 < K_2 < \infty$ such that

$$\|(D_x f(z))^{-1} - (D_x f(x^0))^{-1}\|_p \leq K_2$$

for all $z \in B(x^0)$. It follows that

$$\|(D_x f(z))^{-1}\|_p \leq \|(D_x f(x^0))^{-1}\|_p + K_2$$

for all $z \in B(x^0)$. Let $K = \|(D_x f(x^0))^{-1}\|_p + K_2$. Thus, for all $z \in B(x^0)$, $\|(D_x f(z))^{-1}\|_p \leq K$. This completes the proof. ■

Theorem 2.7.2:

Let x^0 and y^0 be the equilibria of (2.7.3) and (2.7.4), respectively. Let $\varepsilon > 0$. Then $\|y^0 - x^0\|_p \leq \varepsilon K$ if $\|\delta f\|_p \leq \varepsilon$ and $L(x^0, y^0) \subset B(x^0)$, where K is defined in the proof of Lemma 2.7.1.

Proof: Since x^0 and y^0 are equilibria of (2.7.3) and (2.7.4), $g(y^0) = f(y^0) + \delta f(y^0) = 0 = f(x^0)$. It follows that $f(y^0) - f(x^0) = -\delta f(y^0)$. By the *Mean-Value Theorem* [35], there exist z^1, ..., $z^n \in L(x^0, y^0)$ such that

$$f(y^0) - f(x^0) = \begin{bmatrix} D_x f_1(z^1) \\ \vdots \\ D_x f_n(z^n) \end{bmatrix} (y^0 - x^0)$$

$$= \begin{bmatrix} D_x f_1(z_1^1) \\ \vdots \\ D_x f_n(z_n^n) \end{bmatrix} (y^0 - x^0)$$

$$= D_x f(z)(y^0 - x^0)$$

where $z = [z^1_1, ..., z^n_n]^T \in L(x^0, y^0)$. Since $L(x^0, y^0) \subset B(x^0) \subset \mathcal{D}$, $D_x f(z)$ is invertible. Thus,

$$y^0 - x^0 = (D_x f(z))^{-1}(f(y^0) - f(x^0))$$

$$= -(D_x f(z))^{-1}\delta f(y^0)$$

which implies

$$\|y^0 - x^0\|_p \leq \|(D_x f(z))^{-1}\|_p \|\delta f(y^0)\|_p \qquad (2.7.5)$$

Since $\|(D_x f(z))^{-1}\|_p \leq K$ by Lemma 3.1 and $\|\delta f(y^0)\|_p \leq \varepsilon$ for some $\varepsilon > 0$ by assumption, $\|y^0 - x^0\|_p \leq \varepsilon K$. This completes the proof. ■

To apply Theorem 2.7.2, we need to estimate the value of K and the size of $B(x^0)$,

which is generally not easy for the general gray-level input patterns. In the following, we will develop further results under the case of binary desired patterns.

Let $x^0 = [x_1^0, ..., x_n^0]^T$ be a desired binary pattern with $x_i^0 = \upsilon$ or $-\upsilon$ for $0 < \upsilon \in R$. In Section 2.3 and Section 2.4, we have introduced an analytical learning algorithm to find a *symmetric* weight matrix T of (2.7.3) such that x^0 is exactly stored as an equilibrium. Thus, in the following, we may assume that T is known and fixed. It is also shown in the Section 2.2 that x^0 is asymptotically stable if $s_0'(s_0^{-1}(\upsilon)) < \omega_0$, where $s_0(.)$ is the sigmoidal function of each neuron, ω_0 is a small positive real number which can be evaluated exactly.

Let ω_0 be the positive number defined above. Now, let us define

$$\mathcal{D}_{\omega 0} = \{x = [x_1, ..., x_n]^T \in (-\Delta, \Delta)^n \mid s_0'(s_0^{-1}(x_i)) < \omega_0, i = 1, ..., n\} \tag{2.7.6}$$

where Δ is the upper bound of the sigmoidal function s_0. In general, $\mathcal{D}_{\omega 0}$ is non-empty due to the "shape" of this sigmoidal function $s_0(.)$, especially, when the neuron is realized by CMOS inverters or op-amps. Let $\mathcal{B}_\alpha(x^0)$ be a neighborhood of x^0, where α is selected such that $\mathcal{B}_\alpha(x^0) = \{x \in (-\Delta, \Delta)^n \mid \|x - x^0\|_2 < \alpha\} \subset \mathcal{D}_{\omega 0}$. Obviously, α exists since $s_0'(s_0^{-1}(x_i^0)) < \omega_0$ for $i = 1, ..., n$.

Let $x \in \mathcal{B}_\alpha(x^0)$. Since $f(.) = T - R^{-1}S^{-1}(.)$,

$$D_x f(x) = T - R^{-1} D_x S^{-1}(x) \tag{2.7.7}$$

is invertible. We have the following lemma:

Lemma 2.7.3:

Let $z \in \mathcal{B}_\alpha(x^0)$. If $\|\delta T\|_p < 1 / \|(T - R^{-1}D_x S^{-1}(z))^{-1}\|_p$, $T - R^{-1}D_x S^{-1}(z) + \delta T$ is non-singular.

The proof of this lemma directly follows the fact that $\|\delta T(T - R^{-1}D_x S^{-1}(z))^{-1}\|_p \leq \|\delta T\|_p \|(T - R^{-1}D_x S^{-1}(z))^{-1}\|_p < 1$.

Theorem 2.7.4:

There exists an open set $\mathcal{A} \subset R^{n\times n}$ such that for $T \in \mathcal{A}$, there exists $x \in \mathcal{D}_{\omega 0}$ which satisfies $f(T, x) = Tx - R^{-1}S^{-1}(x) = 0$.

Proof: Let $0 \neq z \in \mathcal{D}_{\omega 0}$. There is $T_z \in R^{n\times n}$ such that $f(T_z, z) = T_z z - R^{-1}S^{-1}(z) = 0$ as we have shown in the Section 2.3. Since $M = T_z - R^{-1}D_x S^{-1}(z)$ is non-singular by assumption, by the *Implicit Function Theorem*, there is an open set $\mathcal{A}_z \subset R^{n\times n}$ containing T_z such that for $T \in \mathcal{A}_z$ there is a unique $x \in \mathcal{D}_{\omega 0}$ such that $f(T, x) = Tx - R^{-1}S^{-1}(x) = 0$. Define

$$\mathcal{A} = \bigcup_{z \in \mathcal{D}_{\omega 0}} \mathcal{A}_z \tag{2.7.8}$$

which is an open set. Therefore, for $T \in \mathcal{A}$, there is some $x \in \mathcal{D}_{\omega 0}$ such that $f(T, x) = 0$. This completes the proof. ■

By the *Implicit Function Theorem*, there is a domain $\mathcal{U}_x$ containing x, and a corresponding domain $\mathcal{U}_T$ containing T such that $f(x, T) = 0$. Moreover, there is a unique function, say g, such that $x = g(T)$ within $\mathcal{U}_x \times \mathcal{U}_T$. In addition, there also exists a domain $\mathcal{V}_x$ and a corresponding domain $\mathcal{V}_T$ such that $x \in \mathcal{V}_x$ is an asymptotically stable equilibrium satisfying $f(x, T) = 0$ for some $T \in \mathcal{V}_T$. Our efforts will be focused on the intersection of $\mathcal{U}_x \times \mathcal{U}_T$ and $\mathcal{V}_x \times \mathcal{V}_T$.

In the following, we assume $\boldsymbol{R} = r_0\boldsymbol{I}$ for simplicity. Let $\lambda_1, ..., \lambda_n$ be the eigenvalues of $\boldsymbol{T}$, $\lambda_{max} = max\{\lambda_1, ..., \lambda_n\}$. Let $\boldsymbol{K} = D_x\mathcal{S}^{-1}(x)$ for some $x \in \mathcal{D}_{\omega 0}$. We assume that ω_0 is small enough such that

(1) $\|r_0\boldsymbol{K}^{-1}\boldsymbol{T}\|_2 < 1$,

(2) $\omega_0^{-1}r_0^{-1} > \lambda_{max}$

These two assumptions can be satisfied by choosing a small value of ω_0. Let us look at an example. Suppose the desired pattern is given as $v = [1, -1, 1, -1]^T$, $\mathcal{S}^{-1}(v) = \text{diag}(0.5, -0.5, 0.5, -0.5)$, $r_0 = 10\text{K}\Omega$. It can be found that $\lambda_{max} = 0.5\times10^{-4}$ and $\|r_0\boldsymbol{T}\|_2 = 0.5$. Therefore, the assumption (1) and (2) will be satisfied if $\omega_0 < 2$. In general, as far as stability is concerned, the value of ω_0 is very small, e.g., $\omega_0 = 0.01$. Thus, assumption (1) and (2) will always be satisfied by choosing a sufficiently small value of ω_0.

Lemma 2.7.5:

Let $\boldsymbol{T} \in R^{n\times n}$ be a symmetric matrix. Suppose assumption (1) and (2) are satisfied. Then, for $x \in \mathcal{D}_{\omega 0}$,

$$\|(\boldsymbol{T} - r_0^{-1}D_x\mathcal{S}^{-1}(x))^{-1}\|_2 < \|(\boldsymbol{T} - \omega_0^{-1}r_0^{-1}\boldsymbol{I})^{-1}\|_2$$

Proof: Let $x \in \mathcal{D}_{\omega 0}$. Define $s_0^{-1}(x_i) = k_i$, $i = 1, ..., n$. Thus, $D_x\mathcal{S}^{-1}(x) = \boldsymbol{K} = \text{diag}(k_1, ..., k_n)$. Since $x \in \mathcal{D}_{\omega 0}$, $k_i > \omega_0^{-1}$ for $i = 1, ..., n$. Since $\boldsymbol{T}$ is symmetric, there exists an orthogonal matrix $\boldsymbol{P}$ such that $\boldsymbol{T} = \boldsymbol{P}\Lambda\boldsymbol{P}^T$, where $\Lambda = \text{diag}(\lambda_1, ..., \lambda_n)$ and where λ_i is the *ith* eigenvalue of $\boldsymbol{T}$.

It is observed that

$$\|(T - \omega_0^{-1} r_0^{-1} I)^{-1}\|_2 = \|(P \Lambda P^T - \omega_0^{-1} r_0^{-1} I)^{-1}\|_2$$

$$= \|(\Lambda - \omega_0^{-1} r_0^{-1} I)^{-1}\|_2 = 1 / (\omega_0^{-1} r_0^{-1} - \lambda_{max})$$

Since $\|r_0 K^{-1} T\|_2 < 1$,

$$\|(T - r_0^{-1} K)^{-1}\|_2$$

$$\leq \frac{\left\| r_0 K^{-1} \right\|_2}{1 - \left\| r_0 K^{-1} \right\|_2 \| T \|_2}$$

$$= \frac{r_0 k_{min}^{-1}}{1 - r_0 k_{min}^{-1} \| \Lambda \|_2}$$

$$= 1 / (r_0^{-1} k_{min} - \lambda_{max})$$

Since $k_{min} > \omega_0^{-1}$, $1 / (r_0^{-1} k_{min} - \lambda_{max}) < 1 / (\omega_0^{-1} r_0^{-1} - \lambda_{max})$ which implies that

$$\|(T - r_0^{-1} D_x S^{-1}(x))^{-1}\|_2 < \|(T - \omega_0^{-1} r_0^{-1} I)^{-1}\|_2$$

This completes the proof. ■

Let us define

$$\beta_0 = \frac{\alpha}{\left\| (T - \omega_0^{-1} r_0^{-1} I)^{-1} \right\|_2 (\alpha + \nu \sqrt{n})} \qquad (2.7.9)$$

As we know, $f(T, x^0) = Tx^0 - r_0^{-1}S^{-1}(x^0) = 0$, and $f(T+\delta T, y) = (T + \delta T)y - r_0^{-1}S^{-1}(y)$. In the following, assume that $T+\delta T \in \mathcal{A}$ which is defined in (2.7.8). This assumption is generally satisfied due to the particular nonlinearity of the sigmoidal function, $s_0(.)$.

Theorem 2.7.6:

There exists $y \in \mathcal{B}_\alpha(x^0)$ such that $f(T+\delta T, y) = 0$ if $\|\delta T\|_2 \le \beta_0$.

Proof: Suppose $\|\delta T\|_2 \le \beta_0$. Since $T+\delta T \in \mathcal{A}$, by Theorem 3.4, there is $y \in \mathcal{D}_{\omega 0}$ such that $f(T+\delta T, y) = 0$. Since $\mathcal{B}_\alpha(x^0) \subset \mathcal{D}_{\omega 0}$, there are two cases: $y \in \mathcal{B}_\alpha(x^0)$ or $y \in \mathcal{D}_{\omega 0} \setminus \mathcal{B}_\alpha(x^0)$. The first case is desired. We want to show that $y \notin \mathcal{D}_{\omega 0} \setminus \mathcal{B}_\alpha(x^0)$.

Suppose $f(T+\delta T, y) = 0$ with $y \in \mathcal{D}_{\omega 0} \setminus \mathcal{B}_\alpha(x^0)$. Then

$$(T + \delta T)(x^0 + \delta y) - r_0^{-1}S^{-1}(x^0 + \delta y) = 0$$

where $y = x^0 + \delta y$. It follows that

$$(T - r_0^{-1}D_x S^{-1}(z) + \delta T)\delta y + \delta T x^0 = 0$$

where $z \in L(x^0, y)$. By Lemma 2.7.5,

$$\|\delta T\|_2 \le \beta_0 \le 1 / \|(T - \omega_0^{-1} r_0^{-1} I)^{-1}\|_2$$

$$\le 1 / \|(T - r_0^{-1}D_x S^{-1}(z))^{-1}\|_2$$

By Lemma 2.7.3, the matrix $T - r_0^{-1}D_x S^{-1}(z) + \delta T$ is non-singular. Thus,

$$\delta y = -\,(T - r_0^{-1}D_x S^{-1}(z) + \delta T)^{-1}\delta T x^0$$

It follows that

$$\|\delta y\|_2 = \|(T - r_0^{-1}D_x S^{-1}(z) + \delta T)^{-1}\delta T x^0\|_2$$

$$\leq \|(T - r_0^{-1}D_x S^{-1}(z) + \delta T)^{-1}\|_2 \|\delta T\|_2 \|x^0\|_2$$

$$\leq \frac{\left\| (T - r_0^{-1}D_x S^{-1}(z))^{-1} \right\|_2 \| \delta T \|_2 \nu\sqrt{n}}{1 - \left\| (T - r_0^{-1}D_x S^{-1}(z))^{-1} \right\|_2 \| \delta T \|_2}$$

$$\leq \alpha$$

which implies that $y = x^0 + \delta y \in \mathcal{B}_\alpha(x^0)$, contradicting the assumption $y \in \mathcal{D}_{\omega 0} \setminus \mathcal{B}_\alpha(x^0)$. Thus, there exists $y \in \mathcal{B}_\alpha(x^0)$ such that $f(T+\delta T, y) = 0$. This completes the proof. ■

Let $\varepsilon > 0$. Define

$$\gamma_0 = Min\left\{\beta_0, \frac{\varepsilon}{(\upsilon\sqrt{n} + \alpha)\left\| (T - \omega_0^{-1} r_0^{-1} I)^{-1} \right\|_2}\right\} \qquad (2.7.10)$$

We have the following theorem:

Theorem 2.7.7:

Let x^0 be a binary equilibrium of (3.3), $y^0 \in \mathcal{B}_\alpha(x^0)$ be an equilibrium of (3.4). If

$\|\delta T\|_2 \le \gamma_0$, then,

$$\|y^0 - x^0\|_2 \le \varepsilon \qquad \text{if } \beta_0 \ge \varepsilon / [(\upsilon(n)^{1/2}+\alpha)\|(T - \omega_0^{-1}r_0^{-1}I)^{-1}\|_2]; \tag{2.7.11a}$$

$$\|y^0 - x^0\|_2 \le \beta_0(\upsilon(n)^{1/2}+\alpha)\|(T - \omega_0^{-1}r_0^{-1}I)^{-1}\|_2 \qquad \text{otherwise} \tag{2.7.11b}$$

Proof: Since $x^0, y^0 \in \mathcal{B}_\alpha(x^0)$, $D_x f(z)$ is non-singular for $z \in L(x^0, y^0)$. As we know, $D_x f(z) = T - r_0^{-1}D_x S^{-1}(z)$. By equation (3.5), Lemma 2.7.5, and the fact that $\|y^0\|_2 \le \|x^0\|_2 + \|\delta y\|_2 \le \upsilon(n)^{1/2}+\alpha$,

$$\|y^0 - x^0\|_2 \le (\upsilon(n)^{1/2}+\alpha)\|\delta T\|_2\|(D_x f(z))^{-1}\|_2$$

$$\le (\upsilon(n)^{1/2}+\alpha)\gamma_0\|(T - r_0^{-1}D_x S^{-1}(z))^{-1}\|_2$$

$$\le (\upsilon(n)^{1/2}+\alpha)\gamma_0\|(T - \omega_0^{-1}r_0^{-1}I)^{-1}\|_2$$

If $\beta_0 \ge \varepsilon / [(\upsilon(n)^{1/2}+\alpha)\|(T - \omega_0^{-1}r_0^{-1}I)^{-1}\|_2]$, $\gamma_0 = \varepsilon / [(\upsilon(n)^{1/2}+\alpha)\|(T - \omega_0^{-1}r_0^{-1}I)^{-1}\|_2]$. Thus, $\|y^0 - x^0\|_2 \le \varepsilon$. Otherwise, $\gamma_0 = \beta_0$. In this case, $\|y^0 - x^0\|_2 \le \beta_0(\upsilon(n)^{1/2}+\alpha)\|(T - \omega_0^{-1}r_0^{-1}I)^{-1}\|_2$. This completes the proof. ■

Collecting all the previous results together, we obtain our final major result:

Theorem 2.7.8 (Persistence of Equilibria):

Let x^0 be a binary equilibrium of (2.7.3). Suppose $\mathcal{D}_{\omega 0} \ne \varnothing$ and $\|\delta T\|_2 \le \gamma_0$. Then

(1) There exists an equilibrium y^0 of the perturbed feedback continuous-time neural network (2.7.4) such that $y^0 \in \mathcal{B}_\alpha(x^0)$.

(2) $\|y^0 - x^0\|_2$ satisfies (2.7.11).

(3) y^0 is asymptotically stable.

It is observed that one has to check that $\mathcal{D}_{\omega 0} \neq \varnothing$ in order to apply Theorem 2.7.7. As we stated earlier, $\mathcal{D}_{\omega 0}$ is always non-empty for the sigmoidal function $s_0(.)$. Thus, the equilibria of the feedback continuous-time neural network (2.7.3) and its perturbed network (2.7.4) are very close as long as $\|\delta T\|_2$ is relatively small. Moreover, we are able to estimate the upper bound on $\|y^0 - x^0\|_2$ once we know the upper bound on $\|\delta T\|_2$.

Let us consider an example. Suppose we are given three patterns:

$$v_1 = \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} \qquad v_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \qquad v_3 = \begin{bmatrix} -1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$$

Let the sigmoidal function s_0 be chosen such that $s_0^{-1}(1) = 3$, and $s_0^{-1}(-1) = -3$. By using the analytical recursive algorithm in [2], we find weight matrix T as follows:

$$T = r_0^{-1} \begin{bmatrix} 0.0 & -3 & -3 & -3 \\ -3 & 0.0 & -3 & -3 \\ -3 & -3 & 0.0 & -3 \\ -3 & -3 & -3 & 0.0 \end{bmatrix}$$

It can be easily estimated [2] that $\omega_0 = 0.1$. Thus, $\|(T - \omega_0^{-1} r_0^{-1} I)^{-1}\|_2 = 0.1429 r_0$. Based on the sigmoidal function selected, we choose $\alpha = 1.5$. It follows that $\beta_0 = 3r_0^{-1}$. And, it can be calculated that $\gamma_0 = Min\{3r_0^{-1}, 2\varepsilon r_0^{-1}\}$. It follows that $\gamma_0 = 0.2r_0^{-1}$ if $\varepsilon = 0.1$. By Theorem 3.7, $\|y^0 - x^0\|_2 \leq 0.1$ if $\|\delta T\|_2 \leq 0.2r_0^{-1}$. Obviously, $\|y^0 - x^0\|_2 \leq 0.01$ if $\|\delta T\|_2 \leq 0.02r_0^{-1}$.

2.8 A CMOS Circuit of 2n-State Neuron Operating in Subthreshold

The multiple-state neural network can be used to process gray-level images. In the associative memory network, patterns are stored as asymptotically stable equilibria, and then retrieved by some other patterns. As we show in the Section 2.2, neurons with flat regions are capable of storing gray-level images which are asymptotically stable equilibria of continuous-time feedback neural networks. The key to their stability is the slope of the flat region. In other words, for the given pattern $v = [v_1, ..., v_n]^T$, it is asymptotically stable if

$$s_0'(s_0^{-1}(v_i)) < \frac{1}{r_i \sum_{j=1, j\neq i}^{n} |w_{ij}|} \tag{2.8.1}$$

for $i = 1, 2, ..., n$. Therefore, the pattern, $v = [v_1, ..., v_n]^T$, is asymptotically stable if the sigmoidal function $s_0(.)$ has flat regions at $s_0^{-1}(v_i)$, $i = 1, ..., n$. To make this pattern v asymptotically stable, we can adjust $s_0'(s_0^{-1}(v_i))$ for $i = 1, ..., n$, based on (2.8.1). In practice, it is wise to pre-design the neuron circuit such that the slope $s_0'(s_0^{-1}(v_i))$, for $i = 1, ..., n$, is very small. In this case, (2.8.1) will be easily satisfied

Figure 2.8.1 is a schematic diagram of the $2n$-state neuron circuit. Figure 2.8.2 shows the subcircuit serving as a building block. There is a current mirror shared by all building blocks [59]. In general, n building blocks contribute $2n$ voltage regions where the corresponding currents are flat. Each block has its own bias transistor which provides sufficient bias current. All bias transistors are controlled by a common bias gate voltage V_b. By setting small value of V_b (e.g., 0.7V), the neuron circuit operates in sub-threshold. In each block, the two transistors forming the middle leg are assumed to have identical sizes,

and so do the other remaining transistors.

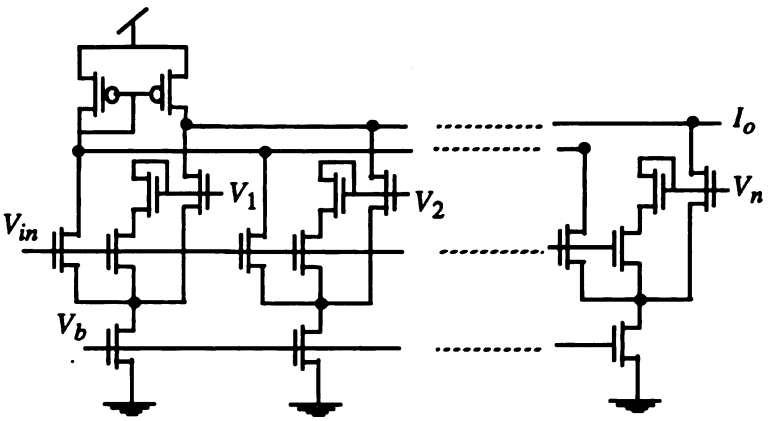


Figure 2.8.1: $2n$-state neuron circuit operating in sub-threshold. There are n building blocks which share a common current mirror. The gate voltage V_b controls all bias transistors.

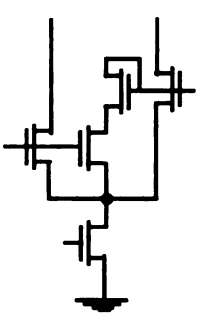


Figure 2.8.2: The sub-circuit serving as a building block.

Figure 2.8.3 is a schematic diagram of the eight-state neuron circuit. This neuron circuit consists of four building blocks. Figure 2.8.4 is the corresponding I_o-V_{in} characteristic. This curve shows consecutive flat regions. First the controllable flat regions at the chosen voltages $V_1 = 1\text{V}$, $V_2 = 2\text{V}$, $V_3 = 3\text{V}$, $V_4 = 4\text{V}$. These regions separate very flat regions which are due to current saturation of each block. Since the slope of the curve in the saturation region is very small, all patterns selected in this region are bound to be asymptotically stable.

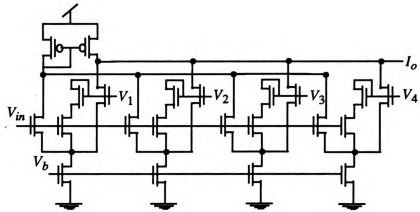


Figure 2.8.3. Schematic diagram of the 8-state neuron circuit.

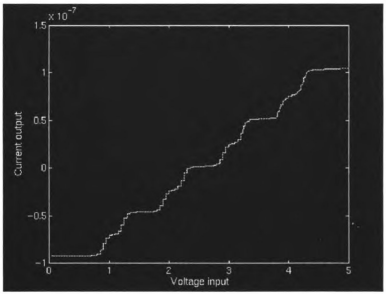


Figure 2.8.4: The I_o-V_{in} characteristic of 8-state neuron with W= 80, w= 1.

To understand the behavior of this $2n$-state neuron circuit, let us analyze the circuit shown in Figure 2.8.5 which is similar to the modified transconductance circuit in [12]. In the sub-threshold region of operation [38], $I_1 = wI_0exp((\kappa V_{in} - V)/V_0)$ and $I_2 = wI_0exp(\kappa V_j - V)/V_0)$, where I_0 is a fabrication parameter, the value of κ is about 0.7, and $V_0 = kT/q$ = 25mV. For simplicity, we will omit V_0 in the expression of current since V_0 basically

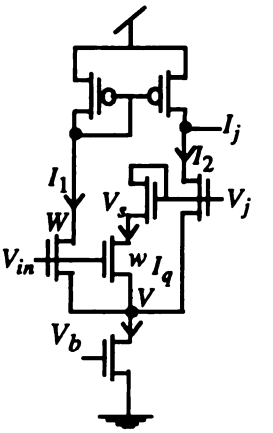


Figure 2.8.5. The modified transconductance circuit.

scales all voltages. The bias current can be expressed as $I_b = wI_0exp(\kappa V_b)(1 - exp(-V))$. The current flowing through the middle two transistors is $I_q = WI_0exp(\kappa V_j)(exp(-V_s) - exp(-V_j)) = WI_0exp(\kappa V_{in})(exp(-V) - exp(-V_s))$. By eliminating V_s, I_q can be solved as

$$I_q = WI_0 \frac{e^{\kappa V_{in}} e^{\kappa V_j} e^{-V}}{e^{\kappa V_{in}} + e^{\kappa V_j}} \tag{2.8.2}$$

Since $I_b = I_1 + I_2 + I_q$, we have

$$e^{-V} = e^{\kappa V_b} \frac{e^{\kappa V_{in}} + e^{\kappa V_j}}{e^{\kappa V_b}(e^{\kappa V_{in}} + e^{\kappa V_j}) + e^{2\kappa V_{in}} + e^{2\kappa V_j} + (2 + \frac{W}{w}) e^{\kappa V_{in}} e^{\kappa V_j}} \tag{2.8.3}$$

And since $I_{oj} = I_1 - I_2 = wI_0(exp(\kappa V_{in}) - exp(\kappa V_j))exp(-V)$, we know

$$I_{oj} = wI_0 e^{\kappa V_b} \frac{e^{2\kappa V_{in}} - e^{2\kappa V_j}}{e^{\kappa V_b}(e^{\kappa V_{in}} + e^{\kappa V_j}) + e^{2\kappa V_{in}} + e^{2\kappa V_j} + (2 + \frac{W}{w}) e^{\kappa V_{in}} e^{\kappa V_j}} \qquad (2.8.4)$$

In the $2n$-state neuron circuit shown in Figure 2.8.1, all building blocks share the same current mirror. Consequently, the output current, I_o, is the summation of all $I_{oj}, j = 1, ..., n$, i.e.,

$$I_o = \sum_{j=1}^{n} I_{oj} = wI_0 e^{\kappa V_b} \sum_{j=1}^{n} \frac{e^{2\kappa V_{in}} - e^{2\kappa V_j}}{e^{\kappa V_b}(e^{\kappa V_{in}} + e^{\kappa V_j}) + e^{2\kappa V_{in}} + e^{2\kappa V_j} + (2 + \frac{W}{w}) e^{\kappa V_{in}} e^{\kappa V_j}} \qquad (2.8.5)$$

The slope of the I_o-V_{in} characteristic at V_j is given by

$$\beta_j = \left.\frac{dI_o}{dV_{in}}\right|_{V_j} = \sum_{j=1}^{n} \frac{\kappa w I_0 e^{\kappa V_b}}{2 + 0.5(\frac{W}{w}) + e^{\kappa(V_b - V_j)}} \qquad (2.8.6)$$

In this paper, for simplicity, we choose $w = 1$. Thus,

$$\beta_j = \left.\frac{dI_o}{dV_{in}}\right|_{V_j} = \sum_{j=1}^{n} \frac{\kappa I_0 e^{\kappa V_b}}{2 + 0.5W + e^{\kappa(V_b - V_j)}} = \sum_{j=1}^{n} \frac{\kappa I_0}{e^{-\kappa V_b}(2 + 0.5W) + e^{-\kappa V_j}} \qquad (2.8.7)$$

It is observed that β_j is inversely related to W which is the ratio of width and length

of those middle transistors. In order to have small value of β_j, we can make those middle transistors having larger ratio of width and length. Figure 2.8.6 shows some curves with W = 20, 40, 60, and 80. It is seen that the slope β_j becomes smaller as W increases. In general, we desire the slope of each flat region to be small. This can be satisfied if we design the $2n$-state neuron circuit with larger value of w (e.g., $W = 80$ or $W/w = 80$.).

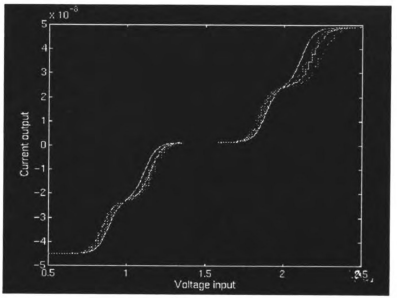


Figure 2.8.6: Four I_o-V_{in} curves of 4-state neuron circuit with different values of W. The curves from left to right have W= 20, 40, 60, and 80. It can be seen that the curve with W = 80 has the smallest slope in the flat region.

Compared with the multiple-state neuron circuit in [60], this circuit is much simpler. Moreover, this circuit operates in the subthreshold region. Thus, the power dissipation is expected to be very small.

2.9 Computer Simulation of 4-State Neural Network with 16 Neurons

We have built a Hopfield-type neural network with 16 neurons. Each neuron has four-state. The four-state neuron is realized by the following sigmoidal function:

$$s(x) = s_1(x) - s_2(x) - s_3(x) + s_4(x) \tag{2.9.1}$$

where

$$s_1(x) = \frac{4e^{2(x-1)}}{e^{2(x-1)} + e^{-2(x-1)}}$$

$$s_2(x) = \frac{4e^{-2(x+1)}}{e^{2(x+1)} + e^{-2(x+1)}}$$

$$s_3(x) = \frac{2e^{-4(x-1)}}{e^{4(x-1)} + e^{-4(x-1)}}$$

$$s_4(x) = \frac{2e^{-4(x+1)}}{e^{4(x+1)} + e^{-4(x+1)}}$$

The characteristic of $s(x)$ is shown in Figure 2.9.1. It can be determined that $s(1) = 1$, $s(-1) = -1$, $s(1.9) = 1.9$, and $s(-1.9) = -1.9$. Thus, the four states of this neuron are defined as: 1.9, 1, -1, -1.9.

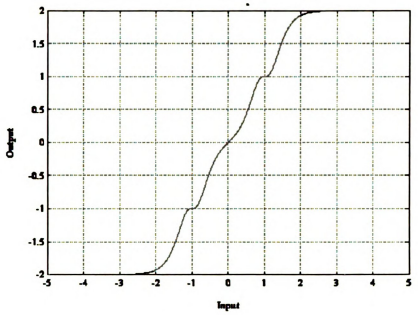
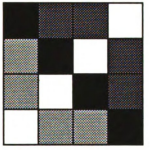
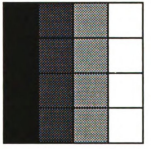


Figure 2.9.1 The characteristic of a 4-state neuron.

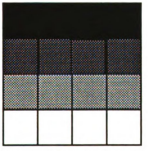
Suppose we are given 4 gray-level patterns:



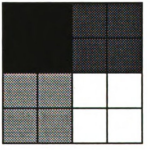
Pattern A



Pattern B

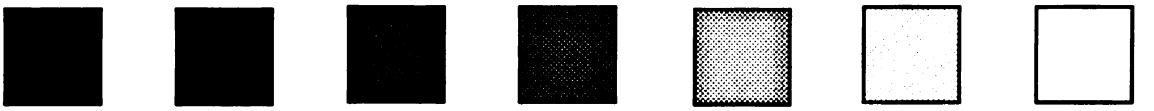


Pattern C



Pattern D

The numerical values associated with the gray-levels are defined as:



1.9 1.5 **1** 0 **-1** -1.5 **-1.9**

Based on this set of gray levels, we obtain the pattern vectors associated with these four desired patterns:

Pattern A: $[1.9, 1, 1, -1.9, -1, 1.9, -1.9, 1, -1, -1.9, 1.9, 1, -1.9, -1, -1, 1.9]^T$

Pattern B: $[1.9, 1, -1, -1.9, 1.9, 1, -1, -1.9, 1.9, 1, -1, -1.9, 1.9, 1, -1, -1.9]^T$

Pattern C: $[1.9, 1.9, 1.9, 1.9, 1, 1, 1, 1, -1, -1, -1, -1, -1.9, -1.9, -1.9, -1.9]^T$

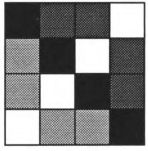
Pattern D: $[1.9, 1.9, 1, 1, 1.9, 1.9, 1, 1, -1, -1, -1.9, -1.9, -1, -1, -1.9, -1.9]^T$

For simplicity, we choose $\boldsymbol{R} = \boldsymbol{C} = \boldsymbol{I}_n$. By applying the recursive algorithm shown in the Section 2.4, we find the weight matrix $\boldsymbol{W}$ which is given as follows:

$\boldsymbol{W}$ =

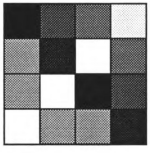
0.00 0.28 0.18 -.24 0.08 0.22 -.21 -.13 0.13 -.06 0.12 -.08 -.14 -.18 -.28 -.25
0.28 0.00 0.10 -.01 0.07 0.13 -.04 0.00 0.00 -.05 -.01 -.07 -.09 -.10 -.13 -.16
0.18 0.10 0.00 0.25 -.08 -.05 -.02 0.03 -.03 -.06 0.13 0.08 -.22 -.19 -.10 -.09
-.24 -.01 0.25 0.00 -.03 -.39 0.42 0.13 -.13 0.06 -.18 0.03 -.22 -.25 0.01 -.36
0.08 0.07 -.08 -.03 0.00 0.19 0.08 0.01 -.01 0.01 -.29 -.22 0.12 0.08 -.07 -.21
0.22 0.13 -.05 -.39 0.19 0.00 -.03 0.22 -.22 -.22 -.19 -.19 -.05 0.05 -.13 0.14
-.21 -.04 -.02 0.42 0.08 -.03 0.00 0.13 -.13 -.01 -.24 -.08 0.01 0.02 0.04 -.10
-.13 0.00 0.03 0.13 0.01 0.22 0.13 0.00 -.33 -.20 -.12 -.01 -.17 -.03 0.00 0.24
0.13 0.00 -.03 -.13 -.01 -.22 -.13 -.33 0.00 0.20 0.12 0.01 0.17 0.03 0.00 -.24
-.06 -.05 -.06 0.06 0.01 -.22 -.01 -.20 0.20 0.00 0.00 -.01 0.16 0.06 0.05 -.18
0.12 -.01 0.13 -.18 -.29 -.19 -.24 -.12 0.12 0.00 0.00 0.29 -.15 -.13 0.01 0.20
-.08 -.07 0.08 0.03 -.22 -.19 -.08 -.01 0.01 -.01 0.29 0.00 -.12 -.08 0.07 0.21
-.14 -.09 -.22 -.22 0.12 -.05 0.01 -.17 0.17 0.16 -.15 -.12 0.00 0.22 0.09 -.08
-.18 -.10 -.19 -.25 0.08 0.05 0.02 -.03 0.03 0.06 -.13 -.08 0.22 0.00 0.10 0.09
-.28 -.13 -.10 0.01 -.07 -.13 0.04 0.00 0.00 0.05 0.01 0.07 0.09 0.10 0.00 0.16
-.25 -.16 -.09 -.36 -.21 0.14 -.10 0.24 -.24 -.18 0.20 0.21 -.08 0.09 0.16 0.00

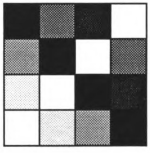
All of those four patterns have been stored as asymptotically stable equilibria. To show this, we choose the Pattern A as an example:



Pattern A

This pattern can be retrieved from the following noisy patterns:





In the circuit implementation, the implemented neuron function is usually not an odd function, like the multiple-state neuron circuit proposed in the Section 2.8. In the following, we will simulate the Hopfield-type feedback neural network with 16 neurons. Each neuron has four states. The output of each individual neuron is positive. This neural function is chosen as follows:

$$s(x) = 5*(tanh(5*(x\text{-}1)) + tanh(5*(x\text{-}2)) + tanh(5*(x\text{-}3)) + tanh(5*(x\text{-}4)))/8 \qquad (2.9.2)$$

$s(x)$ is shown in Figure 2.9.2. The four states are chosen as 5, 3.75, 1.25, 0. It can be determined that $s(4.5) = 5$, $s(3.5) = 3.75$, $s(1.5) = 1.25$, and $s(0.5) = 0$.

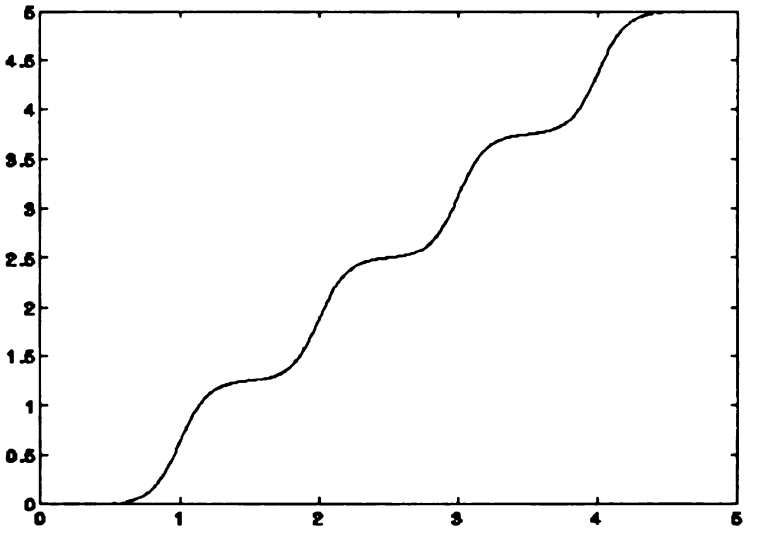
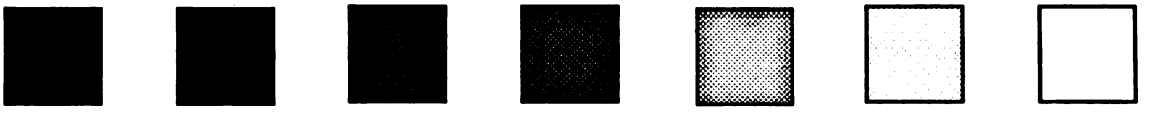


Figure 2.9.2 The characteristic of a 4-state neuron.

The numerical values associated with the gray-levels are defined as:



5 4.25 **3.75** 2.5 **1.25** 0.75 **0**

Based on this set of gray levels, we obtain the pattern vectors associated with those four desired patterns given early:

Pattern A: $[5, 3.75, 3.75, 0, 1.25, 5, 0, 3.75, 1.25, 0, 5, 3.75, 0, 1.25, 1.25, 5]^T$

Pattern B: $[5, 3.75, 1.25, 0, 5, 3.75, 1.25, 0, 5, 3.75, 1.25, 0, 5, 3.75, 1.25, 0]^T$

Pattern C: $[5, 5, 5, 5, 3.75, 3.75, 3.75, 3.75, 1.25, 1.25, 1.25, 1.25, 0, 0, 0, 0]^T$

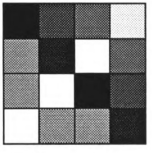
Pattern D: $[5, 5, 3.75, 3.75, 5, 5, 3.75, 3.75, 1.25, 1.25, 0, 0, 1.25, 1.25, 0, 0]^T$

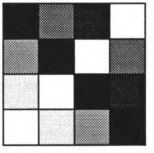
For simplicity, we choose $\boldsymbol{R} = \boldsymbol{C} = \boldsymbol{I}_n$. By applying the recursive algorithm shown in the Section 2.4, we find the weight matrix $\boldsymbol{W}$ which is given as follows:

$\boldsymbol{W}$ =

0.00 0.15 0.14 0.12 0.14 0.21 0.07 0.08 0.17 0.10 0.17 0.09 0.10 0.07 0.05 0.08
0.15 0.00 0.13 0.18 0.13 0.19 0.09 0.10 0.08 0.06 0.07 0.04 0.05 0.04 0.02 0.03
0.14 0.13 0.00 0.34 0.01 0.03 0.09 0.11 0.06 0.04 0.21 0.14 -.08 -.06 0.02 0.03
0.12 0.18 0.34 0.00 0.08 -.16 0.28 0.11 0.01 0.08 -.04 0.05 -.17 -.20 -.06 -.29
0.14 0.13 0.01 0.08 0.00 0.28 0.13 0.07 0.10 0.09 -.18 -.11 0.19 0.13 0.01 -.08
0.21 0.19 0.03 -.16 0.28 0.00 0.08 0.32 -.11 -.10 -.05 -.05 0.09 0.19 0.02 0.28
0.07 0.09 0.09 0.28 0.13 0.08 0.00 0.08 -.01 0.02 -.13 -.04 0.00 -.01 -.02 -.10
0.08 0.10 0.11 0.11 0.07 0.32 0.08 0.00 -.19 -.12 -.01 0.03 -.12 -.02 -.01 0.15
0.17 0.08 0.06 0.01 0.10 -.11 -.01 -.19 0.00 0.27 0.23 0.08 0.27 0.12 0.08 -.08
0.10 0.06 0.04 0.08 0.09 -.10 0.02 -.12 0.27 0.00 0.09 0.02 0.17 0.07 0.04 -.11
0.17 0.07 0.21 -.04 -.18 -.05 -.13 -.01 0.23 0.09 0.00 0.35 -.03 -.01 0.10 0.28
0.09 0.04 0.14 0.05 -.11 -.05 -.04 0.03 0.08 0.02 0.35 0.00 -.07 -.04 0.04 0.15
0.10 0.05 -.08 -.17 0.19 0.09 0.00 -.12 0.27 0.17 -.03 -.07 0.00 0.17 0.05 -.06
0.07 0.04 -.06 -.20 0.13 0.19 -.01 -.02 0.12 0.07 -.01 -.04 0.17 0.00 0.03 0.04
0.05 0.02 0.02 -.06 0.01 0.02 -.02 -.01 0.08 0.04 0.10 0.04 0.05 0.03 0.00 0.05
0.08 0.03 0.03 -.29 -.08 0.28 -.10 0.15 -.08 -.11 0.28 0.15 -.06 0.04 0.05 0.00

All of those four patterns have been stored as asymptotically stable equilibria. In fact, the following two noisy patterns will retrieve the Pattern A:





2.10 A Schematic Layout for an N-Neuron Feedback Neural Network

As we know from section 2.6, the mathematical model for the feedback continuous-time neural network [21] can be expressed as

$$c_i \frac{du_i}{dt} = \sum_{i=1}^{n} w_{ij} v_j + I_i - \frac{1}{r_i} u_i$$

$$v_i = s_i(u_i) \tag{2.10.1}$$

for $i = 1, ..., n$. A schematic layout for a four-neuron neural network is shown in Figure 2.10.1 where the capacitor and resistor connected to each individual neuron are not shown.

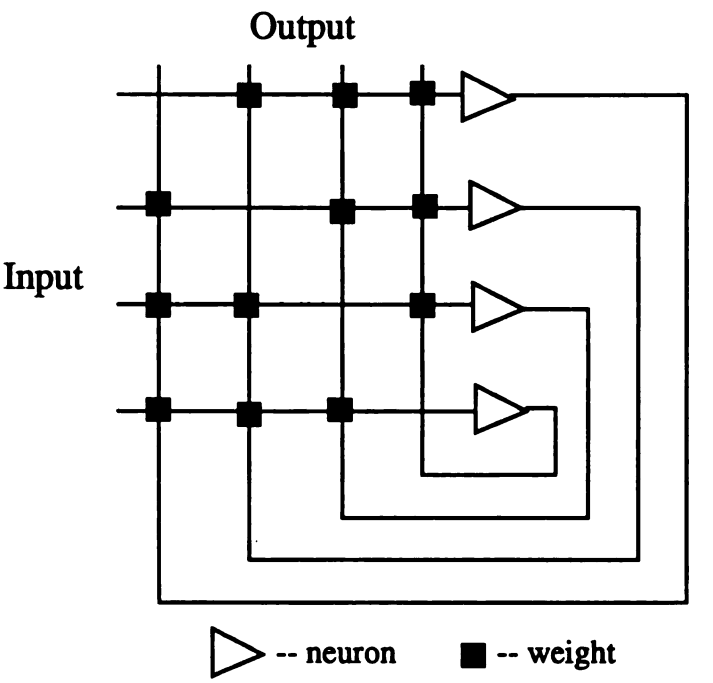


Figure 2.10.1. A schematic layout for a four-neuron network.

The neuron in Figure 2.10.1 can be either two-state or multiple-state. Additional inverters are needed in order to realize the negative weights [21]. The general schematic layout for

an *n*-neuron feedback neural network with four-state neurons are shown in Figure 2.10.2.

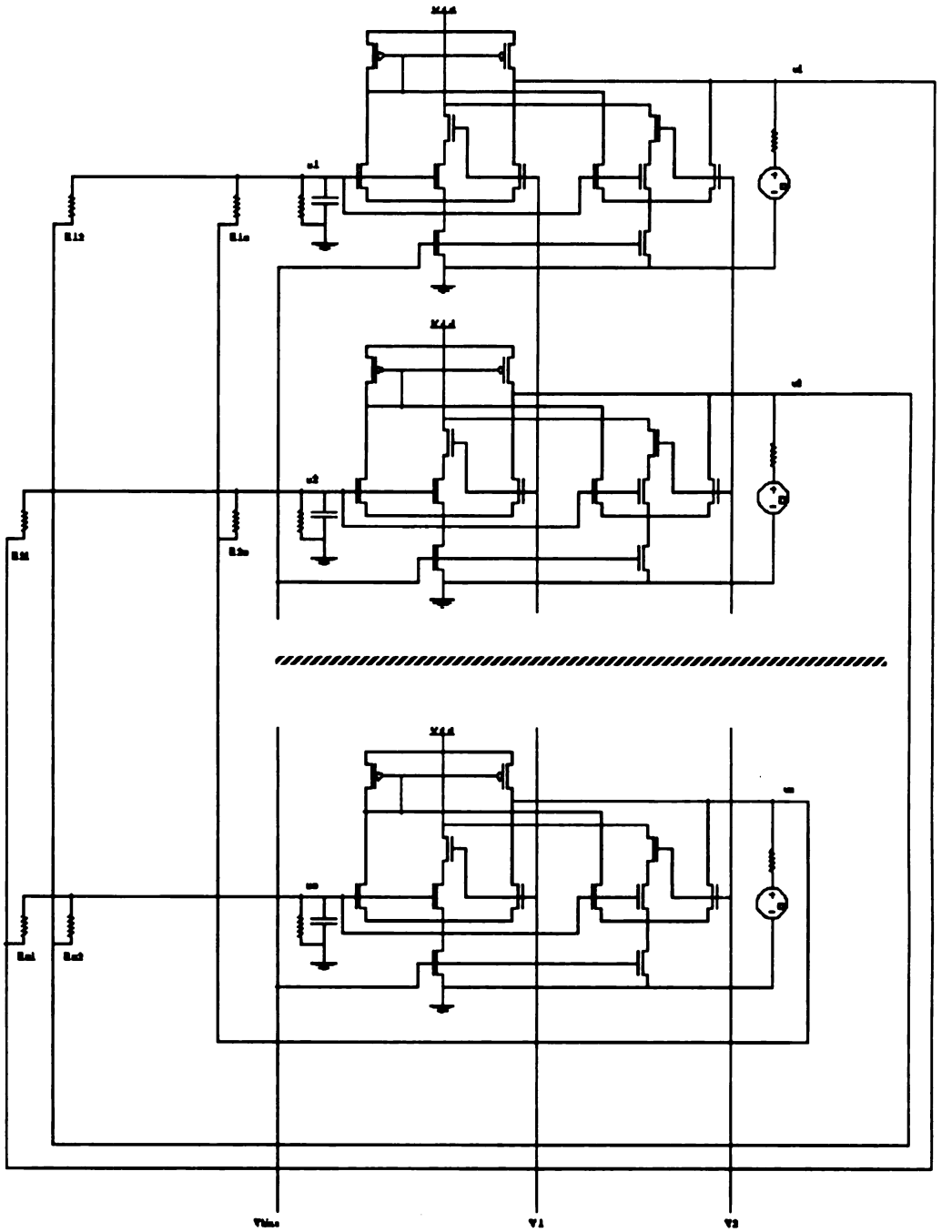


Figure 2.10.2. A schematic layout of an n-neuron network with four-state neurons.

In this general circuit, the voltage states are controlled by V_1 and V_2. The voltage V_{bias} provides the bias voltage for all bias transistors.

CHAPTER 3

Cellular Continuous-Time Feedback Neural Networks with Multiple-State Neurons

3.1 Background

The mathematical model for a Hopfield-type continuous-time neural network is

$$c_i \frac{du_i}{dt} = \sum_{j=1}^{n} w_{ij} v_j - \frac{1}{r_i} u_i$$
$$v_i = s_i(u_i) \tag{3.1.1}$$

for i = 1, 2, ..., n, where $c_i, r_i > 0$, $u_i \in R$ and $v_i \in (-\Delta, \Delta)$ with $0 < \Delta \in R$. $s_i : R \rightarrow (-\Delta, \Delta)$ is a monotone increasing C^1-function with $s_i(0) = 0$, $s_i(x) = \Delta$ (or $-\Delta$) if and only if $x \rightarrow +\infty$ (or $-\infty$). n denotes the number of neurons in the network (3.1.1).

In compact matrix form, (3.1.1) can be written as

$$C\frac{du}{dt} = Wv - R^{-1}u$$
$$v = s(u) \tag{3.1.2}$$

where $u = [u_1, ..., u_n]^T \in R^n$, $v = [v_1, ..., v_n]^T \in R^n$, $C = \text{diag}(c_1, ..., c_n) \in R^{n \times n}$, $R = \text{diag}(r_1, ..., r_n) \in R^{n \times n}$, and $s(u) = [s_1(u_1), ..., s_n(u_n)]^T \in R^n$, and T denotes the transpose of matrix.

For the network with n neurons, the dimension of the pattern vector is n. In the circuit implementation of the Hopfield-type neural network, the size of each network is quite

limited due to the high interconnectivity; in other words, the value of n cannot be too large. In this sense, it is impossible to build a neural network circuit in a single chip with as many neurons as is often required in real applications, such as pattern recognition.

The cellular neural network (CNN) (for example, [10], [33]) provides a useful approach to connect small networks (cells) together. In this large network, each cell is treated as an individual group performing certain tasks. Due to the large number of neurons in the large network, the network capacity is expected to be large.

In the design of CNN, one needs to consider the structure of the large network, its stability and the way to find weights. In this chapter, we will address those basic questions.

3.2 Structure of the Cellular Hopfield-Type Neural Networks

First, we consider the case of two small networks. Suppose we are given two small Hopfield-type neural networks, $S1$ and $S2$, which are described as follows:

$S1$:

$$c_i^1 \frac{du_i^1}{dt} = \sum_{j=1}^{n} w_{ij}^1 v_j^1 + I_i^1 - \frac{1}{r_i^1} u_i^1 \tag{3.2.1}$$

$$v_i^1 = s_i(u_i^1)$$

and $S2$:

$$c_i^2 \frac{du_i^2}{dt} = \sum_{j=1}^{N} w_{ij}^2 v_j^2 + I_i^2 - \frac{1}{r_i^2} u_i^2$$

$$v_i^2 = s_i(u_i^2) \tag{3.2.2}$$

for $i = 1, 2, ..., n$, where n denotes the number of neurons in each network. These two small networks, $S1$ and $S2$, can be viewed as two independent electronic chips with a fixed internal structure. Assume $I_i^1 = I_i^2 = 0$.

Let us combine these two small networks to form a large network. The corresponding large network model is expressed as:

$$\begin{aligned} C^1\frac{du^1}{dt} &= W^1v^1 + A^1v^1 + A^2v^2 - (R^1)^{-1}u^1 \\ v^1 &= s(u^1) \end{aligned} \tag{3.2.3}$$

$$\begin{aligned} C^2\frac{du^2}{dt} &= W^2v^2 + A^2v^1 + A^3v^2 - (R^2)^{-1}u^2 \\ v^2 &= s(u^2) \end{aligned} \tag{3.2.4}$$

where $A^j = \text{diag}(a_1^j, ..., a_N^j)$ for j = 1, 2, 3.

Define $v = [(v^1)^T, (v^2)^T]^T$, $u = [(u^1)^T, (u^2)^T]^T$. Then, u and v satisfy the following equations:

$$\begin{aligned} C\frac{du}{dt} &= Wv + Kv - R^{-1}u \\ v &= S(u) \end{aligned} \tag{3.2.5}$$

where $W = \text{diag}(W^1, W^2)$, $C = \text{diage}(C^1, C^2)$, $R = \text{diag}(R^1, R^2)$, and

$$K = \begin{bmatrix} A^1 & A^2 \\ A^2 & A^3 \end{bmatrix} \qquad S(u) = \begin{bmatrix} s(u^1) \\ s(u^2) \end{bmatrix}$$

The model in (3.2.5) represents the cellular Hopfield-type neural network with two cells. The diagram associated with the model of (3.2.5) is shown in Figure 3.2.1.

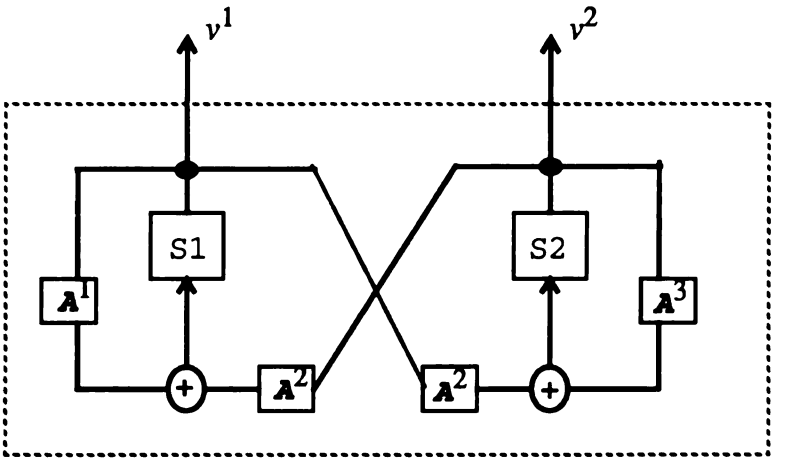


Fig. 3.2.1 Hopfield cellular neural network built on two small network

Now, let us extend the structure of the network with two small networks to M small networks. Suppose we are given M small Hopfield-type neural networks. The cellular Hopfield neural network is described as follows:

$$C^k\frac{du^k}{dt} = W^k v^k + \sum_{l=1}^{M} A_k^l v^k - (R^k)^{-1}u^k$$
$$v^k = s(u^k) \tag{3.2.6}$$

where

$C^k = \text{diag}(c_1^k, ..., c_n^k)$,

$A_k^j = \text{diag}(a_{k1}^j, ..., a_{kn}^j)$; $A^i_j = A^j_i$ for $i, j = 1, 2, ..., M$,

$R^k = \text{diag}(r_1^k, ..., r_n^k)$,

W^k is the weight matrix of the *kth* small network,

$k = 1, ..., M$.

The overall system is defined as:

$$\begin{aligned} C\frac{du}{dt} &= Wv + Kv - R^{-1}u \\ v &= S(u) \end{aligned} \tag{3.2.7}$$

where

$$\begin{aligned} &C = \text{diag}(C^1, ..., C^M), \\ &W = \text{diag}(W^1, ..., W^M), \\ &R = \text{diag}(R^1, ..., R^M), \\ &u = [(u^1)^T, ..., (u^M)^T]^T, \\ &v = [(v^1)^T, ..., (v^M)^T]^T, \\ &S(u) = [(s(u^1))^T, ..., (s(u^M))^T]^T, \end{aligned}$$

$$K = \begin{bmatrix} A_1^1 & A_1^2 & \cdots & A_1^M \\ A_2^1 & A_2^2 & \cdots & A_2^M \\ \cdots & \cdots & \cdots & \cdots \\ A_M^1 & A_M^2 & \cdots & A_M^M \end{bmatrix}$$

It is easy to see that the dimension of K, C, R, W is $Mn \times Mn$, and the dimension of u and v is Mn. This means the cellular Hopfield-type neural network represented in (3.2.7) has Mn neurons. It is observed that the matrix K is symmetric.

The network (3.2.7) can be transformed to

$$\begin{aligned} C\frac{du}{dt} &= (W + K)v - R^{-1}u \\ v &= S(u) \end{aligned} \tag{3.2.8}$$

which has the same structure as the Hopfield-type neural network (3.1.2) if we treat the matrix $W + K$ as the weight matrix. Therefore, the stability results in the Section 2.2 about the Hopfield-type neural network (3.1.2) can be all applied to the network (3.2.8). Moreover, by the same argument in Section 2.5, the capacity of this large network (3.2.7) is at least $0.5Mn$. Thus, the network capacity is increased by at least M times.

In a real large network design, K is a symmetric constant matrix. Theoretically, the elements of K can be arbitrary. But, in the circuit implementation, those values should be small to reduce the network power dissipation since they are equivalent to the conductance. One may choose the elements of K in the same order of network local resistance.

In particular, for the larger network with adjacent connections, the matrix K can be expressed as:

$$K = \begin{bmatrix} O & A_1^2 & O & \cdots & \cdots & O \\ A_2^1 & O & A_2^3 & \ddots & \cdots & O \\ \cdots & \ddots & \cdots & \cdots & \ddots & \cdots \\ O & O & O & \ddots & O & A_{M-1}^M \\ O & O & O & \cdots & A_M^{M-1} & O \end{bmatrix}$$

Thus, the larger network (3.2.7) represents the locally connected network, which is also referred to as cellular neural network [10].

In this locally connected larger network, each individual cell can also be locally connected. The mathematical model of this locally connected smaller network is:

$$C\frac{du}{dt} = Wv - R^{-1}u \tag{3.2.9}$$

where

$$W = \begin{bmatrix} w_{11} & w_{12} & 0 & \cdots & \cdots & 0 \\ w_{12} & w_{22} & w_{23} & \cdots & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & w_{n-1,n-1} & w_{n-1,n} \\ 0 & 0 & 0 & \cdots & w_{n,n-1} & w_{n,n} \end{bmatrix}$$

By the same argument in section 2.3, for a desired pattern, v, the weight matrix W can be by the following equation:

$$Px = b$$

where

$$P = \left[\begin{array}{ccc|ccccc} v_1 & & & v_2 & & & & \\ & v_2 & O & v_1 & v_3 & & O & \\ & & & & v_2 & \cdots & & \\ O & & & & O & & v_{n-2} & v_n \\ & & v_n & & & & & v_{n-1} \end{array}\right]_{n \times (2n-1)}$$

$$x_i = \begin{cases} w_{i,i} & 1 \le i \le n \\ w_{i-n,\, i+1-n} & n+1 \le i \le 2n-1 \end{cases}$$

Therefore, the recursive algorithm presented in the section 2.4 can be applied.

3.3 A Parallel Algorithm for Finding Weights

Let v be an equilibrium point of the cellular Hopfield-type neural network (3.2.7).

Then v satisfies

$$W v + A v - R^{-1} S^{-1}(v) = 0 \tag{3.3.1}$$

which is equivalent to

$$\begin{aligned} & W^1 v^1 + \sum_{l=1}^{M} A_1^l v^1 - (R^1)^{-1} s^{-1}(v^1) = 0 \\ & \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots \\ & W^M v^M + \sum_{l=1}^{M} A_M^l v^M - (R^M)^{-1} s^{-1}(v^M) = 0 \end{aligned} \tag{3.3.2}$$

where $v = [(v^1)^T, ..., (v^M)^T]^T$, T represents the matrix transpose,. It is observed from the above equations that W^k, $k = 1, ..., M$, are independent during the "learning" process, where each W^k represents the weight matrix of the *kth* small network. In other words, W^k can be solved from the following linear equation:

$$W^k v^k = -\sum_{l=1}^{M} A_k^l v^k + (R^k)^{-1} u^k \tag{3.3.3}$$

The algorithm discussed in the Section 2.3 can be used to find the weight matrix W^k for the given pattern v. Since those M matrix equations in (3.3.2) are all identical in terms of weight matrices, all weight matrices can be found in the same way. Moreover, those weight matrices can be found in parallel since all equations are independent in terms of weight matrices. In this sense, the weights for all cells will be found simultaneously.

For multiple patterns case, we can apply the Morris-Odell's theorem to find all weight matrices. The recursive algorithm discussed in the Section 2.4 can be applied in the cellular Hopfield-type continuous-time feedback neural networks.

CHAPTER 4

Dendro-Dendritic Artificial Neural Network with Multiple-State Neurons

4.1 Background

The mathematical model of the dendro-dendritic neural network is given as:

$$c_i \frac{du_i}{dt} = \sum_{j=1, j\neq i}^{n} T_{ij}(u_j - u_i) + T_{ii}(v_i - u_i) + I_i - \frac{1}{r_i}u_i \qquad (4.1.1a)$$

$$v_i = s_i(u_i) \qquad (4.1.1b)$$

for $i = 1, ..., n$, where n denotes the number of neurons in the network; u_i is the input of the *ith* neuron, v_i is the output of the *ith* neuron; c_i is the capacitor and r_i is the resistor connected to the *ith* neuron; $s_i(.)$ the activation function which is a bounded monotone C^1-function. One type of circuit configurations of the dendro-dendritic neural network for three-neuron is shown in Figure 4.1.1 where connections are made by resistors.

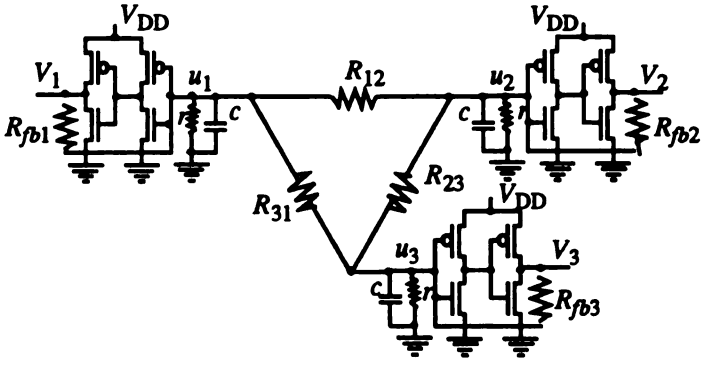


Figure 4.1.1: One type of circuit configurations of DANN with three neurons.

It is reasonable to assume $c_i = c_0$, $r_i = r_0$ and $s_i = s_0$ for $i = 1, ..., n$. Thus, (4.1.1) can be rewritten as follows:

$$c_0 \frac{du_i}{dt} = \sum_{j=1, j\neq i}^{n} T_{ij}(u_j - u_i) + T_{ii}(v_i - u_i) + I_i - \frac{1}{r_0} u_i \tag{4.1.2a}$$

$$v_i = s_0(u_i) \tag{4.1.2b}$$

In compact matrix form, (4.1.2) reads

$$c_0 du/dt = -Wu + Tv + I \tag{4.1.3a}$$

$$v = S(u) \tag{4.1.3b}$$

where $W = [w_{ij}]$, $w_{ij} = -T_{ij}$ for $i \neq j$ and $w_{ii} = \sum_{j=1}^{n} T_{ij} + r_0^{-1} T = \text{diag}(T_{11}, ... T_{nn})$,

$I = [I_1, ..., I_n]^T$, $u = [u_1, ..., u_n]^T$, $v = [v_1, ..., v_n]^T$, and $S(u) = [s_0(u_1), ..., s_0(u_n)]^T$.

The energy function is defined as [49]:

$$E = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} T_{ij} u_i u_j - \sum_{i=1}^{n} T_{ii} \int_0^{u_i} s_0(x)\, dx - \sum_{i=1}^{n} I_i u_i + \frac{1}{2} \sum_{i=1}^{n} \left(r_0^{-1} + \sum_{j=1}^{n} T_{ij} \right) u_i^2 \tag{4.1.4}$$

or in vector form:

$$E(u) = \frac{1}{2} u^T W u - I^T u - \sum_{i=1}^{n} T_{ii} \int_0^{u_i} s_0(x)\, dx \tag{4.1.5}$$

which is continuously decreasing with respect to time t and vanishes only at an equilibrium of the network (4.1.3). This also implies that no limit cycle (or chaos) of (4.1.3) exists. Now define the Hessian matrix at a point u as $J_{\nabla E}(u) = W - TD_u S(u)$.

Since the activation function s_0, of each neuron, is a bounded C^1-function, (4.1.3) has a unique solution for $t \geq 0$ starting from each initial condition $u(0)$. As shown in [49], system (4.1.3) is a gradient-like system where the isolated local minima of (4.1.4) are asymptotically stable. And, by the *Inverse Function Theorem*, u is isolated if u is an equilibrium of (4.1.3) such that $\det(J_{\nabla E}(u)) = \det(W - TD_u S(u)) \neq 0$. Moreover, u is a local minimum of (2.4) if $J_{\nabla E}(u) > 0$. It is also shown in [49] that all trajectories of (4.1.3) converge to a bounded and closed (compact) region. Hence, there is only finite number of isolated equilibria of (4.1.3). In summery, the dendro-dendritic neural network has the following important properties:

(1) For the energy function defined in (4.1.4), $dE(u)/dt \leq 0$, and $dE(u)/dt = 0$ if and only if u is an equilibrium point of (4.1.3).

(2) There exists a unique solution of (4.1.3) for $t \geq 0$ for each initial condition $u(0)$.

(3) Isolated local minimum of $E(u)$ is an asymptotically stable equilibrium of (4.1.3).

(4) There are only finite number of equilibria of (4.1.3).

(5) Let u^0 be an equilibrium point of (4.1.3). u^0 is isolated if $\det(J_{\nabla E}(u^0)) \neq 0$.

(6) Let u^0 be an equilibrium point of (4.1.3). u^0 is a local minimum of function $E(v)$ if and only if the Jacobian matrix $J_{\nabla E}(u^0) > 0$.

(7) All solutions are bounded.

(8) No limit cycles nor other forms of recurrent solutions exit.

Thus, the dendro-dendritic neural network has all the desired properties of the Hopfield-type neural network. But, compared with the Hopfield-type neural network, the dendro-dendritic neural network has the following advantages:

(1) Less number of weights. For an n-neuron network, the Hopfield-type neural network uses a maximum of n^2-n weights, the DANN network uses a maximum of $n(n+1)/2$ weights.

(2) Natural symmetric weights. In the DANN network circuit, $w_{ij} = w_{ji}$ naturally. In the Hopfield-type neural network, this property can never be true due to circuit hardware mismatch.

The circuit shown in Figure 4.1.1 has connections made by resistors. In the neural chip design, resistors take large area. Thus, we expect to replace the resistors by some other circuit devices, like for example MOS transistors. Another type of three-neuron DANN network is shown in Figure 4.1.2 where all neurons are connected by nMOS transistors which are controlled by their gate voltages.

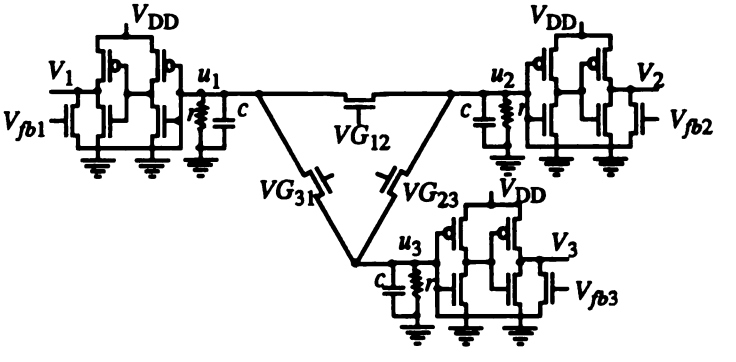


Figure 4.1.2: Another type of circuit configurations of DANN with three neurons.

4.2 Stability Analysis

In the following analysis, we assume s_0: $(-\infty, +\infty) \rightarrow (\Delta_1, \Delta_2)$ for $\Delta_2 \geq \Delta_1 > 0$, $s_0(0) = 0$, and the desired patterns are binary, e.g., $v = [v_1, ..., v_n]^T$ with $v_i = v_-$ or v_+. Corresponding to $v = [v_1, ..., v_n]^T$, we have $u = [u_1, ..., u_n]^T$ with $u_i = u_-$ or u_+, where $s_0(u_-) = v_-$ and $s_0(u_+) = v_+$.

For the energy function defined in (4.1.5), it is easy to verify that

$$\nabla E(u) = Wu - I - TS(u)$$
$$J_{\nabla E}(u) = W - TD_u S(u)$$

where $\nabla E(u)$ is the gradient and $J_{\nabla E}(u)$ is the Hessian of $E(u)$. A stability theorem is stated as follows:

Stability Theorem: Let u^0 be an equilibrium of (4.1.3) where $T_{ij} \geq 0$ $i \neq j$ and $T_{ii} > 0$. Then u^0 is asymptotically stable if $s_0'(u_i^0) < 1 / (r_0 T_{ii})$ for all i.

Proof: Suppose $T_{ij} \geq 0$ for $i \neq j$, $T_{ii} > 0$. Define $B = W - TD_u S(u^0)$. Then, $b_{ii} = w_{ii} + T_{ii} s_0'(u_i^0) = \sum_{j=1}^{n} T_{ij} + r_0^{-1} - T_{ii} s'_0(u_i^0)$, $b_{ij} = w_{ij} = -T_{ij}$ for $i \neq j$. Suppose $s_0'(u_i^0) < 1 / (r_0 T_{ii})$ for all i. Then, $b_{ii} > 0$. It is easy to see that $|b_{ii}| > \sum_{j=1, i \neq j}^{n} |b_{ij}|$. By *Gershgorin Circle Theorem* [13], B has only positive eigenvalues. Therefore, $J_{\nabla E}(u^0) > 0$, which implies u^0 is a local minimum of $E(u)$. Thus, u^0 is asymptotically stable. This completes the proof. ■

4.3 An Analytical Method for Finding Positive Weights via Linear Programming

First, consider the single pattern case. Let $u = [u_1, ..., u_n]^T$ be an image pattern, where $u_i = u_-$ or u_+. In order to be an equilibrium of the dendro-dendritic neural network (4.1.3), u must satisfy:

$$- Wu + TS(u) + I = 0 \tag{4.3.1}$$

which is equivalent to

$$W' u + TS(u) = R^{-1}u - I \tag{4.3.2}$$

where $w_{ij}' = - w_{ij} = T_{ij}$ for $i \neq j$, $w_{ii}' = - w_{ii} + r_0^{-1}$, $T = \text{diag}(T_{11}, ..., T_{nn})$, $R^{-1} = r_0^{-1}E_n$, where E_n is the identity matrix, $I = [I_1, ..., I_n]^T$. For simplicity, one may set $I_i = 0$ for $i = 1$, ..., n. Thus, (4.3.2) results in:

$$W' u + TS(u) = R^{-1}u = r_0^{-1}E_n u \tag{4.3.3}$$

By proper algebraic operations, (4.3.3) can be transformed into

$$Ax = b \tag{4.3.4}$$

where $x = [x_1, ..., x_N]^T$, $N = n(n+1)/2$ (the number of unknowns in (4.3.4)), and $b = r_0^{-1}u$,

$$A = \left[\begin{array}{ccc|c|c|c} s_0(u_1) - u_1 & & 0 & u_2 - u_1 & u_n - u_1 & \\ & s_0(u_2) - u_2 & & u_1 - u_2 & 0 & 0 \\ 0 & & \ddots & 0 & \ddots & \\ & & s_0(u_n) - u_n & & u_1 - u_n & u_{n-1} - u_n \end{array}\right]$$

$$x_i = \begin{cases} T_{i,i} & 1 \le i \le n \\ T_{1,i+1-n} & n+1 \le i \le 2n-1 \\ \vdots & \vdots \\ T_{n-1,n} & i = N \end{cases} \qquad (4.3.5)$$

Without loss of generality, assume $s_0(u_i) \neq u_i$ for $i = 1, ..., n$. From the structure of A in (4.3.4), it is easy to show the following proposition:

Proposition 4.3.1: Let $u = [u_1, ..., u_n]^T$ be an image pattern with $u_i = u_-$ or $u_+ \neq 0$, A as defined in (4.3.5). Then, rank(A) = n.

The minimum-norm solution of (4.3.4) is given by $x = A^+ b$ where $A^+ = A^T(AA^T)^{-1}$. Once x is solved, all weights, T_{ij} for $i, j = 1, ..., n$, can easily be reconstructed via (4.3.5). Those weights will make u exactly stored as an equilibrium of (4.1.3).

Now, consider the case of multiple patterns. Suppose we are given m binary patterns, $u^1, ..., u^m$. For each vector u^i, we have

$$- Wu^i + TS(u^i) = 0$$

which can be transferred to the equivalent matrix equation: $A^i x = b^i$. Thus, in order to find weights, T_{ij} for $i, j = 1, ..., n$, for storing those m patterns, $A^i x = b^i$, $i = 1, ..., m$, must have common solutions. In other words,

$$\begin{bmatrix} A^1 \\ A^2 \\ \cdots \\ A^m \end{bmatrix} x = \begin{bmatrix} b^1 \\ b^2 \\ \cdots \\ b^m \end{bmatrix} \tag{4.3.6}$$

must have solutions. In the Section 2.4, we have shown how to apply the *Morris-Odell's Theorem* to check the existence of solutions and find common solutions of (4.3.6). Thus, we may assume that common solutions of (4.3.6) exist.

Define $A = [(A^1)^T, ..., (A^m)^T]^T$, $b = [(b^1)^T, ..., (b^m)^T]^T$. Then, the common solutions of (4.3.6) is given by $x = A^+ b + N(A)$, where $N(A)$ is the null space of A. Based on the *Stability Theorem* in Section 4.2, we wish to find non-negative solutions of (4.3.6). In other words, we need to find solutions of:

$$\begin{aligned} Ax &= b \\ x &\geq 0 \end{aligned} \tag{4.3.7}$$

It is observed that the solution of (4.3.7) is not unique. The elements of the solution x are the weights in our neural network. It is desired to have small values of those weights so that they can be implementable as large resistive elements to achieve minimized power dissipation in the network. In other words, the problem can be stated as:

$$\text{Minimize} \quad \sum_{i=1}^{n} x_i \tag{4.3.8a}$$

$$\text{s.t.} \quad Ax = b \tag{4.3.8b}$$

$$x \geq 0 \tag{4.3.8c}$$

which is a standard linear programming problem.

The solutions of (4.3.8b) and (4.3.8c) are called feasible solutions. An optimal feasible solution is one which satisfies (4.3.8). An optimal basic feasible solution is an optimal feasible solution which has exactly p non-zero elements, other elements are all zero, where $p = \text{rank}(A)$. It can be shown [34] that

(1) The feasible solution set K is convex.
(2) If K is non-empty, there exists at least one extreme point of K.
(3) The convex set K possesses at most a finite number of extreme points.
(4) Optimal values of the objective function occurs only at extreme points of K.

Without loss of generality, assume $s_0(u_+) > u_+$ and $s_0(u_-) < u_-$. This assumption can be always satisfied since we are free to set the values of u_+ and u_-. With this assumption, we have the following theorem:

Theorem 4.3.2: Let $u = [u_1, ..., u_n]^T$ be a binary pattern with $u_i = u_-$ or u_+. Then, there exists an optimal basic feasible solution of (4.3.8).

Proof : Let A be defined as in (4.3.4). Partition $A = [A_1, A_2]$, where $A_1 = \text{diag}(s_0(u_1)-u_1, ..., s_0(u_n)-u_n)$. Define $x_1 = A_1^{-1} r_0^{-1} u = \text{diag}(1/(s_0(u_1)-u_1), ..., 1/(s_0(u_n)-u_n)) r_0^{-1} u$. Then, $x =$

$[x_1, 0]^T$ is a basic feasible solution of the linear programming problem (4.3.8) with A and b defined as in (4.3.4). Thus, the feasible solution set K is non-empty, which implies that an optimal basic feasible solution of (4.3.8) exists. This proves the result. ■

Theorem 4.2 guarantees the existence of optimal basic feasible solution of (4.3.8) for single input pattern. In the multiple-pattern case, the situation is quite different. How to find conditions which guarantee the existence of the feasible solution of (4.3.8) for multiple-patterns case remains as a future work. Real-time experiments [47] have shown multiple binary patterns can be stored and retrieved.

Among all numerical algorithms about linear programming, the simplex method is the most significant algorithm. It proceeds from one basic feasible solution to another in such a way as to continuously decrease the value of the objective function until a minimum is reached [34]. Standard packages of the simplex method are available. Therefore, it is easy to find weights for the dendro-dendritic neural network once multiple patterns are provided.

4.4 Computer Simulation of a Three-Neuron DANN Network

Consider a three-neuron dendro-dendritic neural network. The binary input pattern takes two values: 5 volts and -5 volts. And define $u_+ = 2$ volts, $u_- = -2$ volts, which implies $s_0(u_+) = 5 > u_+$ and $s_0(u_-) = -5 < u_-$. Suppose a binary pattern $v = [5, 5, -5]^T$ is given. Corresponding to this pattern, $u = [2, 2, -2]^T$. Since the number of neurons is three, the maximum number of weights $N = 6$. The matrix A in (4.3.4) is constructed as:

$$\mathbf{A} = \begin{bmatrix} 3 & 0 & 0 & 0 & -4 & 0 \\ 0 & 3 & 0 & 0 & 0 & -4 \\ 0 & 0 & -3 & 0 & 0 & 4 \end{bmatrix}$$

Let $r_0 = 1$ KΩ. Then $b = r_0^{-1}u = [0.002, 0.002, -0.002]^T$. The linear programming problem is formulated as follows:

$$\begin{aligned} &\textit{Minimize } x_1 + x_2 + x_3 + x_4 + x_5 + x_6, \\ &\quad s.t.\ Ax = b, \\ &\quad\quad x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0, x_5 \geq 0, x_6 \geq 0. \end{aligned}$$

The solution $[x_1, x_2, x_3, x_4, x_5, x_6]^T = [0.00067, 0.00067, 0.00067, 0, 0, 0]^T$ is an optimal basic feasible solution. Thus, the weights are: $T_{11} = 0.00067$, $T_{22} = 0.00067$, $T_{33} = 0.00067$, $T_{12} = 0$, $T_{13} = 0$, $T_{23} = 0$. The resistors of 1.5 KΩ are needed to implement those weights. From the dendro-dendritic neural network (4.1.1), we know there are no connections among the three neurons. In other word, for this example, the network is decoupled.

From a mathematical point of view, $[x_1, x_2, x_3, x_4, x_5, x_6]^T = [0.00067, 0.00067, 0.00067, 0, 0, 0]^T$ is an optimal basic feasible solution which minimizes the quantity $x_1 +$

$x_2 + x_3 + x_4 + x_5 + x_6$. But from an implementation point of view, this optimal basic feasible solution may not yield good robust performance, e.g. fault tolerance, since the network with those weights are basically decoupled. There is a trade-off between small values of weights and good system performance. If we focus on a better system performance, we may only want to find feasible solutions, instead of optimal basic feasible solutions. In this example, we can find a solution $[x_1, x_2, x_3, x_4, x_5, x_6]^T$ = [0.002, 0.002, 0.002, 0.001, 0.001, 0.001]^T which is only a feasible solution. The corresponding weights are: $T_{11} = T_{22} = T_{33} = 0.002$, $T_{12} = T_{13} = T_{23} = 0.001$. In this case, the network is not decoupled. We can also define the objective function (4.8a) as to *minimize* $\{x_1^2 + ...+ x_N^2\}$ and apply nonlinear programming algorithms. This will result in optimal solutions which will not make the network decoupled.

4.5 Network Implementation and Layout of Three-Neuron DANN Network

We will design the DANN network with multiple-state neurons. For simplicity, we will only design a network with three neurons. Each neuron has four states. In the Section 2.8, we have shown the design of the CMOS circuit of $2n$-state neuron. The schematic diagram of four-state neuron with voltage input and voltage output is shown in Figure 4.5.1.

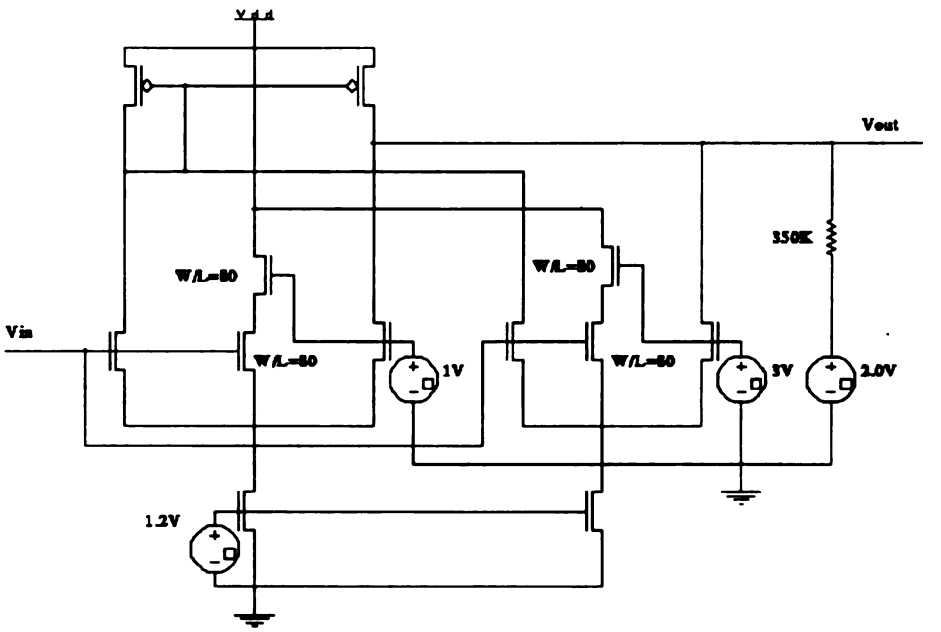


Figure 4.5.1 The schematic diagram of 4-state neuron with voltage input and voltage output.

In this circuit, the resistor 320K and the voltage source 2.0V in the output stage converts the current to voltage. A large resistance is required in order to vary the output voltage over the range of 0 - 5V. The bias voltage is set at 1.2V. One may reduce this bias voltage. The smaller the bias voltage, the larger the resistance in the output stage. In the chip design, one may implement this large resistor outside the chip since this large resistor will require a large layout area.

The PSpice simulation of this four-state neuron circuit is shown in Figure 4.5.2.

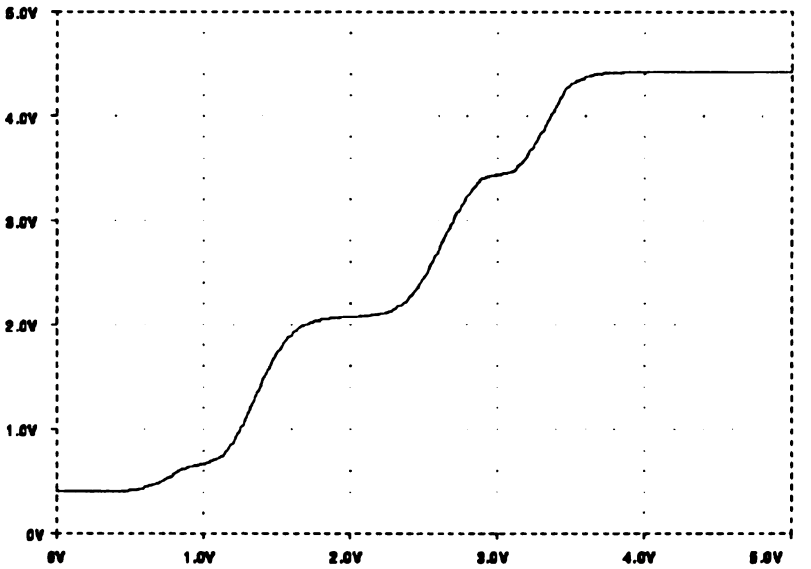


Figure 4.5.2 The PSpice simulation of the circuit in Figure 4.5.1.

In the DANN network, the "sigmoidal" function does not necessarily need to be an odd function. Thus, the four states can be defined at u = 0.5V, 2V, 3V and 4V. With this four-state neuron, we build the DANN network shown in Figure 4.5.3.

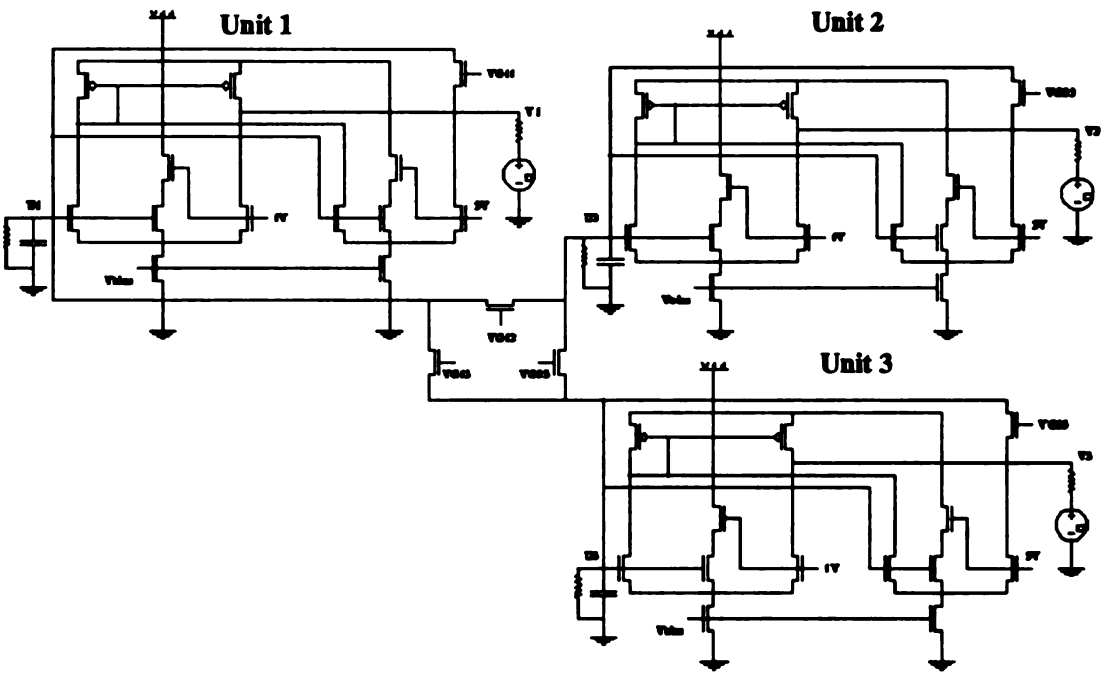


Figure 4.5.3. The DANN network with three neurons.

In this network, each neuron has self-feedback. All neurons are fully connected. These connections are made possible by MOS transistors which are controlled by the gate voltages. These gate voltages can be set globally.

We have made some PSpice simulations for this DANN network. In those simulations, we set all gate voltages at certain level. Some initial voltages are applied to the input nodes. After the transient process is finished, the output voltages of all three neurons converge to desired values. Figure 4.5.4 shows the transient curves of the output voltages of those 3 neurons.

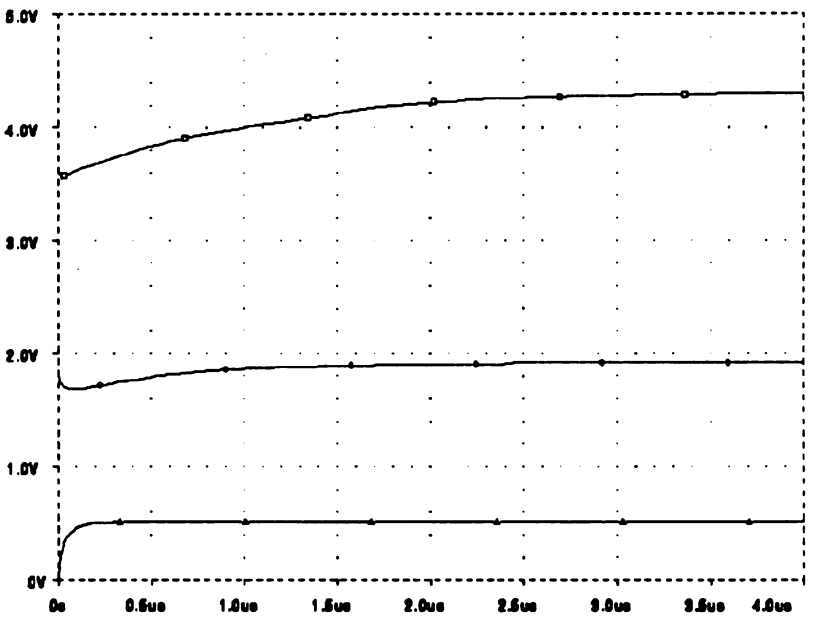


Figure 4.5.4. The transient curves of the output voltages of three neurons.

In this simulation shown in the Figure 4.5.4, we set the initial input voltages of unit 1, unit 2 and uint 3 at 3.5V, 1.8V and 0V, respectively. The steady state of the output of unit 1 converges to 4.306V, unit 2 to 1.912V, and unit 3 to 0.516V. These three voltages, 0.516V, 1.912V and 4.306V are the three states of our multiple-state neuron shown in the Figure 4.5.1 with some small shift in voltage. The gate voltages of self-feedback transis-

tors are all equal to 5.0V. For other three connection transistors, their gate voltages are set at 1.5V.

The network layout of the circuit in Figure 4.5.4 is shown in Figure 4.5.5. In this layout, the resistor and power source in the output stage of each neuron is not included. A large area of the chip is required in order to implement this large resistor. To test this chip, one can externally connect the resistors and power sources to the chip through pins.

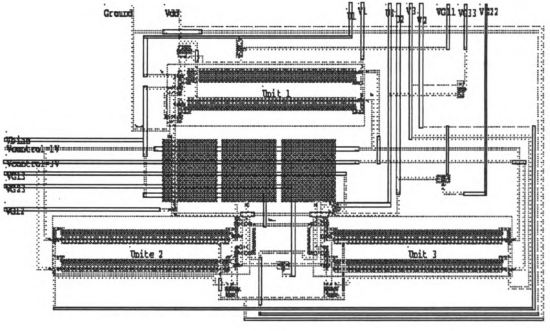


Figure 4.5.5. The layout of the circuit in Figure 4.5.4.

4.6 A Schematic Layout for an N-Neuron DANN Network

As we know from Section 4.1, the mathematical model for the DANN network [49] can be expressed as

$$c_i \frac{du_i}{dt} = \sum_{j=1, j\neq i}^{n} T_{ij}(u_j - u_i) + T_{ii}(v_i - u_i) + I_i - \frac{1}{r_i} u_i$$

$$v_i = s_i(u_i) \tag{4.6.1}$$

for $i = 1, ..., n$. A schematic layout for a four-neuron neural network is shown in Figure 4.6.1 where the capacitor and resistor connected to each individual neuron are not shown.

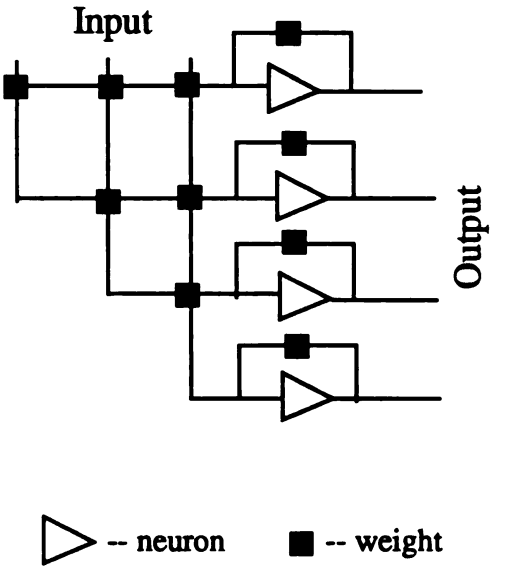


Figure 4.6.1. A schematic layout for afour4-neuron DANN network.

The symbol of neuron in Figure 4.6.1 can be either two-state or multiple-state. The weights can be realized by either resistors or nMOS transistors. The general schematic

layout for an n-neuron feedback neural network with four-state neurons are shown in Figure 4.6.2.

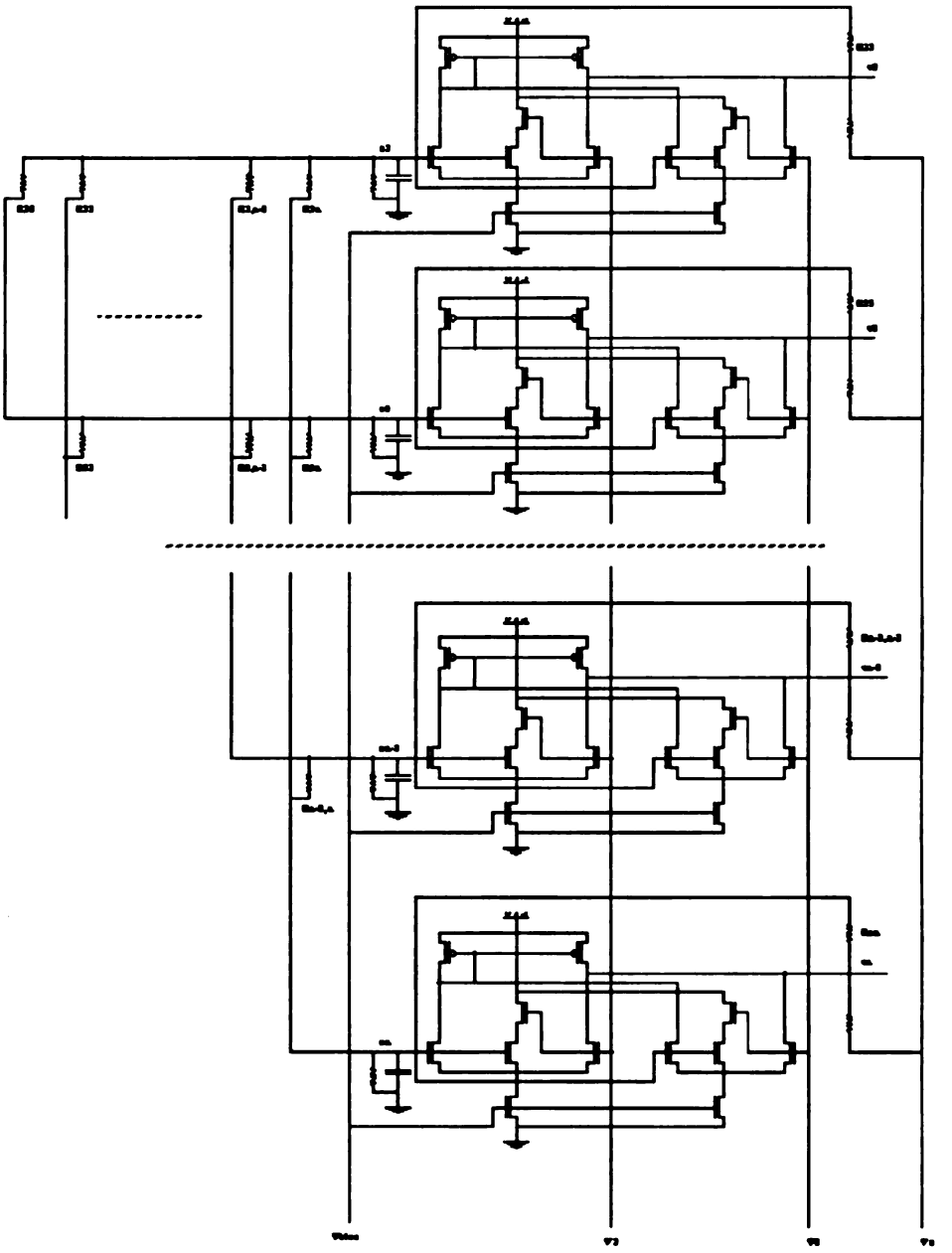


Figure 4.6.2. A schematic layout of an n-neuron DANN with four-state neurons.

In this general circuit, the voltage states are controlled by V_1 and V_2. The voltage V_{bias} provides the bias voltage for all bias transistors.

CHAPTER 5

A Tiny-Chip Layout of a Six-Neuron DANN Network

5.1 Computer Simulation of a Six-Neuron DANN Network

We have built a DANN neural network with six neurons. Each neuron has four-state. The four-state neuron is realized by the following sigmoidal function:

$$s(x) = 0.847(tanh(5(x - 0.5)) + 1) + 0.415(tanh(5(x - 1.5)) + 1) + 0.405(tanh(5(x - 2.5)) + 1) + 0.389(tanh(5(x - 3.5)) + 1) \quad (5.1.1)$$

for $x \in [0, 5]$. In the expression of (5.1.1), all parameters are chosen such that the characteristic of $s(x)$ is approximately matched with the characteristic of the four-state neuron circuit shown in next section. The characteristic of $s(x)$ is shown in Figure 5.1.1. The four states are chosen as: 1V, 2V, 3V and 4V. It can be measures that $s(1) = 1.694$V, $s(2) = 2.514$V, $s(3) = 3.325$V, and $s(4) = 4.103$V.

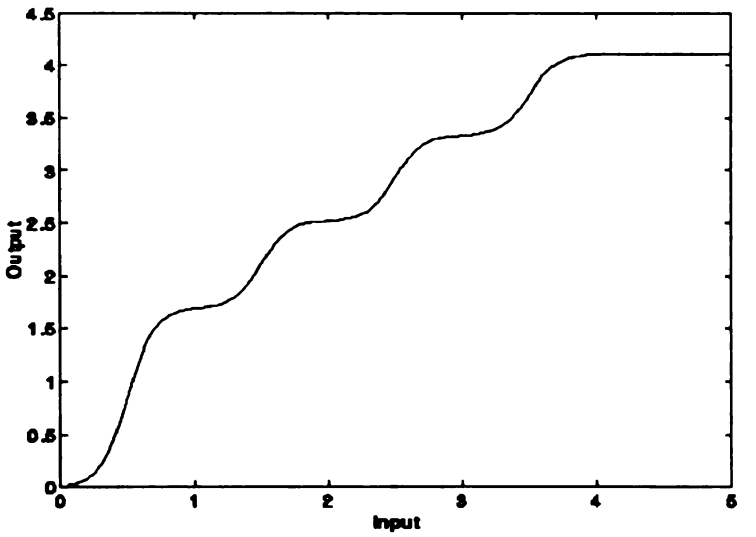
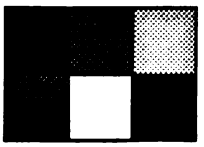


Figure 5.1.1. The characteristic of sigmoidal function $s(x)$.

Suppose we are given the following four-level pattern:



The numerical values associated with those four gray levels are:

4.103 3.325 2.514 1.964

Based on this set of gray levels, the pattern vector is given as

$$[4.103, 3.325, 2.514, 3.325, 1.694, 4.103]^T$$

The corresponding input pattern vector is

$$[4, 3, 2, 3, 1, 4]^T$$

For simplicity, we choose $\boldsymbol{R} = r_0\boldsymbol{I}$. It can be found that

$$x = r_0^{-1}[44.95, 9.51, 3.02, 9.51, 0.014, 44.95, 0.09, 0.09, 0.09, 0.09,$$
$$0.09, 0.09, 0.09, 0.09, 0.09, 0.09, 0.09, 0.09, 0.09, 0.09, 0.09,]^T$$

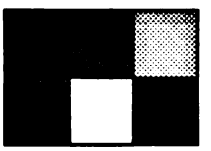
is a feasible solution of (4.3.8). Thus, the weights are:

$$T_{11} = 44.95r_0^{-1}, T_{22} = 9.51r_0^{-1}, T_{33} = 3.02r_0^{-1}, T_{44} = 9.51r_0^{-1},$$
$$T_{55} = 0.014r_0^{-1}, T_{66} = 44.95r_0^{-1}, T_{12} = 0.09r_0^{-1}, T_{13} = 0.09r_0^{-1},$$

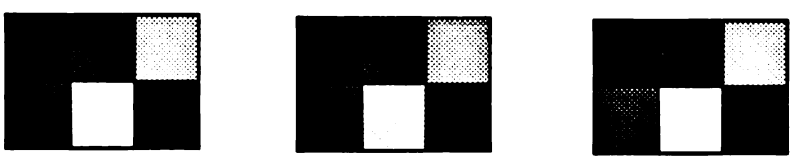
$$T_{14} = 0.09r_0^{-1}, T_{15} = 0.09r_0^{-1}, T_{16} = 0.09r_0^{-1}, T_{23} = 0.09r_0^{-1},$$
$$T_{24} = 0.09r_0^{-1}, T_{25} = 0.09r_0^{-1}, T_{26} = 0.09r_0^{-1}, T_{34} = 0.09r_0^{-1},$$
$$T_{35} = 0.09r_0^{-1}, T_{36} = 0.09r_0^{-1}, T_{45} = 0.09r_0^{-1}, T_{46} = 0.09r_0^{-1}$$
$$T_{56} = 0.09r_0^{-1}.$$

To test whether this pattern has been stored as asymptotically stable equilibrium, we use the pattern itself and its noisy patterns to retrieve the stored pattern. The numerical values for the noisy patterns are defined as:

We found that the given pattern



can be retrieved by itself and the following noisy patterns:



which shows that the given pattern has been indeed stored as asymptotically stable equilibrium.

5.2 PSpice Simulation of a Six-Neuron DANN Network with Resistor-Connections

We have designed a six-neuron DANN network circuit in which all connections of each individual neuron and other neurons are mode by resistors. Each neuron has four states. We use this DANN circuit to store the four-level pattern given in Section 5.1.

The schematic diagram of this six-neuron DANN network is shown in Figure 5.2.1. In this circuit, the resistors of each individual neuron are identical with resistance r_0. Since we store the pattern given in Section 5.1, the values of all resistors are found as:

$$R_{11} = 0.0222r_0,\ R_{22} = 0.105r_0,\ R_{33} = 0.332r_0,\ R_{44} = 0.105r_0,$$
$$R_{55} = 69.4r_0,\ R_{66} = 0.0222r_0,\ R_{12} = 11.11r_0,\ R_{13} = 11.11r_0,$$
$$R_{14} = 11.11r_0,\ R_{15} = 11.11r_0,\ R_{16} = 11.11r_0,\ R_{23} = 11.11r_0,$$
$$R_{24} = 11.11r_0,\ R_{25} = 11.11r_0,\ R_{26} = 11.11r_0,\ R_{34} = 11.11r_0,$$
$$R_{35} = 11.11r_0,\ R_{36} = 11.11r_0,\ R_{45} = 11.11r_0,\ R_{46} = 11.11r_0,$$
$$R_{56} = 11.11r_0.$$

where r_0 is the local resistance. In Figure 5.2.1, the neuron circuit shown by the box serves as a building block. Its schematic diagram is depicted in Figure 5.2.2 in which, in addition to V_{DD} and GND, the signals V_1, V_2, V_o and V_{bias} are shared by all other neurons. The input-output characteristic of the neuron circuit is shown in Figure 5.2.3 where the voltages 1V, 2V, 3V and 4V are chosen as 4 states in the multiple-state neural network. It can be measured that

$$s(1) = 1.694\text{V},\ s(2) = 2.514\text{V},\ s(3) = 3.325\text{V},\ s(4) = 4.103\text{V}$$

Its input-output is very similar to the one in Figure 5.1.1 as far as the four states

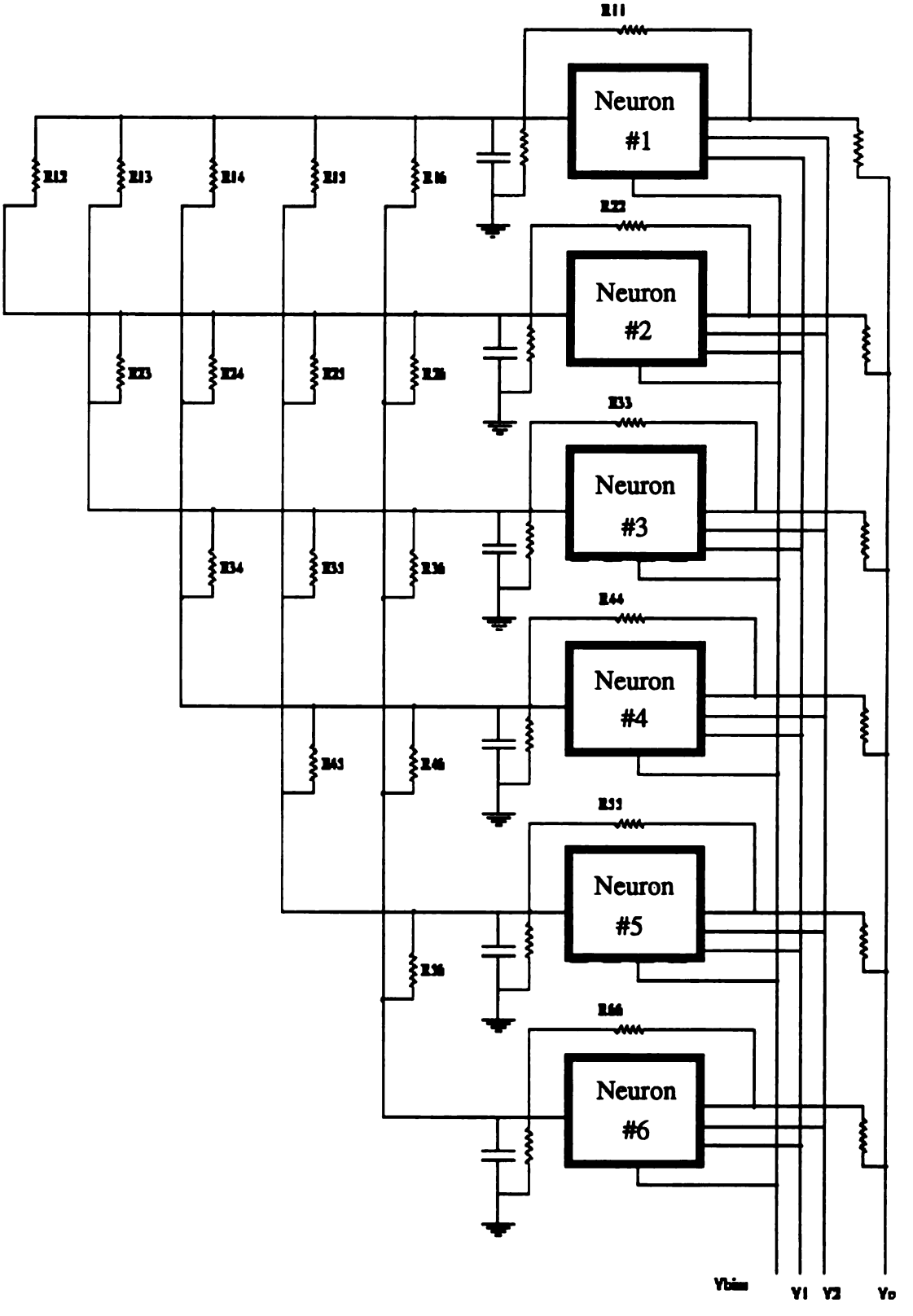


Figure 5.2.1. The schematic diagram of a six-neuron DANN circuit.

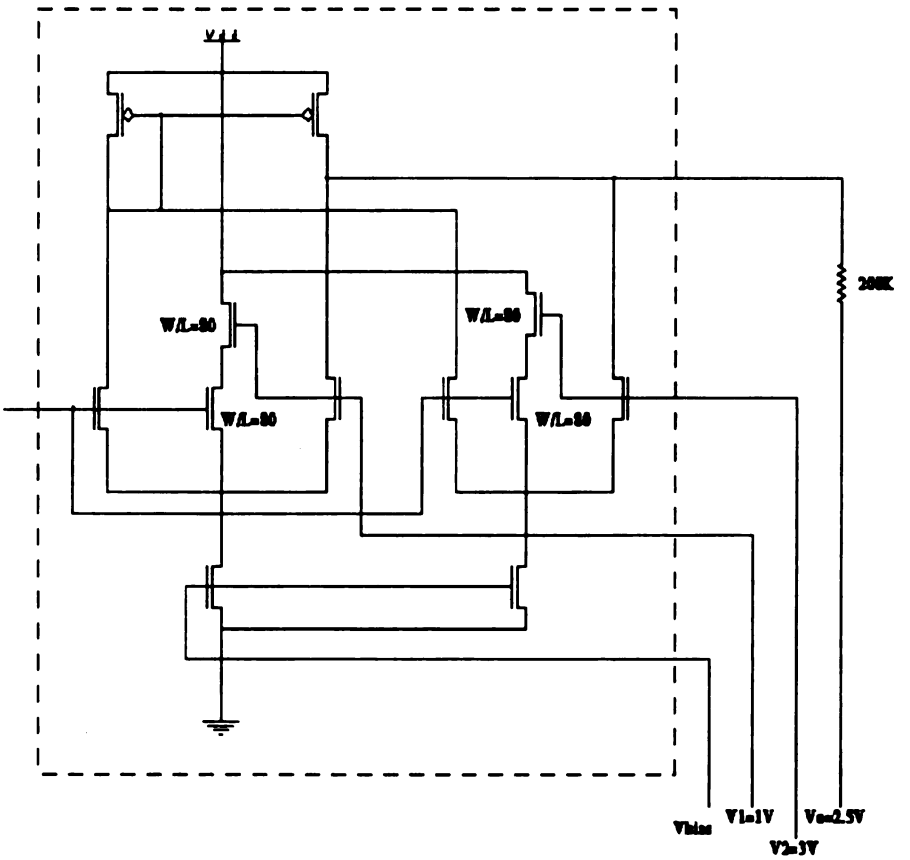


Figure 5.2.2. The schematic diagram of a four-state neuron circuit.

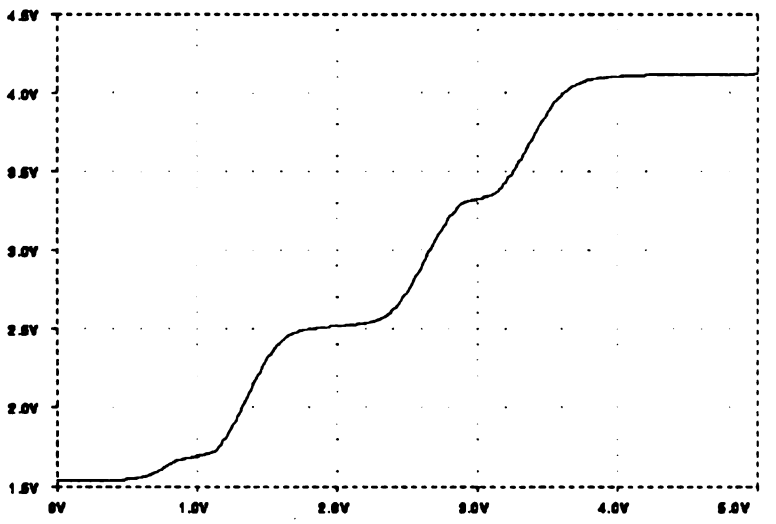


Figure 5.2.3. The input-output characteristic of a four-state neuron circuit.

are concerned. This similarity allows us to use the theoretical calculation results to build our circuit.

To test this circuit, we set the initial voltages at the input nodes of each neuron as: neuron #1 - 4.3V, neuron #2 - 3.2V, neuron #3 - 2.2V, neuron #4 - 2.8V, neuron #5 - 0.9V, neuron #6 - 3.6V. After the transient process is finished, the final voltages at the corresponding nodes converge to: neuron #1 - 4V, neuron #2 - 3V, neuron #3 - 2V, neuron #4 - 3V, neuron #5 - 1V, neuron #6 - 4V, which represents the stored pattern vector: [4, 3, 2, 3, 1, 4]T. This experiment shows that the designed 6-neuron DANN circuit works well as expected. The simulation result is shown in 5.2.4.

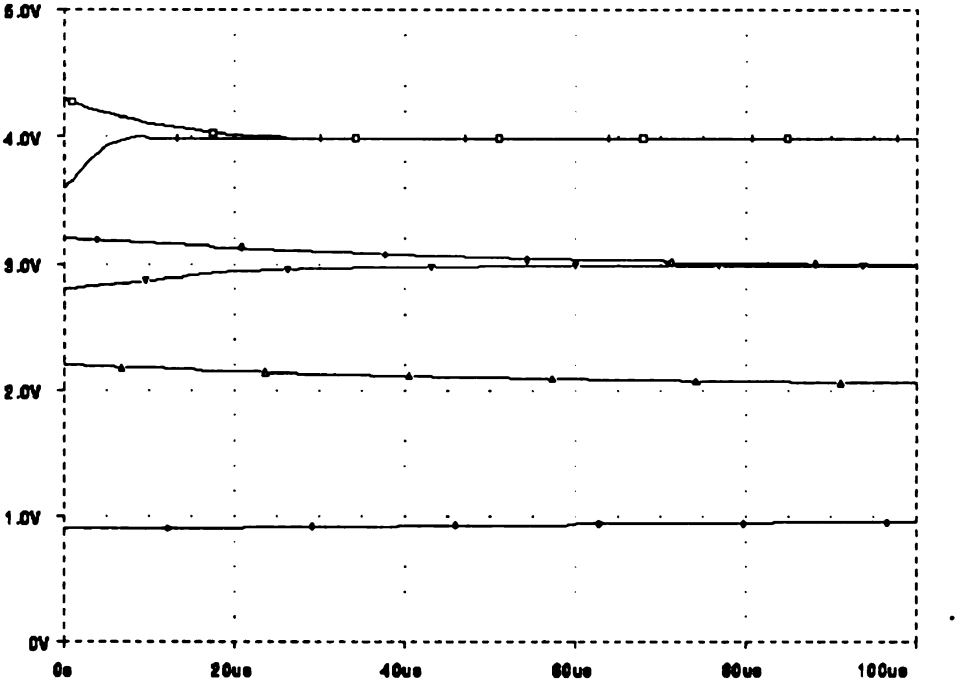


Figure 5.2.4. The simulation of a 6-neuron DANN network circuit.

It is noted that the resistor r_0 of each neuron has to be very large. In this simulation, we choose r_0 to be 1MegΩ.

The network in Figure 5.2.1 also stores other patterns. To show this, we set the initial input voltages at arbitrary values. Figure 5.2.5 shows that the network converges to the pattern $[4, 2, 2, 3, 1, 4]^T$. And Figure 5.2.6 shows that the network converges to the pattern $[4, 2, 2, 2, 1, 4]^T$.

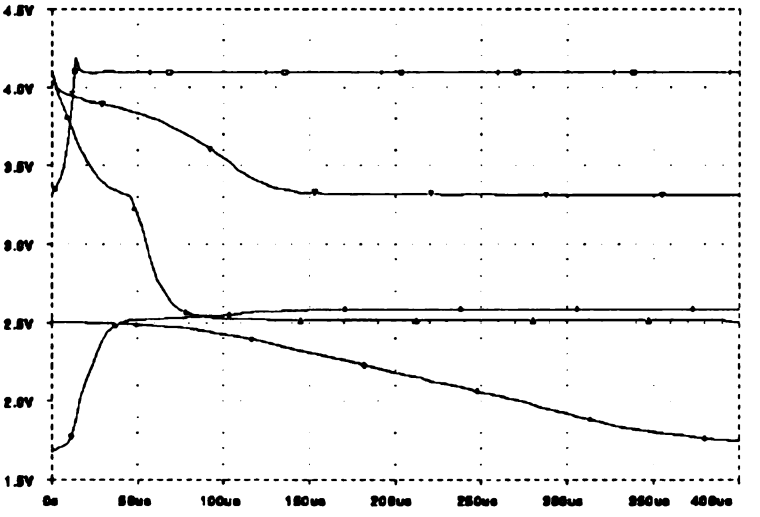


Figure 5.2.5. The network converges to pattern $[4, 2, 2, 3, 1, 4]^T$.

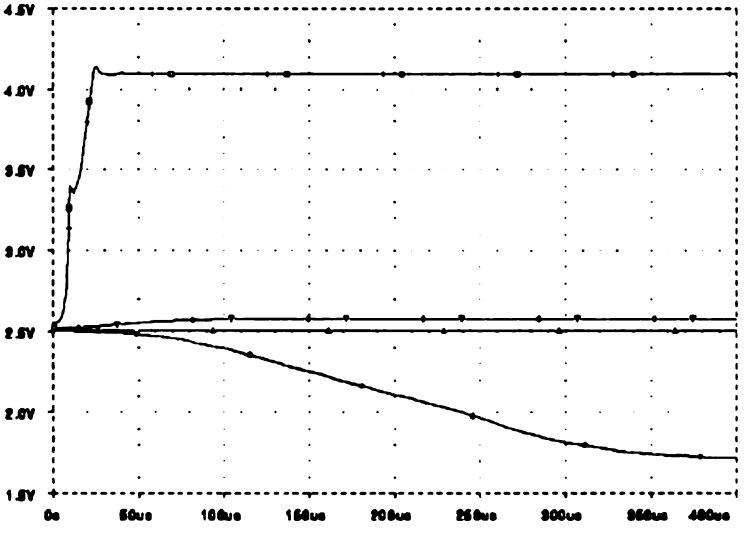


Figure 5.2.6. The network converges to pattern $[4, 2, 2, 2, 1, 4]^T$.

5.3 PSpice Simulation of a Six-Neuron Network with nMOS Transistor-Connections

A neural network can be used to store given patterns by having proper weights. This requires that those weights are programmable. The disadvantage of the circuit with resistor-connections shown in Figure 5.2.1 is that the weights are not programmable.

As shown in [49], there is another type of configuration of DANN where all connections of each individual neuron and other neurons are made by nMOS transistors. In this configuration of DANN, the nMOS transistors are used as the programmable weights. The values of weights can be set on-line by varying the gate voltages of those transistors.

An nMOS transistor used as a programmable weight is shown in Figure 5.3.1.

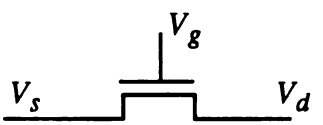


Figure 5.3.1. An nMOS transistor used as a programmable weight.

For the fixed gate voltage, the characteristic of I_{ds} - V_{ds} is plotted in Figure 5.3.2 where V_{gs} = 5V. The level of I_{ds} mainly depends on V_{gs}, V_{ds}, and W/L. There are basically two regions in the I_{ds} - V_{ds} region: ohmic region and saturation. The nMOS transistor can be used as a nonlinear resistive device if it operates in the ohmic region. The resistance of a nMOS for fixed Vgs is determined by:

$$R_{ds} = \frac{dV_{ds}}{dI_{ds}} = \frac{1}{\frac{dI_{ds}}{dV_{ds}}} \tag{5.3.1}$$

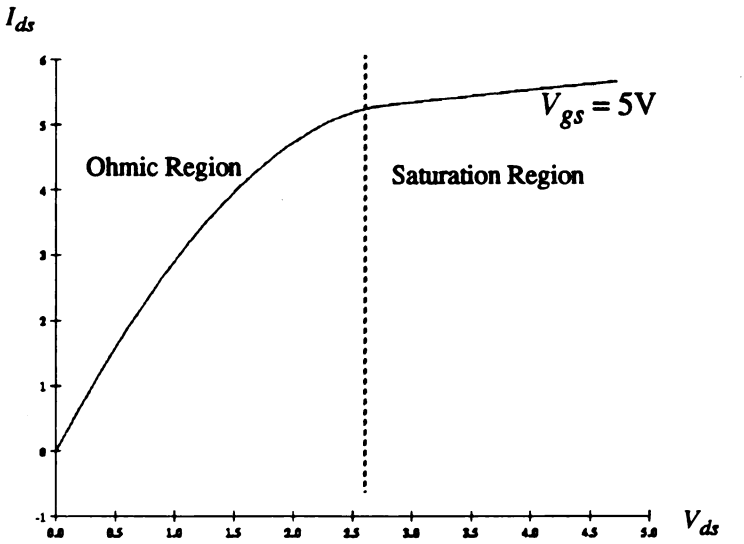


Figure 5.3.2. The I_{ds} - V_{ds} characteristic with V_{gs} = 5V.

Therefore, the level of R_{ds} depends on V_{gs}, V_{ds}, and W/L, too. The larger V_{gs}, the smaller R_{ds}. And, R_{ds} decreases as W/L increases.

The schematic diagram of a six-neuron DANN network with nMOS transistor-connections is depicted in Figure 5.3.3. In this circuit, the neuron circuit shown by the box serves as a building block. Its schematic diagram is depicted in Figure 5.2.2 in which, in addition to V_{DD} and GND, the signals V_1, V_2, V_o and V_{bias} are shared by all other neurons. The input-output characteristic of the neuron circuit is shown in Figure 5.2.3 where the voltages 1V, 2V, 3V and 4V are chosen as 4 states in the multiple-state neural network. It can be measured that

$$s(1) = 1.694V,\ s(2) = 2.514V,\ s(3) = 3.325V,\ s(4) = 4.103V$$

Its input-output characteristic is very similar to the one in Figure 5.1.1 as far as the four states are concerned.

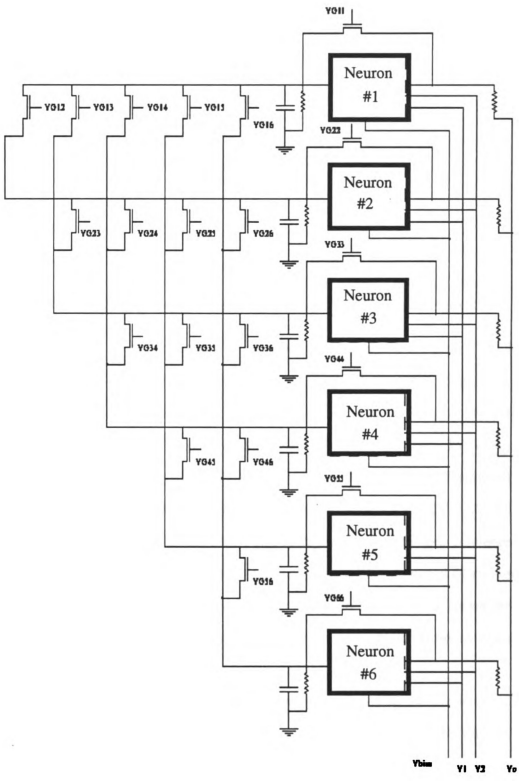


Figure 5.3.3. The schematic diagram of a six-neuron DANN with nMOS transistor-connections.

A PSpice simulation of this DANN network in Figure 5.3.3 is shown in Figure 5.3.4.

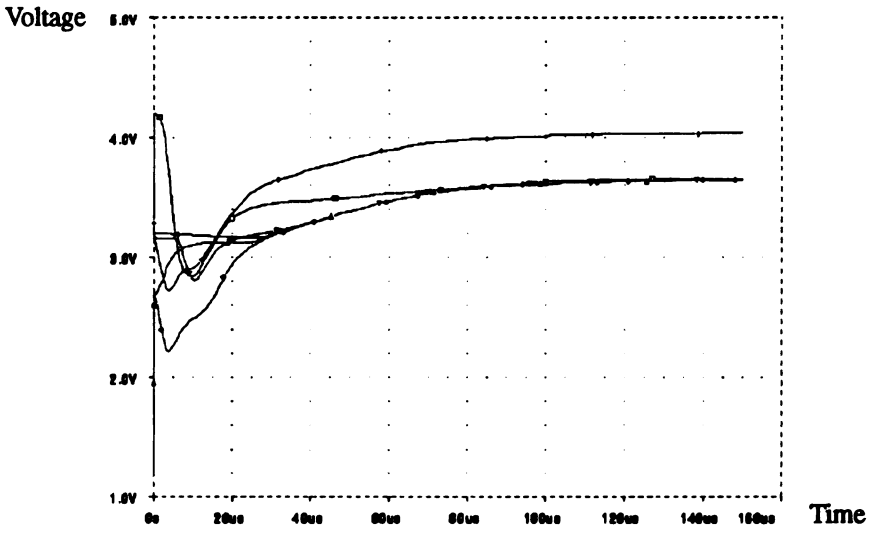


Figure 5.3.4. A simulation result of the six-neuron DANN network in Figure 5.3.3.

In this simulation, the resistor r_0 of each individual neuron is chosen as 1000K which is the same as it is in the simulation of Figure 5.2.4. All the gate voltages are: VG_{11} = 4V, VG_{22} = 3V, VG_{33} = 2V, VG_{44} = 3V, VG_{55} = 2V, VG_{66} = 5V, VG_{12} = 3V, VG_{13} = 2V, VG_{14} = 4V, VG_{15} = 1V, VG_{16} = 4V, VG_{23} = 2V, VG_{24} = 3V, VG_{25} = 1V, VG_{26} = 3V, VG_{34} = 2V, VG_{35} = 1V, VG_{36} = 2V, VG_{45} = 1V, VG_{46} = 3V, VG_{56} = 1V. The initial input voltages in the order of neuron #1 - neuron # 6 are: 4.2V, 3.2V, 2.2V, 3.2V, 1.2V, 4.2V. From Figure 5.3.4, we know that the final input voltage of neuron #6 does converge to about 4.1V which is expected. The final voltages of neuron #2 and neuron #4 converge to about 3.6V. The other three final input voltages converge to 3.7V as well. As we see, the voltage states of each neuron in the circuit of Figure 5.3.3 has shifted from desired states.

This simulation shows that the six-neuron DANN with nMOS transistor-connections has the potential to store gray-level pattern. Due to the nonlinearity of nMOS transistor used as resistors, some on-chip learning circuits have to be designed if we expect the DANN network with nMOS transistor-connections store desired gray-level patterns. This work is left for the future research.

For the same set of gate voltages, we test the network in Figure 5.3.3 with different initial conditions. Figure 5.3.5 shows the simulation result with the initial input voltages in the order of neuron #1 to neuron #6: 4V, 1V, 1V, 2V, 2V, 3V.

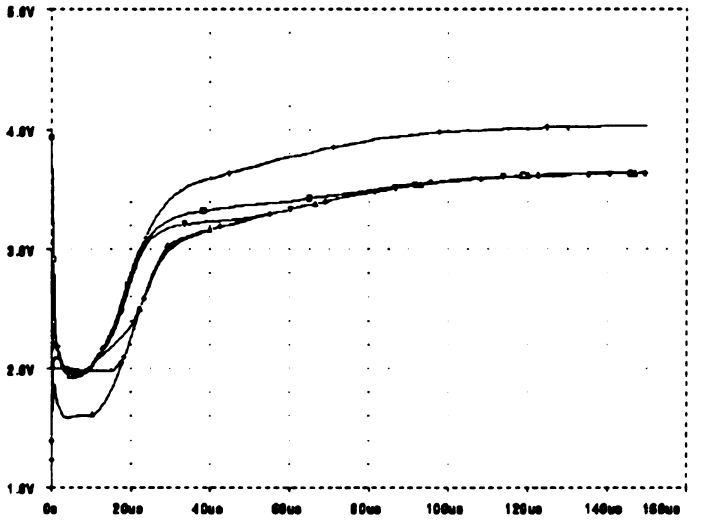


Figure 5.3.5. Simulation of the network in Figure 5.3.3 with initial input voltages: 4V, 1V, 1V, 2V, 2V, 3V.

Figure 5.3.6 is another simulation of the same network with the set of initial input voltages given as: 3V, 2V, 1V, 4V, 4V, and 3V.

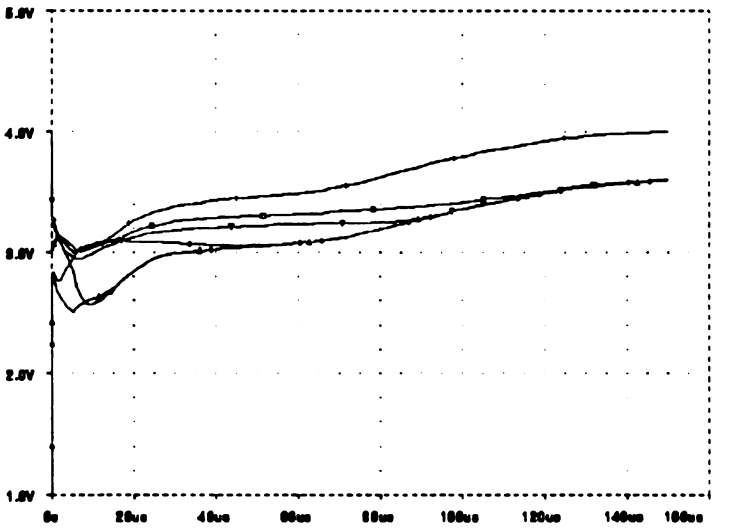


Figure 5.3.6. Simulation of the network in Figure 5.3.3 with initial input voltages: 3V, 2V, 1V, 4V, 4V, 3V.

We also test the network in Figure 5.3.3 with different gate voltages with the initial input voltages given as the desired patterns: 4V, 3V, 2V, 3V, 1V, 4V. Figure 5.3.7 shows the simulation with the gate voltages of all self-feedback transistors given as 5V, all others are held the same as before.

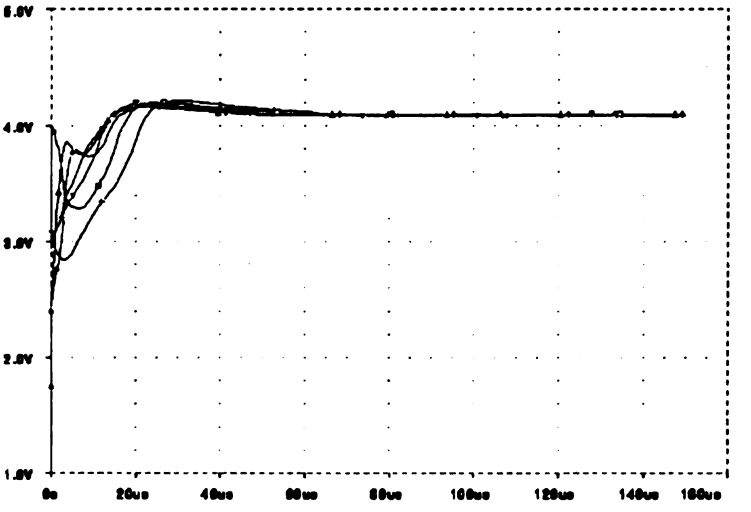


Figure 5.3.7. Simulation of the network in Figure 5.3.3 with the gate voltages of all self-feedback transistors set at 5V.

Figure 5.3.8 shows another simulation of the same network with the gate voltages of all self-feedback transistors set as 1V, all others are the same as before.

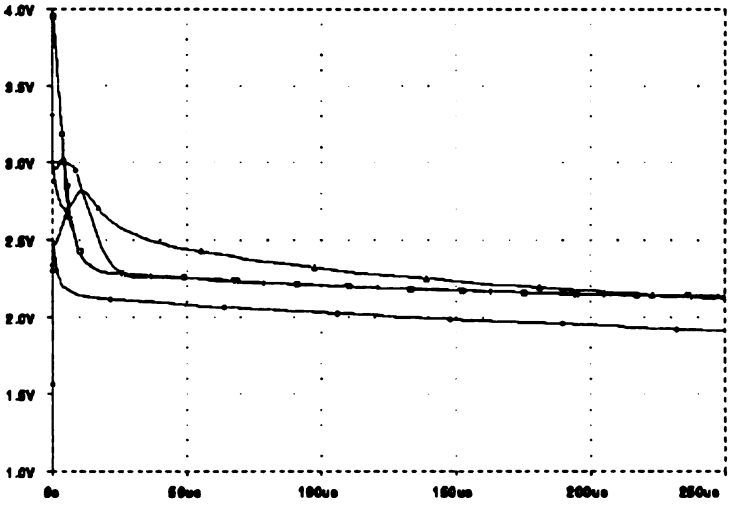


Figure 5.3.7. Simulation of the network in Figure 5.3.3 with the gate voltages of all self-feedback transistors set at 1V.

5.4 A Tiny-Chip Layout of a Six-Neuron DANN Network

A VLSI layout was designed for a six-neuron DANN network circuit on MOSIS Tiny-Chip as a prototype chip. This layout is fabricated by MOSIS using an n-well 2μm CMOS process with 2.2 × 2.2 mm2 chip area in a 40-pin package.

Among those 40 pins, 4 pins are used as V_{DD} and *GND*. The rest of 36 pins can be used to access the internal signals. For this 6-neuron DANN network, there are total 2×6 + 6×(6+1)/2 = 33 gate voltages. Thus, 33 pins will be used to access those gate voltages and input, output voltages. The other three pins are used to connect V_{bias}, and two-state voltages, V_1 = 1V and V_2 = 3V. Those three voltage signals are shared by all six neurons. The resistor and voltage source in the output stage of each neuron are connected to the chip externally.

The schematic diagram of this six-neuron DANN circuit is depicted in Figure 5.4.1, which has been designed to maximally use the limited chip area. Each neuron is represented by the black box which circuit is shown in Figure 5.2.2. Based on this configuration, the VLSI layout of this six-neuron DANN network is designed, which is shown in Figure 5.4.2. In this layout, the resistors and capacitors of each individual neuron are not included. The pin configuration of this DANN chip is shown in Figure 5.4.3.

This six-neuron DANN chip will be sent to MOSIS for fabrication. All internal signals can be accessed through pins. To test this chip, one needs to connect a 200K resistor and 2.5V power source to the output of each individual neuron to convert four-state output current signal to four-state voltage signal. The bias voltage has to be set at 1.2V in order to have the four-state input-output characteristic of each neuron. This characteristic can be measured by setting all gate voltages to 0V and measuring input-output relation.

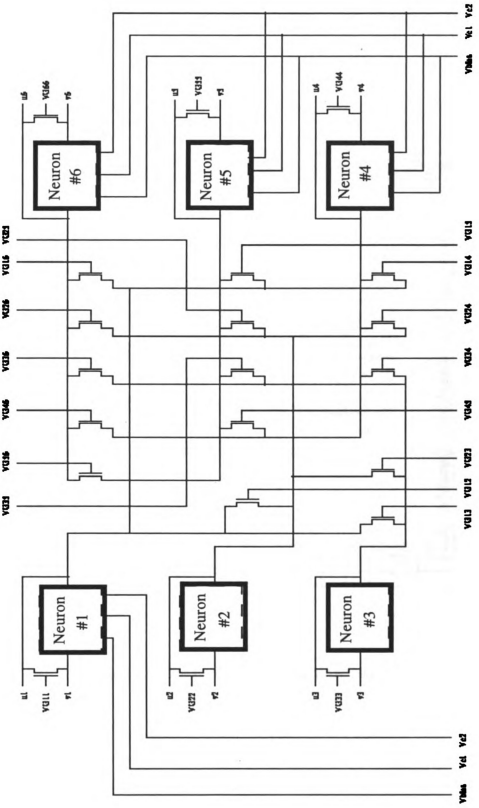


Figure 5.4.1. The schematic diagram of a six-neuron DANN network.

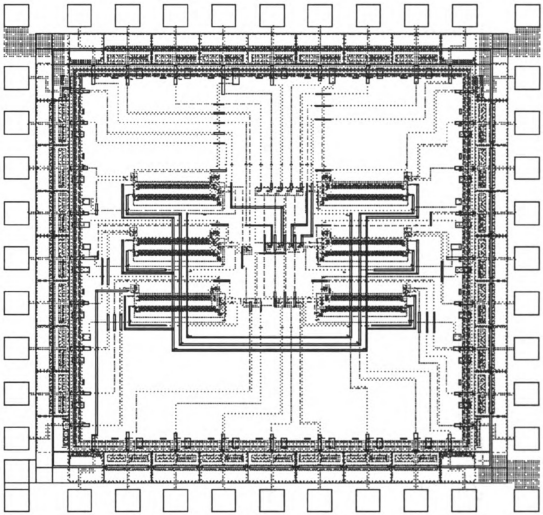


Figure 5.4.2. The layout of a six-neuron DANN chip.

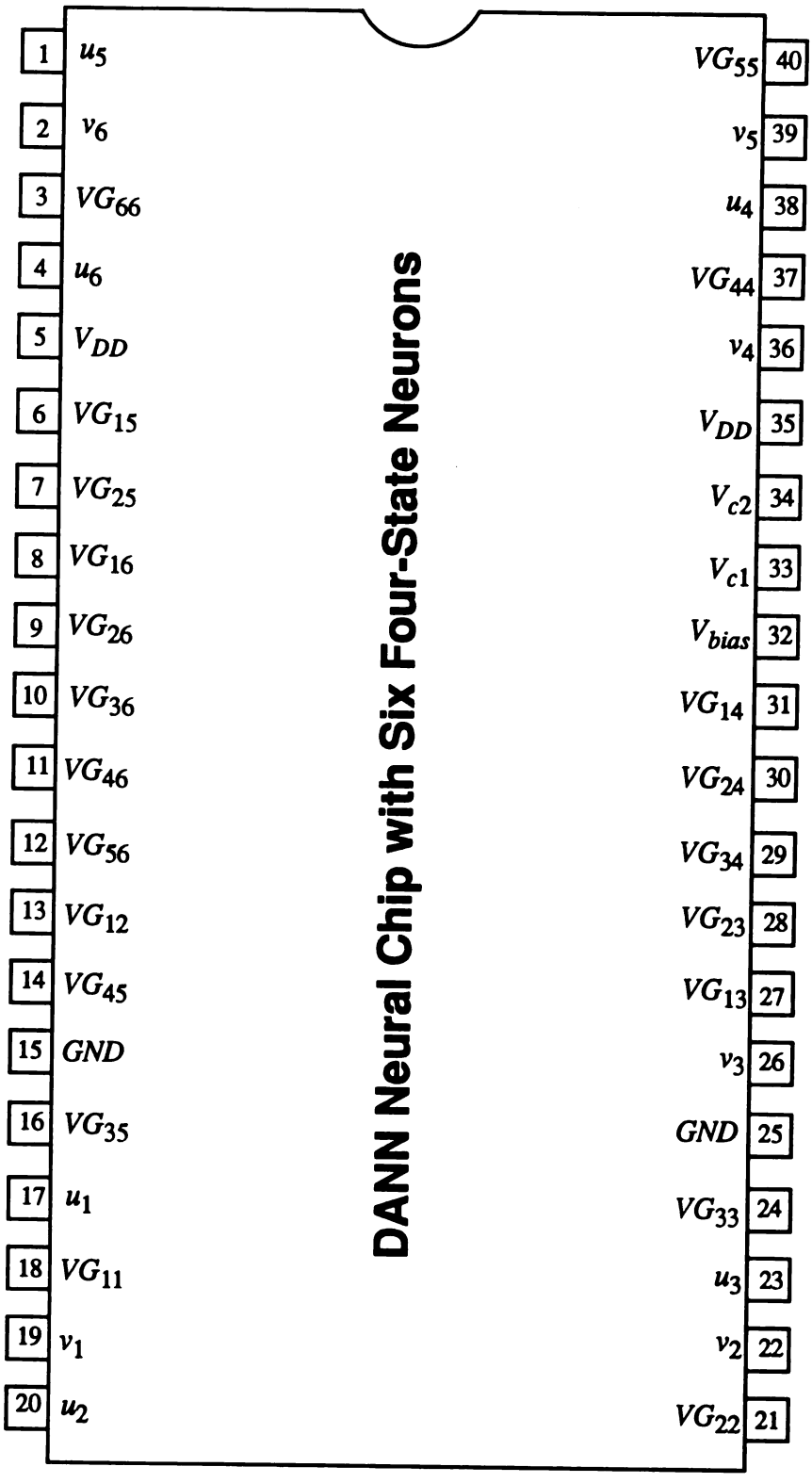


Figure 5.4.3. The pin configure of the six-neuron DANN chip.

CHAPTER 6

Summary, Conclusion, and Future Work

This work has focused on the design and analysis of two types of continuous-time feedback neural networks, namely, the Hopfield-type feedback neural network and the Dendro-dendritic artificial neural network (DANN). The networks considered in this dissertation mainly have multiple-state neurons.

For the Hopfield-type continuous-time feedback neural network with multiple-state neurons, we have proposed an analytical recursive algorithm to find weights which will exactly store the desired gray-level patterns as asymptotically stable equilibria. We introduce a criterion which will ensure the designed Hopfield-type neural network to be implementable. Moreover, the resistance in the network can be made as large as necessary. We estimate the variation of equilibria based on the variation of weights.

The cellular Hopfield-type feedback neural network is also considered. We propose a large network structure which connects some small identical networks (cells) together. A parallel recursive algorithm for this type of locally connected neural networks is introduced. This parallel algorithm makes all cells capable of finding their own weights simultaneously.

For the DANN network, we show the stability of the network with positive weights. We show that multiple patterns can be stored as asymptotically equilibria of DANN with only positive weights. We further formulate the problem of finding all positive weights as the standard linear programming problem. Thus, based on given patterns, all positive weights can be easily found.

We have designed the CMOS VLSI multiple-state neuron circuit which operates in subthreshold. With this neuron, the network power dissipation is reduced. As a prototype, a DANN network with three neurons has been designed. Each neuron has four states. The PSpice simulation has shown that this prototype DANN network works well. A six-neuron DANN Tiny-Chip has been designed and fabricated by MOSIS.

The following problems are left for future research:

(1) The persistence of equilibria for DANN with transistor-connections.

(2) The persistence of equilibria for Hopfield-type network with transistor-connections.

(3) Existence of optimal feature solutions of DANN for multiple patterns.

(4) Design a new output stage of the multiple-state neuron circuit to convert the current to voltage.

(5) Design an on-chip learning circuit to find gate voltages for DANN with multiple-state neurons and transistor-connections.

Identifying applications is the key to the future of the continuous-time feedback neural networks. Feedback continuous-time feedback neural networks have shown their potential in various applications such as optimization, vision, control, edge detection, and computer numerical computations, to name a few.

References

[1] Eyad H. Abed, "A simple proof of stability on the center manifold for Hopf bifurcation", *SIAM Review*, Vol. 30, No. 3, Sep. 1988, pp. 487-491.

[2] Eyad H. Abed, "Local bifurcation control", pp. 225-241 in *Dynamic Systems Approaches to Nonlinear Problems in Systems and Circuits*, ed. Fathi M.A. Salam, Mark L. Levi, SIAM, Philadelphia 1988.

[3] Dirk Aeyels, "Stabilization of a class of nonlinear systems by a smooth feedback control", *Systems & Control Letters* 5 (1985), pp. 289-294.

[4] Phillip E. Allen, Douglas R. Holberg, *CMOS Analog Circuit Design*, Holt, Rinehart and Winston, Inc., 1987.

[5] Stephen T. Barnard, "A Stochastic Approach to Stereo Vision", in *Proc. National Conference on AI* (AAAI-86) (Philadephia, PA., Aug. 11-15), pp. 676-679.

[6] Adi Ben-Israel, Thomas N. E. Greville, *Generalized Inverses: Theory And Applications*, Robert E. Krieger Publishing Company, 1980.

[7] Andrew Blake, Andrew Zisserman, *Visual Reconstruction*, The MIT Press, 1987.

[8] R.W. Brockett, "Asymptotic stability and feedback stabilization", pp. 181-191 in *Differential Geometric Control Theory*, ed. R.W. Brockett, R.S. Millman and H.J. Sussmann, Birkhauser, Boston (1983).

[9] Shui-Nee Chow, Hack K. Hale, *Methods of Bifurcation Theory*, Springer-Verlag, 1982.

[10] Leon O. Chua, L. Yang, "Cellular neural networks: Theory", *IEEE Trans. on Circuits Syst.*, Vol. 35, pp. 1257-1272, 1988.

[11] Leon O. Chua, Tamas Roska, "Stability of a Class of Nonreciprocal Cellular Neural Networks", *IEEE Trans. on Circuits Syst.*, Vol. 37, No. 12, December 1990.

[12] Tobi Delbruck, ""Bump" Circuits for Computing Similarity and Dissimilarity of Analog Voltages", *CNS Memo* 10, California Institute of Technology, Computation and Neural Systems Program, May, 1991.

[13] D.K. Faddeev, V.N. Faddeeva, *Computational Methods of Linear Algebra*, W.H. Freeman and Company, 1963.

[14] Jay A. Farrell, Anthony N. Michel, "A synthesis procedure for Hopfield's continuous-time associative memory", *IEEE Trans. on Circuits Syst.*, Vol. 37, July 1990.

[15] W. Eric L. Grimson, "Computational Experiments with a Feature Based Stereo Algorithm", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-7, No. 1, Jan. 1985, pp. 17-34.

[16] L.T. Grujic, A.N. Michel, "Exponential stability and trajectory bounds of neural network under structural variations", *IEEE Trans. on Circuits Syst.*, Vol. 38, No. 10, Oct. 1991.

[17] Jack K. Hale, *Ordinary Differential Equations*, Robert E. Krieger Publishing Company, 1980.

[18] J. Hertz, A. Krogh, R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley Publishing Company, 1991.

[19] M.W. Hirsch, S. Smale, *Differential Equations, Dynamical Systems and Linear Algebra*, Academic Press, 1974.

[20] J.J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities,", in *Proc. Natl.Acad. Sci. U.S.A.*, Vol. 79, pp. 2554-2558, 1982.

[21] J.J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," in *Proc. Natl. Acad. Sci. U.S.A.*, Vol. 81, pp. 3088-3092, 1984.

[22] Zahid Hussian, *Digital Image Processing - Practical Applications of Parallel Processing Techniques*, Ellis Horwood, 1991.

[23] Hassan K. Khalil, *Nonlinear Systems*, Macmillan Publishing Company, 1992.

[24] N.N. Krasovskii, J.L. Brenner, *Stability of Motion*, Stanford University Press, 1963.

[9] Solomon Lefschetz, *Differential Equation: Geometric Theory* (2nd edition), Interscience Publishers, 1963.

[25] Solomon Lefschetz, *Differential Equation: Geometric Theory* (2nd edition), Interscience Publishers, 1963.

[26] J.H. Li, A.N. Michel, W. Porod, "Qualitative analysis and synthesis of a class of neural networks", *IEEE Trans. on Circuits Syst.*, Vol. 35, No. 8, Aug. 1988.

[27] Bo Ling, Fathi M.A. Salam, "A cellular network formed of Hopfield networks", in *Proc. of the 35th Midwest Symposium on Circuits and Systems*, Washington, D.C., Aug. 9-12, 1992.

[28] Bo Ling, Fathi M.A. Salam, "Persistence of equilibria under weight variation of feedback continuous-time neural network", *Proc. of the IEEE International Symposium on Circuits and Systems, Chicago, Illinosis, 1993.*

[29] Bo Ling, Fathi M.A. Salam, "Parameter determination for an implementable feedback neural network", *Proc. of the IEEE International Symposium on Circuits and Systems, Chicago, Illinosis, 1993.*

[30] Bo Ling, Fathi M.A. Salam, "An analytical learning algorithm for the dendro-dendritic artificial neural network via linear programming", *Proc. of the IEEE International Conference on Neural Networks*, San Francisco, California, 1993.

[31] Bo Ling, Fathi M.A. Salam, "State feedback stabilization of nonlinear system via the neural network approach", *Proc. of the American Control Conference*, San Francisco, California, 1993.

[32] Bo Ling, Fathi M.A. Salam, "2n-state CMOS neuron circuit operating in subthreshold", *Proc. of the 1993 World Congress on Neural Networks*, Portland, Oregon, July, 1993.

[33] Bo Ling, Fathi M.A. Salam, Shanti Vedula, "Analog VLSI feature matching circuit

for stereo vision process", in *Proc. of the 35th Midwest Symposium on Circuits and Systems*, Detroit, Michigan, August, 1993.

[34] David G. Luenberger, *Introduction to Linear and Nonlinear Programming*, Addison-Wesley Publishing Company, 1965.

[35] Jerrold E. Marsden, *Elementary Classical Analysis*, W.H. Freeman and Company, 1974.

[36] Larry Matthies, Takeo Kanade, "Kalman Filter-based Algorithms for Estimating Depth from Image Sequences", *International Journal of Computer Vision*, 3, 209-236, 1989.

[37] W.S. McCulloch, W. Pitts, Bull Biophys., 5, 1943, pp. 115-133.

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

[39] A,N, Michel, J.A. Farrell, W. Porod, "Qualitative analysis of neural networks", *IEEE Trans. on Circuits Syst.*, Vol. 36, No. 2, Feb. 1989.

[40] Kenneth S. Miller, Donald M. Leskiw, *An Introduction to Kalman Filtering with Applications*, Robert E. Krieger Publishing Company, 1987.

[41] Gerald L. Morris, Patrick L. Odell, "Common Solutions for n Matrix Equations with Applications", *Journal of the Associations for Computing Machinery*, Vol. 15, No.2, April 1968, pp. 272-274.

[42] C. Neti, M.H. Schneider, E.D. Young, "Maximally fault tolerant neural networks", *IEEE Trans. on Neural Networks*, Vol. 3, No. 1, Jan. 1992.

[43] J.M. Ortega, W.C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, 1970.

[44] Alex P. Pentland, "Dynamic Vision", *Pattern Recognition by self-organization neural network*, ed. Stephen Grossberg, MIT Press, 1991.

[45] Tomaso Poggio, Vincent Torre, Christof Koch, "Computational vision and regularization theory", *Nature*, Vol. 317, Sep. 1985.

[46] Fathi M.A. Salam, Y. Wang, M. Choi, "On the Analysis of Dynamic Feedback Neural Nets," *IEEE Trans. on Circuits Syst.*, Vol. 38, No. 2, Feb. 1991.

[47] Fathi M.A. Salam, Y. Wang, "A Real-Time Experiment Using a 50-Neuron CMOS Analog Silicon Chip with On-Chip Digital Learning", *IEEE Trans. on Neural Networks*, Vol. 2, No. 4, July 1991.

[48] Fathi M.A. Salam, "A model of neural circuits for programmable VLSI implementation of the synaptic weights for feedback neural nets", *1989 IEEE International Symposium on Circuits and Systems*, Portland, Oregon, May 1989, pp. 849-851.

[49] Fathi M.A. Salam, "New artificial neural models: basic theory and characteristics", *1990 IEEE International Sym posium on Circuits and Systems*, New Orleans, Louisiana, May 1990, pp. 200-203.

[50] Fathi M.A. Salam, Yiwen Wang, "Neural circuits for programmable analog MOS VLSI implementation", *Proc. of 32nd Midwest Symposium on Circuits and Systems, Champaign, Illinois, August, 1989.*

[51] Fathi M.A. Salam, Yiwen Wang, "Learning scheme for analog CMOS neural chips", *Memorandum* No. MUS/EE/A90/07, Department of Electrical Engineering, Michigan State University, East Lansing, MI 48824, 1990.

[52] Michael Spivak, *Calculus on Manifolds _ A Modern* Approach to Classical Theorems of Advanced *Calculus*, New York, W.A. Benjamin, 1965.

[53] M. Stevenson, R. Winter, B. Widrow, "Sensitivity of feedforward neural networks to weight error", *IEEE Trans. on Neural Networks*, Vol. 1, No. 1, March 1990.

[54] Demetri Terzopoulos, Andrew Witkin, Michael Kass, "Symmetry-Seeking Models and 3D Object Reconstruction", *International Journal of Computer Vision*, 1, 211-221, 1987.

[55] J. Tsinias, N. Kalouptsidis, "Output feedback stabilization", *IEEE Trans. Automat. Contr.*, Vol. 35, pp. 951-954, Aug. 1990.

[56] M. Vidyasagar, *Nonlinear Systems Analysis*, Prentice-Hall, 1978.

[57] Y. Wang and F. Salam, "Custom Analog VLSI Neural Chip with On-Chip Digital Learning for Pattern/Character Recognition," *the second international conference on Fuzzy Logic and Neural Networks*, Iizuka, Japan, July 1992.