ANALYTICAL FORMULATION FOR STATIC AND TEMPORAL PATTERN ASSOCIATION: NEURAL ARCHITECTURES AND ALGORITHMS

By

Jiansheng Hou

A DISSERTATION

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

DOCTOR OF PHILOSOPHY

Department of Electrical Engineering

1997

STA
N

Curre

structure

using off-

(matrix).

This di

used for st

neural net

set as stab

determine

matrix is e

for augmer

The pro

adapted for

function an

ABSTRACT

ANALYTICAL FORMULATION FOR STATIC AND TEMPORAL PATTERN ASSOCIATION: NEURAL ARCHITECTURES AND ALGORITHMS

By

Jiansheng Hou

Currently, most Artificial Neural Network (ANN) models have standard layered structure of interconnected neurons where the connections (weights) are computed using off-line or on-line methods. The process of finding an appropriate weight (matrix), if it exists, is usually computationally expensive.

This dissertation focuses on new design and training of artificial neural networks used for storing static or temporal patterns. A product-of-norm model of artificial neural networks is presented. It is shown that this model can store an arbitrary data set as stable system equilibria – without the need for performing training process to determine the weights. Consequently, the process of iteratively finding the weight matrix is eliminated. The need of defining the weights is in fact replaced by defining (or augmenting) the architecture.

The properties of this new model are investigated, and various modifications are adapted for practical applications. In particular, the use of the log form of the energy function and the subgrouping method are introduced to reduce the computational

effort.

penalt

Thi

recurre

propos

back-pr

effort. Moreover, methods for recognizing shifted images are proposed by adding penalty terms to the energy function.

This investigation also includes ways to improve the current algorithms for training recurrent networks to store temporal patterns. Specifically, two algorithms are proposed to avoid the backward integration mode in the time-dependent recurrent back-propagation method.

I wo
his cont
Michiga
guidanc
Newhou
for thei
I am
of Elect
I am
and enc
been ab
warmest

ACKNOWLEDGMENTS

I would like to express my sincere thanks to my advisor, Dr. Fathi M. A. Salam, for his continued support and guidance throughout the years of my graduate studies at Michigan State University. I also want to thank the other members of my Ph.D guidance committee members, Dr. Hassan Khalil, Dr. John Deller, Dr. Sheldon Newhouse, Dr. Robert Nowak, Dr. Frank Hoppensteadt and Dr. R. L. Tummala, for their valuable comments and suggestions.

I am grateful for the financial support from the Case Center and the Department of Electrical Engineering of Michigan State University.

I am especially grateful to my wife, Jian Yan, for her love, understanding, support and encouragement, particularly in the last years of this research. I would not have been able to finish this work without her support. Finally, I want to express my warmest gratitude to my parents for their continued support and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CH

IN

1.1

Since t

human

have b

industr

though

many

so easi

by digi

year-ol

compu

to hum

- R

  th

CHAPTER 1

INTRODUCTION

1.1 Overview

Since the birth of the first electronic digital computer — the Von Neumann machine, human civilization has entered a new era of information processing. Today, computers have been used in almost every area in our life, from home to office, from science to industry. It is hard to imagine what life will look like without computers today. Even though the digital computers are so advanced and so widely used, they still have many limitations when compared to human brains. Many things, which can be done so easily by human brains, can be very difficult, if not impossible, to be accomplished by digital computers. A good example is the processing of visual information: an one-year-old baby is much faster at recognizing human faces and objects than the best computers today [HKP90]. The following distinguished features have been attributed to human brain (neural networks):

- Robust and fault tolerant. Nerve cells die every moment and this does not affect the performance of neural networks significantly.

- Flexible and adaptive. It can easily adjust to a new environment by "learning". It does not have to be programmed like a digital computer.
- High degree of parallelism and distributed storage.
- The ability to process information that is fuzzy, probabilistic or inconsistent.
- Compactness.

These features have inspired generations of researchers to study the human brain — neural networks, to investigate the principles of how human brain works and to build models to mimic the behavior of the human brain. This lead to the birth of "artificial neural network" (ANN) — " the next generation of computers".

The study of artificial neural network has gone through several circles of up and down. It began in the 1940's with McCullough and Pitt. Second peak happened in the 1960's with Rosenblatt's perceptron convergence theorem. It began to explode again in the 1980's inspired by John Hopfield's energy approach and the modern era of multi-layered neural networks and back-propagation algorithm. Researchers have been investigating the possible applications of artificial neural networks in all areas. Information association using artificial neural networks is a major field which researchers have studied extensively. The research results can be used in a wide range of real world applications, such as image association, pattern recognition, associative memory and etc..

Associative memory is one of the early themes of the research works on artificial neural networks. It is such a kind of network that presents the same output when similar inputs are applied. Many artificial neural network architectures and algorithms have been proposed and studied by researchers. Some of the proposed networks

use fee

networ

1.2

Some m

1. th

2. the

usi

While it

systems

prohibiti

network

Consequ

order of

computa

the insu

the data

In th

associat

to overc

on redu

of ANN

too.

use feedforward structures, while the majority use feedback structures (the recurrent networks). The focus of this dissertation research is on the recurrent networks.

1.2 Problem Statement

Some major issues in designing functional neural networks are [SB91, SWC91]:

1. the ability to store (or learn) any number of pre-specified set of data successfully;
2. the determination of a set of parameters (or weights) which would archive 1 using efficient off-line or on-line computations.

While it is possible to employ analogous approaches to the algorithms employed in systems theory, control and adaptive control, the amount of computation may be prohibitive by present implementation media. In particular, for n-neuron neural network update methods, one must update or determine in the order of n^2 weights. Consequently, most learning or weight updating methods can successfully store in the order of n patterns while updating about n^2 weights. In addition, the weight updating computation is largely time consuming. Another limitation of current algorithms is the insufficient ability to store arbitrary set of data. Many of the algorithms require the data to be stored meet certain prerequisite conditions.

In this thesis, current ANN architectures and algorithms used in static information association will be analyzed. New ANN architectures and algorithms will be presented to overcome the limitations identified in the analysis. Specifically, this thesis will focus on reducing the computation complexity of the networks and increasing the capacity of ANN networks. Implementation of new models via hardware will be investigated too.

AN

great p

extensi

Ways t

be inves

1.3

Goal 1:

Objecti

Significa

Approach

ANN networks used for temporal pattern association is another area which has great potential applications. Most algorithms proposed so far are, to certain degree, extensions of the back-propagation method. The computation is very expensive. Ways to improve the performance of ANN used in temporal pattern association will be investigated in this thesis too.

1.3 Research Goals of the Thesis

Goal 1: *Improve the performance of ANN used for static pattern association*

Objective: Analyze Current ANN network architectures, models and algorithms, especially recurrent networks will be analyzed, so that their limitations can be identified. New architectures and algorithms will be presented to overcome the limitations identified.

Significance: Analysis of current approaches is a very important step. Through this step, advantages and limitations of current ANN networks used for association of static patterns will be identified. Thus new architectures and algorithms can be generated. A new architecture with better performance will have a wide range of applications. It is a step further in mimicking the behavior of neural networks.

Approach: Current models and methods of ANN used for association of static patterns, especially feedback networks, will be investigated first. The capacity, speed, convergence and computation complexity of various nets will be studied. A new ANN architecture will be proposed

Goal 2:

Objecti

Significa

Approac

with a new "energy" function. The resulting network will be a gradient-like system, thus it will have nice stable behavior. The proposed system will be able to store arbitrary analog data set. The determination of network parameters (weights) will be simplified using this model. Static patterns will be stored as stable equilibria of the system so that given enough hint (input) the network will be able to retrieve the correct pattern. Simulations will be performed to verify the theoretic results and demonstrate the potential usage of the new model.

Goal 2: *Generalization and extension of the proposed model.*

Objective: The proposed model will be generalized. Several ways of modifying the "energy" function will be investigated so that the resulting network can be tailored for various application needs.

Significance: Generalization of the "energy" function can gain insights of the property of the new model. Investigation of ways to modify the "energy" function will lay a foundation for the new model to be used in real world applications.

Approach: Different forms of the "energy" function will be explored to simplify the model and improve the performance. Ways to modify the model so that it can be easily modularized and parallelized will be investigated. Methods of modifying the model so that it can be used in image association and recognition will be studied. In particular, variations of the "energy" function to recognize shifted images will

Goal 3:

Objecti

Significa

Approa

Goal 4:

Objecti

Signific

be investigated. Software simulations will be performed to verify the correctness of theoretical result and computation complexity.

Goal 3: *Micro-electronic hardware implementation*

Objective: Study ways to implement the proposed network in hardware; particularly, VLSI technology. Basic circuits building blocks will be presented.

Significance: Implementing the new ANN model in hardware makes it possible to employ this model in real world applications. The implementation process can also provide feedback about directions to improve the theoretical model.

Approach: Analog MOS transistors biased in subthreshold region will be used as the basic elements of circuits for the new ANN model. Circuit building blocks will be designed and verified using "Pspice". Small networks will be built.

Goal 4: *Extensions to for temporal pattern association*

Objective: Analyze current models and algorithms used for temporal patterns association. Improve the performance of ANN by proposing new models or algorithms.

Significance: ANNs used for temporal pattern association have great potential usage in real world applications.

Appro

1.4

The ma

- A
  mo
  fixe
  equ
  use
- Va
  as
  ing
- Sev
  are
  ular
  dem
- Basi
  usin

Approach: Current models and algorithms for temporal pattern association will be surveyed first, then the limitations and advantages of each approach will be analyzed. New algorithms will be proposed to improve the performance and overcome the limitations identified.

1.4 Major Contributions

The major contributions of this work can be summarized as follows:

- A new ANN architecture — the product-of-norm model — is proposed. This model eliminates the dynamic learning of the weight connections. Weights are fixed instead. This network can store arbitrary set of data as stable system equilibria. Thus the data are guaranteed to be recallable. This model can be used in a wide range of real world applications.

- Various methods of simplifying the product-of-norm model are presented, such as using the log form of the "energy" function, the infinity norm and subgrouping method. The simplified network can be easily parallelized and modularized.

- Several ways of tailoring the "energy" function of the product-of-norm model are proposed so that the model can be used in specific applications. In particular, tailoring the "energy" function for recognizing slightly shifted images is demonstrated.

- Basic building blocks for implementing the product-of-norm model in hardware using VLSI technology are proposed.

• N
ba
ne
in
de
va
are

Overa
for gener
become s

1.5

The rest
Chapt
tion of st
Chapt
norm mo
equations
is investig
analyzed,
Chapte
function p
function.

- Numerical algorithms are proposed which extend the time-dependent recurrent back-propagation method used in storing temporal patterns using recurrent networks. They eliminate the need of integration backward in time as needed in the original time-dependent recurrent back-propagation method. Rules are developed for setting the final boundary conditions of the co-state and the initial values of the neurons of the time-dependent recurrent back-propagation method are derived using optimal control theory.

Overall, this work improves the current methods for artificial neural networks used for general association. Arbitrary data set can be stored and retrieved. "Learning" become simpler.

1.5 Organization of the Thesis

The rest of the dissertation is organized as follows:

Chapter 2 gives a basic introduction to artificial neural networks used in association of static patterns and identifies limitations of current approaches.

Chapter 3 presents a new model of artificial neural network — the product-of-norm model of ANN. A new "energy" function is proposed and the system dynamic equations are derived from the "energy" function. The stability of this new system is investigated. The properties of the new model using different kinds of norms are analyzed, especially the L_2 norm.

Chapter 4 further investigates the ways to modify and generalize the "energy" function proposed for the new model, specifically the log form of the "energy" function. The subgrouping method is proposed to localize the connections. It

also der

purpose.

Cha

transisto

of the ne

Chap

lem usin

each me

limitation

Finall

direction.

also demonstrates the ways of tailoring the "energy" function for image recognition purpose. Simulations are proved.

Chapter 5 presents some circuit building blocks for the new model using analog transistors biased in the subthreshold region. "Pspice" simulations are given for some of the networks. A one-neuron-/two-pattern network of the new model is also showed.

Chapter 6 analyzes the current approaches for temporal pattern association problem using artificial neural networks. It identifies the limitations and advantages of each method. Finally it presents some new algorithms to overcome some of the limitations.

Finally, chapter 7 summarizes the whole thesis, points out the future research direction.

CH

ART

NET

INF

2.1 B

Some of th

- How

  big is

  the ra

  the ne

- Will t

  store

CHAPTER 2

ARTIFICIAL NEURAL NETWORKS FOR INFORMATION ASSOCIATION

2.1 Background

Some of the major concerns or criteria in designing associative neural networks are:

- How many patterns can a network store? Can it store arbitrary patterns? How big is the network capacity (we define the capacity of an associative network as the ratio of the number of patterns it can store over the number of neurons in the network) ?
- Will there be any spurious patterns stored? In other words, does the network store patterns which are not desired?

• H
be
• Ho
pa
• Spe
retr
• Car

These

In genera

temporal

Many

and studie

memory. S

majority u

in the later

for store st

2.2 T

The analog

butions of t

for VLSI im

of a Hopfield

- How are the patterns stored? What is the training method? Can the network be trained on-line?
- How easy is it for the network to store additional patterns without altering the patterns which have already been stored?
- Speed of the network. How fast can it be trained? How fast can a pattern be retrieved?
- Can the network be easily implemented in hardware/software?

These criteria measure the merit of a designed associative artificial neural network. In general, the above criteria apply to associative networks for both static and temporal patterns.

Many artificial neural network architectures and algorithms have been proposed and studied by researchers. Most of them are focused on the applications of associative memory. Some of the networks proposed use the feedforward structure. While the majority use the feedback structure (the recurrent structure). Our main interest is in the later. In the following sections, we will study some of the popular ANNs used for store static or temporal patterns.

2.2 The Hopfield Model

The analog Hopfield model [Hop84] is a recurrent network. One of the major contributions of this model is that it can be easily built with analog circuit and it is suitable for VLSI implementation. Figure 2.1 shows a circuit model of an element (neuron) of a Hopfield model. Figure 2.2 shows a Hopfield network. In Hopfield model, each

processi

a resist

transfer

finite co

$i \in \{1.2.$

are expr

where I_i i

Let $R_{i,j} =$

The system

processing unit (neuron) i, $i = 1, 2, \ldots, n$, is an amplifier with a capacitor C_i and a resistor ρ_i at the input node. The processing node uses a sigmoid function S to transfer input u_i to output v_i. Node (neuron) i is connected to node (neuron) j via a finite conductance T_{ij}. The connection matrix $T = [T_{ij}]$ is symmetric and $T_{ii} = 0$ for $i \in \{1, 2, \ldots, n\}$. The system dynamic equations can be derived by using KCL, and are expressed as:

$$\begin{aligned} C_i \frac{du_i}{dt} &= \sum_{j=1}^{n} T_{ij}(v_j - u_i) - \frac{u_i}{\rho_i} + I_i \\ &= \sum_{j=1}^{n} T_{ij} v_j - (\sum_{j=1}^{n} T_{ij} + \frac{1}{\rho}) u_i + I_i \qquad i = 1, 2, \ldots, n, \end{aligned} \tag{2.1}$$

where I_i is an input bias current to neuron i, and

$$v_j = S(u_j) \qquad j = 1, 2, \ldots, n. \tag{2.2}$$

Let $R_{ij} = \frac{1}{T_{ij}}$ and

$$\begin{aligned} \frac{1}{R_i} &= \frac{1}{\rho_i} + \sum_{j=1}^{n} \frac{1}{R_{ij}} \\ &= \frac{1}{\rho_i} + \sum_{j=1}^{n} T_{ij}. \end{aligned} \tag{2.3}$$

The system differential equations become:

$$C_i \frac{du_i}{dt} = \sum_{j=1}^{n} T_{ij} v_j - \frac{u_i}{R_i} + I_i \qquad i = 1, 2, \ldots, n. \tag{2.4}$$

The

The deri

as:

where v_i =

Since C_i >

When $\frac{dE}{dt}$ =

So u_i is a e

stable point

system. [SW

Hopfield

The "energy" function of the system is defined as

$$E = -\sum_{i=1}^{n}\sum_{j=1}^{n} T_{ij} v_i v_j - \sum_{i=1}^{n} I_i v_i + \sum_{i=1}^{n} \frac{1}{R_i} \int_0^{v_i} S^{-1}(\eta_i) d\eta_i. \tag{2.5}$$

The derivative of the energy function along the trajectory of the system can be derived as:

$$\frac{dE}{dt} = -\sum_{i=1}^{n} \frac{1}{c_i} \frac{dS(u_i)}{du_i} \left(\frac{\partial E}{\partial v_i}\right)^2 \tag{2.6}$$

where $v_i = S(u_i)$ and S is a sigmoid function. Thus

$$\frac{dS}{du} > 0. \tag{2.7}$$

Since $C_i > 0$ and $\left(\frac{\partial E}{\partial v_i}\right)^2 \geq 0$,

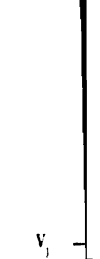
$$\frac{dE}{dt} \leq 0 \quad \text{for} \quad t \geq 0. \tag{2.8}$$

When $\frac{dE}{dt} = 0$

$$\begin{aligned} \frac{\partial E}{\partial v_i} &= 0 \qquad i = 1, 2, \ldots, n \\ \Longrightarrow \quad & -c_i \frac{du_i}{dt} = 0. \end{aligned}$$

So u_i is a equilibrium point of the system. Therefore the system will converge to a stable point (equilibrium point) starting from any point. This system is a gradient-like system. [SWC91] has more detailed analysis of this system.

Hopfield network functions as associative memory by storing desired patterns as



stable equ

input "sim

the stable

However, i

network ca

ways of fin

by Barbosa

There i

can be desc

We omit th

The Hop

• The st

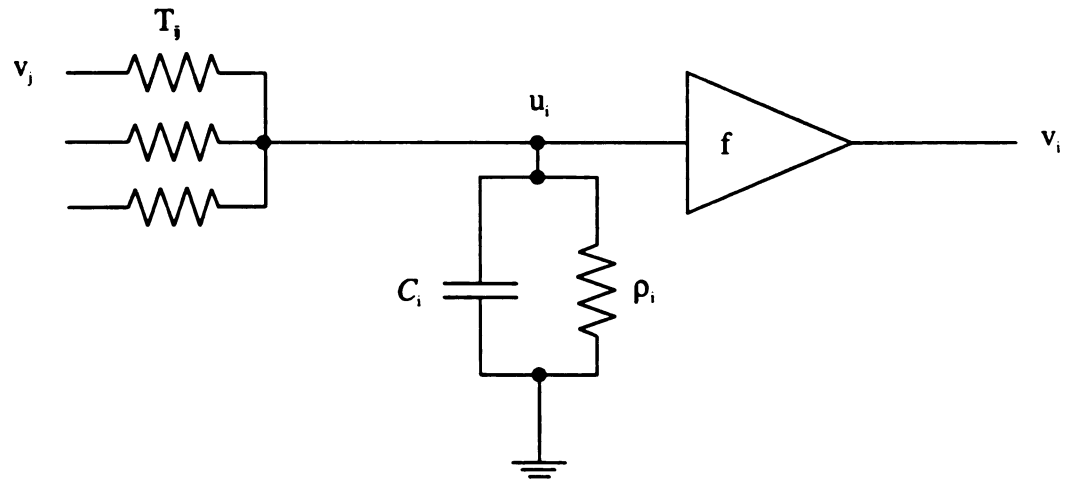


Figure 2.1. A neuron of an analog Hopfield model

stable equilibrium points of the system. Thus, when the system is presented with an input "similar" to one of the stored patterns (within the region of attraction of one of the stable equilibrium point), the system will converge to the closest stored pattern. However, it is difficult to determine the symmetric connection (weight) matrix T. The network cannot store arbitrary patterns either. There are many papers published on ways of finding the weight matrix for a set of given patterns, such as the one proposed by Barbosa [Bar91], and the one by Tseng *et al* [THL93].

There is a discrete version of Hopfield model too [Hop82]. The system dynamics can be described by:

$$V_i = \text{sign}\left(\sum_j T_{ij} V_j\right) \tag{2.9}$$

We omit the detailed analysis here.

The Hopfield-type model has the following limitations:

- The storage capacity is relatively small, i.e. for a fixed number of neurons the

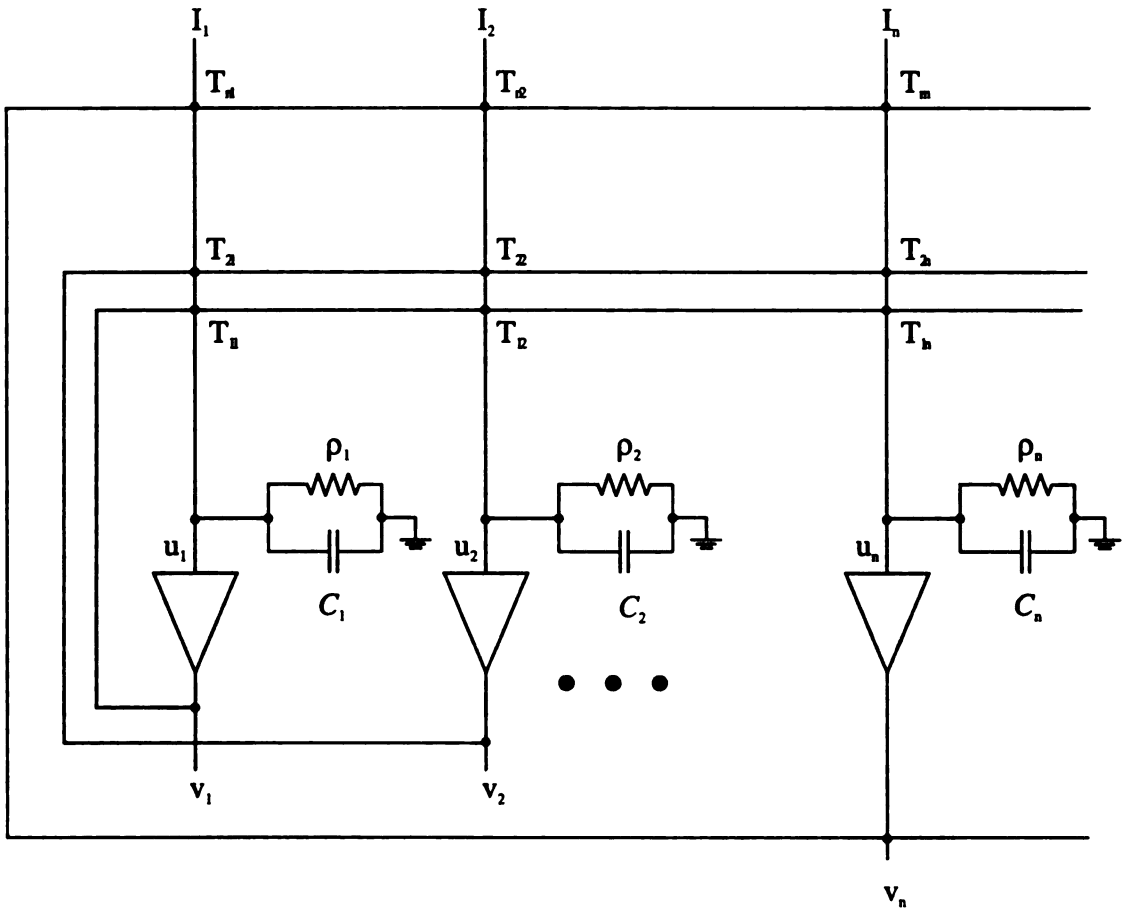


Figure 2.2. A network of Hopfield model

nu

- Gi

im

se

Ho

wh

- On

2.3 H

Bart Kosk

network (

pattern) a

connected

When the

this: the o

function: t

M^T: the ou

$A = S(M^T$

one). Give

given B as

proved that

connection

way to find

number of "storable" patterns is small.

- Given an arbitrary set of patterns to be stored, it is difficult and maybe impossible to find a weight matrix. That is, the network cannot store arbitrary sets of patterns. For example, assume $I_i = 0$, then the system equilibria of the Hopfield model must satisfy the condition: $F^T(V^i)V^j = F^T(V^j)V^i$, for $i \neq j$; where $F(V) = R^{-1}S^{-1}(V)$ [SWC91].

- On-line learning is difficult and computationally expensive.

2.3 Bidirection Associative Memory

Bart Kosko extended the original Hopfield type associative memory to a two-layer network [Kos88], as shown in Figure 2.3. Each layer can represent a vector (or pattern) and the two layers can have different number of neurons. The two layers are connected through the weight matrix (connection matrix) M and its transpose M^T. When the system is given an initial input to layer one, say A', the system works like this: the output of the layer two is calculated by $B' = S(MA')$, $S()$ is a threshold function; the output of layer two is then fed to layer one again through connection M^T; the output of layer one then become $A'' = S(M^TB)'$ ... until $B = S(MA)$ and $A = S(M^TB)$. A and B are the stored patterns of the network (may not be a desired one). Given pattern A as the input, the network can retrieve pattern B; similarly, given B as a input the network can retrieve pattern A for layer one. Kosko [Kos88] proved that any real matrix M is bidirectionally stable (i.e. use matrix M as the connection matrix, the resulting network is stable). Kosko proposed the following way to find the connection matrix: for a given set of patterns (A_1, B_1), (A_2, B_2), ...,

(Am.

wher

defin

Ho

guaran

(A_m, B_m), matrix M can be constructed as follows:

$$M = \sum_{i=1}^{n} X_i^T Y_j, \tag{2.10}$$

where X_i and Y_i are the bipolar form of patterns A_i and B_i respectively and are defined as:

$$x_{ij} = \begin{cases} a_{ij} & \text{if } a_{ij} > 0 \\ -1 & \text{if } a_{ij} \leq 0 \end{cases}, \tag{2.11}$$

$$y_{ij} = \begin{cases} b_{ij} & \text{if } b_{ij} > 0 \\ -1 & \text{if } b_{ij} \leq 0 \end{cases}. \tag{2.12}$$

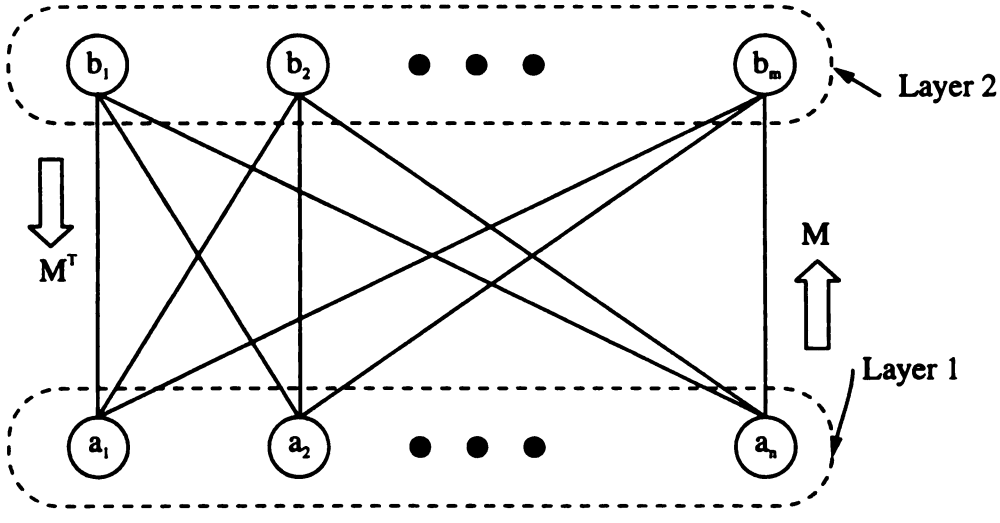


Figure 2.3. Structure of Bidirection Associative Memory

However, as pointed by Wang *et al.* [WCM89], this proposed matrix cannot guarantee the recall of the desired patterns, i.e. some of the desired patterns may

not be

Training

single pa

proposed

idea is t

In [WCJ

connectic

associativ

also prop

for a give

The a

limitation

It is also

2.4 R

t

Many alg

store/gene

recurrent

current alg

section, a

analysis wi

recurrent n

not be stable equilibrium points of the system. Wang *et al.* proposed "multiple Training Method" to build the matrix M, which can guarantee a successful recall of a single pair of patterns [WCM90]. To guarantee the recall of all training pairs, Wang proposed a method called "Dummy Augmentation Encoding" [WCM90]. The basic idea is to increase the dimension of the stored patterns to make them "noise-free". In [WCJ91], Wang proposed a necessary and sufficient condition for a generalized connection matrix (also called correlation matrix in Wang's paper) of a bidirectional associative memory (BAM) which guarantees the recall of all the training pairs. He also proposed to use linear programming techniques to find such matrix. However, for a given set of patterns this condition may not be satisfied.

The advantage of BAM is its ability to store patterns in pairs. The major limitation with BAM is that it cannot guarantee to store an arbitrary set of patterns. It is also difficult to find the connection matrix if it exists at all.

2.4 Existing Models for Storing Temporal Patterns

Many algorithms have been proposed for training ANNs so that they can store/generate temporal patterns. To be able to generate temporal patterns, A recurrent network structure is usually required. Our main interest here is to extend current algorithms and to improve the performance of current methods. In this section, a brief introduction will be given to some popular algorithms. Detailed analysis will be given in later chapters. Our focus is on the algorithms of training a recurrent network to store/generate temporal patterns.

Back-P

Back

[RHWS

so that

correspo

connecti

the origi

feedforw

The limit

The leng

requires t

Real-tim

A met

general re

training a

of the "en

the deriva

method is

computati

needs larg

Time-de

This n

neural net

function"

Back-Propagation Through Time

Back-propagation through time was proposed by Rumelhart, Hinton and Williams [RHW86]. This method "unfold" a fully connected recurrent network through time, so that the network can be represented by a feedforward network with each layer corresponding to a state of the recurrent network at certain time constant. The connections between each layer are the same, and they are the weight matrix of the original recurrent network. Therefore, the popular back-propagation method of feedforward network can be slightly modified to be used to train this kind of network. The limitation is that this algorithm cannot be used to train arbitrary length patterns. The length of the training patterns need to be fixed and known. This method also requires too much computer resources and is difficult to implement in hardware.

Real-time Recurrent Learning

A method proposed by Williams and Zipser [WZ89b, WZ89a] can be used to train general recurrent networks. It does not require the patterns to have fixed length. The training algorithms use a gradient descent method to update weights. The derivative of the "energy" function with respect to weights is calculated by integrating forward the derivative of the outputs of the neurons with respect to weights. Thus this method is a forward propagation method. The drawbacks with this method are: it is computationally very expensive, each time step requires $O(n^4)$ of operation. It also needs large memory storage.

Time-dependent Recurrent Back-propagation

This method was proposed by Pearlmutter [Pea89] to train general recurrent neural networks. Other researchers developed similar method, such as the "adjoint function" method proposed by Toomarian and Barhen [TB91]. The time-dependent

recurre

method

using a

"energy

(adjoint

it needs

and hard

The Gr

Sun. (

function

it used a

computat

equation

simulatio

In the

networks

the limit

the ways

patterns.

recurrent back-propagation method is in essence an extension to the back-propagation method used in feed forward networks. This method updates the weight/connections using a gradient-descent algorithm too. However, this time, the derivative of the "energy" function with respect to weights is calculated using a co-state variable (adjoint system). The computation per time step is in the order of $O(n^2)$. However, it needs to integrate the co-state backward in time. Thus difficult to be used on-line and hard to be implemented in hardware.

The Green's Function Method

Sun, Chen and Lee proposed another way of finding the derivative of the "energy" function with respect to weights [SCL91]. Instead od using back-propagation method, it used a "Green" function. This method can be integrated forward in time. The computation steps are in the order of $O(n^3)$. This method needs to solve a linear equation at each time step, which may not have a unique solution in numerical simulations.

In the rest of this dissertation, we will first propose a new model of artificial neural networks for association of static patterns. This new model will overcome some of the limitations we have pointed out in current approaches. Then we will investigate the ways to improve current methods and models used for association of temporal patterns.

Cl

Tl

M

N

3.1

As c

funct

store

the d

(1) b

D

the p

els, t

Pins

CHAPTER 3

THE PRODUCT-OF-NORM MODEL OF ARTIFICIAL NEURAL NETWORKS

3.1 Motivation

As can be seen from previous chapters, two of the major issues in the design of functional artificial neural networks (ANN) are [SB91, SWC91]: (1) the ability to store (or learn) any number of pre-specified set of data (patterns) successfully; (2) the determination of a set of parameters (or weights, connections) which would achieve (1) by using efficient off-line or on-line computations.

Dynamic learning has been a more attractive way in (adaptively) computing the parameters of an ANN. For (static) feedforward artificial neural network models, the error-back-propagation is one of such dynamic methods [RHW86]. Pineda [Pin87, Pin88, Pin89] and some other researchers have ported the back-propagation

techr

prop

H

expe

is lir

know

T

first.

meth

The

netw

capa

I

of pa

of da

its p

the s

will

com

techniques to recurrent networks too. Other learning methods such as the one proposed by Perlmutter [Pea89] in essence are error back-propagation method too.

However, most of the proposed algorithms proposed so far are computationally expensive and the number of sets of data (patterns) which can be reliable learned is limited. Moreover, additional data may be inadvertently learned making what is known as spurious states.

The usual models of ANN fix the architecture and the number of units (neurons) first, then seek to "compute" parameter values, either through off-line or on-line methods, which render a given set of data stable equilibria of input/output pair. The structure is not dependent on the number of elements of the set of data. These networks spend lots of time in learning stage and are well known to have limited capacity in storing arbitrary set of data.

In this proposed model, we make trade-offs: we eliminate the dynamic learning of parameters but make the number of units or neurons dependent on the number of data to be stored. The resulting network does not "waste" any time on adapting its parameters and the given patterns are guaranteed to be the stable equilibria of the system (i.e. they are guaranteed to be recalled). However, the network's "size" will depend on the number of desired data to be stored. The network becomes more complex: the connectivity or interaction among neurons increases.

3.2

We st

with:

where

V

T

syste

is bo

3.2 Proposed Mathematical Model

We start the network by proposing a new general "energy" function of the network with:

$$E = \frac{1}{p} f(m) \prod_{l=1}^{m} ||e^l||_p^p, \tag{3.1}$$

where $e^l \triangleq [(v_1^{*l} - v_1), (v_2^{*l} - v_2), \ldots, (v_n^{*l} - v_n)]$;

$||e^l||_p$ means the p-th norm of vector e^l with $1 < p < \infty$,

($p = 1$ and $p = \infty$ need special treatment) ;

v_j : the output of the j-th neuron;

n : the number of neurons of the system;

m : the number of patterns (data set) to be stored;

v^{*l} : the l-th specified pattern;

v_j^{*l} : the j-th component of vector $\mathbf{V}^{*l}$;

f : a scaling function and $f(m) > 0$;

u_j : the input to the j-th neuron, and $v_j = S(u_j)$;

S : a sigmoid function (a bounded, monotone increasing diffeomorphism);

In particular we have used $S = \tanh()$ in our simulations.

We define the system dynamic equations by

$$\dot{u}_i = -\frac{\partial E}{\partial v_i} \qquad i = 1, 2, \ldots, n. \tag{3.2}$$

This is a gradient-like system. It preserves all the characteristics of a gradient system, such as no oscillations and global convergence. Since $E \geq 0$, the "energy" is bounded from below. Thus, the system is guaranteed to converge to its limit set

which

3.3

For th

is: if

recall

Theo

V^{*l} =

in thi

asym

Proo

Si

then

and

If

ien

which is composed of equilibria only [HS74].

3.3 Stability Analysis

For the new artificial neural network architecture, the first issue we want to address is: if the desired patterns are the stable equilibria of the system, i.e. can they be recalled? Theorem 3.1 answers this question.

Theorem 3.1 *For the system defined by equation* (3.1) *and equation* (3.2), *suppose* $\mathbf{V}^{*l} = [v_1^{*l}, v_2^{*l}, \ldots, v_n^{*l}]^T$, $l = 1, 2, \ldots, m$, *are the desired/pre-specified patterns* (*note: in this paper we use bold face to represent vectors*), *Then* $\mathbf{V}^{*l}$, $l = 1, 2, \ldots, m$, *are asymptotically stable equilibria of the system, i.e. they are guaranteed recallable.*

Proof:

Since

$$\mathbf{e}^l(\mathbf{V}) = [(v_1^{*l} - v_1), (v_2^{*l} - v_2), \ldots, (v_n^{*l} - v_n)]^T,$$

then

$$\mathbf{e}^l(\mathbf{V}^{*l}) = [0, 0, \ldots, 0]^T$$

and

$$||\mathbf{e}^l||_p = 0 \Longrightarrow E(\mathbf{V}^{*l}) = 0.$$

If

$$\mathbf{V} \neq \mathbf{V}^{*l} \qquad \forall l = 1, 2, \ldots, m,$$

then

$$||\mathbf{e}^l(\mathbf{V})||_p > 0 \qquad \forall l = 1, 2, \ldots, m.$$

Since $\frac{1}{p}$

Therefor

assume V

pattern

that $E($V

$\mathbf{V} \neq \mathbf{V}^{*}$

system ca

where $\frac{\partial v_i}{\partial u_i}$

Therefore,

cally stabl

The se

pre-specifi

From t

Lyapunov

Since $\frac{1}{p} > 0$ and $f(m) > 0$, thus

$$E(\mathbf{V}) > 0 \qquad \forall \mathbf{V} \notin \{\mathbf{V}^{*1}, \mathbf{V}^{*2}, \ldots, \mathbf{V}^{*m}\}.$$

Therefore, $\mathbf{V}^{*l}$, $l = 1, 2, \ldots, m$, are isolated minima of the "energy" function (we assume $\mathbf{V}^{*l}$, $l = 1, 2, \ldots, m$, are distinct). Consider a small nerghberhood Ω around pattern k such that $\mathbf{V}^{*k}$ is the only system equilibrium point inside Ω, It is obvious that $E(\mathbf{V})$ is continuously differentiable over Ω. $E(\mathbf{V}^{*k}) = 0$ and $E(\mathbf{V}) > 0$ for $\mathbf{V} \neq \mathbf{V}^{*k}$, $\mathbf{V} \in \Omega$. The derivative of the "energy" function along the trajectory of the system can be expressed as:

$$\frac{dE(\mathbf{V})}{dt} = \sum_{i=1}^{n} \frac{\partial E}{\partial v_i} \frac{\partial v_i}{\partial u_i} \dot{u}_i \tag{3.3}$$

$$= -\sum_{i=1}^{n} \left(\frac{\partial E}{\partial v_i} \right)^2 \frac{\partial v_i}{\partial u_i}, \tag{3.4}$$

where $\frac{\partial v_i}{\partial u_i} = \frac{\partial S(u_i)}{\partial u_i}$. By our choice of the function S, we have $\frac{\partial S(u_i)}{\partial u_i} > 0$. Thus

$$\frac{dE(\mathbf{V})}{dt} < 0 \qquad \text{for} \quad \mathbf{V} \neq \mathbf{V}^{*k} \tag{3.5}$$

Therefore, according to *Lyapunov's* stability theorem [Kha91], $\mathbf{V}^{*k}$ is an asymptotically stable equilibrium point of the system.

The second question about the new system is if the regions of attraction for each pre-specified patterns can be estimated, i.e. how stable are the desired patterns?

From the proof above we know that the "energy" function E can serve as a *Lyapunov* function for the equilibrium point $\mathbf{V}^{*k}$ over the region $\Omega(\mathbf{V}^{*k})$. Thus we

can ca

$k = 1.2$

In t

will ana

3.4

When u

can be v

The syst

$\dot{u}$

From

pattern, s

can calculate the *Lyapunov surface* to estimate the regions of attract for each $\mathbf{V}^{*k}$, $k = 1, 2, \ldots, m$.

In this section, we investigated the overall stability of the new architecture. We will analyze this new model for specific norms in the following sections.

3.4 The Product-of-norm Model Using L_2 Norm

When using L_2 norm ($p = 2$) in the product-of-norm model, the "energy" function can be written as:

$$\begin{aligned} E &= \frac{1}{2} f(m) \prod_{l=1}^{m} ||\mathbf{e}^l(\mathbf{V})||_2^2 \\ &= \frac{1}{2} f(m) \prod_{l=1}^{m} ||\mathbf{V}^{*l} - \mathbf{V}||_2^2 \\ &= \frac{1}{2} f(m) \prod_{l=1}^{m} \left[(v_1^{*l} - v_1)^2 + (v_2^{*l} - v_2)^2 + \cdots + (v_n^{*l} - v_n)^2 \right]. \end{aligned} \tag{3.6}$$

The system dynamic equations now become

$$\begin{aligned} \dot{u}_i &= f(m) \sum_{l=1}^{m} (v_i^{*l} - v_i) \prod_{\substack{k=1 \\ k \neq l}}^{m} ||\mathbf{V}^{*k} - \mathbf{V}||_2^2 \\ &= f(m) \sum_{l=1}^{m} (v_i^{*l} - v_i) \prod_{\substack{k=1 \\ k \neq l}}^{m} \left[\sum_{j=1}^{n} (v_j^{*k} - v_j)^2 \right] \qquad i = 1, 2, \ldots, n. \end{aligned} \tag{3.7}$$

From the above system dynamic equations, it is obvious that at any desired pattern, say $\mathbf{V}^{*l}$, $\dot{\mathbf{U}} = 0$. It should also be observed that for a vector $\mathbf{V}$ to be a

system

its com

Otherw

or

Thus $\dot{u}_i$

range of

the follo

Theore

- *Th*
- *Fo*

(3.

Note

system equilibrium point, i.e.

$$0 = f(m)\sum_{l=1}^{m}(v_i^{*l} - v_i)\prod_{\substack{k=1\\k\neq 1}}^{m}\left[\sum_{j=1}^{n}(v_j^{*k} - v_j)^2\right],$$

its components must satisfy the following conditions:

$$\begin{cases} \min\left\{v_i^{*l} : l = 1,2,\ldots,m\right\} \leq v_i & i = 1,2,\ldots,n \\ \max\left\{v_i^{*l} : l = 1,2,\ldots,m\right\} \geq v_i & i = 1,2,\ldots,n. \end{cases} \tag{3.8}$$

Otherwise, there will exist an $i \in \{1,2,\ldots,n\}$ such that

$$(v_i^{*l} - v_i)\prod_{\substack{k=1\\k\neq l}}^{m}\sum_{j=1}^{n}(v_j^{*k} - v_j)^2 > 0 \qquad \forall l \in \{1,2,\ldots,m\} \tag{3.9}$$

or

$$(v_i^{*l} - v_i)\prod_{\substack{k=1\\k\neq l}}^{m}\sum_{j=1}^{n}(v_j^{*k} - v_j)^2 < 0 \qquad \forall l \in \{1,2,\ldots,m\}. \tag{3.10}$$

Thus $\dot{u}_i \neq 0$, and $\mathbf{V}$ is not a system equilibrium point. Inequality (3.8) specifies the range of the locations of all the system equilibria. To summarize the above, we have the following theorem:

Theorem 3.2 *For the dynamic system defined by equation* (3.7)

- *The desired patterns* $\mathbf{V}^{*l}$, $l = 1,2,\ldots,m$, *are the asymptotically stable equilibria.*
- *For a vector* $\mathbf{V}$ *to be a system equilibrium point, it must satisfy the inequality* (3.8).

Note that this theorem can be applied to general pth norm also.

3.4.1

For on

vestiga

and th

Theore

ties:

- *T*
- *T*

  T
- *S*

  fo

  be

Proof:

Let

3.4.1 One-neuron/m-pattern Case

For one-neuron/m-pattern case, the system's characteristic can even be further investigated. When $n = 1$, the "energy" function becomes

$$E = \frac{1}{2} f(m) \prod_{l=1}^{m} (v^{*l} - v)^2 \tag{3.11}$$

and the system dynamic equation becomes:

$$\dot{u} = f(m) \sum_{l=1}^{m} \left[(v^{*l} - v) \prod_{\substack{k=1 \\ k \neq l}}^{m} (v^{*k} - v)^2 \right]. \tag{3.12}$$

Theorem 3.3 *The system represented by equation* (3.12) *has the following properties*:

- *The system has a total of* $2m - 1$ *equilibrium points.*
- *The pre-specified patterns* $\mathbf{v}^{*l}$. $(l = 1, 2, \ldots, m)$ *are asymptotic stable equilibria. The other* $m - 1$ *equilibria are unstable.*
- *Suppose* v^{*l}, $(l = 1, 2, \ldots, m)$, *are distinct and* $v^{*1} < v^{*2} < \cdots < v^{*m}$, *then for every* $i \in \{1, 2, \ldots, m-1\}$, *there is an unstable equilibrium point* $v^{*(i,i+1)}$ *between* v^{*i} *and* $v^{*(i+1)}$.

Proof:

Let us define a new function:

$$H(v) = f(m) \sum_{l=1}^{m} \left[(v^{*l} - v) \prod_{\substack{k=1 \\ k \neq l}}^{m} (v^{*k} - v)^2 \right]. \tag{3.13}$$

Then t

Since H

That m

Rew

where w

It is obv

Thus v^*

be given

At a

Then the roots of $H(v)$ are the equilibria of the system defined by equation (3.12). Since $H(v)$ is a polynomial of order $2m-1$, it generally has a total of $2m-1$ roots. That means system (3.12) will have at most $2m-1$ equilibria.

Rewrite $H(v)$ as

$$\begin{aligned} H(v) &= f(m)\left[\prod_{l=1}^{m}(v^{*l}-v)\right]\left[\sum_{l=1}^{m}\prod_{\substack{k=1\\k\neq l}}^{m}(v^{*l}-v)\right] \\ &= H_0(v)H_1(v), \end{aligned} \tag{3.14}$$

where we define

$$H_0(v) = f(m)\left[\prod_{l=1}^{m}(v^{*l}-v)\right], \tag{3.15}$$

$$H_1(v) = \left[\sum_{l=1}^{m}\prod_{\substack{k=1\\k\neq l}}^{m}(v^{*l}-v)\right]. \tag{3.16}$$

It is obvious that at the specified patterns v^{*l} $(l=1,2,\ldots,m)$,

$$H_0(v^{*l}) = 0 \qquad \text{and} \qquad H(v^{*l}) = 0. \tag{3.17}$$

Thus v^{*l}, $l=1,2,\ldots,m$, are equilibria of the system. The other $m-1$ equilibria will be given by the roots of function $H_1(v)$.

At a specific pattern v^{*i},

$$\begin{aligned} H_1(v^{*i}) &= \sum_{l=1}^{m}\prod_{\substack{k=1\\k\neq l}}^{m}(v^{*k}-v^{*i}) \\ &= \prod_{\substack{k=1\\k\neq i}}^{m}(v^{*k}-v^{*i}) \end{aligned}$$

Since

i.e. sig

Because

interval

number

these ro

The

been sho

1. can b

The

where c

Evalu

$$= \left[\prod_{k=1}^{i-1}(v^{*k} - v^{*i})\right]\left[\prod_{k=i+1}^{m}(v^{*k} - v^{*i})\right]. \tag{3.18}$$

Since $v^{*k} - v^{*i} < 0$ for every $k \in \{1, 2, \ldots, i-1\}$,

$$\text{sign}\left(\prod_{k=1}^{i-1}(v^{*k} - v^{*i})\right) = (-1)^{i-1},$$

i.e. $\text{sign}(H_1(v^{*i})) = (-1)^{i-1}$. Therefore

$$H_1(v^{*i})H_1(v^{*(i+1)}) < 0 \qquad i = 1, 2, \ldots, m-1.$$

Because $H_1(v)$ is a continuous function of v, there must be a root $v^{*(i,i+1)}$ in each interval $(v^{*i}, v^{*(i+1)})$, $i = 1, 2, \ldots, m-1$, such that $H_1(v^{*(i,i+1)}) = 0$. The total number of roots we have now is $2m-1$. Since $H(v)$ is a polynomial of order $2m-1$, these roots must be the only equilibria of the system.

The stability of the system at the pre-specified patterns v^{*i}, $i = 1, 2, \ldots, m$, has been shown at previous section. The instability of patterns $v^{*(i,i+1)}$, $i = 1, 2, \ldots, m-1$, can be proved as follows:

The function $H_1(v)$ may now be expressed as

$$H_1(v) = c\prod_{q=1}^{m-1}(v^{*(q,q+1)} - v), \tag{3.19}$$

where $c > 0$, and c is a constant.

Evaluating the derivative of equation (3.14) with respect to v at $v^{*(r,r+1)}$, we obtain

$$\frac{d}{dv}H(v^{*(r,r+1)}) = H_0(v^{*(r,r+1)})\frac{d}{dv}H_1(v^{*(r,r+1)}), \tag{3.20}$$

where

Theref

Since

thus

Theref

So they

Fig

neuron

are glo

equatio

points

roots b

cross th

where

$$\begin{aligned}\frac{d}{dv}H_1(v^{*(r,r+1)}) &= -c\prod_{\substack{q=1\\q\neq r}}^{m-1}(v^{*(q,q+1)}-v^{*(r,r+1)})\\ &= (-1)^{r-1}c\prod_{\substack{q=1\\q\neq r}}^{m-1}|v^{*(q,q+1)}-v^{*(r,r+1)}|. \qquad (3.21)\end{aligned}$$

Therefore

$$\text{sign}\left(\frac{d}{dv}H_1(v^{*(r,r+1)})\right) = (-1)^{r-1}. \qquad (3.22)$$

Since

$$\begin{aligned}\text{sign}\left(H_0(v^{*(r,r+1)})\right) &= \text{sign}\left(f(m)\left[\prod_{l=1}^{m}(v^{*l}-v^{*(r,r+1)})\right]\right)\\ &= (-1)^r, \qquad (3.23)\end{aligned}$$

thus

$$\frac{d}{dv}H(v^{*(r,r+1)}) = \frac{d}{dv}H_1(v^{*(r,r+1)})H_0(v^{*(r,r+1)}) < 0. \qquad (3.24)$$

Therefore, $v^{*(r,r+1)}$, $r = 1, 2, \ldots, m-1$, are the maxima of the "energy" function. So they are unstable equilibria. QED

Figure 3.1 and 3.3 show two example plots of the "energy" function for two one-neuron/m-pattern cases. As we can see from the plots, all the desired patterns are global minima of the "energy" function. The corresponding system differential equations are plotted in Figure 3.2 and 3.4 where we can see that all the desired points are the roots at which the plots cross the x-axis with a negative slope. All the roots between the desired ones are between two consecutive stable ones and the plots cross the x-axis at those points with a positive slope.

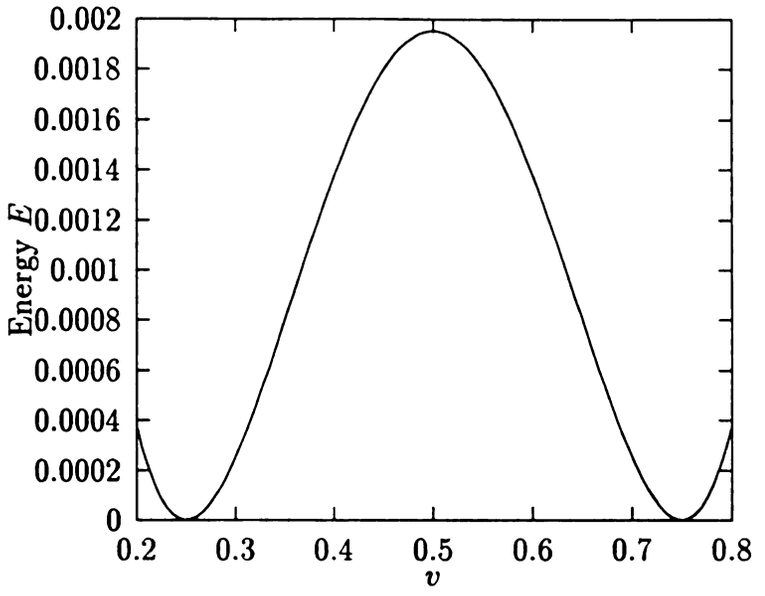


Figure 3.1. The plot of "energy" function $E = 0.5(0.25 - x)^2(0.75 - x)$

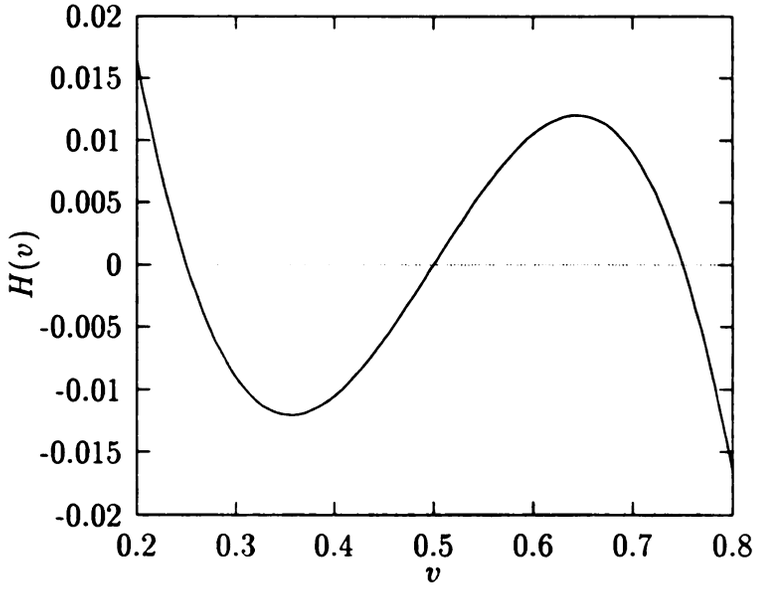


Figure 3.2. The plot of function $H(v)$ with $v^{*1} = 0.25$ and $v^{*2} = 0.75$

Figure

Figur

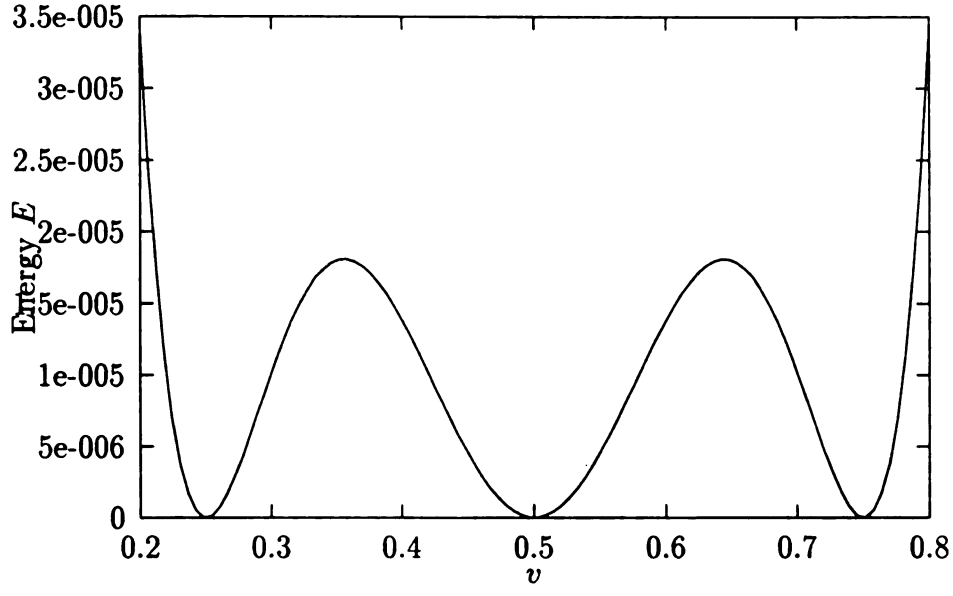


Figure 3.3. The plot of "energy" function $E = 0.5(0.25 - x)^2(0.5 - x)^2(0.75 - x)^2$

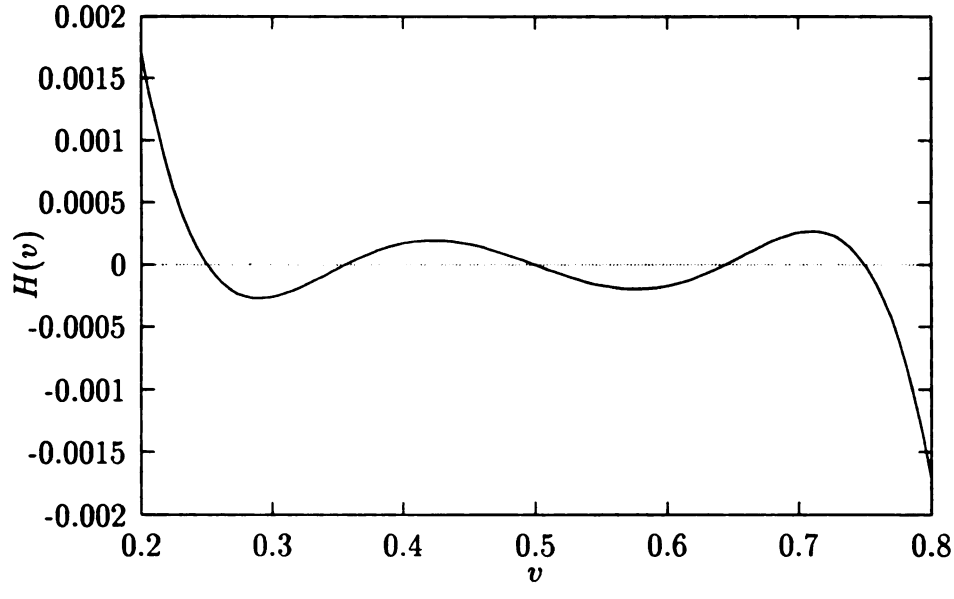


Figure 3.4. The plot of function $H(v)$ with $v^{*1} = 0.25$ $v^{*2} = 0.5$ and $v^{*3} = 0.75$

Le

neuron

each s

For ex

Withou

$m-1$

and the

The loc

stable e

this way

3.4.2

From t

location

cases for

Let us look back again at the proof of theorem (3.3). We find that for the one-neuron/m-pattern case, we can even further control the regions of attractions for each specified pattern by specifying the unstable equilibria between the stable ones. For example, suppose we want to store $v^{*1}, v^{*2}, \cdots$, and v^{*m} as the stable equilibria. Without loosing generality, we assume $v^{*1} < v^{*2} < \cdots < v^{*m}$, then we can select $m-1$ points: $v^1, v^2, \cdots, v^{m-1}$, such that

$$v^{*1} < v^1 < v^{*2} < v^2 < v^{*3} < \cdots < v^{m-1} < v^{*m}; \tag{3.25}$$

and the system differential equation can be constructed as

$$\begin{aligned} \dot{u} &= (v^{*1} - v)(v^1 - v)(v^{*2} - v)\cdots(v^{*m} - v) \\ &= \left[\prod_{l=1}^{m}(v^{*l} - v)\right]\left[\prod_{k=1}^{m-1}(v^k - v)\right]. \end{aligned} \tag{3.26}$$

The locations of $v^1, v^2, \cdots$, and v^{m-1} will control the regions of attractions for those stable equilibria. Figure 3.5 and figure 3.6 show two example functions $H(v)$ defined this way.

3.4.2 The Number of Equilibria for Product-of-norm Networks

From theorem (3.3), we have already known the number of equilibria and their locations for the case of one-neuron/m-pattern. Here we will investigate some special cases for two-neuron networks.

Figure
0.3 an

Figure
equili

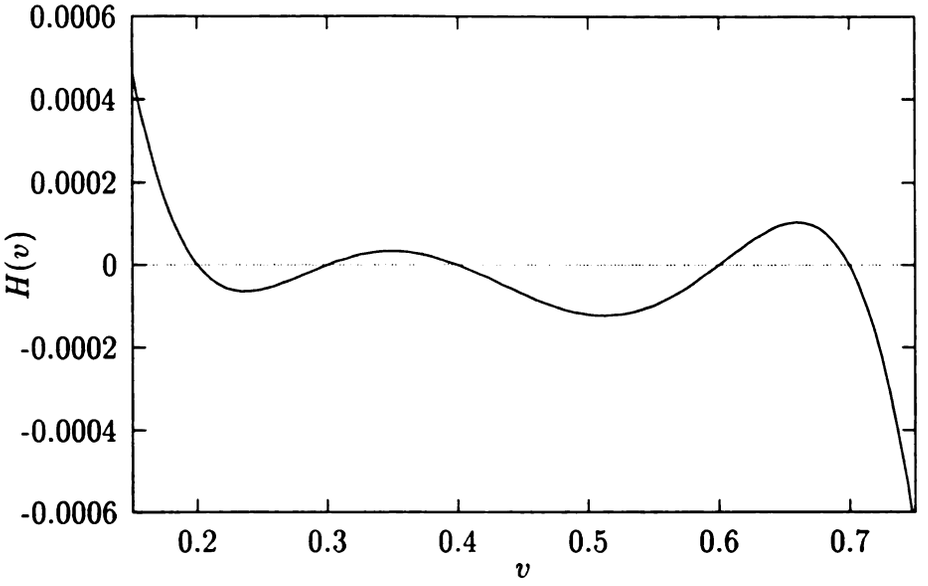


Figure 3.5. An $H(v)$ function with stable equilibria 0.2, 0.4, 0.7 and unstable ones at 0.3 and 0.6

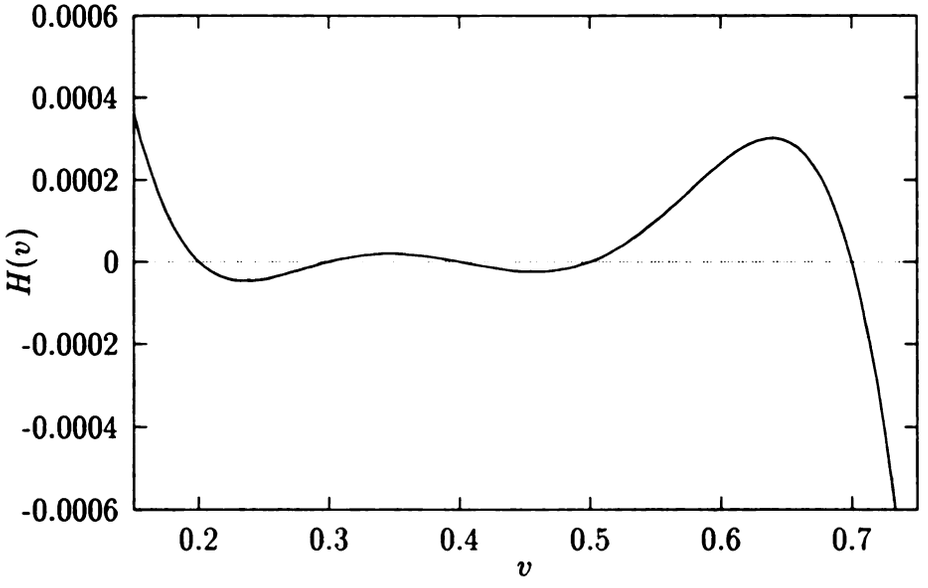


Figure 3.6. An $H(v)$ function with stable equilibria 0.2, 0.4, 0.7 and unstable equilibria at 0.3, 0.5

When

Since the

number o

the syster

For th

It is obvi

is stable.

For t

the syste

The

Therefo

When $n = 2$, the system differential equations are :

$$\begin{cases} \dot{u}_1 = \sum_{l=1}^{m}(v_1^{*l} - v_1) \prod_{\substack{k=1 \\ k\neq l}}^{m} [(v_1^{*k} - v_1)^2 + (v_2^{*k} - v_2)^2] \\ \dot{u}_2 = \sum_{l=1}^{m}(v_2^{*l} - v_2) \prod_{\substack{k=1 \\ k\neq l}}^{m} [(v_1^{*k} - v_1)^2 + (v_2^{*k} - v_2)^2] \end{cases} . \tag{3.27}$$

Since the shift or rotation of the axes does not change the relative locations and the number of equilibria of the system, we can always transform the system to simplify the system equations in the analysis.

For the case of $n = 2$ and $m = 1$, The system is trivial

$$\begin{cases} \dot{u}_1 = (v_2^* - v_1) \\ \dot{u}_2 = (v_2^* - v2) \end{cases} . \tag{3.28}$$

It is obvious that the system only has one equilibrium point and the equilibrium point is stable.

For the case of $m = 2$, since we can always transform the coordinates to simplify the system, we can assume, without losing generality, $v_2^{*1} = v_2^{*2} = 0$.

The system equations are

$$\begin{cases} \dot{u}_1 = (v_1^{*1} - v_1)[(v_1^{*2} - v_1)^2 + v_2^2] + (v_1^{*2} - v_1)[(v_1^{*1} - v_1)^2 + v_2^2] \\ \dot{u}_2 = -v_2[(v^{*2} - v_1)^2 + v_2^2] - v_2[(v_1^{*1} - v_2)^2 + v_2^2] \end{cases} . \tag{3.29}$$

Therefore, at system equilibria we have

$$u_2 = 0 \qquad \text{and} \qquad (v_1^{*1} - v_1)(v_1^{*2} - v_1)^2 + (v_1^{*2} - v_1)(v^{*1} - v_1)^2 = 0.$$

So $r_1 = r^*$

For th

only inves

straight li

The sy

and

At an e

So $v_1 = v^{*1}, v^{*2}$ or $\frac{v_1^{*1}+v_1^{*2}}{2}$. Thus the system has a total of three equilibria.

For the case of $m = 3$ and $n = 2$, the system becomes complicated. Here we only investigate the special case: the three pre-specified patterns are located on one straight line. Without losing generality, we assume $v_2^{*1} = v_2^{*2} = v_2^{*3} = 0$ and $v_1^{*3} = 0$.

The system differential equations become

$$\begin{aligned}\dot{u}_1 &= (v_1^{*1} - v_1)[(v_1^{*2} - v_1)^2 + v_2^2][v_1^2 + v_2^2] \\ &\quad +(v_1^{*2} - v_1)[(v_1^{*1} - v_1)^2 + v_2^2][v_1^2 + v_2^2] \\ &\quad +(-v_1)[(v_1^{*2} - v_1)^2 + v_2^2][(v_1^{*1} - v_1)^2 + v_2^2] \qquad (3.30)\end{aligned}$$

and

$$\begin{aligned}\dot{u}_2 &= -v_2\{[v_1^{*2} - v_1)^2 + v_2^2][(v_1^{*3} - v_1)^2 + v_2^2] \\ &\quad +[v_1^2 + v_2^2][(v_1^{*1} - v_1)^2 + v_2^2] \\ &\quad +[(v^{*1} - v_1)^2 + v_2^2][(v_1^{*2} - v_1)^2 + v_2^2]\}. \qquad (3.31)\end{aligned}$$

At an equilibrium point $v_2 = 0$ and

$$\begin{aligned}0 &= (v_1^{*1} - v_1)[(v_1^{*2} - v_1)^2][v_1^2] + \\ &\quad (v_1^{*2} - v_1)[(v_1^{*1} - 1)^2][v_1^2] - v_1[(v_1^{*2} - v_1)^2][(v_1^{*1} - v_1)^2] \\ 0 &= v_1^2(v_1^{*2} - v_1) + v_1^2(v_1^{*1} - v_1) - v_1(v_1^{*2} - v_1)(v_1^{*1} - v_1). \qquad (3.32)\end{aligned}$$

Thus $r_1 =$

The syster

3.5 T

For the cas

exist. The

software or

Since th

system grac

Thus $v_1 = 0$ or

$$0 = v_1(v_1^{*2} - v_1) + v_1(v_1^{*1} - v_1) - (v_1^{*2} - v_1)(v_1^{*1} - v_1) \tag{3.33}$$

$$\Longrightarrow \qquad v_1 = \frac{-(v_1^{*1} + v_1^{*2}) \pm \sqrt{(v_1^{*1})^2 - v_1^{*1}v_1^{*2} + (v_1^{*2})^2}}{-3}. \tag{3.34}$$

The system has a total of five equilibria.

3.5 The Case of L_∞ Norm

For the case of $p = \infty$ (L_∞ norm), the derivative of the norm does not always exist. The "energy" function is not continuous. This case need special treatment in software or hardware implementations. Here we define the "energy" function as:

$$\begin{aligned} E &= f(m)\prod_{l=1}^{m} ||\mathbf{V}^{*l} - \mathbf{V}||_\infty \\ &= f(m)\prod_{l=1}^{m} \max_j |v_j^{*l} - v_j|. \end{aligned} \tag{3.35}$$

Since the partial derivative $\frac{\partial}{\partial v_i}||\mathbf{V}^{*l} - \mathbf{V}||_\infty$ does not always exist, to make the system gradient-like, we define

$$\frac{\partial}{\partial v_i}||\mathbf{V}^{*l} - \mathbf{V}||_\infty \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } |v_i^{*l} - v_i| \neq \max_j |v_j^{*l} - v_j| \\ -1 & \text{if } v_i^{*l} - v_i = -\max_j |v_j^{*l} - v_j| \\ 1 & \text{if } v_i^{*l} - v_i = \max_j |v_j^{*l} - v_j| \end{cases} \tag{3.36}$$

$$\stackrel{\text{def}}{=} \delta_{li}. \tag{3.37}$$

The syste

Since the

simulation

Figu

equilibr

system

system

of the

compu

The system differential equations are defined as

$$\dot{u}_i = f(m)\left[\sum_{i}^{m} \delta_{ki} \prod_{\substack{l=1 \\ l\neq k}}^{m} \max_{j} |v_j^{*l} - v_j|\right]. \tag{3.38}$$

Since the $\dot{u}_i$ is not continuous, we need to choose $f(m)$ to be very small in software simulations.

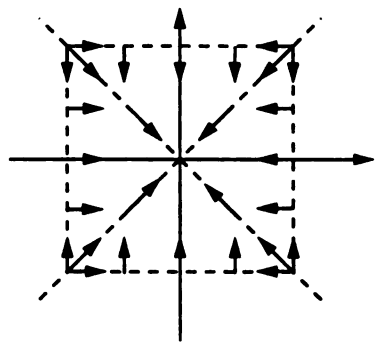


Figure 3.7. An example vector field for L_∞ norm

Figure 3.7 shows how the vector field of this system will look like around a stable equilibrium point in two dimension case. As it can be seen from the figure that the system has limit number of directions to go at any point. If we fix the step size fo the system, the system will become a grid system. The system will move from one node of the grid to another. This may make it easier to simulate the system using digital computers.

3.6 Simulation Results

To verify the correctness of the theoretic results and demonstrate how this model can be used in practical applications, we present several simulation results of this model used in image recognition and association. In the following examples, we use the outputs of neurons to represent pixels of images. Each pixel is shown as a square in the figures below. The neuron output value of 1 represents a full back square in the figures, the value of 0 represents a full white square and values between 0 and 1 represent black squares proportional to the values. We use L_2 norm in the simulations.

In example one, we considered a 304-neuron network. For each desired image, we stored the image and an assigned code in the network. This way we can read the output of the network by reading the outputs of the neurons representing the codes. To compare its performance with BAM [WCM89, WCM90, WCJ91], we have used the same image patterns as those used in [WCM89].

Figure 3.8 shows an example where six image-code pairs are stored. When the network was given the initial input as shown in the figure, the network retrieved the corresponding images correctly.

In example two, we used a network with 37011 (169x219) neurons to store two facial images as shown in figure 3.9. Figure 3.10 shows that given a part of an image, the network was able to retrieve the who image correctly.

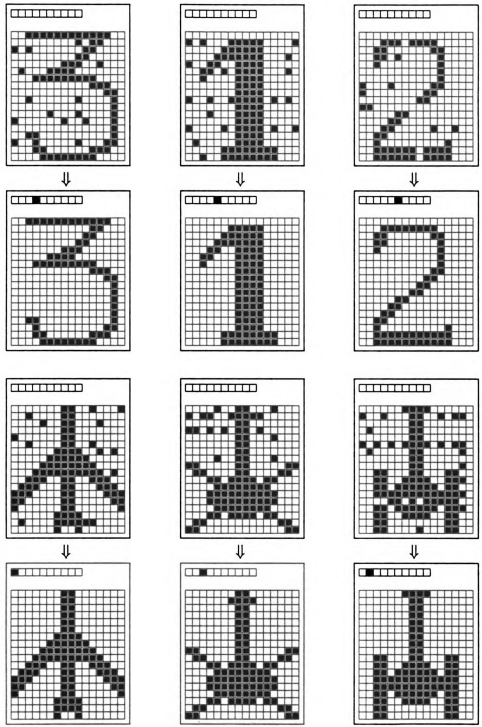


Figure 3.8. Initial and final outputs of the network with six pre-specified patterns

Figure 3.9. Stored two facial images

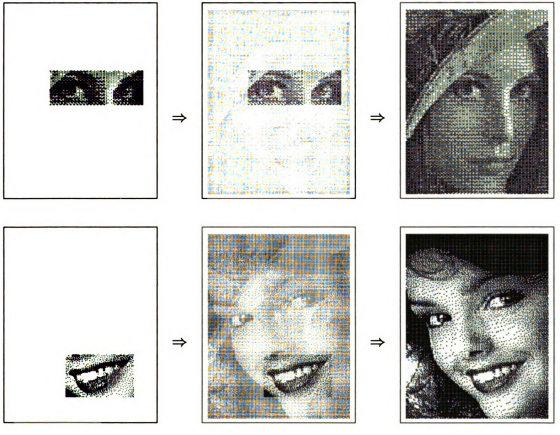


Figure 3.10. Image retrieving process

CHAPTER 4

GENERALIZATION AND VARIATIONS OF THE "ENERGY" FUNCTION FOR PRODUCT-OF-NORM MODEL

In general, the proposed product-of-norm model is very complex: neurons are fully connected to one another. For the L_2 norm case, the order of the system increases as the number of data sets increases. It is also difficult to modularize or parallelize the system in software simulations or hardware implementations. In this chapter, we will investigate various ways to simplify the system. We will also present some example ways of tailoring the "energy" function for specific applications. Our focus here is the L_2 norm form of the product-of-norm model. However, the techniques discussed here can be applied to other forms of norm too.

4.1 The Log Form of the "Energy" Function

If we take log() on both sides of the original "energy" function (3.1). We get a new "energy" function as:

$$\tilde{E} \triangleq \log E = f_1(m) + \sum_{l=1}^{m} \log ||\mathbf{e}^l||_p^p, \tag{4.1}$$

where f_1 is a function of m. It is a scaling function like $f(m)$.

Now the system differential equations can be calculated as:

$$\begin{aligned} \dot{u}_i &= -\frac{\partial \tilde{E}}{\partial v_i} \\ &= -\sum_{l=1}^{m} \frac{1}{||\mathbf{e}^l||_p^p} \frac{\partial}{\partial v_i} ||\mathbf{e}^l||_p^p \qquad i = 1, 2, \ldots, n. \end{aligned} \tag{4.2}$$

These dynamic equations describe a gradient-like system. Different from the original product-of-norm model (3.1), this system has been changed to a sum-of-norm form. In the system differential equations, the terms corresponding to each pattern are now computed separately and then added together. However, the "energy" function is no longer bounded below. Fortunately, the points at which the "energy" function goes to negative infinity are the pre-specified patterns. In simulations, the system can stop at certain point according to the specified accuracy. Thus in simulation, even though the "energy" function is not bounded below, starting from any initial point, the system will still "converge" to one of its stored patterns.

To avoid the problem that the "energy" function is not bounded from below, we

can modify the "energy" function as follows:

$$E \triangleq \sum_{l=1}^{m} \left[\log(||\mathbf{e}^l||_p^p + c_l)\right], \tag{4.3}$$

where $c_l > 0, \quad l = 1, 2, \ldots, m$, are pre-chosen constants.

The new system differential equations become

$$\dot{u}_i = -\sum_{l=1}^{m} \frac{1}{||\mathbf{e}^l||_p^p + c_l} \frac{\partial}{\partial v_i} ||\mathbf{e}^l||_p^p \qquad i = 1, 2, \ldots, n. \tag{4.4}$$

This system "energy" function now is bounded from below. In addition, it preserves the features of gradient systems. The terms corresponding to each pre-specified pattern in the system differential equations are added together. However, the pre-specified patterns will generally no longer be system equilibria. We will address the relation between the equilibria of the new log-form system and the original product-of-norm system next.

Lemma 4.1 *For a system with "energy" function*

$$E = \prod_{l=1}^{m} [||\mathbf{e}^l||_p^p + c_l], \tag{4.5}$$

where $c_l > 0$ $(l = 1, 2, \ldots, m)$, *and* c_l *are constants. The system differential equations are defined by:*

$$\dot{u}_i = -\sum_{l=1}^{m} \left(\frac{\partial}{\partial v_i} ||\mathbf{e}^l||_p^p \prod_{\substack{k=1 \\ k \neq l}}^{m} \left[||\mathbf{e}^k||_p^p + c_k\right] \right) \qquad i = 1, 2, \ldots, n. \tag{4.6}$$

If $\mathbf{V}^*$ *is a (isolated) local energy minima of the system* (4.5), *then* $\mathbf{V}^*$ *is a (isolated)*

local minima of the "energy" function of (4.3).

Proof:

To distinguish the two "energy" functions, we refer to the "energy" defined by equation (4.3) as $\bar{E}$ in this proof. Since $\mathbf{V}^*$ is a local minima of the "energy" function of (4.5), there exists a small neighborhood Ω of $\mathbf{V}^*$, $\mathbf{V}^* \in \Omega$, such that for any point of $\bar{\mathbf{V}} \neq \mathbf{V}^*$ and $\bar{\mathbf{V}} \in \Omega$, we have

$$E(\bar{\mathbf{V}}) - E(\mathbf{V}^*) \geq 0. \tag{4.7}$$

For the system (4.5), since $E > 0$,

$$\bar{E}(\bar{\mathbf{V}}) = \log E(\bar{\mathbf{V}}) \text{ and } \bar{E}(\mathbf{V}^*) = \log E(\mathbf{V}^*). \tag{4.8}$$

Thus

$$\begin{aligned} \bar{E}(\bar{\mathbf{V}}) - \bar{E}(\mathbf{V}^*) &= \log E(\bar{\mathbf{V}}) - \log E(\mathbf{V}^*) \\ &= \log \frac{E(\bar{\mathbf{V}})}{E(\mathbf{V}^*)}. \end{aligned} \tag{4.9}$$

Since

$$E(\bar{\mathbf{V}}) - E(\mathbf{V}^*) \geq 0, \tag{4.10}$$

$$\Longrightarrow \quad E(\bar{\mathbf{V}}) \geq E(\mathbf{V}^*). \tag{4.11}$$

Thus

$$\frac{E(\bar{\mathbf{V}})}{E(\mathbf{V}^*)} \geq 1 \tag{4.12}$$

and

$$\bar{E}(\bar{\mathbf{V}}) - \bar{E}(\mathbf{V}^*) = \log \frac{E(\bar{\mathbf{V}})}{E(\mathbf{V}^*)} \geq 0. \tag{4.13}$$

Therefore, $\mathbf{V}^*$ is a local minima of the new "energy" function (4.3) too. It can been seen from the proof above that if we change the inequalities to strict form we can prove the case of isolated "energy" minima. We omit the proof here. QED.

Lemma 4.2 *A pattern* $\mathbf{V}^*$ *is an asymptotically stable equilibrium of the system defined by* (4.4)/(4.6) *iff it is an isolated local minima of the "energy" function defined by* (4.3)/(4.5).

Proof:

Firstly, we want to show that if $\mathbf{V}^*$ is an asymptotically stable equilibrium of the system (4.4)/(4.6) it must be a local "energy" minima of (4.3)/(4.5). We prove it by contradiction method.

Case 1. Suppose $\mathbf{V}^*$ is an asymptotically stable equilibrium of (4.4)/(4.6), but it is not a minima of the "energy" function (4.3)/(4.5). Define a small neighborhood D around $\mathbf{V}^*$, such that $\mathbf{V}^*$ is the only system equilibrium point inside D. Consider a new function $\tilde{E} \triangleq E(\mathbf{V}^*) - E(\mathbf{V})$. It is obvious that $\tilde{E}$ is continuously differentiable over D. Since $\mathbf{V}^*$ is not a local minima of the energy function E, there exits some $\mathbf{V}_0$ with arbitrarily small $||\mathbf{V}_0 - \mathbf{V}^*||$ and

$$E(\mathbf{V}_0) < E(\mathbf{V}^*) \Longrightarrow \tilde{E}(\mathbf{V}_0) > \tilde{E}(\mathbf{V}^*) = 0. \tag{4.14}$$

Define a set β as

$$\beta = \{\mathbf{V} \in B_r : \tilde{E}(\mathbf{V}) > 0\} \tag{4.15}$$

where the ball $B_r \triangleq \{\mathbf{V} \in R^n : ||\mathbf{V} - \mathbf{V}^*|| \leq r\}$ and $B_r \subset D$. Along the trajectory of the system

$$\begin{aligned} \dot{\dot{E}} &= -\dot{E} \\ &= -\sum_i \frac{\partial E}{\partial v_i}\frac{\partial v_i}{\partial u_i}\dot{u}_i \\ &= \sum_i \frac{\partial v_i}{\partial u_i}\dot{u}_i^2 \\ &> 0 \quad \text{for} \quad \mathbf{V} \in \beta. \end{aligned} \tag{4.16}$$

According to *Chetavev's* theorem [Kha91], $\mathbf{V}^*$ is unstable. Contradiction! thus, $\mathbf{V}^*$ must be a local energy minima.

Case 2. Suppose $\mathbf{V}^*$ is a local energy minima of the system (4.3)/(4.5) but not an isolated one. Then for any small neighborhood of $\mathbf{V}^*$ defined by $\Omega = \{\mathbf{V} : ||\mathbf{V}-\mathbf{V}^*|| < \epsilon\}$, $\epsilon > 0$, there exits a local energy minima $\mathbf{V}_0 \in \Omega$ such that $\frac{\partial E}{\partial \mathbf{V}} = 0 \longrightarrow \dot{\mathbf{U}}|_{\mathbf{V}_0} = 0$. Thus $\mathbf{V}^*$ is not asymptotically stable. contradiction! Therefore, $\mathbf{V}^*$ has to be an isolated local energy minima.

Now we will show that if $\mathbf{V}^*$ is an isolated "energy" minima, then it is an asymptotically stable equilibrium point.

Since $\mathbf{V}^*$ is an isolated "energy" minima,

$$\frac{\partial E}{\partial \mathbf{V}}|_{\mathbf{V}^*} = 0. \tag{4.17}$$

Thus $\mathbf{V}^*$ is an equilibrium of the system (4.4)/(4.6). Consider an small neighborhood Ω around $\mathbf{V}^*$ such that $\mathbf{V}^*$ is the only system equilibrium point inside Ω. Then consider the continuously differentiable function $E(\mathbf{V})$ over Ω, $E(\mathbf{V}^*) = 0$ and

$E(\mathbf{V}) > 0$ for $\mathbf{V} \neq \mathbf{V}^*$, $\mathbf{V} \in \Omega$. Thus E can serve as a *Lyapunov* function candidate over Ω. Along the trajectory of the system,

$$\begin{aligned} \frac{dE(\mathbf{V})}{dt} &= -\frac{d\tilde{E}(\mathbf{V})}{dt} \\ &< 0 \quad \text{for} \quad \mathbf{V} \neq \mathbf{V}^*, \mathbf{V} \in \Omega \end{aligned} \tag{4.18}$$

Thus $\mathbf{V}^*$ is an asymptotically stable equilibrium of the system (4.4)/(4.6). QED

Lemma 4.3 *Suppose* $\mathbf{V}^{*l}$, $l = 1, 2, \ldots, m$, *are the pre-specified patterns of the system defined by equations* (4.5) *and* (4.6). *Given any small positive constant* r, *there always exist a set of constants* $c_l > 0$, $l = 1, 2, \ldots, m$, *such that, for* $l = 1, 2, \ldots, m$, *the system* (4.5) *has a local "energy" minima* $\bar{\mathbf{V}}^{*l}$ *and* $||\mathbf{V}^{*l} - \bar{\mathbf{V}}^{*l}|| < r$; $\bar{\mathbf{V}}^{*l}$ *is an equilibrium point of system* (4.6).

Proof:

For simplicity, let us start by choosing

$$0 < c_l < 1 \qquad l = 1, 2, \ldots, m.$$

Define a small neighborhood Ω of $\mathbf{V}^{*1}$ as:

$$\Omega = \{(\mathbf{V}^{*1} + \Delta\mathbf{V}) : ||\Delta\mathbf{V}|| \leq r_1,\ r > r_1 > 0\} \tag{4.19}$$

such that

$$\mathbf{V}^{*l} \notin \Omega, \quad l = 2, 3, \ldots, m. \tag{4.20}$$

Define another neighborhood Ω_1 of $\mathbf{V}^{*1}$ as:

$$\Omega_1 = \{(\mathbf{V}^{*1} + \Delta\mathbf{V}) : r_1 \geq ||\Delta\mathbf{V}|| \geq r_2,\ r_1 > r_2 > 0\}. \tag{4.21}$$

The open neighborhood Ω_2 is defined as

$$\Omega_2 = \Omega - \Omega_1. \tag{4.22}$$

Consider the system "energy" function (4.5) at pattern $\mathbf{V}^{*1}$:

$$\begin{aligned} E(\mathbf{V}^{*1}) &= \left[||\mathbf{e}^1(\mathbf{V}^{*1})||_p^p + c_1\right] E_{n-1}(\mathbf{V}^{*1}) \\ &= c_1 E_{n-1}(\mathbf{V}^{*1}), \end{aligned} \tag{4.23}$$

here $||\mathbf{e}^1(\mathbf{V}^{*1})||_p = 0$ and we define

$$E_{n-1} = \prod_{l=2}^{m} \left[||\mathbf{e}^l||_p^p + c_l\right]. \tag{4.24}$$

At a point $\mathbf{V} = \mathbf{V}^{*1} + \Delta\mathbf{V},\ \mathbf{V} \in \Omega_1$,

$$E(\mathbf{V}) = [\Delta e + c_1] E_{n-1}(\mathbf{V}), \tag{4.25}$$

where $\Delta e = ||\mathbf{e}^1(\mathbf{V})||_p^p$. Therefore,

$$\begin{aligned} E(\mathbf{V}) - E(\mathbf{V}^{*1}) &= [\Delta e + c_1] E_{n-1}(\mathbf{V}^{*1} + \Delta\mathbf{V}) - c_1 E_{n-1}(\mathbf{V}^{*1}) \\ &= \Delta e E_{n-1}(\mathbf{V}^{*1} + \Delta\mathbf{V}) + c_1 E_{n-1}(\mathbf{V}^{*1} + \Delta\mathbf{V}) - \end{aligned}$$

$$
\begin{aligned}
& c_1 E_{n-1}(\mathbf{V}^{*1}) \\
> \; & \Delta e \tilde{E}_{n-1}(\mathbf{V}^{*1}+\Delta\mathbf{V}) + c_1 \tilde{E}_{n-1}(\mathbf{V}^{*1}+\Delta\mathbf{V}) - \\
& c_1 \bar{E}_{n-1}(\mathbf{V}^{*1}). \qquad (4.26)
\end{aligned}
$$

where $\bar{E}_{n-1}(\mathbf{V}) = \prod_{l=2}^{m} \left[||\mathbf{e}^l||_p^p + 1 \right]$ and $\tilde{E}_{n-1}(\mathbf{V}) = \prod_{l=2}^{m} ||\mathbf{e}^l||_p^p$. For $\mathbf{V}^{*1}+\Delta\mathbf{V} \in \Omega_1$,

$$
\Delta e > 0, \quad \tilde{E}_{n-1}(\mathbf{V}^{*1}+\Delta\mathbf{V}) > 0 \text{ and } \quad c_1 > 0,
$$

Since Δe and $\tilde{E}_{n-1}(\mathbf{V}^{*1}+\Delta\mathbf{V})$ are continuous with respect to $\Delta\mathbf{V}$ over the compact set Ω_1, there exists $a > 0$, such that $\Delta e \tilde{E}_{n-1}(\mathbf{V}^{*1}+\Delta\mathbf{V}) > a$ for every point $(\mathbf{V}^{*1}+\Delta\mathbf{V}) \in \Omega_1$. Similarly, there exists $b > 0$ such that $\bar{E}_{n-1}(\mathbf{V}^{*1}) < b$ for $\mathbf{V}^{*1}+\Delta\mathbf{V} \in \Omega_1$. Therefore, for $\mathbf{V}^{*1}+\Delta\mathbf{V} \in \Omega_1$, we can always find a constant $c_1 > 0$ such that

$$
c_1 < \frac{a}{b} \quad \text{and} \quad c_1 < 1. \qquad (4.27)
$$

Thus, for any $(\mathbf{V}^{*1}+\Delta\mathbf{V}) \in \Omega_1$

$$
\begin{aligned}
\Delta e E_{n-1}(\mathbf{V}^{*1}+\Delta\mathbf{V}) + c_1[E_{n-1}(\mathbf{V}^{*1}+\Delta\mathbf{V}) - E_{n-1}(\mathbf{V}^{*1})] \; & > \; a - c_1 b \\
& > \; 0. \qquad (4.28)
\end{aligned}
$$

Since $E(\mathbf{V})$ is continuous over the compact set Ω, $E(\mathbf{V})$ must have a minima $\bar{\mathbf{V}}^{*1} \in \Omega$ such that $E(\bar{\mathbf{V}}^{*1}) \leq E(\mathbf{V})$ for every $\mathbf{V} \in \Omega$. From inequality (4.28), we know

$$
||\bar{\mathbf{V}}^{*1} - \mathbf{V}^{*1}|| < r_2 < r, \qquad \text{i.e. } \bar{\mathbf{V}}^{*1} \in \Omega_2. \qquad (4.29)
$$

That is $\bar{\mathbf{V}}^{*1}$ is a local "energy" minima. Since this is a gradient like system, $\bar{\mathbf{V}}^{*1}$ is a equilibrium point of the system.

Similarly we can prove the existence of c_l, $l = 2,3,\ldots,m$. we omit the detail here. QED

From lemma 4.1 and lemma 4.3 we have the following theorem:

Theorem 4.1 *Suppose* $\mathbf{V}^{*l}$, $l = 1,2,\ldots,m$, *are the pre-specified patterns of the system defined by equations* (4.3). *Given any small positive constant* r, *there always exist a set of constants* $c_l > 0$, $l = 1,2,\ldots,m$, *such that, for* $l = 1,2,\ldots,m$, *the system* (4.3) *has a equilibrium point* $\bar{\mathbf{V}}^{*l}$ *and* $||\mathbf{V}^{*l} - \bar{\mathbf{V}}^{*l}|| < r$.

For the case of L_2 norm, we can further study the stability of the equilibria of the new system.

Lemma 4.4 *For case of* L_2 *norm, suppose* $\mathbf{V}^{*l}$, $l = 1,2,\ldots,m$, *are the pre-specified distinct patterns of the system defined by equations* (4.5) *and* (4.6). *Given any small positive constant* ϵ, *there always exist a set of constants* $c_l > 0$, $l = 1,2,\ldots,m$, *such that, for* $l = 1,2,\ldots,m$, *the system* (4.6) *has a corresponding asymptotically stable equilibrium point* $\bar{\mathbf{V}}^{*l}$ *and* $||\mathbf{V}^{*l} - \bar{\mathbf{V}}^{*l}|| < \epsilon$.

Proof:

For L_2 norm, equation (4.6) becomes

$$\dot{u}_i = -2\sum_{l=1}^{m}\left((v_i^{*l} - v_i)\prod_{\substack{k=1\\k\neq l}}^{m}\left[||\mathbf{e}^k||_2^2 + c_k\right]\right) \qquad i = 1,2,\ldots,n. \tag{4.30}$$

It can be written in general vector form as

$$\dot{\mathbf{U}} = f(\mathbf{U}), \tag{4.31}$$

where f is a C^1 vector field. Similarly, by taking $f(m) = 2$, we can write equation (3.7) as

$$\dot{\mathbf{U}} = f^*(\mathbf{U}). \tag{4.32}$$

Note that when $c_l = 0$, $l = 1, 2, \ldots, m$, we have $f = f^*$.

We have already showed $\mathbf{V}^{*l}$, $l = 1, 2, \ldots, m$, are asymptotically stable equilibria of the system (3.7). The *Jacobian* matrix of system (3.7) evaluated at a specified pattern $\mathbf{V}^{*l}$ is

$$Df(\mathbf{U})|_{\mathbf{V}^{*l}} = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix} \tag{4.33}$$

where

$$\lambda_i = -\prod_{\substack{k=1 \\ k \neq l}}^{m} \left[\sum_{j=1}^{n} (v_j^{*k} - v_j^{*l})^2\right] \left[\left.\frac{\partial v_i}{\partial u_i}\right|_{v_i^{*l}}\right] < 0. \tag{4.34}$$

Therefore $\mathbf{V}^{*l}$, $l = 1, 2, \ldots, m$, are hyperbolic equilibria. It is obvious that $Df(\mathbf{U})|_{\mathbf{V}^{*l}}$ is invertible.

We define the C^1-*norm* $||h||_{C^1}$ of a vector field $h \in v(R^n)$ ($v(R^n)$ means all the vector field over R^n) to be the least upper bound of all the numbers of $||h(\mathbf{U})||$ and $||Dh(\mathbf{U})||$, for $\mathbf{U} \in R^n$. According to the perturbation theory [HS74], for any $\epsilon > 0$ there always exists a neighborhood $\sigma = \{g : ||g - f^*||_{C^1} < r, \quad r > 0\} \subset v(R^n)$

of f^* such that for any $g \in \sigma$ there is a unique equilibrium $\bar{\mathbf{V}}^{*1}$ of $\dot{\mathbf{U}} = g(\mathbf{U})$ and $||\bar{\mathbf{V}}^{*1} - \mathbf{V}^{*1}|| < \epsilon$.

Since $||f - f^*||_{C^1}$ is a continuous function of c_l, $l = 1, 2, \ldots, m$, and $||f - f^*||_{C^1} = 0$ when $c_l = 0$, $l = 1, 2, \ldots, m$, we can always find a $c_1 = c_1^* > 0$ such that the vector field

$$f^{(1)} \in \sigma, \tag{4.35}$$

where

$$f^{(1)}() = f()|_{c_2 = c_3 = \cdots = c_m = 0}. \tag{4.36}$$

Therefore, there is a unique equilibrium point $\bar{\mathbf{V}}^{*1}$ of $\dot{\mathbf{U}} = f^{(1)}(\mathbf{U})$ and $||\bar{\mathbf{V}}^{*1} - \mathbf{V}^{*1}|| < \epsilon$. Further more $\bar{\mathbf{V}}^{*1}$ is asymptotically stable and hyperbolic.

Now consider a vector field $f^{(2)}$ defined by

$$f^{(2)}() = f()|_{c_1 = c_1^*, c_3 = c_4 = \cdots = c_m = 0} \tag{4.37}$$

Since $\bar{\mathbf{V}}^{*1}$ is a hyperbolic equilibrium of $\dot{\mathbf{U}} = f^{(1)}(\mathbf{U})$, the eigenvalues of the *Jacobian* matrix $Df^{(1)}|_{\bar{\mathbf{V}}^{*1}}$ will have nonzero real parts. Thus, $Df^{(1)}|_{\bar{\mathbf{V}}^{*1}}$ is invertible. According to the perturbation theorems, there exists a neighborhood σ_1 of $f^{(1)}$ such that for any $g_1 \in \sigma_1$ there is a unique equilibrium point $\overline{\bar{\mathbf{V}}}^{*1}$ of $\dot{\mathbf{U}} = g_1(\mathbf{U})$ and $||\overline{\bar{\mathbf{V}}}^{*1} - \bar{\mathbf{V}}^{*1}|| < \epsilon_1 < \epsilon$. Since $||\bar{\mathbf{V}}^{*1} - \mathbf{V}^{*1}|| < \epsilon$, we can always make ϵ_1 small enough so that $||\overline{\bar{\mathbf{V}}}^{*1} - \mathbf{V}^{*1}|| < \epsilon$. Further more, $\overline{\bar{\mathbf{V}}}^{*1}$ is hyperbolic and asymptotically stable. Since $||f^{(2)}||_{C^1}$ is a continuous function of c_2, and $f^{(2)} = f^{(1)}$ when $c_2 = 0$, there exists a open set $(0, b_1)$ with $b_1 > 0$ such that for very $c_2 \in (0, b_1)$, $f^{(2)} \in \sigma_1$. Thus, there exits

a unique asymptotically stable equilibrium point $\bar{\bar{\mathbf{V}}}^{*1}$ of $\dot{\mathbf{U}} = f^{(2)}(\mathbf{U})$ and

$$||\bar{\bar{\mathbf{V}}}^{*1} - \mathbf{V}^{*1}|| < \epsilon. \tag{4.38}$$

Since $\mathbf{V}^{*2}$ is a hyperbolic asymptotically stable equilibrium point of $f^{(1)}$ too (this can be verified like those of f^*, we omit the detail here), we can similarly show that there exists a set $(0, b_2)$ with $b_2 > 0$ such that for every $c_2 \in (0, b_2)$, there exits a unique asymptotically stable equilibrium point $\bar{\mathbf{V}}^{*2}$ of $\dot{\mathbf{U}} = f^{(2)}(\mathbf{U})$ and

$$||\bar{\mathbf{V}}^{*2} - \mathbf{V}^{*2}|| < \epsilon. \tag{4.39}$$

We can choose $c_2 \in (0, \min(b_1, b_2))$ to have both (4.38) and (4.39) hold.

Similarly, we can show that there exist $c_3 > 0$, $c_4 > 0$, ..., $c_m > 0$ such that there exits a unique asymptotically stable equilibrium $\bar{\mathbf{V}}^{*l}$, $l = 3, 4, \ldots, m$ and $||\mathbf{V}^{*l} - \bar{\mathbf{V}}^{*l}|| < \epsilon$. QED

From lemma (4.1), (4.2) and (4.4) we have the following theorem:

Theorem 4.2 *For case of L_2 norm, suppose $\mathbf{V}^{*l}$, $l = 1, 2, \ldots, m$, are the pre-specified distinct patterns of the system defined by equations* (4.3) *and* (4.4). *Given any small positive constant ϵ, there always exist a set of constants $c_l > 0$, $l = 1, 2, \ldots, m$, such that, for $l = 1, 2, \ldots, m$, the system* (4.4) *has a corresponding asymptotically stable equilibrium point $\bar{\mathbf{V}}^{*l}$ and $||\mathbf{V}^{*l} - \bar{\mathbf{V}}^{*l}|| < \epsilon$.*

The proof is straight forward. We omit it here.

From theorem 4.2, we know that we can use the new log form system to store patterns as close to (not equal to) the desired ones as we want.

For the new system, the right hand side of the equations (4.4) is in the form of summation, and the effects of the desired patterns are additive. This would make it easier to implement the model in standard software/hardware modules which can be easily parallelized. This parallelized architecture of modules is depicted in Figure 4.1. As can be seen from the picture, each processing module works on one pattern alone. Therefore, the process can be parallelized. Some simulations have shown that, compare to those networks which use the original "energy" function, networks using this log-form "energy" function converge (retrieve) faster. It can also be observed that it is easier for this system to "learn" new patterns. To "learn" (store) a new pattern, the original structure of the system does not have to be changed, only a new module is added to the original network. The stored patterns of the old system are still stored (to certain degree) in the new system.

4.2 Localization of the Product-of-norm Model

4.2.1 Motivation

The original product-of-norm system model described in equation (3.7) is fairly complex: the feedbacks are nonlinear; the structure is fully connected. As the number of units (neurons) increases, the number of connections will increase in the order of n^2. This will limit the size of the network in implementations. In software, the limitation will appear in the time to simulate a reasonable-sized network. In hardware, the limitation will manifest itself in the full connectivity. Thus it appears to be difficult to directly implement this model in hardware using present technology. Therefore it is necessary to simplify the original networks.

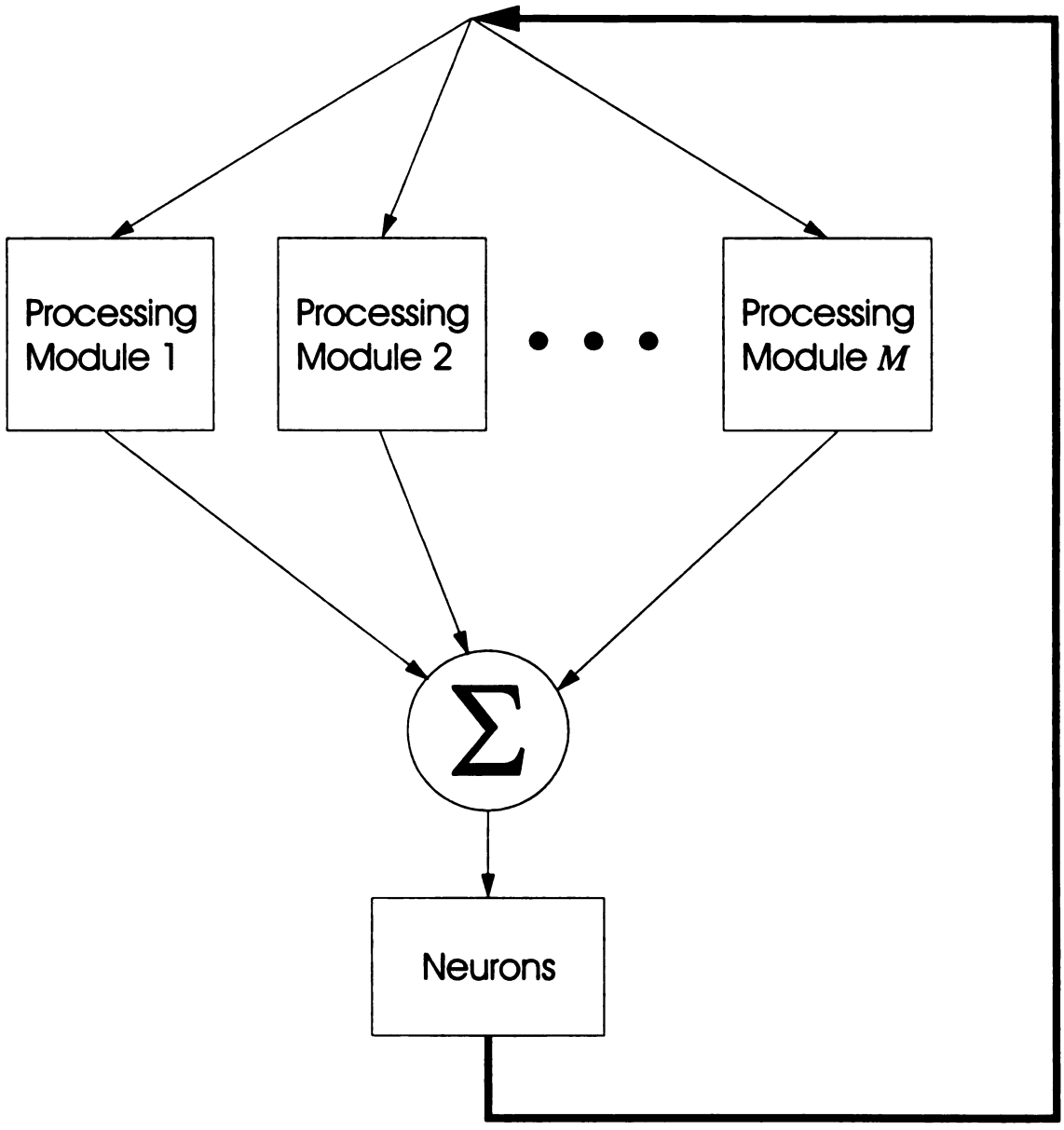


Figure 4.1. A modularized structure of Product-of-norm model

In last section we investigated the way to use log-formed "energy" function to modularize the original system. In this section we will explore other approaches to simplify the network. Our first goal here is to localize the connectivity. As oppose to haphazard structure, we seek to select a local structure that in addition to preserving some of the delineated features of the original product-of-norm model, would only add a limited number of spurious equilibria.

The specific approach rests on retaining the gradient systems property while rendering the pre-specified patterns as isolated minima of the "energy" function. Hence it can be guaranteed that the pre-specified patterns will always be asymptotic stable equilibria of the system.

4.2.2 Subgrouping Method

A naive way to simplify the network is to divide the whole network into smaller subnetworks which are isolated from one another. One builds each subnetwork separately and assembles the small networks together to form the large network. Figure 4.2 depicts an example of a network grouped this way. The "energy" function which describes this model is:

$$E = \frac{1}{2}\left[\prod_{l=1}^{m}\sum_{j\in N_1}(v_j^{*l}-v_j)^2+\prod_{l=1}^{m}\sum_{j\in N_2}(v_j^{*l}-v_j)^2 + \cdots+\prod_{l=1}^{m}\sum_{j\in N_p}(v_j^{*l}-v_j)^2\right]. \tag{4.40}$$

Where N_k, for $k = 1, 2, \ldots, p$, are the index of neuron in subnet k (p is the total number of subnetworks). $N_i \cap N_j = \phi$, for $i \neq j$ and $i, j \in \{1, 2, \ldots, p\}$. The system

dynamic equations are:

$$\dot{u}_i = -\frac{\partial E}{\partial v_i} \qquad i = 1, 2, \ldots, n \tag{4.41}$$

i.e.

$$\dot{u}_i = f(m)\left[\sum_{l=1}^{m}(v_i^{*l} - v_i)\prod_{\substack{k=1\\k\neq l}}^{m}\sum_{\substack{j\in N_h\\ i\in N_h}}(v_j^{*k} - v_j)^2\right]. \tag{4.42}$$

Since at the pre-specified pattern $\mathbf{V}^{*l}$, the "energy" $E(\mathbf{V}^{*l}) = 0$ and $E(\mathbf{V}) > 0$ for $\mathbf{V} \neq \mathbf{V}^{*l}$, $l = 1, 2, \ldots, m$, $\mathbf{V}^{*l}$ are the isolated "energy" minima of the system. Therefore they are asymptotic stable equilibria.

However, all the combinations of the minima of the separated p subnets are stable equilibria of the system also. This leads to the generation of more so-called spurious equilibria. Thus the region of attractions for each equilibrium point will become smaller than that in the original product-of-norm model.

In this model, each neuron is only directly connected to neurons in its own subset. Compared to the original model, the connectivity has dropped dramatically. A similar approach is to divide the original network into smaller ones too. However, this time each small subnetworks is coupled with one or more other subnetworks. That is, the subnets overlap with one another. Figure 4.3 shows one example structure of such network whose original network is divided into four subnetworks. The neurons which are marked in dark color belong to two subnets, while the rest belong to one subnet only.

The motivation of building subnetworks with overlap is to overcome the generation of spurious equilibria of the previous model. In the previous model, each subnetwork works almost independently. Thus any combination of the "energy" minima of each subnet composes one system stable equilibrium point. While in this new model,

the "energy" minima of one subnet will affect the minima of other subnets. Thus intuitively, it should have fewer spurious equilibria than the previous model does.

Compared to the previous model, this model will have more connections. The network will be a little more complex. However, compared to the original product-of-norm model, the connectivity is much less and the structure is much simpler. The system is still a gradient-like system and retains the features of a gradient system. The choice of the size of each fully connected subnetworks and the degree of overlapping with their "neighbor" subnetworks should be chosen judiciously for each application, i.e. they should be application dependent. We observe that there will always be a trade-off between the connectivity of these subnetworks and the application-related performance.

The "energy" function for this new approach can be formalized as:

$$E = \frac{1}{2}\left[\prod_{l=1}^{m}\sum_{j\in N_1'}(v_j^{*l}-v_j)^2+\prod_{l=1}^{m}\sum_{j\in N_2'}(v_j^{*l}-v_j)^2 + \cdots + \prod_{l=1}^{m}\sum_{j\in N_p'}(v_j^{*l}-v_j)^2\right], \tag{4.43}$$

where p is the total number of subnetworks. $N = \{1,2,\ldots,n\}$, $N_i' \subset N$ for every $i \in \{1,2,\ldots,p\}$. $N_i' \cap \{N_1', N_2', \ldots, N_{i-1}', N_{i+1}', \ldots, N_p'\} \neq \phi$ for $i = 1,2,\ldots,p$, and $\cup_{i=1}^{p} N_i' = N$.

If we define $\Gamma(i)$ to be the set of subnets which contains neuron i, that is

$$\Gamma(i) \subset \{N_1', N_2', \ldots, N_p'\} \tag{4.44}$$

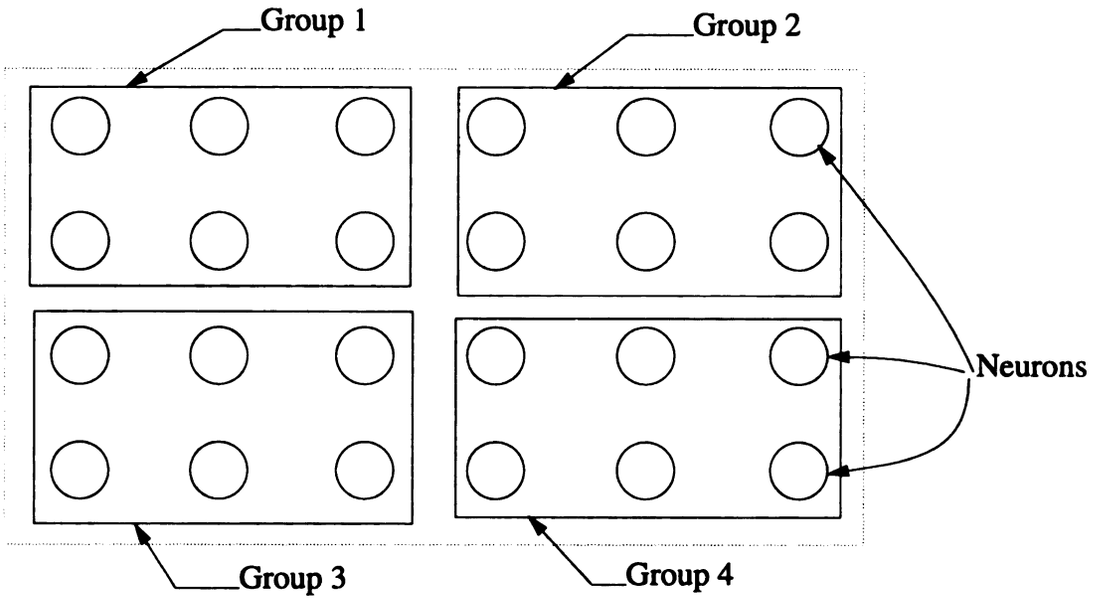


Figure 4.2. An example structure of dividing a big network into isolated smaller subnetworks

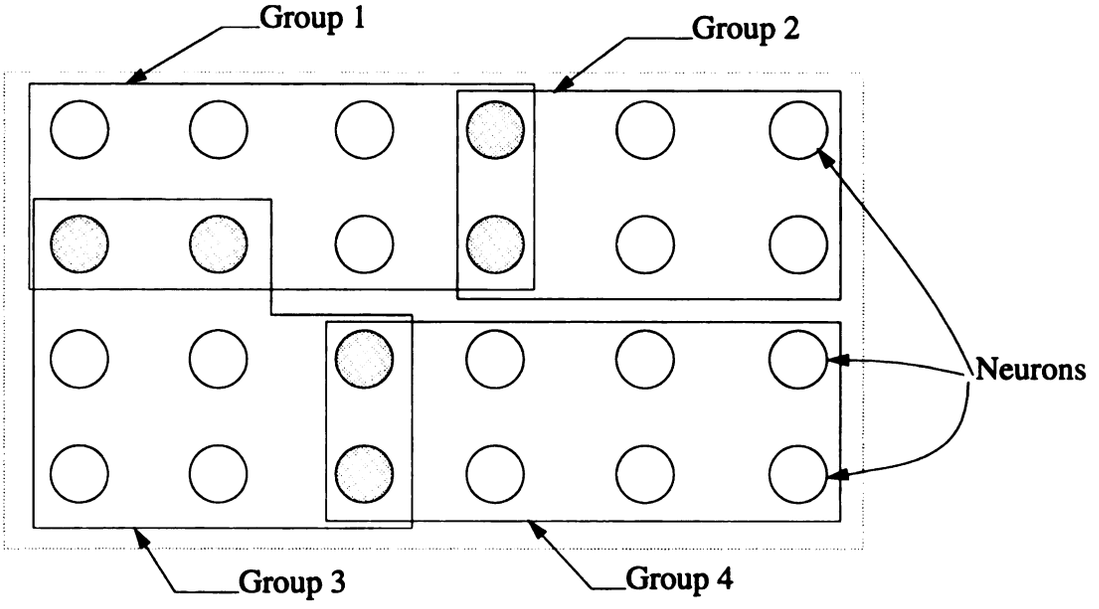


Figure 4.3. An example structure of dividing a big network into overlapped smaller subnetworks

and

$$i \in N'_k \Longleftrightarrow N'_k \subset \Gamma(i), \tag{4.45}$$

then, the system dynamic equation for the model can be expressed as

$$\dot{u}_i = f(m) \sum_{N'_q \in \Gamma(i)} \sum_{l=1}^{m} \left\{ (v_i^{*l} - v_i) \prod_{\substack{k=1 \\ k \neq l}}^{m} \sum_{j \in N'_q} (v_j^{*k} - v_j)^2 \right\}, \qquad i = 1, 2, \ldots, n. \tag{4.46}$$

As we will show in the simulation examples later, if the portion of neurons corresponding to those coupling neurons (neurons belonging to more than one group, shown in dark color in Figure 4.3) are the same for all the pre-specified patterns, the coupling neurons may lose their coupling effects, i.e. they may fail to pass information between the two overlapping subnetworks, therefore, they cannot prevent the generation of spurious equilibria.

4.2.3 Simulation Results

To investigate the performance of the two new models, we built a network with 288 neurons using different models presented in previous section to store three patterns. Figure 4.4 shows the patterns/images to be stored by the network. Figure 4.5 shows the two test patterns which we used as input to the network.

Network one is built using the "energy" function described by equation (4.40). Figure 4.6 shows the grouping used in this network, which does not use overlapping method. Network two is built using the "energy" function (4.43). Figure 4.7 shows the grouping used in this network. In the simulation, both networks successfully retrieved the correct patterns when applied with the test patterns shown in figure 4.5.

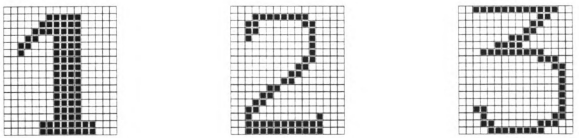


Figure 4.4. Three stored patterns

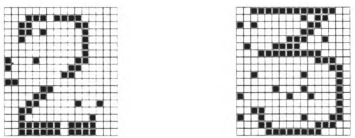


Figure 4.5. Test patterns

Figure 4.8 and figure 4.9 show the retrieve process for these two network when a given input is composed of half of the image "1" and half of the image of"3". The network which was built using overlap method successfully retrieved a stored image "1", while the one with no overlap failed to retrieve a stored image.

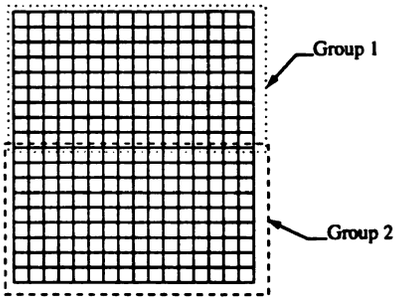


Figure 4.6. No over-lap grouping

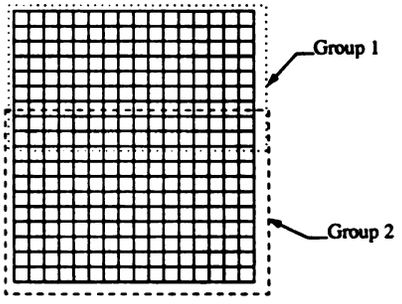


Figure 4.7. Over-lap grouping

From the simulation results presented above, we can make the following remarks:

- The grouping of the neurons affects the performance of the network. The

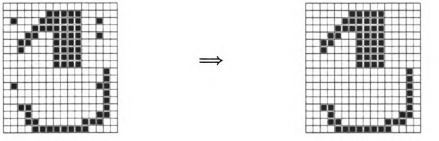


Figure 4.8. The retrieving process of the network using non-overlap method

Figure 4.9. The retrieving process of the network using overlap method

grouping is dependent on the patterns to be stored.

- The model of overlapping groups has better overall performance for correct retrieval of the stored patterns. This demonstrated that the overlapping among subnetworks is essential for prohibition of the generation of spurious equilibria and the enlargement of the basin of attraction for the desired patterns.

4.3 Modified Product-of-norm Models for Recognizing Shifted/Translated Input Patterns

4.3.1 Motivation

The original product-of-norm model described in equations (3.7) cannot retrieve the correct patterns effectively when the input pattern is a shifted/translated stored pattern. An example network which stored patterns shown in figure 4.4 failed to retrieve the correct pattern when given a shifted version of the stored pattern as shown in figure 4.10. In this section we will explore various approaches to modify the original model, so that it can tolerate (to certain degree) shifted/translated images while retaining the features of being a gradient-like system. This section also serves as an example of how the "energy" function can be tailored to suit specific application needs.

4.3.2 Modified "Energy" Functions

We seek to modify the "energy" function to take the shift/translation effects into account. Let us assume the images to be stored are in rectangular form and the

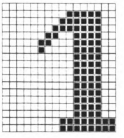


Figure 4.10. Shifted input pattern

neurons are indexed in such a way that the images are represented row by row, that is, the first row of the image corresponds to neuron 1 through c (c is the number of columns of the image), and the second row corresponds to neuron $c+1$ through $2c$, $\cdots$. We propose a modified "energy" function as follows:

$$E = \frac{1}{2} f(m) \prod_{l=1}^{m} \sum_{j=1}^{n} (v_j^{*l} - v_j)^2 + \frac{1}{2} K \prod_{l=1}^{m} \sum_{i=1}^{r} \left[\sum_{j=(i-1)c+1}^{ic} (v_j^{*l} - v_j) \right]^2 \tag{4.47}$$

where K : is a constant and $K > 0$. Used here as a scaling factor;

c : the number of columns of the digital image/pattern;

n : total number of neurons of the network, which equals to the total number of pixels of the stored image;

r : the number of rows of the digital image/pattern;

m : number of images/patterns to be stored.

In comparison with the original "energy" function (3.7), this modified "energy" function has an additional term, which actually measures the error (i.e. the difference between the current output of the neurons and the stored images) of each row as a whole. This term serves as a penalty function. It is zero when the input pattern is a horizontally shifted version of one of the stored patterns, while is non-negative otherwise. It is obvious that at a stored pattern $\mathbf{V}^{*l}$, $l = 1, 2, \ldots, m$, $E = 0$. Thus $\mathbf{V}^{*l}$ is an isolated minima of the "energy" function. Therefore, $\mathbf{V}^{*l}$, $l = 1, 2, \ldots, m$, are stable equilibria of the system. When the current state of the system is close to a shifted version of a stored pattern, the added term will be relatively small and make E small also compared to the case of a random image as input. This, in a sense, builds a "energy" valley to attract t hose shifted images to the correct patterns stored. This can also be viewed as a simple feature mapping, the goal is to choose such maps that will separate input patterns in feature space while being shift-invariant.

The "energy" function introduced this way may introduce more local "energy" minima (spurious equilibria) and in general will not be able to recognize all the shifted versions of the pre-specified patterns. The value of K becomes crucial when it is used for different applications. The new dynamic equations now become:

$$\begin{aligned} \dot{u}_i &= f(m) \sum_{l=1}^{m} (v_i^{*l} - v_i) \prod_{\substack{k=1 \\ k \neq l}}^{m} \sum_{j=1}^{n} (v_j^{*k} - v_j)^2 + \\ & K \sum_{l=1}^{m} \left[\sum_{i=J(i)}^{J(i)+c} (v_j^{*l} - v_j) \right] \left\{ \prod_{\substack{k=1 \\ k \neq l}}^{m} \sum_{p=1}^{r} \left[\sum_{j=(i-1)c+1}^{pc} (v_j^{*k} - v_j) \right]^2 \right\} \\ & \text{for} \qquad i = 1, 2, \ldots, n, \end{aligned} \tag{4.48}$$

where $R(i)$ is the index number of the row which contains neuron i, and $J(i)$ is the

index of the first neuron of $R(i)$.

Another way of modifying the "energy" function to achieve the same goal is to add the penalty function as follows:

$$E = \frac{1}{2} f(m) \prod_{l=1}^{m} \left\{ \sum_{j=1}^{n} (v_j^{*l} - v_j)^2 + K \sum_{i=1}^{r} \left[\sum_{j=(i-1)c+1}^{ic} (v_j^{*l} - v_j) \right]^2 \right\} \tag{4.49}$$

The "energy" function of this form can be used in conjunction with the log modification method described earlier to achieve faster speed. It will also be easier to be parallelized or be modularize in software simulation or hardware implementation.

The effects of this term can be controlled via the coefficient K. Later simulations showed that varying this constant is a very effective way of changing the regions of attractions for each stable equilibrium point. While, technically speaking, the values of K should be computed from a nonlinear version of the *Lagrange* multiplier approach, we opted for simplicity and practical implementation in software in our treatment.

It should be clear from the discussion above that we can similarly construct an "energy" function such that the resulting neural network would be able to recognize vertically shifted patterns as well. Here we omit the detailed derivation.

To modify the network so that it is shift-invariant, i.e. it is able to recognize both horizontally and vertically shifted images, we can obviously combine the two penalty functions used for horizontal shift and vertical shift together. Here we want to present a different approach. We propose to divide the network, more precisely

the neurons whose outputs represent an image, into small regions as we did in the subgroup method (See Figure 4.11 for example). Each region consists of a number of neurons which form a rectangular area of the image. For a region, I, obtained this way, we define the error for region I corresponding to the stored pattern $\mathbf{V}^{*l}$ as:

$$e_I^l = \left[\sum_{i\in I}(v_i^{*l} - v_i)\right]^2, \tag{4.50}$$

where $\mathbf{V}$ is the current output vector of the neuron and v_i is its ith component.

We use the summation of the errors of all the regions as the penalty term and add it to the original "energy" function. The new "energy" function can be expressed as:

$$\begin{aligned} E &= \frac{1}{2}f(m)\prod_{l=1}^{m}\sum_{j=1}^{n}(v_j^{*l} - v_j)^2 + \\ &\quad \frac{1}{2}K\prod_{l=1}^{m}\sum_{I\in G}\left[\sum_{i\in I}(v_i^{*l} - v_i)\right]^2, \end{aligned} \tag{4.51}$$

where $K > 0$ and G is the set of all the regions. The penalty term can also be added as in equation (4.49) to take the advantage of the log forms later. The new system differential equations are (assuming regions not overlapping with one another):

$$\begin{aligned} \dot{u}_i &= f(m)\sum_{l=1}^{m}(v_i^{*l} - v_i)\prod_{\substack{k=1\\k\neq l}}^{m}\sum_{j=1}^{n}(v_j^{*k} - v_j)^2 \\ &\quad +K\sum_{l=1}^{m}\left\{\sum_{j\in I_i}(v_j^{*l} - v_j)\prod_{\substack{k=1\\k\neq l}}^{m}\sum_{I\in G}\left[\sum_{j\in I}(v_j^{*k} - v_j)\right]^2\right\}, \end{aligned} \tag{4.52}$$

where I_i is the region which contains neuron i.

The motivation for choosing such an error function can be explained as follows:

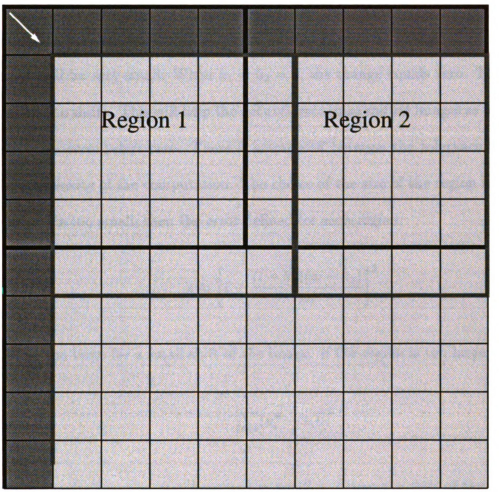


Figure 4.11. An example way of dividing the network into small regions

Referring to the grouping of neurons shown in Figure 4.11, if the input image is shifted one box (pixel) in the direction shown by the arrow, the changes in the error for each region will be in the order of $(\frac{9}{25})^2$. In general, if the region is w pixel wide and l pixel high (long) and the shift is h_1 pixels horizontally and h_2 pixels vertically, the change of errors for each region will be in the order of $\left[1 - \frac{(l-h_1)(w-h_2)}{wl}\right]^2$. When h_1 and h_2 is much smaller than w and l respectively, the change of errors for each region will be very small. When $h_1 = h_2 = 0$, the change equals zero. Thus it is not sensitive to shifts. This will help the network recognize shifted images as we explained in the horizontal-shift case. There is a trade-off between the tolerance to shift and the complexity of the computation. The choice of the size of the region is important. If w or l is too small, then the error defined for each region:

$$e \approx \left[1 - \frac{(l-h_1)(w-h_2)}{wl}\right]^2 \tag{4.53}$$

will be too large for a small shift of the image. If the region is too large, the term

$$\left[\sum (v_i^{*l} - v_i)\right]^2 \tag{4.54}$$

in the penalty function may become too small to detect the shift of the image.

4.3.3 Simulation Results

To test the performance of the proposed modifications, we tried to build three 288-neuron networks using different "energy" functions. We used the images shown in Figure 4.12 as images to be stored and used images shown in Figure 4.13 as test

input patterns. Using these patterns, we tested three networks built using different models. Network one was built using the "energy" function (3.6). Network two was built using the "energy" function (4.47). Network three was built using the "energy" function described in (4.51).

Table 4.1 compares the retrieve results for each case. As can be seen from the table, the proposed new network was able to recognize the shifted images in addition to those input images which are blurred by noises. However, we noticed that the modified network converges slower than the original model does in simulation. This is due to the complexity of the added connections.

Table 4.1. Simulation results

	Test Patterns					
	(a)	(b)	(c)	(d)	(e)	(f)
model 1	×	×	√	√	√	√
model 2	√	√	×	√	√	√
model 3	√	√	√	√	√	√

Note: √ means successful and × means failed.

4.3.4 Discussion and Remarks

So far we have surveyed current models and algorithms for neural networks used for information association, such as associative memory, especially in the application of character recognition and image retrieval. Our focus was on recurrent networks. We

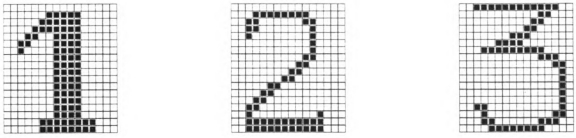


Figure 4.12. Stored patterns

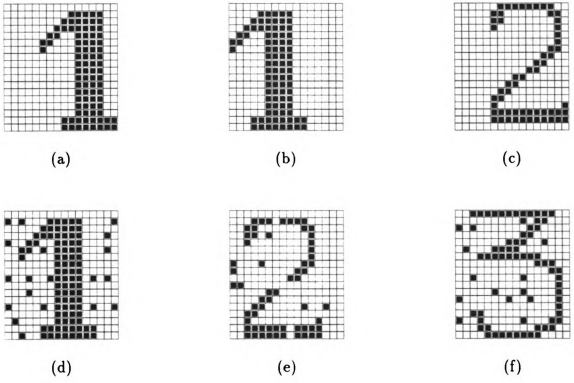


Figure 4.13. Test patterns

proposed a product-of-norm model which has the following features:

- The system eliminates the dynamic "learning" process. It can also be viewed as that the learning process becomes "static".
- The stored patterns are built into the network by design. In theory, such network can store any pattern as the system equilibria. There is no limit as to how many patterns can be stored.
- The system is a gradient system which has no oscillation. The system is globally convergent.

Some of the limitations of this network are: the original network is fully connected and is very complex, making it difficult to be implemented in hardware.

We also investigated the product-of-norm model using different norms. Based on the original production-of-norm model, we proposed several modified network models. All of them retain the property of being a gradient-like system while still being able to store the desired patterns as stable equilibria. Especially, we investigated ways of tailoring the "energy" function for the application of image retrieval. We proposed ways of modifying the "energy" function of the original network to control the regions of attractions for each pattern, so that the network can recognize slightly shifted images.

We proposed ways to simplify the network, specifically, we proposed to use "log" form of the "energy" function. Such networks can be easily parallelized in software simulation and modularized in hardware implementation. To store additional patterns, the network does not have to be built from scratch, it only needs to add some new modules to the original one.

We analyzed various ways to divide the original network into subnetworks so that the connectivity of the network can be reduced while we can still store the desired patterns. We demonstrated that the grouping with overlapping has better performance.

We want to remark here: the simulation/application examples shown here are for demonstration purpose only. They illustrated how this model may be used in real life applications. The examples here are not intended for real image associations. Some of the methods are heuristic.

As demonstrated by the examples, the new proposed models have great potential of being used in a wide range of real life applications: such as associative memories, categorization, character recognition and image association, etc..

CHAPTER 5

THE BUILDING BLOCKS FOR MICRO-ELECTRONIC CIRCUIT REALIZATION OF THE PRODUCT-OF-NORM MODEL

5.1 Motivations

One of the goals of the research on artificial neural networks is to understand how human brain works so that we can use hardware (circuits) to mimic the human brain to solve problems in real life applications. With today's technology of very large scale integrated (VLSI) circuits, it is possible to fabricate tens of millions of devices (transistors) interconnected on a single wafer. This makes it possible to build complex ANN circuits for real world applications where digital computers have limited use.

In this chapter, we will investigate ways of implementation of the product-of-norm

model using analog transistors biased in the subthreshold region. There are several advantages with circuits operating in the subthreshold region [Mea89]:

- Power dissipation is extremely low – from 10^{-12} to 10^{-6} watt for a typical circuit;
- In the subthreshold region, a MOS transistor's drain current saturates in a few KT/q (thermal voltage), allowing the transistor to operate as a current source over most of the voltage ranging from near ground to V_{dd};
- The exponential nonlinearity is an ideal computation primitive for many applications.

First, we will propose some basic circuits which can be used as building blocks for the implementation of product-of-norm model of ANNs. We will then present some examples of ways of implementation of small product-of-norm model. Please note that the example circuits presented here are for demonstration purpose only. They are not targeted to specific real world applications.

5.2 Background and Basic Circuits

5.2.1 MOS Transistors in Subthreshold Region

The drain current of an N-channel metal-oxide-semiconductor (MOS) transistor biased in the subthreshold region can be expressed as [Mea89]:

$$I = I_0 e^{\frac{-q\kappa V_g}{kT}} (e^{\frac{qV_s}{kT}} - e^{\frac{qV_d}{kT}}), \tag{5.1}$$

where V_g : gate voltage.

V_s : souce voltage.

V_d : drain voltage.

I_0 : a constant for a given MOS transistor.

$\frac{kT}{q}$: thermal voltage.

When all voltages are normalized to quantity of $\frac{kT}{q}$, the above equation can be written as:

$$\begin{aligned} I &= I_0 e^{\kappa V_g}(e^{-V_s} - e^{-V_d}) \\ &= I_0 e^{\kappa V_g - V_s}(1 - e^{-V_{ds}}), \end{aligned} \tag{5.2}$$

where V_{ds} is the voltage between the drain and source nodes. When V_{ds} is large enough, we have

$$I \approx I_{sat} = I_0 e^{\kappa V_g - V_s} \tag{5.3}$$

where I_{sat} is the so-called saturation current of the transistor. Figure 5.1 shows the relationship of the saturation current vs gate voltage.

5.2.2 Current Mirrors

Figure 5.2 shows two diode-connected transistors, their gates are connected to their drains respectively. For a typical process, I_0 is so small that even the smallest drain current used (10^{-12} amp) requires V_{gs} approximately equal to 0.4 volt [Mea89], thus makes the transistor well intro the saturation region of the drain current. Higher drain current requires higher V_{gs} (V_{ds}). Thus for any useful V_{gs}, the drain current of

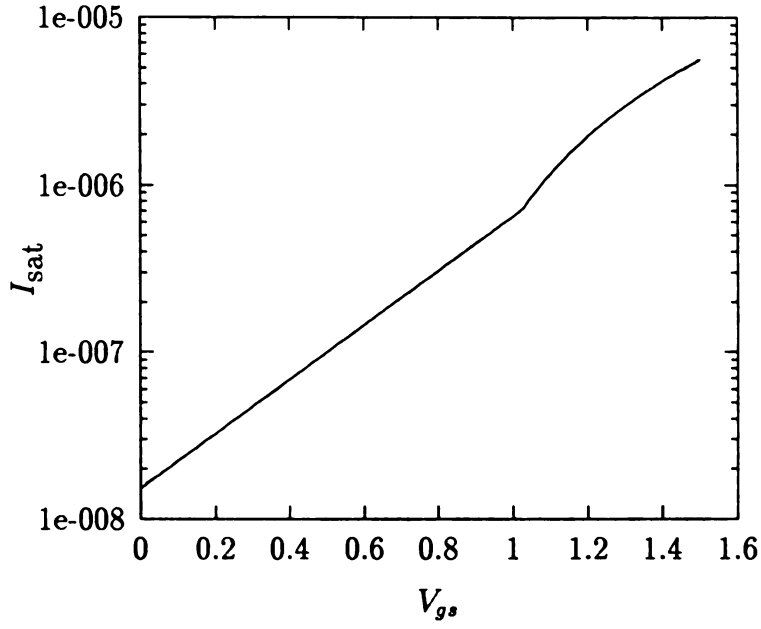


Figure 5.1. I_{sat} vs V_{gs}

the transistor has the same exponential dependence on the gate-source voltage. The current can be controlled by V_{gs}.

In circuit design, it is often necessary to change the sign (direction) of a current. Figure 5.3 shows two types of current mirrors using diode connected transistors. The input current I_{in} biases a diode-connected transistor Q_1. The resulting V_{gs} is just sufficient to bias the second transistor Q_2 to a saturation current I_{out} equal to I_{in}. As long as Q_2 stays in saturation region, I_{out} is nearly independent of the drain voltage of Q_2.

5.2.3 Differential Pair

Figure 5.5 shows a circuit which is usually called "differential pair". The circuit uses the difference of the voltages as the input. Since we will use this kind of circuits quite

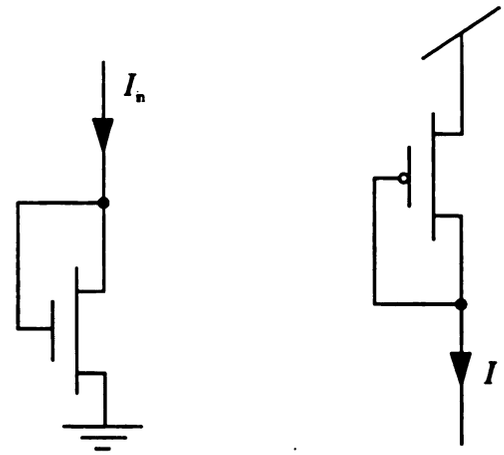


Figure 5.2. Two diode-connected transistors

often, I will give a brief analysis here.

Assume all transistors are working in the saturated region, the saturation current can be calculated as:

$$I_{sat} = I_0 e^{\kappa V_g - V_s}. \tag{5.4}$$

Thus we have

$$I_1 = I_0 e^{\kappa V_1 - V}, \tag{5.5}$$

$$I_2 = I_0 e^{\kappa V_2 - V}. \tag{5.6}$$

Since

$$\begin{aligned} I_b &= I_1 + I_2 \\ &= I_0 e^{-V}(e^{\kappa V_1} + e^{\kappa V_2}), \end{aligned} \tag{5.7}$$

we have,

$$e^{-V} = \frac{I_b}{I_0} \frac{1}{e^{\kappa V_1} + e^{\kappa V_2}}. \tag{5.8}$$

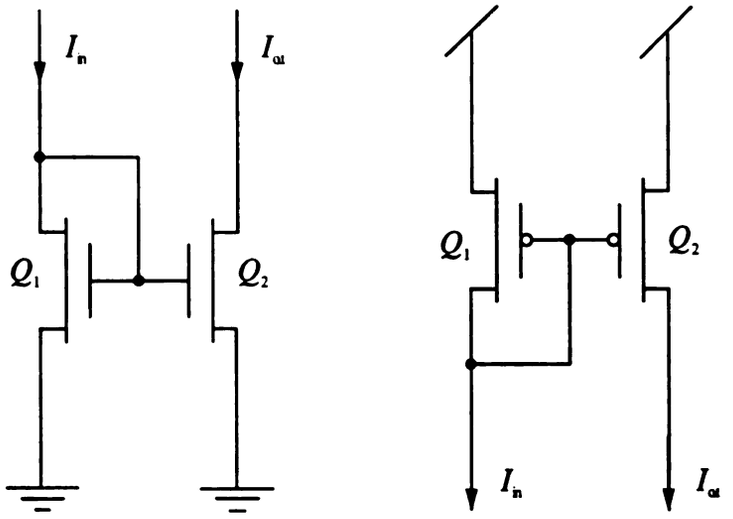


Figure 5.3. Two types of current mirrors

Substitute e^{-V} back to equation (5.5) and (5.6), we obtain the following:

$$I_1 = I_b \frac{e^{\kappa V_1}}{e^{\kappa V_1} + e^{\kappa V_2}} \tag{5.9}$$

$$I_2 = I_b \frac{e^{\kappa V_2}}{e^{\kappa V_1} + e^{\kappa V_2}}, \tag{5.10}$$

and

$$\begin{aligned} I_1 - I_2 &= I_b \frac{e^{\kappa V_1} - e^{\kappa V_2}}{e^{\kappa V_1} + e^{\kappa V_2}} \\ &= I_b \tanh \frac{\kappa(V_1 - V2)}{2}. \end{aligned} \tag{5.11}$$

When $V_1 - V_2$ is small enough,

$$I_1 - I_2 \approx I_b \frac{\kappa}{2}(V_1 - V_2). \tag{5.12}$$

Figure 5.4 shows the relationship of $I_1, I_2, I_1 - I_2$ vs $V_1 - V_2$. Using current mirror,

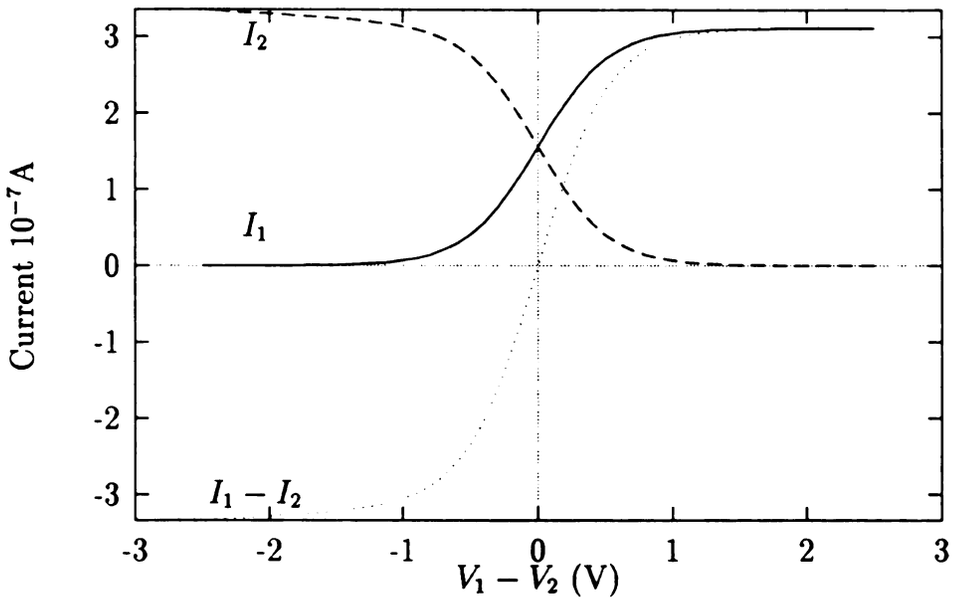


Figure 5.4. I_1, I_2 and $I_1 - I_2$ vs $V_1 - V_2$

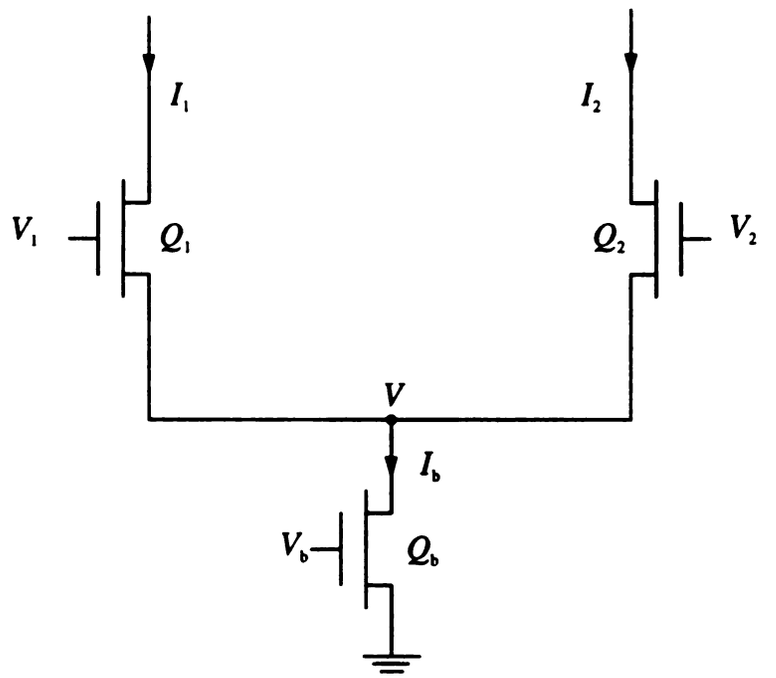


Figure 5.5. Differential pair

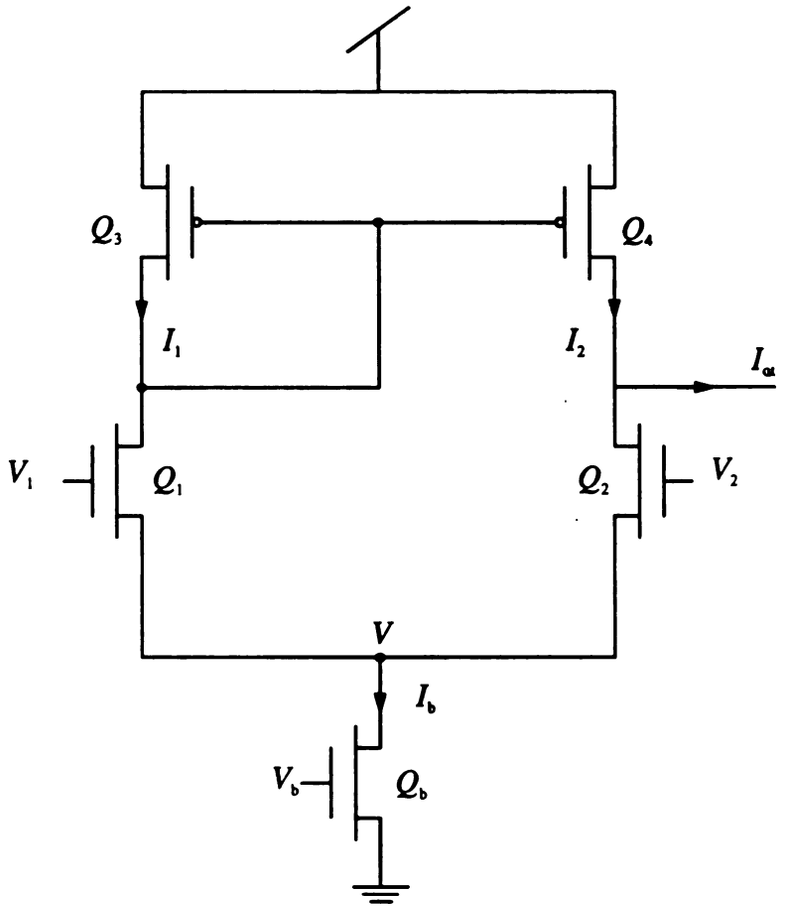


Figure 5.6. Transconductance amplifier

we can easily get $I_1 - I_2$ as output. As shown in Figure 5.6, where

$$I_{\text{out}} = I_1 - I_2 = I_b \tanh \frac{\kappa(V_1 - V_2)}{2}. \tag{5.13}$$

The circuit shown in Figure 5.6 is called transconductance amplifier.

5.2.4 Current-voltage Multiplier

Figure 5.7 shows a prototype of current-voltage multiplier circuits. Compared to the transconductance amplifier shown in the last section, the circuit output I_{out} now is drawn through two current mirrors instead of directly from the differential pair. Therefore, the output does not disturb the operation of the differential pair. It is obvious that the output current is:

$$I_{\text{out}} = I_1 - I_2 = I_b \tanh \frac{\kappa(V_1 - V_2)}{2}. \tag{5.14}$$

When $V_1 - V_2$ is small enough,

$$I_{\text{out}} \approx \frac{1}{2} I_b (V_1 - V_2). \tag{5.15}$$

Compared with the differential pair, I_b now is controlled by the input current I_{in} via a current mirror. In normal operation, the bias transistor will be in saturation region. Thus $I_b = I_{\text{in}}$ and

$$I_{\text{out}} = I_{\text{in}} \tanh \frac{\kappa(V_1 - V_2)}{2} \tag{5.16}$$

$$\approx \frac{\kappa}{2} I_{\text{in}} (V_1 - V_2) \tag{5.17}$$

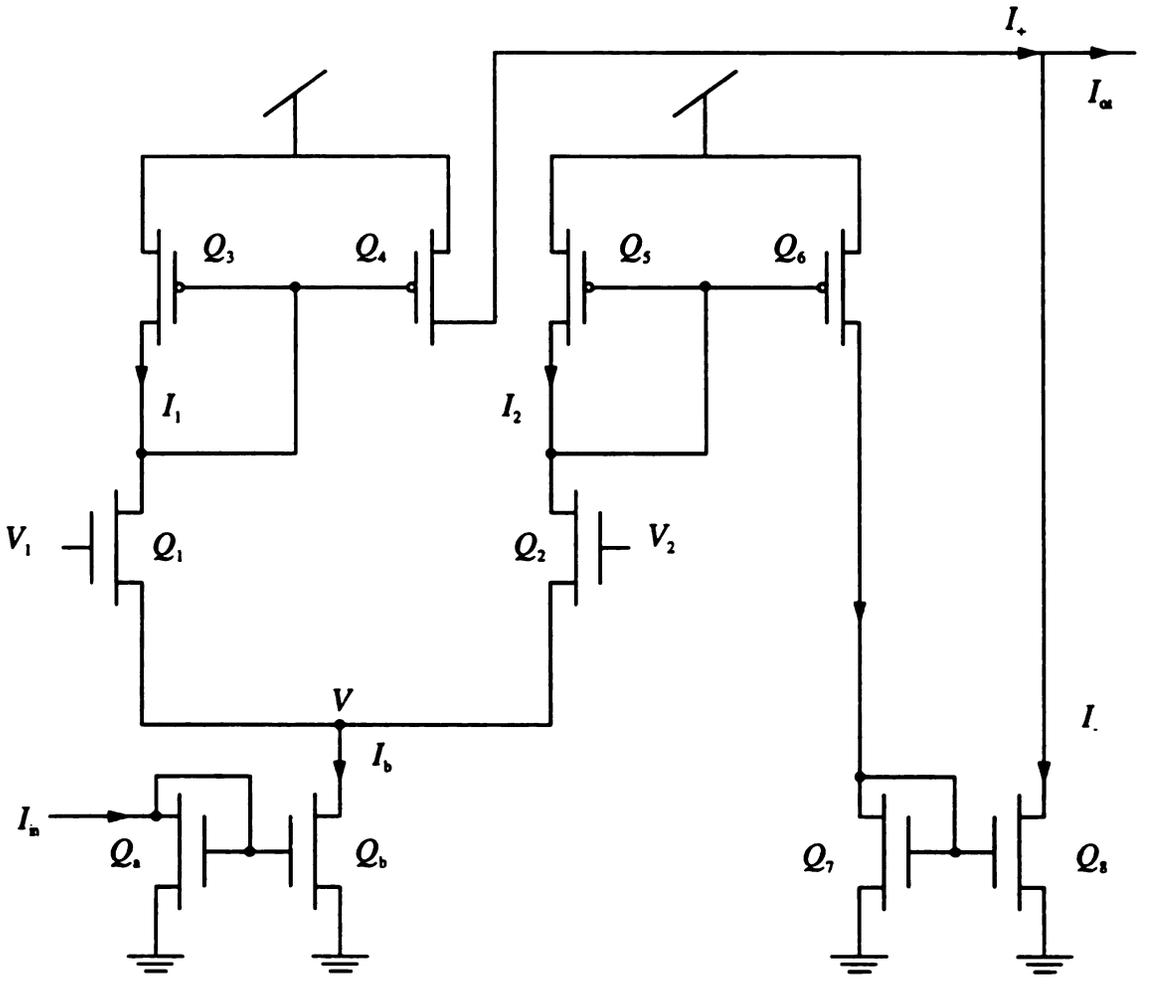


Figure 5.7. Current-voltage multiplier

So I_{out} is the product of the input current and the difference between the input voltages.

Figure 5.9 shows a pspice simulation for this circuit. We use the schematic symbol shown in Figure 5.8 to denote this circuit. Since I_b cannot be negative, this circuit is a two-quadrant multiplier.

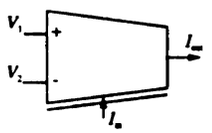


Figure 5.8. Schematic symbol for current-voltage multiplier

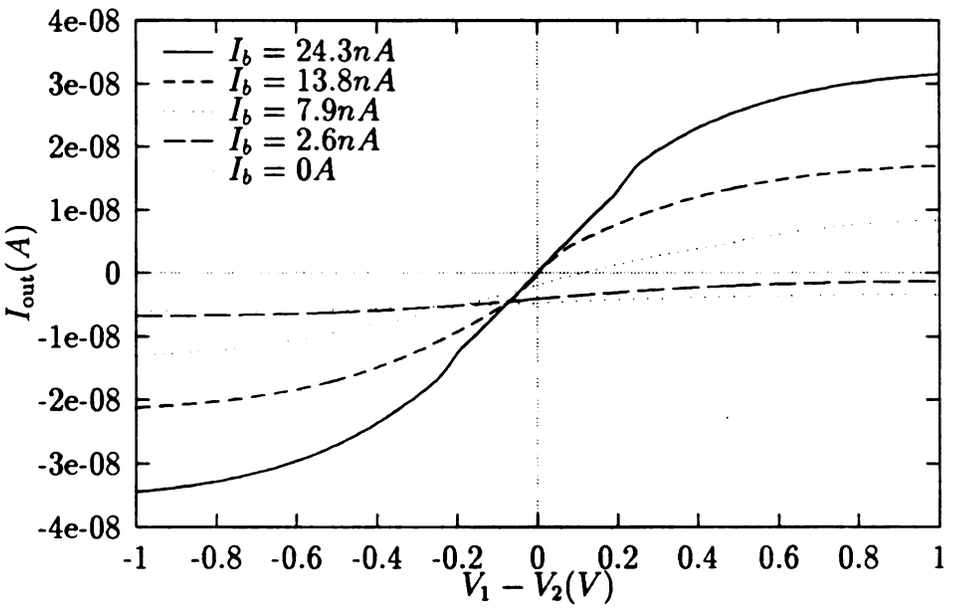


Figure 5.9. The plot of a current-voltage multiplier

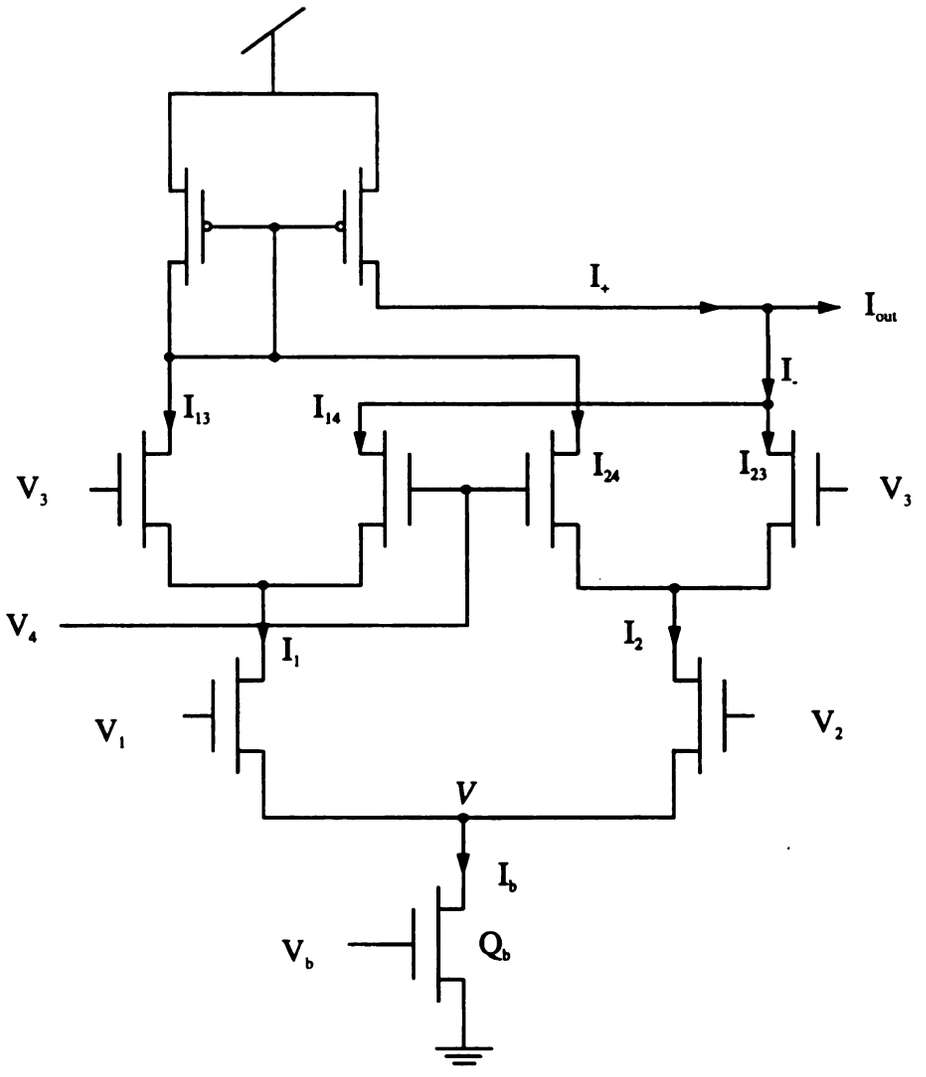


Figure 5.10. Gilbert multiplier

5.2.5 Four-quadrant Multiplier

Figure 5.10 shows a four-quadrant multiplier which is also called the "Gilbert Multiplier". From the analysis of the differential pair, we know

$$I_1 = I_b \frac{e^{\kappa V_1}}{e^{\kappa V_1} + e^{\kappa V_2}}$$

$$= \frac{I_b}{2}(1 + \tanh\frac{\kappa(V_1 - V_2)}{2}), \tag{5.18}$$

$$I_2 = \frac{I_b}{2}(1 - \tanh\frac{\kappa(V_1 - V_2)}{2}). \tag{5.19}$$

Similarly, we can obtain the following currents:

$$I_{13} = \frac{I_1}{2}(1 + \tanh\frac{\kappa(V_3 - V_4)}{2}), \tag{5.20}$$

$$I_{14} = \frac{I_1}{2}(1 - \tanh\frac{\kappa(V_3 - V_4)}{2}), \tag{5.21}$$

$$I_{23} = \frac{I_2}{2}(1 + \tanh\frac{\kappa(V_3 - V_4)}{2}), \tag{5.22}$$

$$I_{24} = \frac{I_2}{2}(1 - \tanh\frac{\kappa(V_3 - V_4)}{2}). \tag{5.23}$$

Thus

$$I_+ = \frac{I_1 + I_2}{2} + \frac{I_1 - I_2}{2}\tanh\frac{\kappa(V_3 - V_4)}{2} \tag{5.24}$$

and

$$I_- = \frac{I_1 + I_2}{2} - \frac{I_1 - I_2}{2}\tanh\frac{\kappa(V_3 - V_4)}{2}. \tag{5.25}$$

Therefore,

$$\begin{aligned} I_{\text{out}} &= I_+ - I_- \\ &= (I_1 - I_2)\tanh\frac{\kappa(V_3 - V_4)}{2} \\ &= I_b \tanh\frac{\kappa(V_1 - V_2)}{2}\tanh\frac{\kappa(V_3 - V_4)}{2}. \end{aligned} \tag{5.26}$$

When $V_1 - V_3$ and $V_3 - V_4$ are small enough, we have

$$I_{\text{out}} \approx I_b\frac{\kappa^2}{4}(V_1 - V_2)(V_3 - V_4). \tag{5.27}$$

Figure 5.11 shows a pspice simulation of this circuit.

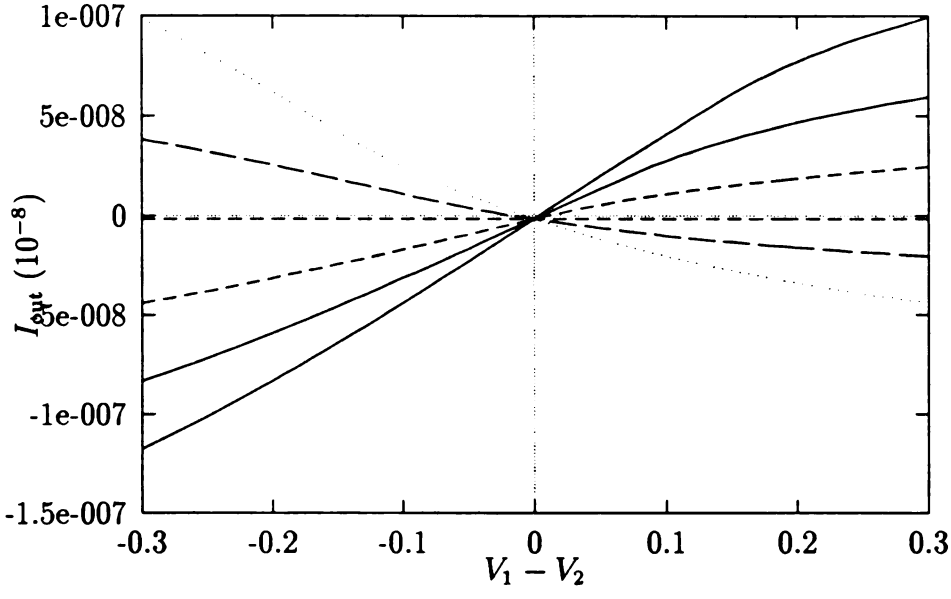


Figure 5.11. Spice simulation of a Gilbert-multiplier

In the following section, we will need to calculate $(V_1 - V_2)^2$ for the product-of-norm model. We can use this four-quadrant multiplier to achieve that by setting $V_3 = V_1$ and $V_4 = V_2$. Figure 5.12 shows the schematic symbol for this square voltage-square circuit.

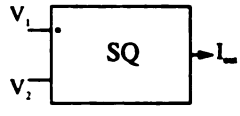


Figure 5.12. Schematic symbol of a voltage square circuit

5.3 An Example Circuit of One-neuron/two-pattern Product-of-norm Model

As we discussed in section 3.4.1 for one-neuron/m-pattn case, the product-of-norm model's dynamic equations can be written as:

$$\dot{u} = f(m) \sum_{l=1}^{m} \left[(v^{*l} - v) \prod_{\substack{k=1 \\ k \neq l}}^{m} (v^{*k} - v)^2 \right]. \tag{5.28}$$

when $m = 2$, the equation becomes

$$\dot{u} = f(m)[(v^{*1} - v)(v^{*2} - v)^2 + (v^{*2} - v)(v^{*1} - v)^2]. \tag{5.29}$$

If we use voltage to represent the output (state of the neurons), the terms $(v^{*l} - v)^2$ can be easily calculated with the voltage-square circuit.

Figure 5.13 shows an example circuit of the one-neuron/two-pattern case. As can be seen, the circuit structure is symmetric. The state of the neuron is represented by the voltage v over the capacitor C. For the voltage-square circuit, v^* and v are given as input. The output current I_1 is then given by

$$I_1 = I_b \tanh \frac{\kappa(v^{*1} - v)}{2} \tanh \frac{\kappa(v^{*1} - v)}{2}. \tag{5.30}$$

This current is then fed to a current-voltage multiplier X_2 through a current mirror. Thus the output current I_2 of the current-voltage multiplier can be written as:

$$I_2 = I_1 \tanh \frac{\kappa(v^{*2} - v)}{2}$$

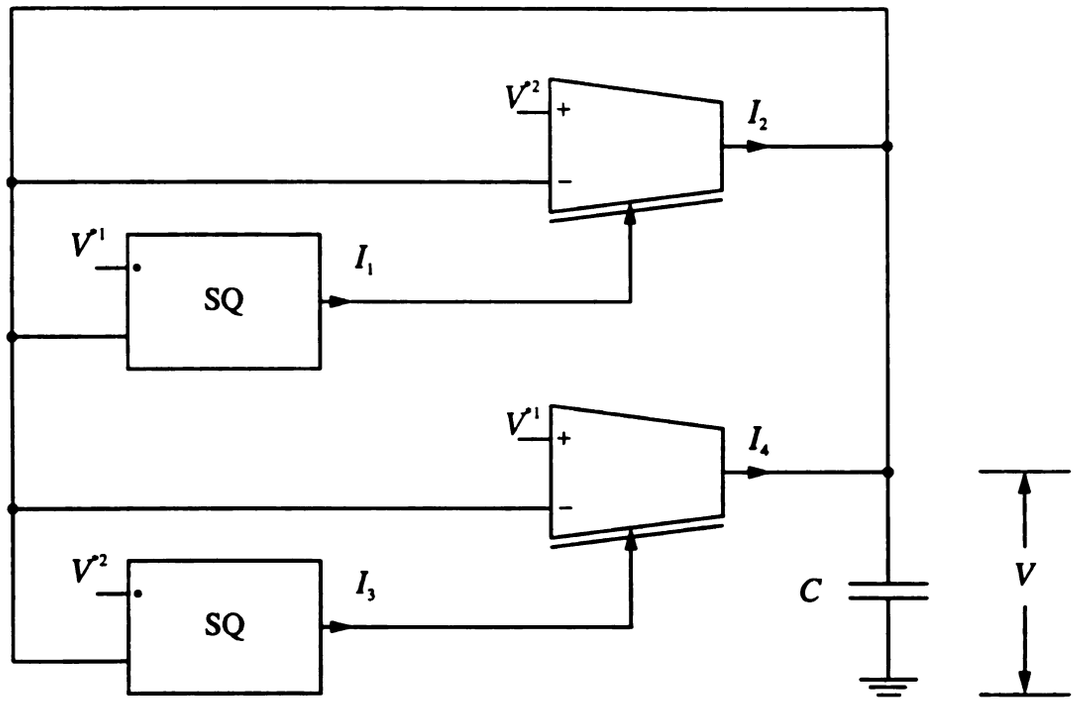


Figure 5.13. One-neuron/two-pattern circuit

$$= I_b \left[\tanh \frac{\kappa(v^{*1} - v)}{2}\right]^2 \tanh \frac{\kappa(v^{*2} - v)}{2}. \tag{5.31}$$

When $v^{*1} - v$ and $v^{*2} - v$ are small, I_2 can be written as

$$I_2 = \alpha(v^{*2} - v)(v^{*1} - v)^2, \tag{5.32}$$

where α is a constant which incorporates the effects of I_b and κ. Similarly, we can show that the current I_4 can be expressed as

$$\begin{aligned} I_4 &= I_b[\tanh \frac{\kappa(v^{*2} - v)}{2}]^2 \tanh \frac{\kappa(v^{*1} - v)}{2} \\ &\approx \alpha(v^{*1} - v)(v^{*2} - v)^2. \end{aligned} \tag{5.33}$$

Currents I_2 and I_4 are then summed up and fed to the capacitor C. Thus the voltage over the capacitor has the following dynamics:

$$C\dot{v} = I_2 + I_4. \tag{5.34}$$

When $v^{*1} - v$ and $v^{*2} - v$ are small, we have

$$C\dot{v} = \alpha(v^{*1} - v)(v^{*2} - v)^2 + (v^{*2} - v)(v^{*1} - v)^2. \tag{5.35}$$

The formula above is a differential equation for a one-neuron/two-pattern product-of-norm model.

The PSpice code and simulation examples are presented in Appendix A.

CHAPTER 6

ARTIFICIAL NEURAL NETWORKS FOR TEMPORAL-PATTERN ASSOCIATION

6.1 Introduction

In previous chapters, we studied networks that could store static patterns and were used in applications like image recognition, pattern classification and associative memory. In this chapter, we will investigate neural networks that can store/learn temporal patterns. This type of networks can be used in many applications, such as speech recognition, temporal pattern generation and forecast.

Learning or storing temporal patterns can be viewed as an extension to the general association of static patterns. Temporal patterns can be discrete or continuous. The

problem of learning temporal patterns can be classified into three different tasks [HKP90]:

1. **Sequence Recognition:** Here one wants to produce a particular output pattern when (or perhaps just after) a specific input pattern is seen). There is no need to reproduce the input pattern. This is appropriate for speech recognition problems where the output might indicate the word just spoken. It can also be used for pattern categorization.

2. **Sequence Reproduction:** In this case, the network must be able to generate the rest of the sequence itself when it receives part of it. This is the generalization of auto-association or pattern completion of dynamic patterns. It would be appropriate for learning a set of songs or for predicating the future course of a time series from examples.

3. **Temporal Association:** In this case, a particular pattern must be produced in response to a specific input pattern. The input and output patterns might be quite different, so this is the generalization of hetero-association to dynamic patterns. It includes , as special case, pure pattern generation and the previous two cases.

6.2 Methodology Review

There are many methods proposed so far to address some of the tasks mentioned above. The first task — sequence recognition — does not necessarily require a recurrent network. For the other two tasks, recurrence is usually needed. In this

section, we will review some of the models currently available.

6.2.1 Tapped Delay Lines

Tapped delay line networks turn a temporal sequence into a spatial pattern on the input layer of a feedforward network. Therefore, the conventional back-propagation method can be used to learn and recognize sequences. Figure 6.1 shows a typical structure of such network. In this type of network, values of $x(t)$, $x(t-\Delta)$, ..., $x(t-(m-1)\Delta)$ from an input signal $x(t)$ were presented simultaneously to the input layer of the network. The values are usually obtained by feeding the signal into a delay line that is tapped at various intervals. In a synchronous network, a shift register can be used. The tapped delay line network has been widely used in speech recognition problem [Wai89, WHH+89].

There are several drawbacks of this approach used for sequence recognition [Moz89]. The length of the delay line or shift register must be chosen in advance to accommodate the longest possible sequences. It cannot handle arbitrary-length patterns. The input signal must be properly registered in time with the clock controlling the delay line and must arrive at exactly the correct rate.

Tank and Hopfield [TH87] suggested a way of compensating the last limitations mentioned above. The idea is to replace the fixed delays by filters that broaden the signal in time as well as to delay it.

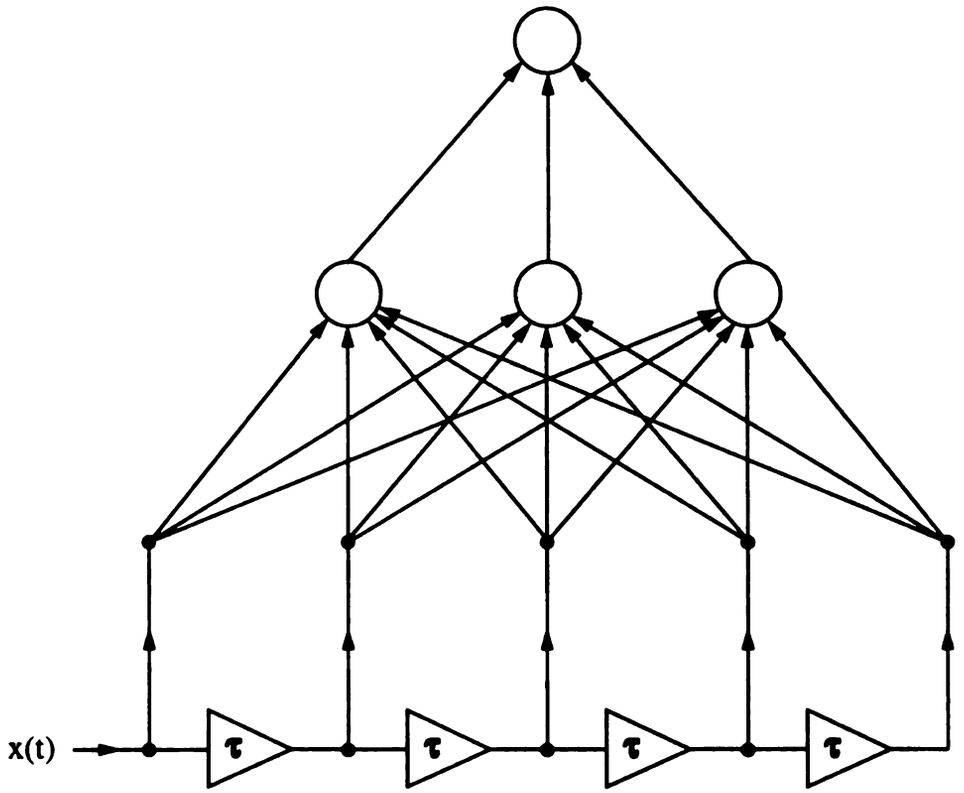


Figure 6.1. A time-delay neural network

6.2.2 Context Units (Partially Recurrent Networks)

There are several approaches falling into this category. The idea of the approach here is to introduce a carefully chosen set of feedback connections in a feedforward network. The recurrence lets the network remember cues from the recent past but does not complicate the training. In most cases, the feedback connections are fixed, not trainable, so back-propagation algorithm may easily be used for training.

Figure 6.2 shows several popular architectures of this type of network which can recognize sequences on the basis of its state at the end of the sequences. In some cases it can generates sequences too. For each net, it is not known what kind of patterns the network will be able to generate in advance. Bert DeVries proposed a Gamma model which introduces subnetworks as short-term memories to be added to the network [VP92].

6.2.3 Back Propagation Through Time

This type of networks is usually fully connected. Figure 6.3 shows a simple example of such network. If the patterns we are interested are of a maximum length, we can turn such network into a feedforward network by unfolding it through time. The idea is to represent the state of the network at different time by a layer in a feedforward network. Thus if the maximum length of the patterns concerned is T, then the network will have T layers with each layer connected by the same weight (connection) matrix. Figure 6.4 shows the unfolded version of the network shown in Figure 6.3 with four time steps. The back-propagation method can be modified to be used for training this new feedforward network. The desired patterns of the

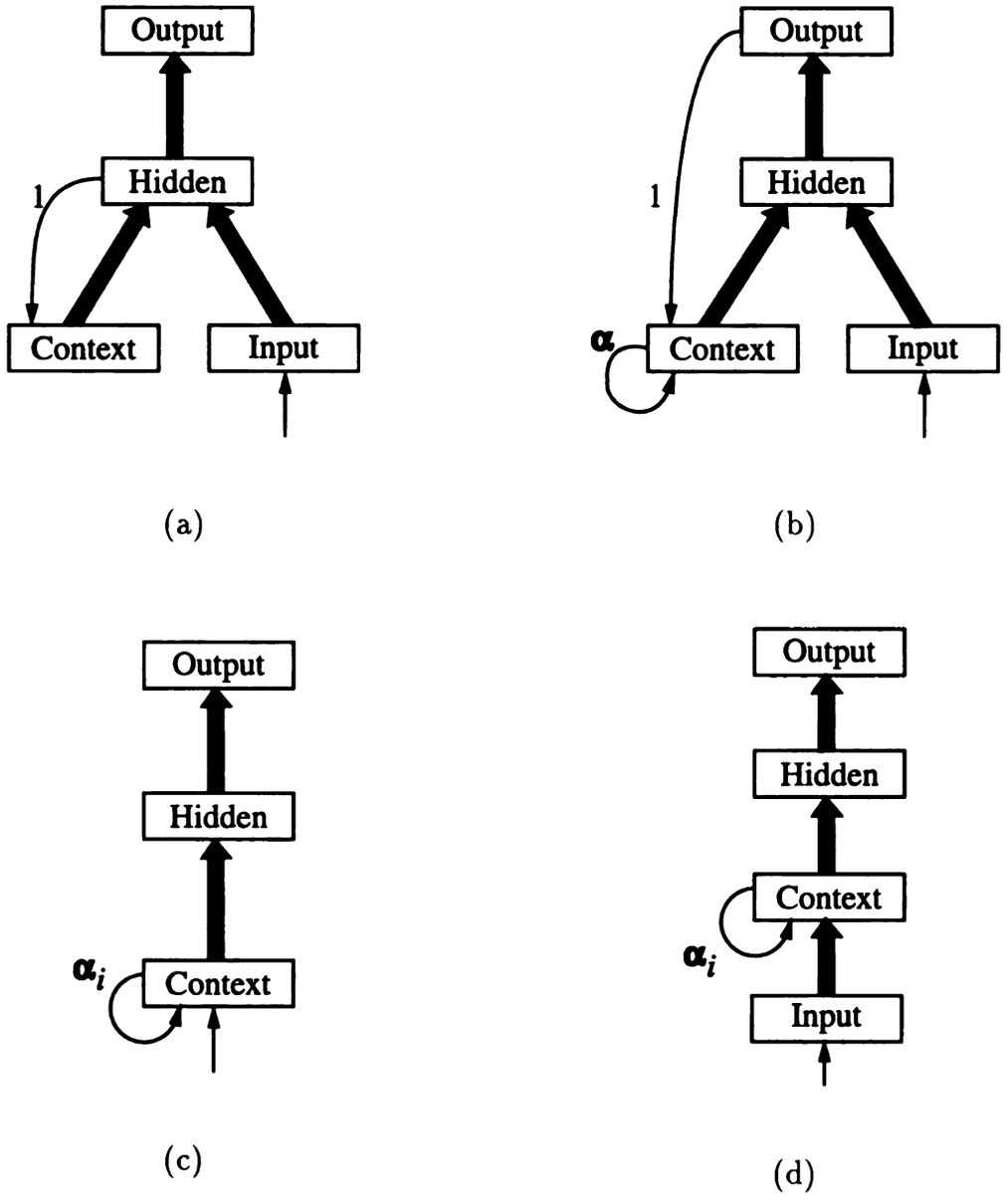


Figure 6.2. Architectures of networks using context units. Shaded arrows represent fully connected connections

original net at different time become the desired values of different layer of the new feedforward net. The main problem with this approach is the need for large computer resources. It is probably very difficult to be implemented in hardware too. For a long sequence or for a sequence of unknown length, this approach becomes impractical. It cannot be used for continuous patterns either.

6.2.4 Real-Time Recurrent Learning

Williams and Zipser [WZ89b, WZ89a] proposed a way to construct a learning rule for general recurrent networks. This method can handle patterns of fixed or indefinite length. It can be run on-line too. Here is a brief description of this method.

Assume the system difference equation is

$$V_i(t) = g(h_i(t-1)) = g(\sum_j w_{ij} V_j(t-1) + \zeta_i(t-1)), \tag{6.1}$$

where $\zeta_i(t)$ is the desired (target) output of unit i at time t. By differentiating this equation with respect to w_{ij}, we have

$$\frac{\partial V_i(t)}{\partial w_{pq}} = g'(h_i(t-1)) \left[\delta_{ip} V_q(t-1) + \sum_j w_{ij} \frac{\partial V_j(t-1)}{\partial w_{pg}} \right]. \tag{6.2}$$

The desired values $\zeta_k(t)$ may be defined for some k's and t's only. The error on neuron k at time t is defined by

$$E_k(t) = \begin{cases} \zeta_k(t) - V_k(t) & \text{if } \zeta_k \text{ is defined at } t \\ 0 & \text{otherwise} \end{cases}. \tag{6.3}$$

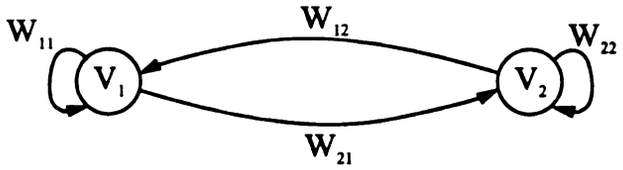


Figure 6.3. A general recurrent network

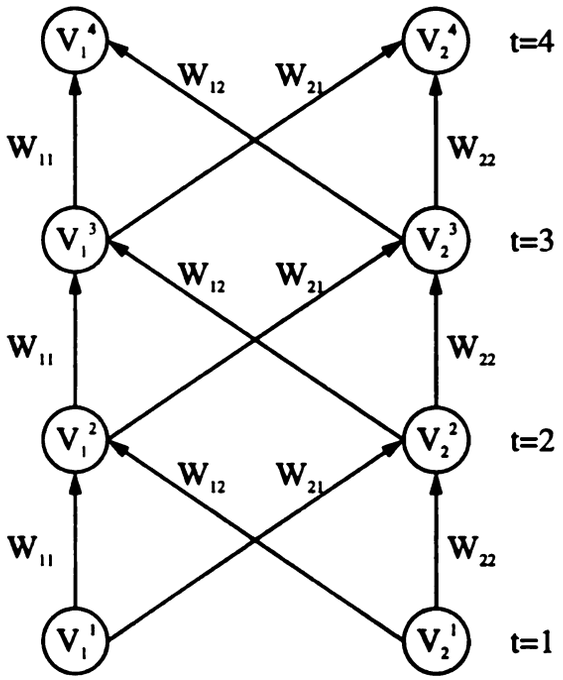


Figure 6.4. A unfolded recurrent network for 4 time steps

The system "energy" function (the overall network error) is

$$E = \sum_{t=0}^{T} E(t), \tag{6.4}$$

where

$$E(t) = \frac{1}{2}\sum_{k}[E_k(t)]^2. \tag{6.5}$$

Thus the gradient-decent weight updating rule can be written as

$$\begin{aligned} \Delta W_{pq}(t) &= -\eta \frac{\partial E(t)}{\partial W_{pq}} \\ &= \eta \sum_{k} E_k(t) \frac{\partial V_k(t)}{\partial W_{pq}}, \end{aligned} \tag{6.6}$$

where η is a positive constant. We have

$$\Delta W_{pq} = \sum_{t} \Delta W_{pq}(t). \tag{6.7}$$

Therefore, if we know $\frac{\partial V_i(0)}{\partial W_{pq}}$, we can solve this equation recursively. Because we assume that the initial state of the network has no functional dependence on the weight, we have

$$\frac{\partial V_i(0)}{\partial w_{pq}} = 0. \tag{6.8}$$

Williams and Zipser [WZ89b] found that updating the weight after each time step instead of waiting until the sequence is ended (at $t = T$) worked well if η was sufficiently small. They called it *real-time recurrent learning.*

This system requires large memory storage, and is computationally expensive. The number of operations for each time step is $O(n^4)$.

6.2.5 Time-dependent Recurrent Back-propagation

Pearlmutter proposed a training algorithm for general continuous-time recurrent networks [Pea89]. Some other researchers derived the same result independently using different approaches [SCL91]. The general system differential equations are described by

$$T_i \frac{dy_i}{dt} = -y_i + \sigma(x_i) + I_i, \qquad i = 1, 2, \ldots, n, \tag{6.9}$$

where $x_i = \sum_j w_{ji} y_j$ is the total input to unit i, y_i is the state of unit i, T_i is the time constant of unit i, σ is a sigmoid function, w_{ij}, $i, j = 1, 2, \ldots, n$, are the weights, $y_i(t_0)$ and the driving functions $I_i(t)$ are the inputs to the system.

The "energy" function is defined by

$$E = \int_{t=0}^{T} \sum_{k \in O} (y_k(t) - f_k(t))^2 dt, \tag{6.10}$$

where O is the index of the output neurons, $f_k(t)$ is the desired output of neuron k at time t, T is the final time of concern. The functional derivative

$$e_k(t) = \frac{\partial E}{\partial y_k(t)} = [y_k(t) - f_k(t)]. \tag{6.11}$$

The derivative of E with respect to weight w_{ij} is defined by (See [Pea89] for detail)

$$\frac{\partial E}{\partial w_{ij}} = \frac{1}{T_i} \int_0^T y_i \sigma'(x_j) z_j dt, \tag{6.12}$$

where Z is the *Lagrange multiplier* (or co-state variable):

$$z_i(t) = \frac{\partial^+ E}{\partial y_i(t)}, \tag{6.13}$$

where ∂^+ denotes an ordered derivative. The dynamic equations of the co-state is

$$\frac{dz_i}{dt} = \frac{1}{T_i} Z_i - e_i - \sum_j \frac{1}{T_j} W_{ij} \sigma'(x_j) z_j \tag{6.14}$$

where

$$e_i = \frac{\partial E}{\partial y_i} = y_i(t) - f_i(t). \tag{6.15}$$

Equation (6.14) can be derived using finite difference approximation [Pea89] or calculus of variation and *Lagrange multipliers* [SCL91]. We omit the derivation detail here. Similarly, we can find the derivative of the "energy" E with respect to the time constant T_i:

$$\frac{\partial E}{\partial T_i} = -\frac{1}{T_i} \int_0^T Z_i \frac{dy_i}{dt} dt. \tag{6.16}$$

Therefore, we can use gradient-descent method to update w_{ij} and T_i.

In the training process, we first integrate equation (6.9) forward from time t_0 to $t_1 = T$. Then set the boundary condition $Z_i(t_1) = 0$, and integrate the system Z backward from t_1 to t_0. In the backward process, $\sigma'(x_j) y_i$ and $\frac{dy_i}{dt}$ are also integrated. Thus at the end, we can compute $\frac{\partial E}{\partial w_{ij}}$ and $\frac{\partial E}{\partial T_i}$. $\sigma'(x_i)$ and y_i are stored in the forward pass and replayed in the backward pass.

This algorithm can be used for training general continuous-time recurrent networks. It is related to other methods mentioned above. It can derive the real-

time recurrent learning method. It may also be viewed as an extension of the back-propagation through time, or extension to recurrent back-propagation.

The drawback of this approach is that it needs to integrate backward in time. Thus it needs to store many intermediate values. Therefore, it needs large storage and it is difficult to be implemented in hardware.

6.2.6 The Green's Function Method

Sun, Chen and Lee [SCL92] proposed a method that uses a Green function to construct the derivative of the "energy" function with respect to weight w_{ij}. By doing so, they avoided a lot of redundant computations. The resulting algorithm does not need backward integration while the operations per time step is in the order of $O(n^3)$, which is one order faster than that of the real-time recurrent learning.

For a general system described by

$$\dot{x}(t) = F(x(t), W, I(t)), \tag{6.17}$$

where W is a matrix representing the set of weights and all other adjustable parameters. The nonlinear function F may look like

$$F(x(t), W, I(t)) = -x(t) + g(W, x) + I(t). \tag{6.18}$$

The "energy" function can be given as

$$E(x, x^*) = \int_{t_0}^{t_f} e(x(t), x^*(t))dt, \tag{6.19}$$

where $x^*(t)$ is the target value at time t. Using gradient descent method, the weight is modified according to

$$\Delta W \propto -\frac{\partial E}{\partial W} = -\int_{t_0}^{t_f} \frac{\partial e}{\partial x} \cdot \frac{\partial x}{\partial w} dt. \tag{6.20}$$

The on-line form is

$$\Delta W(t) = -\eta \left(\frac{\partial e}{\partial x} \cdot \frac{\partial x}{\partial W} \right), \tag{6.21}$$

where η is a positive constant. By using Green's function (we omit the derivation here, see [SCL92]) the weight updating becomes

$$\Delta W = -\eta(v(t)\Pi(t)), \tag{6.22}$$

where Π is a third order tensor with the following dynamics:

$$\begin{cases} \dfrac{d\Pi}{dt} &= U(t)\dfrac{\partial F}{\partial W} \\ \Pi(t_0) &= 0 \end{cases}. \tag{6.23}$$

v is a vector and can be obtained by solving the following linear equation:

$$v(t)U(t) = \frac{\partial e}{\partial x}. \tag{6.24}$$

$U(t)$ is a matrix and is defined by

$$\begin{cases} \dfrac{dU(t)}{dt} + U(t)\dfrac{\partial F}{\partial x} &= 0 \\ U(t_0) &= 1 \end{cases}. \tag{6.25}$$

From systems science point of view, this method is basically the transition matrix method applied to the sensitivity function with the initial condition set to zero ($V(t)$ in [SCL92] is the transition matrix).

The advantage of this method is that it can be used on-line and the operations required per time step is only $O(n^3)$.

Overall, each algorithm has its own advantages and disadvantages. Current approaches still have the following limitations:

- Require large storage.
- Computationally expensive
- Difficult to implement

For training general recurrent networks, the time-dependent recurrent back propagation approach is probably the best unless on-line learning is needed [HKP90]. It can be used for both continuous-time and discrete-time systems. When on-line learning is needed, the Green's function method is a better choice.

In the rest of this chapter, we will focus on algorithms used for general recurrent networks. We will several propose ways to improve the original time-dependent recurrent back-propagation algorithm.

6.3 Motivation

Among the training algorithms proposed for recurrent networks so far, the time-dependent recurrent back-propagation algorithm proposed by Pearlmutter and some other researchers has the following advantages:

- The system is very general and can be applied to a wide range of applications.
- It can derive many other learning algorithms.
- The adjoint variable Z has clear physical meaning.
- The operations required per time step is only $O(n^2)$.
- It can be easily extended using variational approach to solve other problems beside trajectory learning.

However, the original algorithms did not fully address the following problem:

- **The effects of the initial state:** The training algorithm does not specify how the initial state of those neurons without desired values should be chosen (if the initial values of those neurons can be controlled). If the initial values are chosen arbitrarily during the training process, the retrieval process can have totally different dynamics.
- **The final values of the system:** How to control the final values of the system? The final value of a pattern may be more important than the patterns of the intermediate states. Pearlmutter's original algorithm assumed a free ending problem and set $Z_{t_f} = 0$.
- **Avoiding the backward integration:** How to avoid the backward integration without increasing the computational complexity too much?

In the following sections we will try to further investigate the areas mentioned above.

6.4 The Initial Value Problem

The original algorithm of time-dependent recurrent back-propagation does not specify how to set the initial values of all the neurons. It assumes that the initial values are given and fixed. While in real life applications, the initial values of some of the neurons may not be given, and are changeable. If the initial values used in the retrieval process are different from those used in the training process, the system will behave differently. Therefore, it is natural to ask how we should set those initial values.

Let us take the XOR problem described in [Pea89] as an example, Figure 6.5 shows the structure of such network. The network has a total of four neurons, with two neurons in input layer, one in output layer and the other one in hidden layer. The input neurons have external connections. The goal is to select a proper set of weights so that the relationship between the external inputs and the output of the output neuron is XOR. Here the initial values of the neurons can be given to arbitrary values. Obviously, some choices of the initial values will make the training easier while others will not.

The co-state $Z_i(t)$ used in [Pea89] is defined by

$$Z_i(t) = \frac{\partial^+ E}{\partial y_i(t)}. \tag{6.26}$$

That is, $Z_i(t)$ is the ordered derivative of E with respect to the system state $y_i(t)$. Thus

$$Z_i(0) = \frac{\partial^+ E}{\partial y_i(0)}. \tag{6.27}$$

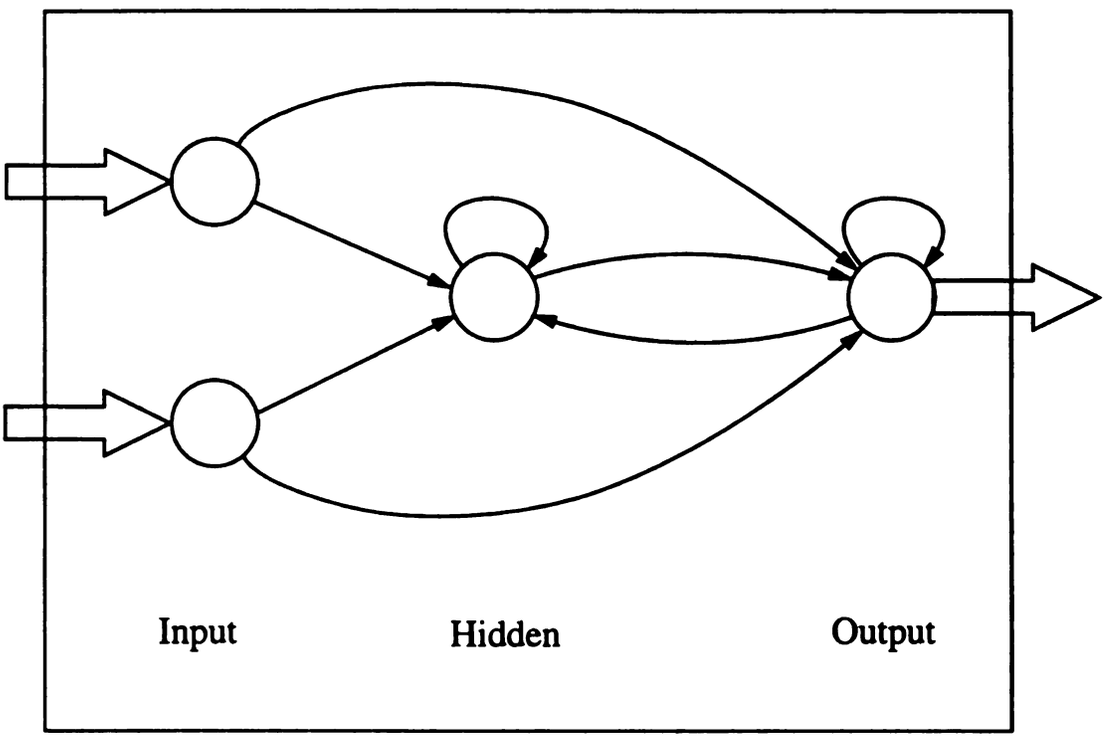


Figure 6.5. Network topology of XOR problem

Since $\frac{\partial^+ E}{\partial y_i(0)}$ represents the change in E over the time period T caused by infinitesimal change of $y_i(0)$, we can use gradient-descent method to update $y_i(0)$ in the training process to find the optimal initial values by using

$$\frac{dy_i(0)}{dt} = -\eta z_i(0), \tag{6.28}$$

where η is a positive constant.

6.5 The Choice of the Final Value of the Co-state

For a general system defined by:

$$\dot{Y} = f(X, I, t), \tag{6.29}$$

where $X = W^T Y$. If we define the performance index function as

$$J(t) = \Phi(Y(t_f), t_f) + \int_0^{t_f} L(X, I, t)dt, \tag{6.30}$$

and the final state has constraint is

$$\psi(Y(t_f), t_f) = 0. \tag{6.31}$$

The Hamiltonian is defined as

$$H(Y, I, t) = L(X, I, T) - Z^T f(X, I, t). \tag{6.32}$$

The boundary condition can be derived using optical control theory [LS95]:

$$\begin{cases} Y(t_0) \text{ is given} \\ (\phi_x + \psi_x^T v - Z)^T|_{t_f} dy(t_f) + (\phi_t + \psi_k^T v + H)|_{t_f} dt_f = 0 \end{cases}, \tag{6.33}$$

where v is also a *Lagrange Multiplier*. Since the final t_f is fixed, we have

$$\begin{aligned} (\phi_x + \psi_x^T v - Z) &= 0 \\ Z|_{t_f} &= (\phi_x + \psi_x^T v)|_{t_f}. \end{aligned} \tag{6.34}$$

Equation (6.34) specifies how to find the final value of the co-state Z. The boundary condition used in original time-dependent recurrent back-propagation is a special case of equation (6.34) which can be derived by setting $\phi_x = 0$ and $\psi_x^T = 0$:

$$Z|_{t_f} = 0. \tag{6.35}$$

In real applications, we may have constrains on the final state and $\phi(y(t_f), t_f)$ may not be a constant either. A good example is to train a network to learn a circular trajectory. It is desired to let the final state equal to the initial state. This can be emphasized by given a constraint function ϕ.

6.6 Avoiding Backward Integration in Time-dependent Recurrent Back-propagation: Method I

One major limitation of the original time-dependent recurrent back-propagation method is that it needs to integrate the co-state Z backward. In the forward process, the state variables and some other variables need to be stored so that they can be played back during the backward integration. Therefore, this algorithm requires large storage and is difficult to be implemented in hardware. There are other methods available from optimal control theory [BcH75]. However, each method has its own limitations, for example, the method of guessing the initial $Z(t_0)$ and simulate Z forward, the difficulty lies in guessing the $Z(t_0)$ to be close enough to the correct one. In this section, we will present a numerical algorithm that can avoid the backward integration.

The system equations concerned can be written in matrix form:

$$\begin{cases} \dot{Y}./T &= \zeta - Y + \sigma(X) \\ X &= W^T Y \end{cases} . \tag{6.36}$$

Here we define the operator "./" as $A_{n\times 1}./T_{n\times 1} \triangleq [a_i/T_i]_{n\times 1}$. ζ is the input vector.

The co-state (the *Lagrange multiplier*) Z has the following dynamics:

$$\dot{Z} = Z./T - \bar{E} - \{W\mathrm{diag}[\sigma'(x)./T]\}Z. \tag{6.37}$$

Here diag[V] is defined as a diagonal matrix with the diagonal elements equal to the corresponding elements of vector V. $\bar{E}$ is defined as

$$\bar{E} = Y^* - Y, \tag{6.38}$$

where Y^* is the desired value. The derivative of the "energy" E with respect to weight W can be written as:

$$\frac{\partial E}{\partial W} = \int_{t_0}^{t_1} Y[\text{diag}[\sigma'(x)./T]Z]^T dt. \tag{6.39}$$

Or in other form:

$$\frac{\partial E}{\partial W^T} = \int_{t_0}^{t_1} \text{diag}[\sigma'(x)./T]ZY^T dt. \tag{6.40}$$

In the following discussion, we define

$$\psi = \text{diag}[\sigma'(x)./T]. \tag{6.41}$$

Thus

$$\frac{\partial E}{\partial W^T} = \int_{t_0}^{t_1} \psi ZY^T dt \tag{6.42}$$

and the derivative of the "energy" with respect to T is

$$\frac{\partial E}{\partial T} = -\int_{t_0}^{t_1} \text{diag}[Z]\dot{Y} dt. \tag{6.43}$$

Using the first-order approximation, the dynamic equation of the co-state becomes

(See [Pea89])

$$\begin{aligned} Z_n &= (I - \Delta t./T)Z_{n+1} + \Delta t\bar{E}_n + \Delta t(W\mathrm{diag}[\sigma'(X_n)./T]Z_{n+1}) \\ &= \{I - \Delta t./T + \Delta t(W\mathrm{diag}[\sigma'(x_n)./T])\}Z_{n+1} + \Delta t\bar{E}_n. \end{aligned} \tag{6.44}$$

Where I is an identity matrix. Define

$$\begin{cases} \mu_n &= I - \Delta t(I./T - W\mathrm{diag}[\sigma'(x_n)./T]) \\ B_n &= \Delta t\bar{E}_n \end{cases}. \tag{6.45}$$

Then,

$$Z_n = \mu_n Z_{n+1} + B_n. \tag{6.46}$$

Write equation (6.42) in difference form:

$$\frac{\partial E}{\partial W^T} = \Delta t \sum_{k=1}^{N} \psi_k Z_k Y_k^T. \tag{6.47}$$

Here we assume the final time $T = N\Delta t$. At time t_1, we have

$$Z_1 = \mu_1 Z_2 + B_1 \tag{6.48}$$

$$\mu_1 = I - \Delta t\,[I./T - W\mathrm{diag}[\sigma(X_1)./T]] \tag{6.49}$$

$$B_1 = \Delta t E_1. \tag{6.50}$$

If we denote

$$\left(\frac{\partial E}{\partial W^T}\right)_k = \sum_{l=1}^{k} \psi_l Z_l Y_l^T, \tag{6.51}$$

we have

$$\left(\frac{\partial E}{\partial W^T}\right)_1 = \psi_1 Z_1 Y_1^T \tag{6.52}$$

$$= \psi_1(\mu_1 Z_2 + B_1) Y_1^T \tag{6.53}$$

$$= \psi_1 \mu_1 Z_2 Y_1^T + \psi_1 B_1 Y_1^T. \tag{6.54}$$

At t_2 we have

$$Z_2 = \mu_2 Z_3 + B_2 \tag{6.55}$$

$$\begin{aligned}\left(\frac{\partial E}{\partial W^T}\right)_2 &= \psi_2 Z_2 Y_2^T + \left(\frac{\partial E}{\partial W^T}\right)_1 \\ &= \psi_2 Z_2 Y_2^T + \psi_1 \mu_1 Z_2 Y_1^T + \psi_1 B_1 Y_1^T \\ &= \psi_2(\mu_2 Z_3 + B_2) Y_2^T + \psi_1 \mu_1 (\mu_2 Z_3 + B_2) Y_1^T + \psi_1 B_1 Y_1^T \\ &= \psi_2 \mu_2 Z_3 Y_2^T + \psi_1 \mu_1 \mu_2 Z_3 Y_1^T + \psi_2 B_2 Y_2^T + \psi_1 \mu_1 B_2 Y_1^T + \psi_1 B_1 Y_1^T. \end{aligned} \tag{6.56}$$

Similarly, at t_3:

$$\begin{aligned}\left(\frac{\partial E}{\partial W^T}\right)_3 &= \psi_3 Z_3 Y_3^T + \left(\frac{\partial E}{\partial W^T}\right)_2 \\ &= \psi_3 \mu_3 Z_4 Y_3^T + \psi_2 \mu_2 \mu_3 Z_4 Y_2^T + \psi_1 \mu_1 \mu_2 \mu_3 Z_4 Y_1^T + \\ &\quad \psi_3 B_3 Y_3^T + \psi_2 \mu_2 B_3 Y_2^T + \psi_1 \mu_1 \mu_2 B_3 Y_1^T + \\ &\quad \psi_2 B_2 Y_2^T + \psi_1 \mu_1 B_2 Y_1^T + \\ &\quad \psi_1 B_1 Y_1^T. \end{aligned} \tag{6.57}$$

If we consider $\frac{\partial E}{\partial W^T} = \left[\frac{\partial E}{\partial w_{ij}}\right]_{n\times n}$ one column at a time and define

$$\frac{\partial E}{\partial W_k} = \begin{bmatrix} \frac{\partial E}{\partial W_{k1}} \\ \frac{\partial E}{\partial W_{k2}} \\ \vdots \\ \frac{\partial E}{\partial W_{kn}} \end{bmatrix}. \tag{6.58}$$

Then at time t_1 we have

$$\left(\frac{\partial E}{\partial W_k}\right)_{t_1} = \psi_1 \mu_1 Y_1^k Z_2 + \psi_1 B_1 Y_1^k. \tag{6.59}$$

At t_2:

$$\begin{aligned} \left(\frac{\partial E}{\partial W_k}\right)_{t_2} &= \left[\psi_2 \mu_2 Y_2^k + \psi_1 \mu_1 \mu_2 Y_1^k\right] Z_3 + \\ &\quad \left[\psi_2 Y_2^k + \psi_1 \mu_1 Y_1^k\right] B_2 \\ &\quad + \psi_1 B_1 Y_1^k. \end{aligned} \tag{6.60}$$

We define

$$\beta_1^k = \psi_1 Y_1^k \tag{6.61}$$

$$\begin{aligned} \beta_2^k &= \left[\psi_2 Y_2^k + \psi_1 \mu_1 Y_1^k\right] \\ &= \psi_2 Y_2^k + \beta_1^k \mu_1. \end{aligned} \tag{6.62}$$

Then at time t_3

$$\begin{aligned}\left(\frac{\partial E}{\partial W_k}\right)_{t_3} &= \left[\psi_3 Y_3^k + \psi_2\mu_2 Y_2^k + \psi_1\mu_1\mu_2 Y_1^k\right][\mu_3 Z_4 + B_3] + \beta_2^k B_2 + \beta_1^k B_1 \\ &= \left[\psi_3 Y_3^k + \beta_2^k\mu_2\right][\mu_3 Z_4 + B_3] + \beta_2^k B_2 + \beta_1^k B_1. \qquad (6.63)\end{aligned}$$

Thus by defining

$$\beta_n^k = \psi_n Y_n^k + \beta_{n-1}^k \mu_{n-1}, \qquad n = 2, 3, \ldots, \qquad (6.64)$$

we have

$$\begin{aligned}\left(\frac{\partial E}{\partial W_k}\right)_t &= \beta_t^k \mu_t Z_{t+1} + \sum_{l=1}^{t} \beta_l^k B_l \\ &= \beta_t^k \mu_t Z_{t+1} + Q_t^k, \qquad (6.65)\end{aligned}$$

where

$$B_t = \Delta t[e_i(t)]_{n\times 1} \qquad (6.66)$$

$$\mu_t = I - \Delta t(I + W\mathrm{diag}[\sigma'(x)./T]) \qquad (6.67)$$

$$Q_t^k = \sum_{l=1}^{t} \beta_l^k B_l. \qquad (6.68)$$

At the final step $t = N$ and

$$\left(\frac{\partial E}{\partial W_k}\right)_N = \beta_N^k \mu_N Z_T + Q_N^k. \qquad (6.69)$$

If we use the same boundary condition as used by Pearlmutter in [Pea89]:

$$Z_T = 0, \tag{6.70}$$

we could further simplify equation (6.69) as:

$$\left(\frac{\partial E}{\partial W_k}\right)_N = \sum_{l=1}^{N} \beta_l^k B_l \qquad k = 1, 2, \ldots, n. \tag{6.71}$$

Thus

$$\left(\frac{\partial E}{\partial W^T}\right) \approx \left(\frac{\partial E}{\partial W_k}\right)_N. \tag{6.72}$$

6.6.1 Simulation Procedure

In simulation, from initial time t_0 to the final time T, we need to calculate the following at each time step (interval Δt):

First we need to simulate the system difference equation (6.36) forward, which will calculate the values of Y_t, E_t and X_t. Then ψ_t can be computed using equation (6.41). Variables μ_t and B_t can be found using equation (6.45). We need to calculate β_t^k using equation (6.64) for each $k = 1, 2, \ldots, n$. Thus we can obtain Q_t^k using equation (6.68) for each $k = 1, 2, \ldots, n$. At the final step T, we can compute the derivative of the "energy" with respect to weight W using

$$\left(\frac{\partial E}{\partial W_k}\right)_t = \beta_t^k \mu_t Z_{t+1} + Q_t^k, \tag{6.73}$$

and

$$\left(\frac{\partial E}{\partial W^T}\right) = \Delta t \left(\frac{\partial W}{\partial W_k}\right)_T . \tag{6.74}$$

Therefore, we can update the weight using gradient-descent method:

$$\Delta W = -\eta \left(\frac{\partial E}{\partial W}\right) \tag{6.75}$$

$$W_{\text{new}} = W_{\text{old}} + \Delta W, \tag{6.76}$$

here η is a positive constant.

Thus this algorithm needs only the forward integration. If we select $Z_T = 0$, this algorithm can be further simplified and used on-line too. The computation complexity can be estimated as follows:

For each time step, calculating Y, $\bar{E}$ and X needs $O(2n^2)$ operations. Computing ψ needs $O(n)$ operations. Updating μ and B requires $O(4n^2)$ operations. The total operations for calculating β^k, $k = 1, 2, \ldots, n$, is $O(n^4 + n^3)$. Calculating Q also needs $O(n^3)$ operations per time step. Therefore, the total operations per time step will be $O(n^4)$, which is the same order as the real-time recurrent method used by Williams and Zipser [WZ89a]. The advantages of this method are that it can use the Z_T for different boundary conditions and it avoids backward integration. However, the computation complexity is still high.

Figure 6.6 to figure 6.9 show the trajectories of an example XOR circuit, as shown in figure 6.5, trained using the above method.

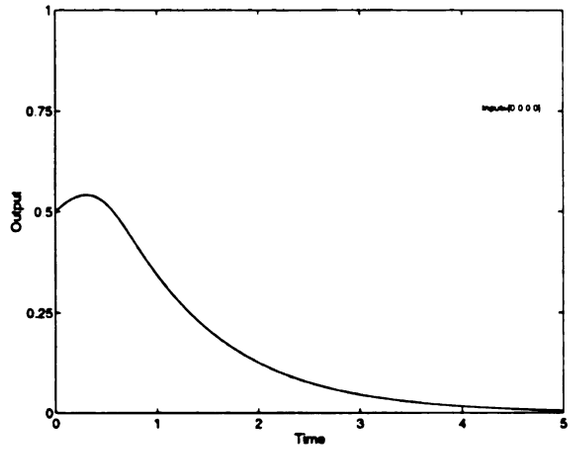


Figure 6.6. XOR network with input = $[0, 0]$

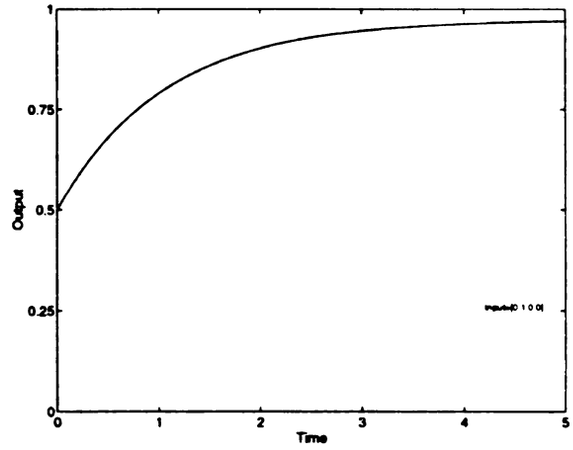


Figure 6.7. XOR network with input = $[0, 1]$

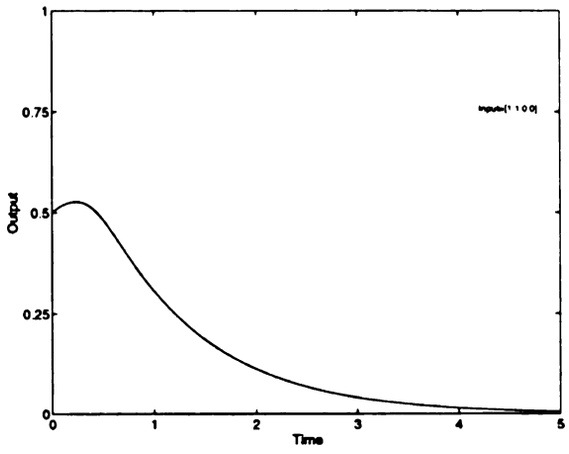


Figure 6.8. XOR network with input = [1, 1]

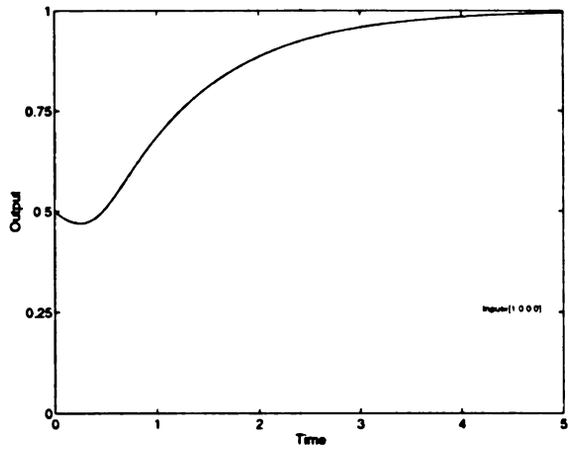


Figure 6.9. XOR network with input = [1, 0]

6.7 Avoiding Backward Integration: Method II

In last section, we presented one way of avoiding the backward integration used in time-dependent recurrent back-propagation method. In this section, we will present another numerical algorithm, which avoids backward integration too while not increasing computational complexity too much.

From last section, we know that the co-state dynamical equations can be written in vector form as:

$$\dot{Z} = Z./T - \bar{E} - (W\text{diag}[\sigma'(X)./T])Z, \tag{6.77}$$

where $Z./T \triangleq [z_{ij}/t_j]$ and diag$[X]$ is a diagonal matrix with diagonal elements equal to the element of vector X. In the following discussion, we denote $\psi \triangleq \text{diag}[\sigma'(X)./T]$.

The system defined by equation (6.77) is a linear time-varying system. It can be written in general form as

$$\dot{Z} = A(t)Z + B(t). \tag{6.78}$$

The transition matrix of this system $\Phi(t, t_0)$ can be calculated using

$$\dot{\Phi} = A(t)\Phi \qquad \text{with } \Phi(t_0, t_0) = I. \tag{6.79}$$

and we have

$$\begin{aligned} Z &= \Phi(t, t_0)Z_{t_0} + \int_{t_0}^{t} \Phi(t, \tau)B(\tau)d\tau \\ &= \Phi(t, t_0)Z_{t_0} + \Phi(t, t_0)\int_{t_0}^{t} \Phi^{-1}(\tau, t_0)B(\tau)d\tau. \end{aligned} \tag{6.80}$$

Thus

$$\begin{aligned}\frac{\partial E}{\partial W^T} &= \int_{t_0}^{t_f} \psi Z Y^T dt \\ &= \int^{t_f} -t_0\psi[\Phi(t,t_0)Z_{t_0} + \Phi(t,t_0)\int_{t_0}^{t}\Phi^{-1}(\tau,t_0)B(\tau)d\tau]Y^T dt \\ &= Z_{t_0}\underbrace{\int_{t_0}^{t_f}\psi\Phi(t,t_0)dt}_{\text{term 1}} + \underbrace{\int_{t_0}^{t_f}\psi\Phi(t,t_0)\int_{t_0}^{t}\Phi^{-1}(\tau,t_0)B(\tau)d\tau dt}_{\text{term 2}}. \qquad (6.81)\end{aligned}$$

Both term 1 and term 2 can be calculated forward. If we define $\Psi = \Phi^{-1}$, then it can be simulated with

$$\dot{\Psi} = -A(t)\Psi \qquad \text{with } \Psi(t,t_0) = I \qquad (6.82)$$

Z_{t_0} can be calculated at final time t_f using

$$Z_{t_0} = \Phi^{-1}(t_f,t_0)[Z_{t_f} - \int_{t_0}^{t_f}\Phi(t_f,\tau)B(\tau)d\tau]. \qquad (6.83)$$

We now show how to implement the idea above numerically. We use the following approximation with sufficiently small time step Δt:

$$\frac{\bar{Z}_{n+1} - \bar{Z}_n}{\Delta t} = \dot{\bar{Z}}_n, \qquad (6.84)$$

where $\bar{Z}_n = \bar{Z}_{t=n\Delta t}$. Therefore,

$$\begin{aligned}\bar{Z}_{n+1} &= \bar{Z}_n + \Delta t\dot{\bar{Z}}_n \\ &= \bar{Z}_n + \Delta t(\bar{Z}_n./T - \bar{E}_n - (W\text{diag}[\sigma'(X_n)./T])\bar{Z}_n). \qquad (6.85)\end{aligned}$$

If we define

$$\begin{cases} \nu_n &= I + \Delta t I./T - \Delta t(W \mathrm{diag}[\sigma'(X_n)./T]) \\ C_n &= -\Delta t \bar{E}_n \end{cases}, \tag{6.86}$$

we have

$$\bar{Z}_{n+1} = \nu_n \bar{Z}_n + C_n. \tag{6.87}$$

From equation (6.87), we have:

$$\begin{aligned} \bar{Z}_n &= \nu_{n-1}\bar{Z}_{n-1} + C_{n-1} \\ &= \nu_{n-1}[\nu_{n-2}\bar{Z}_{n-2} + C_{n-2}] + C_{n-1} \\ &= \nu_{n-1}\nu_{n-2}\bar{Z}_{n-2} + \nu_{n-1}C_{n-2} + C_{n-1} \\ &= \cdots\cdots\cdots\cdots \\ &= \nu_{n-1}\nu_{n-2}\cdots\nu_1\bar{Z}_1 + \nu_{n-1}\nu_{n-2}\cdots\nu_2 C_1 + \nu_{n-1}\nu_{n-2}\cdots\nu_3 C_2 + \cdots + C_{n-1}. \end{aligned}$$

Thus

$$\bar{Z}_k = p_{k-1}\bar{Z}_1 + q_{k-1} \qquad\qquad k = 2, 3, \ldots, N, \tag{6.88}$$

where we define:

$$q_k = \nu_k q_{k-1} + C_k \tag{6.89}$$

$$p_k = \nu_k p_{k-1}, \tag{6.90}$$

with

$$q_1 = C_1 \tag{6.91}$$

$$p_1 = \nu_1. \tag{6.92}$$

The partial derivative of total "energy" with respect to weight can be approximated in the following way:

$$\frac{\partial E}{\partial W^T} = \Delta t \sum_{k=1}^{N} \psi_k \bar{Z}_k Y_k^T. \tag{6.93}$$

Here we assume $T = \Delta t N$. For simplicity, we omit the Δt in the following derivations. If we define

$$\left(\frac{\partial E}{\partial W^T}\right)_k = \sum_{l=1}^{k} \psi_l \bar{Z}_l Y_l^T, \tag{6.94}$$

then, at each time step we have

$$\left(\frac{\partial E}{\partial W^T}\right)_1 = \psi_1 \bar{Z}_1 Y_1^T \tag{6.95}$$

$$\left(\frac{\partial E}{\partial W^T}\right)_2 = \psi_1 \bar{Z}_1 Y_1^T + \psi_2[\nu_1 \bar{Z}_1 + C_1] Y_2^T \tag{6.96}$$

$$\left(\frac{\partial E}{\partial W^T}\right)_3 = \psi_1 \bar{Z}_1 Y_1^T + \psi_2[\nu_1 \bar{Z}_1 + C_1] Y_2^T + \psi_3[\nu_2 \nu_1 \bar{Z}_1 + \nu_2 C_1 + C_2] Y_3^T \tag{6.97}$$

$$\left(\frac{\partial E}{\partial W^T}\right)_k = \sum_{l=1}^{k} \psi_l p_{l-1} \bar{Z}_1 Y_l^T + \sum_{l=2}^{k} \psi_l q_{l-1} Y_l^T \tag{6.98}$$

with $p_0 = I$.

Consider the ith column of $\left(\frac{\partial E}{\partial W^T}\right)_k$, which is

$$\begin{aligned} &\psi_1 \bar{Z}_1 Y_1^i + \psi_2 \nu_1 \bar{Z}_1 Y_2^i + \cdots + \psi_k p_{k-1} \bar{Z}_1 Y_k^i \\ &+\psi_2 q_1 Y_2^i + \psi_3 q_2 Y_3^i + \cdots + \psi_k q_{k-1} Y_k^i. \end{aligned} \tag{6.99}$$

By defining:

$$R_k^i = R_{k-1}^i + \psi_k p_{k-1} Y_k^i \tag{6.100}$$

with

$$R_0 = 0 \tag{6.101}$$

and

$$Q_k = \sum_{l=2}^{k} \psi_l q_{l-1} Y_l^T, \tag{6.102}$$

we have

$$\left(\frac{\partial E}{\partial W^T}\right)_k^i = R_k^i \bar{Z}_1 + Q_k^i. \tag{6.103}$$

Therefore, if $\bar{Z}_1$ is known, we can integrate the system forward. However, the boundary condition we know is $\bar{Z}_N$, not $\bar{Z}_1$.

Now, let us consider simulating the co-state system backward using difference equations. For sufficiently small Δt, we have the following approximations:

$$\frac{\tilde{Z}_n - \tilde{Z}_{n+1}}{\Delta t} = -\dot{\tilde{Z}}_{n+1} \tag{6.104}$$

from equation (6.104), we have:

$$\begin{aligned} \tilde{Z}_n &= \tilde{Z}_{n+1} - \Delta t \dot{\tilde{Z}}_{n+1} \\ &= \tilde{Z}_{n+1} - \Delta t(\tilde{Z}_{n+1}./T - \bar{E}_{n+1} - [W \mathrm{diag}[\sigma'(X_{n+1})./T]]\tilde{Z}_{n+1}). \end{aligned}$$

We assume the simulation time period is $[0, t_1]$, and $t_1 = N\Delta t$, N is an integer.

Define

$$\mu_{n+1} = I - \Delta t I./T + \Delta t(W\text{diag}[\sigma'(X_{n+1})./T]) \tag{6.105}$$

and

$$B_{n+1} = \Delta t \bar{E}_{n+1}. \tag{6.106}$$

We have

$$\tilde{Z}_n = \mu_{n+1}\tilde{Z}_{n+1} + B_{n+1}. \tag{6.107}$$

Thus, use equation (6.107) recursively:

$$\begin{aligned}
\tilde{Z}_1 &= \mu_2\tilde{Z}_2 + B_2 \\
&= \mu_2(\mu_3\tilde{Z}_3 + B_3) + B_2 \\
&= \mu_2\mu_3\tilde{Z}_3 + \mu_2 B_3 + B_2 \\
&= \cdots\cdots\cdots\cdots \\
&= \mu_2\mu_3\cdots\mu_k\tilde{Z}_k + \mu_2\mu_3\cdots\mu_{k-1}B_k + \mu_2\mu_3\cdots\mu_{k-2}B_{k-1} + \cdots + B_2.
\end{aligned}$$

The equation above can be written in general format as

$$\tilde{Z}_1 = U_k\tilde{Z}_k + D_k \qquad k = 2, 3, \ldots, N, \tag{6.108}$$

where we define:

$$U_k = U_{k-1}\mu_k \qquad k = 3, 4, \ldots, N \tag{6.109}$$

$$D_k = D_{k-1} + U_{k-1}B_k \qquad k = 3, 4, \ldots, N \tag{6.110}$$

$$D_2 = B_2 \tag{6.111}$$

$$U_2 = \mu_2. \tag{6.112}$$

When the time step Δt is sufficiently small, the two systems $\tilde{Z}$ and $\bar{Z}$ should be so close that $\tilde{Z}_t \approx \bar{Z}_t$. Therefore, we have

$$\tilde{Z}_N = \bar{Z}_N \tag{6.113}$$

$$\tilde{Z}_1 = \bar{Z}_1. \tag{6.114}$$

Thus equation (6.103) can be written as

$$\left(\frac{\partial E}{\partial W^T}\right)_k^i = R_k^i(U_N \tilde{Z}_N + D_N) + Q_k^i. \tag{6.115}$$

At $k = N$, it becomes

$$\left(\frac{\partial E}{\partial W^T}\right)_N^i = R_N^i(U_N \tilde{Z}_N + D_N) + Q_N^i. \tag{6.116}$$

Since R, U, D and Q all can be calculated forward in time, we can simulate this system forward in time and at time T calculate the quantity $\left(\frac{\partial E}{\partial W^T}\right)$. This way we can avoid the backward integration.

6.7.1 Updating the Time-constant T

The partial derivative of total "energy" with respect to T can be approximated using:

$$\frac{\partial E}{\partial T} = -\Delta t \sum_{k=1}^{N} \text{diag}[\dot{Y}_k]\bar{Z}_k. \tag{6.117}$$

As with the updating of weight, we omit Δt in the following derivation. Therefore, at time step 1 we have

$$\left(\frac{\partial E}{\partial T}\right)_1 = -\text{diag}[\dot{Y}_1]\bar{Z}_1.$$

At step 2,

$$\begin{aligned}\left(\frac{\partial E}{\partial T}\right)_2 &= -\text{diag}[\dot{Y}_2]\bar{Z}_2 \\ &= -\text{diag}[\dot{Y}_2](\nu_1\bar{Z}_1 + C_1).\end{aligned}$$

At step 3

$$\begin{aligned}\left(\frac{\partial E}{\partial T}\right)_3 &= -\text{diag}[\dot{Y}_3]\bar{Z}_3 \\ &= -\text{diag}[\dot{Y}_3](p_2\bar{Z}_1 + q_2).\end{aligned}$$

Define

$$S_k = S_{k-1} + \text{diag}[\dot{Y}_k]p_{k-1} \qquad k = 1, 2, \ldots, N \tag{6.118}$$

$$J_k = J_{k-1} + \text{diag}[\dot{Y}_k]q_{k-1} \qquad k = 2, 3, \ldots, N, \tag{6.119}$$

with initial conditions

$$S_0 = 0, \tag{6.120}$$

$$J_1 = J_0 = 0. \tag{6.121}$$

Then

$$\left(\frac{\partial E}{\partial T}\right)_{\text{total}} = -S_N \bar{Z}_1 - J_N. \tag{6.122}$$

Therefore, we can calculate S and J forward. At the final step T, we calculate $\left(\frac{\partial E}{\partial T}\right)_{\text{total}}$.

6.7.2 Computation Procedures

From the derivation above, we can summarize the computation flow as follows:

Start with $k = 0$, at time step k:

Step 1. Calculate Y and E (for $k = 0, 1, \ldots, N-1$) using:

$$Y_{k+1} = Y_k + \Delta t[\sigma(X_k) + \zeta_k + Y_k]./T$$

$$E_k = Y_k - Y_k^\star$$

$$X_k = W^T Y_k.$$

Step 2. Compute ψ:

$$\psi_k = \text{ diag } [\sigma'(X_k)./T].$$

Step 3. Calculate ν and C:

$$\begin{cases} \nu_k &= I + \Delta t I./T - \Delta t W \psi_k \\ C_k &= -\Delta t \bar{E}_k \end{cases}.$$

Step 4. Calculate μ and B:

$$\begin{cases} \mu_k &= I - \Delta t I./T + \Delta t W \psi_k \\ B_k &= \Delta t \bar{E}_k \end{cases}$$

Step 5. U and D (for $k = 2, 3, \ldots, N$) can be calculated with

$$U_k = U_{k-1}\mu_k \tag{6.123}$$

$$D_k = D_{k-1} + U_{k-1}B_k \tag{6.124}$$

Step 6. Compute q and p (for $k = 1, 2, \ldots, N-1$) using

$$q_k = \nu_k q_{k-1} + C_k \tag{6.125}$$

$$p_k = \nu_k p_{k-1}. \tag{6.126}$$

or using

$$p_k = U_{k+1}^{-1} \tag{6.127}$$

$$q_k = U_{k+1}^{-1} D_{k+1}. \tag{6.128}$$

Step 6a. If we want to update T (for $k = 1, 2, \ldots, N$), we also need to calculate

$$S_k = S_{k-1} + \text{diag}[\dot{Y}_k]p_{k-1}$$

$$J_k = J_{k-1} + \text{diag}[\dot{Y}_k]q_{k-1}.$$

Step 7. Calculate Q:

$$\begin{aligned} Q_k &= Q_{k-1} + \psi_k q_{k-1} Y_k^T \\ Q_0 &= Q_1 = 0. \end{aligned}$$

Step 8. For each column l, $\quad l = 1, 2, \ldots, n$, calculate

$$R_k^l = R_{k-1}^l + \psi_k p_{k-1} Y_k^l \tag{6.129}$$

$$R_0^l = \psi_0 Y_0^l. \tag{6.130}$$

Step 9. Loop back to step 1 until $k = N$. Then, go to step 10.

Step 10. Calculate Z_1:

$$Z_1 = U_N Z_N + D_N$$

Step 11. For each column l, $l = 1, 2, \ldots, n$, calculate

$$\left(\frac{\partial E}{\partial W^T}\right)_{\text{total}}^l = R_N^l Z_1 + Q_N^l.$$

Step 11a. If we want to update T, we also need to calculate

$$\left(\frac{\partial E}{\partial T}\right)_{\text{total}} = -S_N Z_1 - J_N. \tag{6.131}$$

Step 12. Calculate new weight W with gradient-decent method.

$$W = W_{\text{pre}} - \alpha \left(\frac{\partial E}{\partial W}\right)_{\text{total}}.$$

Calculate T with

$$T = T_{\text{pre}} - \beta \left(\frac{\partial E}{\partial T}\right)_{\text{total}}.$$

Where α and β are updating rates for weight and time factor respectively. Loop back to step 1 until the total "energy" is smaller than the desired value.

6.7.3 Simulation Examples

For comparison purpose, we try to solve the same XOR problem as used in [Pea89]. The network topology is shown in figure 6.5. The input layer neurons have external input I_1 and I_2. The goal is to choose a proper set of weights so that the output of the network is the XOR of the two input signals between time $t = 2$ and $t = 3$.

We define the "energy" to be $E = \sum_k \int_2^3 (y_0^{(k)} - d^{(k)})^2 dt$, where k ranges over the four cases, $d^{(k)}$ is the desired output for case k, y_0 is the state of the output neuron. The input signals I_1 and I_2 were set to be $[1,1]$, $[1,0]$, $[0,1]$ and $[0,0]$ respectively for each case. The sigmoid function is chosen as $\sigma(x) = (1 + e^{-x}) - 1$. The initial values Y of neurons are set to be $I + \sigma(0)$. Initial weights W are chosen randomly from $(-1,1)$.

As with Pearlmutter's case, when weight W and time factor T were updated simultaneously, T tended to go to 0 too quickly. In our simulations, we usually choose T's update factor to be very small. Figure 6.10 to figure 6.13 show the network

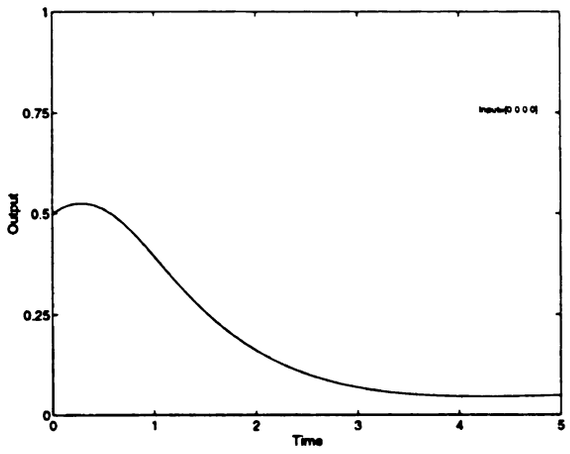


Figure 6.10. XOR network with input = [0, 0]

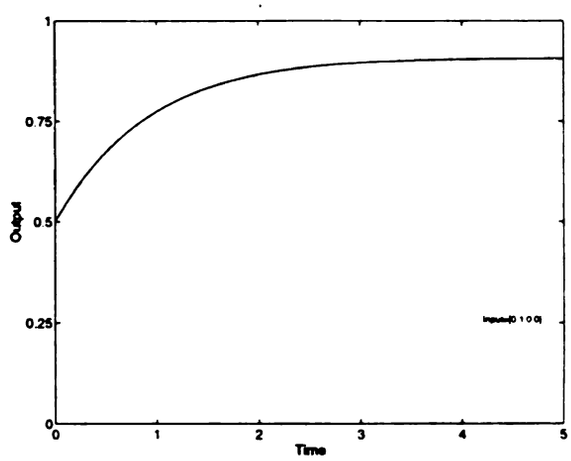


Figure 6.11. XOR network with input = [0, 1]

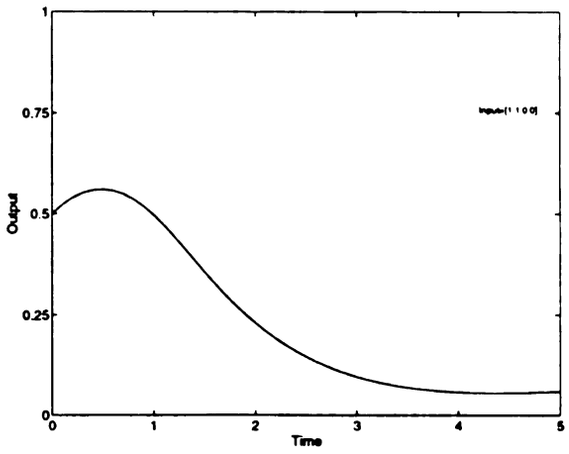


Figure 6.12. XOR network with input = [1, 1]

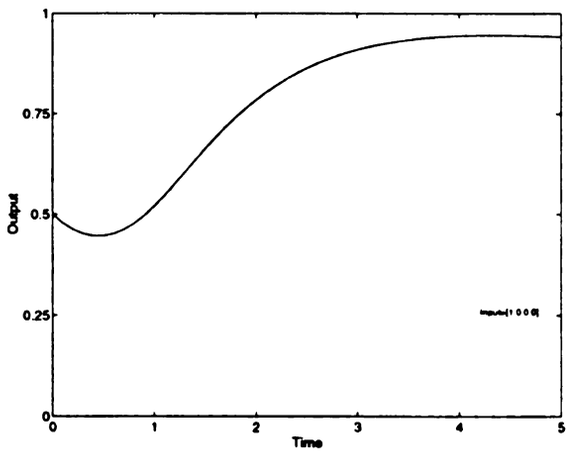


Figure 6.13. XOR network with input = [1, 0]

dynamics after it has been trained.

6.7.4 Comparison and Discussion

The number of computations in each time step can be estimated as follows: For each time step k, we need to calculate step 1 to 8. For step 1, the number of arithmetic computations is $O(2n^2)$. For step 2, the number of computation steps is $O(n)$. Step 3 and step 4 each needs $O(4n^2)$ computations. Step 5 needs $O(2n^3)$. Step 6 needs $O(2n^3)$. Step 6a needs $O(2n^2)$ number of computations. Step 7 needs $O(2n^2)$. Step 8 needs $O(2n^3)$. Thus the total number of computations is in the order of $O(4n^3)$.

In the simulation process, we need to store variables: U, D, q, p, S, J, Q, and R. The added storage is $O(n^3)$.

Compare the proposed algorithm with the original time-dependent recurrent back-propagation method, the proposed one avoids the backward integration, thus the memory storage request is reduced. The price is the increase of computations per time step.

Compared with other methods, This algorithm retains some of the advantages of the original time-dependent recurrent back-propagation method. The system used is very general and can be applied to many applications. The co-state Z has clear physical meaning. The boundary condition Z_T can be chosen according to the system constraint.

Toomarian and Barhen proposed a way to avoid the backward integration of the adjoint-function method [TB91]. They proposed similar way of finding Z_{t_0} using Z_T. However, the computation may be numerically unstable.

Table 6.1 shows the comparison of each method.

Table 6.1. Comparison of the algorithms

	TDRBP	RTRL	Green's function	proposed 1	proposed 2
Computations	$O(n^2)$	$O(n^4)$	$O(n^3)$	$O(n^4)$	$O(n^3)$
Backward integration	yes	no	no	no	no
On-line	no	yes	yes	yes	yes
Remarks	co-state can be set according to system constraint			co-state can be set according to system constraint	co-state can be set according to system constraint

CHAPTER 7

SUMMARY AND CONCLUSION

7.1 Summary

Artificial neural networks have developed very rapidly in the past decade. They have great potential of solving many problems which current digital computers cannot solve efficiently or cannot solve at all. One of the fundamental problem of designing artificial neural networks is to determine a set of weights such that the network can store or learn patterns or signals. When an input is applied to the network, the network can generate "desired" output. This behavior is referred to as "association". A simple association is to store static patterns and to retrieve the stored patterns when some clues are given to the network as input. Associative memory is such a kind of application. A more complex case is to store and to generate temporal patterns.

Numerous architectures and algorithms have been proposed so far for these kinds of tasks. Most of the proposed models have a pre-fixed structure. The weights are computed through on-line/off-line algorithms, which are usually computationally

expensive. In some cases, the weight matrix may not exist at all for a particular set of patterns.

In this thesis, the product-of-norm model of artificial neural network architecture is proposed. The new structure is based on gradient system theory using a special "energy" function. Specifically, the product of the norm of the errors is used as the "energy" function. The system dynamical equations are defined using gradient-descent method. The resulting system is a gradient system and is globally stable. By using the product of the norm of the errors as the "energy" function, the desired patterns are guaranteed to be the global energy minima of the system. Thus they can be retrieved from the network. A distinguished feature of this network is that the weights are fixed by design. The weights learning is eliminated. The trade-off is the increasing of the network complexity. The stability of the product-of-norm model was analyzed using *Lyapunov* theory. Both theoretic proof and simulation results were given in chapter 3 to show the ability of the network to store and retrieve arbitrary set of patterns. For the one neuron case, the locations of the unstable system equilibria were given too. Based on the analysis, another way of design one neuron networks with pre-set stable and unstable equilibria was proposed. The possibility of using the infinity norm in the product-of-norm model was also discussed.

In chapter 4, various techniques of generalizing of the "energy" function of the product-of-norm model were proposed. Ways to simplify the network structure were investigated too. It was shown that the system could be easily parallelized and modularized using the log form of the "energy" function. Using the log form "energy" function, the system can learn new patterns by simply adding modules to the existing network. The subgrouping method was proposed to localize the connections of the

network, so that the network could be simplified and the computations required could be reduced. Several ways of subgrouping were discussed, in general we found that the method of dividing the network into subgroups with overlapping neurons had better performance. chapter 4 showed that the product-of-norm system could be used to store and recognize digital image patterns by adding a penalty function to the original "energy" function. In particular, a method of dividing the images into small regions is discussed. The penalty function was calculated corresponding to each region. The resulting network was able to recognize both vertically and horizontally slightly shifted images.

Chapter 5 investigated possible ways to implement the product-of-norm model in hardware. Analog transistors biased in subthreshold region were proposed as basic building elements. Several building blocks were discussed and an example one-neuron/two-pattern network was shown.

Artificial neural networks used for association of temporal patterns were studied in chapter 6. Among training algorithms available for recurrent networks so far, time-dependent recurrent back-propagation is the best if on-line learning is not concerned. One of the limitations of this method is that it needs to integrate the co-state backward in time. Two algorithms were proposed to avoid the backward pass. The penalty is the increase of computation complexity. However, they are still among the simplest methods. Ways to set the initial values for the neurons and the final conditions for the co-state Z were analyzed also using optimal control theory.

7.2 Future Research Directions

The product-of-norm model is relatively complex. It requires high order feedbacks. Future work should be focused on further simplifying the network. Current hardware implementation example for product-of-norm model is not scalable for large networks. Hardware implementation of the product-or-norm model is another area which needs more investigation.

APPENDIX

APPENDIX A

THE PSPICE CODE AND SIMULATIONS FOR A ONE-NEURON/TWO-PATTERN CIRCUIT

In chapter 5, we presented some basic circuits structures to be used in implementing the product-of-norm model. In this chapter, we will list the PSpice code used for the simulations of the circuits. The following PSpice library file contains the definitions of some basic circuits: n-mirror, p-mirror, current-voltage multiplier and voltage-square circuits.

```
**************************************************************************
* PSPice ciruit library: neuron.lib
*
* This library contains sub-circuits which may be used to
* build product-of-norm model neural networks
*
```

```
* Jiansheng hou
* Sun 02-23-1997 1:49:28 pm
*
**************************************************************************

**************************************************************************
*
* MOSFET used in this library
*
**************************************************************************
.MODEL nfet NMOS LEVEL=2.00000
+LD=0.2500U TOX=401.000E-10
+NSUB=6.257042E+15 VTO=0.771327 KP=5.60600E-05
+GAMMA=0.53 PHI=0.60 UO=650.996 UEXP=0.153507
+UCRIT=41931.5 DELTA=0.472546 VMAX=67660.8 XJ=0.2500000U
+LAMBDA=3.089401E-02 NFS=4.834708E+12 NEFF=1
+NSS=1.000000E+10 TPG=1.00000 RSH=29.600 CGSO=3.229249E-10
+CGDO=3.229249E-10 CGBO=6.726429E-10 CJ=1.13650E-4
+MJ=0.68620 CJSW=5.31870E-10 MJSW=0.26510 PB=0.80000

.MODEL pfet PMOS LEVEL=2.00000
+LD=0.209610U TOX=401.000E-10
+NSUB=6.722092E+15 VTO=-0.78821 KP=2.23000E-05
+GAMMA=0.5486 PHI=0.60 UO=259 UEXP=0.182346
+UCRIT=12594.5 DELTA=0.844423 VMAX=41222.6 XJ=0.2500U
+LAMBDA=4.981694E-02 NFS=1.000E+11 NEFF=1.001
+NSS=1.000000E+10 TPG=-1.00000 RSH=90.700 CGSO=2.707532E-10
+CGDO=2.707532E-10 CGBO=5.839625E-10 CJ=2.513600E-4 MJ=0.547100
+CJSW=2.944700E-10 MJSW=0.318800 PB=0.80

**************************************************************************
*
* Sub-circuits
*
**************************************************************************

************************************************
*  Pmos current mirror
*
*               +----- Iin
*               | +--- Iout
*               | | +- Vdd
*               | | |
*               | | |
.subckt pmirror 1 2 3
```

```
m1 1 1 3 3 pfet L=12u W=4u
m2 2 1 3 3 pfet L=12u W=4u
.ends

.subckt pmirror2 1 2 3
m1 1 1 3 3 pfet L=7u W=4u
m2 2 1 3 3 pfet L=7u W=4u
.ends


.subckt pmirror3 1 2 3
m1 1 1 3 3 pfet L=24u W=2u
m2 2 1 3 3 pfet L=24u W=2u
.ends


************************************************
*
* N MOS mirror
*
* 1 ---Iin 2 -- Iout, 3 --- GND
.subckt nmirror 1 2 3
m1 1 1 0 0 nfet L=12u W=2u
m2 2 1 0 0 nfet L=12u W=2u
.ends


* 1 ---Iin, 2 --- Iout, 3 --- GND
.subckt nmirror2 1 2 3
m1 1 1 0 0 nfet L=12u W=8u
m2 2 1 0 0 nfet L=12u W=8u
.ends

* 1 ---Iin 2 --- Iout, 3 --- GND
.subckt nmirror3 1 2 3
m1 1 1 0 0 nfet L=24u W=1u
m2 2 1 0 0 nfet L=24u W=1u
.ends

* 1 ---Iin 2 ---  Iout, 3 --- GND
.subckt nmirror4 1 2 3
m1 1 1 0 0 nfet L=14u W=3u
m2 2 1 0 0 nfet L=14u W=3u
.ends
```

```
*1 ---Iin 2---Iout, 3---GND
.subckt n_cascode_mirror 1 2 3
m1 1 1 3 3 nfet L=10u W=14u
m2 2 1 4 3 nfet L=10u W=14u
m3 5 5 3 3 nfet L=6u W=14u
m4 4 5 3 3 nfet L=6u W=14u
.ends

* 1 ---Iin 2 --- Iout, 3 --- GND
.subckt n_nmirror 1 2 3
m1 1 1 3 3 nfet L=1200u W=1u
m2 2 1 3 3 nfet L=1200u W=1u
.ends


* 1 ---Iin 2 --- Iout, 3 --- GND
.subckt w_nmirror 1 2 3
m1 1 1 0 0 nfet L=2000u W=16u
m2 2 1 0 0 nfet L=2000u W=16u
.ends

************************************************
* Wide range IV multiplier
*
* 1 --- Iout, 2 --- Iin,
* 3 --- V1,   4.--- V2
* 5 ---Vdd,   6---gnd
.subckt iv_mul_wid 1 2 3 4 5 6
m1 8 3 7 0 nfet
m2 9 4 7 0 nfet
*x0 2 7 0 nmirror
x0 2 7 0 n_nmirror
x1 8 1 5  pmirror
x2 9 10 5  pmirror
x3 10 1 0 nmirror
.ends

************************************************
* IV multiplier
*
* 1 --- Iout, 2 --- Iin,
* 3 --- V1,   4 --- V2,
* 5 --- Vdd,  6 ---gnd
*
* 6 is not really used
```

```
*
.subckt iv_mul 1 2 3 4 5 6
m1 8 3 7 0 nfet  L=4u W=2u
m2 9 4 7 0 nfet  L=4u W=2u
x0 2 7 0  nmirror3
x1 8 1 5  pmirror2
x2 9 10 5 pmirror2
x3 10 1 0 nmirror
.ends


************************************************
* 1 --- Iout, 2 --- Iin,
* 3 --- V1,   4 ---V2,
* 5 ---Vdd, 6---gnd
*
* 6 is not really used
*
.subckt new_iv_mul 1 2 3 4 5 6
m1 8 3 7 0 nfet  L=4u W=2u
m2 9 4 7 0 nfet  L=4u W=2u
x0 2 7 0  nmirror2
x1 8 1 5  pmirror2
x2 9 10 5 pmirror2
x3 10 1 0 nmirror
.ends

************************************************
* 1 --- Iout, 2 --- Iin,
* 3 --- V1,   4 ---V2,
* 5 ---Vdd,    6---gnd
*
* 6 is not really used
*
.subckt new_iv_mul2 1 2 3 4 5 6
m1 8 3 7 0 nfet  L=4u W=2u
m2 9 4 7 0 nfet  L=4u W=2u
x0 2 7 0  nmirror2
x1 8 1 5  pmirror2
x2 9 10 5 pmirror2
x3 10 1 0 n_cascode_mirror
.ends


************************************************
```

```
* Negative IV multiplier
*
* 10 --- Iout, 2 --- Iin,
* 3 ---V1,     4 ---V2,
* 5 ---Vdd,    6 ---gnd
*
* 6 is not really used
*
.subckt neg_iv_mul 10 2 3 4 5 6
m1 8 3 7 0 nfet  L=4u W=2u
m2 9 4 7 0 nfet  L=4u W=2u
x0 2 7 0  nmirror2
x1 8 1 5  pmirror2
x2 9 10 5 pmirror2
x3 1 10 0 n_cascode_mirror
.ends

************************************************
* Negative IV multiplier
*
* 10 --- Iout, 2 --- Iin,
* 3 ---V1,     4 ---V2,
* 5 ---Vdd,    6 ---gnd
*
* 6 is not really used
*
.subckt neg_iv_mul2 10 2 3 4 5 6
m1 8 3 7 0 nfet  L=4u W=2u
m2 9 4 7 0 nfet  L=4u W=2u
x0 2 7 0  nmirror2
x1 8 1 5  pmirror2
x2 9 10 5 pmirror2
x3 1 10 0 nmirror3
.ends

************************************************
* Test IV multiplier
*
* 1 --- Iout, 2 --- Iin,
* 3 --- V1,   4 ---V2,
* 5 ---Vdd,   6 ---gnd
.subckt iv_mul_test 1 2 3 4 5 6
m1 8 3 7 0 nfet
m2 1 4 7 0 nfet
x0 2 7 0  nmirror
```

```
x1 8 1 5  pmirror
*x2 9 10 5 pmirror
*x3 10 1 0 nmirror
.ends

************************************************
* N MOS pair
*
* 1---V1, 2---v2, 3---Iout, 4---Iin1, 5---Iin2
.subckt n_pair 1 2 3 4 5 6
m1 4 1 3 0 nfet  L=4u W=3u
m2 5 2 3 0 nfet  L=4u W=3u
.ends


*********************************************
* gilbert multiplier
* 1-v1, 2-v2, 3-v3, 4-v4, 5-Ib, 6-Iout, 7-Vdd, 8-gnd
*
.subckt gilbert_mul 1 2 3 4 5 6 7 8
x0 3  4  10  12 6  8 n_pair
x1 3  4  11  6 12 8 n_pair
x2 1  2  9   10 11 8 n_pair
x3 12 6  7   pmirror
m1 9  5  8   8  nfet
.ends

*************************************************
* Voltage square circuit
*
*1---v1, 2---v2, 3---Iout, 4---Ib, 8---Vdd, 9---gnd
.subckt v_square 1 2 3 4 8 9
x0 1 2 10 5 6 9 n_pair
x1 1 2 11 6 5 9 n_pair
x2 1 2 12 10 11 9 n_pair
x3 5 3 8  pmirror3
x4 6 7 8  pmirror
x5 7 3 9  nmirror2
m1 12 4 9 9 nfet L=4u W=3u
.ends

*****************************************************************
* another voltage square circuit
*
*1---v1, 2---v2, 3---Iout1, 7--Iout2,  4---Ib, 8---Vdd, 9---gnd
```

```
*9 is not really used
*
.subckt new_v_square 1 2 3 7 4 8 9
x0 1 2 10 5 6 0 n_pair
x1 1 2 11 6 5 0 n_pair
x2 1 2 12 10 11 0 n_pair
x3 5 3 8  pmirror3
x4 6 7 8  pmirror3
*x5 7 3 0  nmirror2
m1 12 4 0  0 nfet L=4u W=3u
.ends

*****************************************************************
* v_sqaure circuit with two current outputs
*1---v1, 2---v2, 3---Iout1, 7---Iout2, 4---Ib, 8---Vdd, 9---gnd
.subckt v_square_2out 1 2 3 7 4 8 9
x0 1 2 10 5 6 0 n_pair
x1 1 2 11 6 5 0 n_pair
x2 1 2 12 10 11 0 n_pair
x3 5 3 8  pmirror
x4 6 7 8  pmirror
*x5 7 3 0  nmirror
m1 12 4 0  0 nfet  L=4u W=3u
*m1 is the bias mosfet
.ends

*********************************************
* test voltage square circuit
*
*1---v1, 2---v2, 4---Ib, 8---Vdd, 9---gnd
.subckt v_squaretest 1 2 3 4 8 9
x0 1 2 10 5 6 0 n_pair
x1 1 2 11 6 5 0 n_pair
x2 1 2 12 10 11 0 n_pair
x3 5 3 8 pmirror
x4 6 7 8 pmirror
x5 7 3 0 nmirror
m1 12 4 0  0 nfet
.ends

*********************************************************
* Another test voltage square circuit
*
*1---v1, 2---v2, 3---Iout, 4---Ib, 8---Vdd, 9---gnd
.subckt v_squaretest2 1 2 3 4 8 9
```

```
x0 1 2 10 5 6 0 n_pair
x1 1 2 11 6 5 0 n_pair
x2 1 2 12 10 11 0 n_pair
x3 6 3 8  pmirror
x4 5 7 8  pmirror
x5 7 3 0 w_nmirror
mtest1 7 3 0 0 nfet L=12u W=4u
mtest2 7 3 0 0 nfet L=12u W=4u
m1 12 4 0  0 nfet
.ends
```

The following is the PSPice file for a one-neuron/two-pattern circuit.

```
1n2p circuit
*
*  This circuit is a one-neuron/two-pattern  circuit
*  of the product-of-norm neural network model
*

***********************************************
* The structure
***********************************************

x0 6 4 80 81 9 1 0 new_v_square
x1 61 82 5 6 1 0 new_iv_mul
x2 62 83 5 6 1 0 neg_iv_mul2
r1 80 82 0.00001
r2 81 83 0.00001
r5 61 6  0.00001
r6 62 6  0.00001

x3 6 5 70 71 9 1 0 new_v_square
x4 63 72 4 6 1 0 new_iv_mul
x5 64 73 4 6 1 0 neg_iv_mul2
r3 70 72 0.00001
r4 71 73 0.00001
r7 63 6  0.00001
r8 64 6  0.00001


***********************************************
* The settings
***********************************************
```

```
vd 1 0 5v
vc 6 0 1v
vb 9 0 0.5v
v_star1 4 0 0.6v
v_star2 5 0 0.4v

************************************************
* The subcircuits
************************************************

**********************************
* N MOS current mirror
* 1 ---Iin 2 --- Iout, 3 --- GND
.subckt nmirror8 1 2 3
m1 1 1 0 0 nfet L=20u W=2u
m2 2 1 0 0 nfet L=20u W=2u
.ends

**********************************
* N MOS current mirror
* 1 ---Iin 2 --- Iout, 3 --- GND
.subckt nmirror9 1 2 3
m1 1 1 0 0 nfet L=7u W=5u
m2 2 1 0 0 nfet L=7u W=5u
.ends

**********************************
* Negative IV multiplier
*
.subckt neg_iv_mul 10 2 3 4 5 6
m1 8 3 7 0 nfet  L=4u W=2u
m2 9 4 7 0 nfet  L=4u W=2u
x0 2 7 0  nmirror2
x1 8 1 5  pmirror2
x2 9 10 5 pmirror2
x3 1 10 0 n_cascode_mirror
.ends

************************************************
* Another negative IV multiplier
*
* 10 --- Iout, 2 --- Iin,
* 3 --- V1,    4 ---V2,
* 5 ---Vdd,    6 ---gnd
*
```

```
* 6 is not really used
*
.subckt neg_iv_mul2 10 2 3 4 5 6
m1 8 3 7 0 nfet  L=4u W=2u
m2 9 4 7 0 nfet  L=4u W=2u
x0 2 7 0  nmirror2
x1 8 1 5  pmirror2
x2 9 10 5 pmirror2
x3 1 10 0 nmirror9
.ends

************************************************
* IV multiplier
*
* 1 --- Iout, 2 --- Iin,
* 3 --- V1,   4 ---V2,
* 5 ---Vdd,   6 ---gnd
*
* 6 is not really used
*
.subckt new_iv_mul 1 2 3 4 5 6
m1 8 3 7 0 nfet  L=4u W=2u
m2 9 4 7 0 nfet  L=4u W=2u
x0 2 7 0  nmirror2
x1 8 1 5  pmirror2
x2 9 10 5 pmirror2
x3 10 1 0 nmirror8
.ends

.lib c:/pspice/hou/neuron.lib

.TEMP 25
.dc vc 0 1.0 0.01
.probe
.end
```

Setting $v^{*1} = 0.4V$ and $v^{*2} = 0.7V$, we plot the response of the circuit to different initial voltages in Figure A.1. Figure A.2 shows the simulations of the same circuit with $v^{*1} = 0.3V$ and $v^{*2} = 0.75V$. As can be seen from the figures, the circuit stored two stable points, which are very close to the desired points. Because the circuit function we derived can only work in certain range, the resulting circuit cannot store arbitrary data.

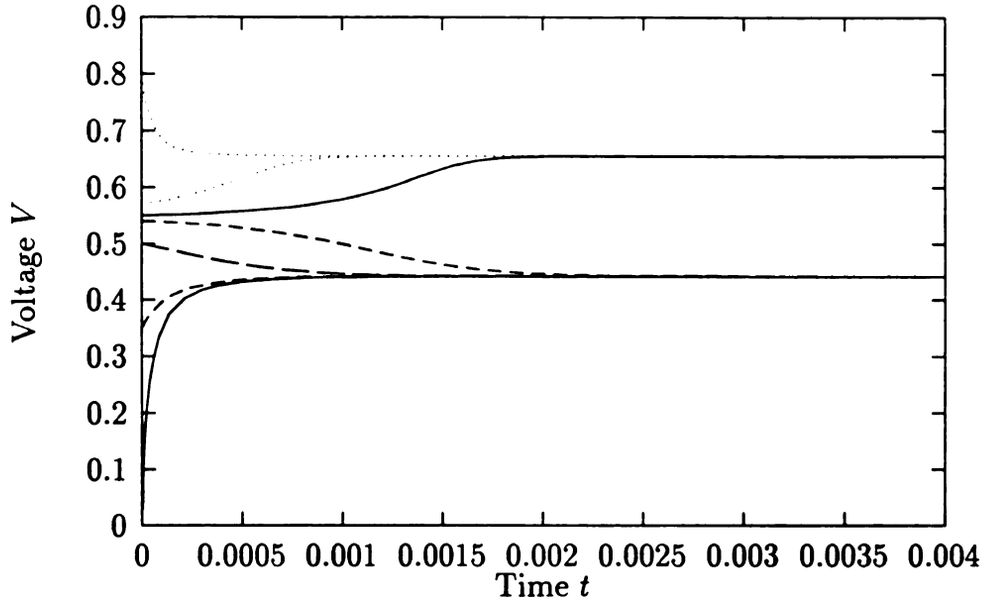


Figure A.1. The response of the one-neuron/two-pattern circuits to different initial voltages. $v^{*1} = 0.4V$ and $v^{*2} = 0.7V$.

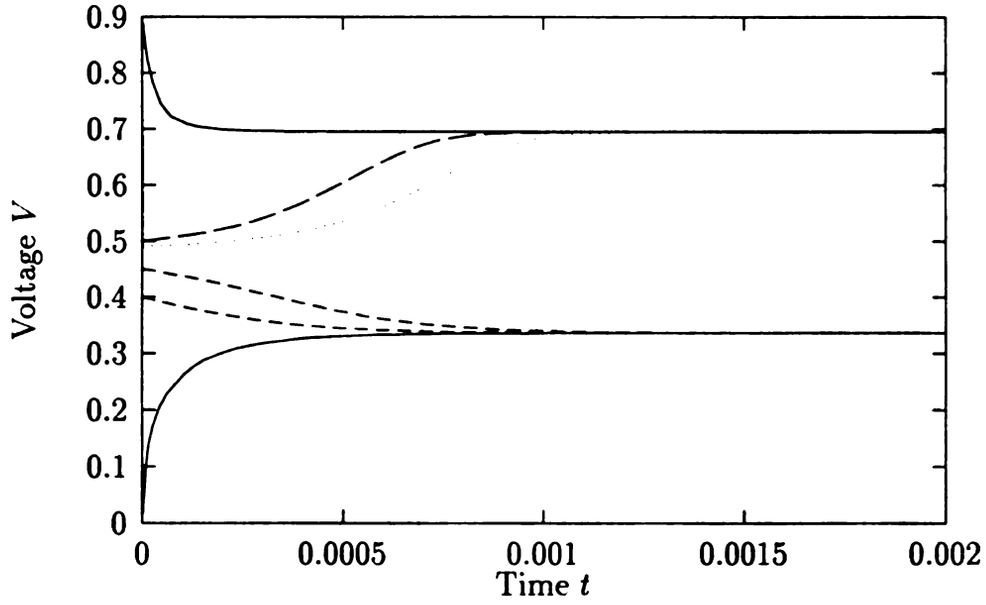


Figure A.2. The response of the One-neuron/two-pattern circuits to different initial voltages. $v^{*1} = 0.3V$ and $v^{*2} = 0.75V$

BIBLIOGRAPHY

BIBLIOGRAPHY

[Bar91] Valmir C. Barbosa. Learning in analog hopfield networks. In *Proceedings of IJCNN 1991*, volume ii, pages 183–186, Seattle, July 1991.

[BcH75] Arthur E. Bryson, Jr. and Yu chi Ho. *Applied Optical Control Optimization, estimation, and control.* Ohemisphere Publishing Corporation, Washington, D.C., 1975.

[HKP90] John Hertz, Anders Krogh, and Richard G. Palmer. *Introduction To The Theory of Neural Computation.* Addison-Wesley Publishing Company, 1990.

[Hop82] John Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proceedings of the National Academy of Sciences*, volume usa 79, pages 2554–2558, 1982.

[Hop84] John J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. In *Proceedings of the National Academy of Sciences*, volume usa 81, pages 3088–3092, 1984.

[HS74] Morris W. Hirsch and Stephen Smale. *Differential Equations, Dynamical Systems, and Linear Algebra.* Academic Press, Inc, 1974.

[Kha91] Hassan K. Khalil. *Nonlinear Systems.* Macmillan Publishing Company, 1991.

[Kos88] Bart Kosko. Bidirectional associative memories. *IEEE Trans. On Systems, Man, and Cybernetics*, 18(1):49–60, Jan/Feb 1988.

[LS95] Frank L. Lewis and Vassilis L. Syrmos. *Optical Control.* John Wiley & Sons, Inc., 1995.

[Mea89] Carver Mead. *Analog VLSI and Neural Systems.* Addision-wesley Publishing Company, 1989.

[Moz89] M. C. Mozer. A focused back-propagation algorithm for temporal pattern recognition. *Complex Systems*, 3:349–381, 1989.

[Pea89] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. In *International Joint Conference on Neural Networks, Washington 1989*, volume II, pages 365–372, Seattle, Washington, 1989.

[Pin87] Fernando J. Pineda. Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 59:2229–2232, 1987.

[Pin88] Fernando J. Pineda. Dynamics and architecture for neural computations. *Journal of Complexity*, 4:216–245, 1988.

[Pin89] Fernando J. Pineda. Recurrent back-propagation and the dynamical approach to adaptive neural computation. *Neural Computation*, 1:161–172, 1989.

[RHW86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the microstructures of cognition*, pages 318–362. MIT Press, Cambridge, 1986.

[SB91] Fathi M. A. Salam and Shi Bai. A new feedback neural network with supervised learning. *IEEE Trans. On Neural Networks*, 1(1):170–173, Jan 1991.

[SCL91] Guo-Zheng Sun, Hsing-Hen Chen, and Yee-Chun Lee. A fast on-line learning algorithm for recurrent neural networks. In *Proceedings of IJCNN 1991*, volume ii, pages 13–18, Seattle, July 1991.

[SCL92] Guo-Zheng Sun, Hsing-Hen Chen, and Yee-Chun Lee. Greens' function method for fast on-line learning algorithms of recurrent neural networks. In *Advance in Neural Information Processing Systems*, volume 4, pages 333–340. Morgan Kaufmann Publishing Inc., 1992.

[SWC91] Fathi M. A. Salam, Yiwen Wang, and Myung-Ryul Choi. On the analysis of dynamic feedback neural nets. *IEEE Trans. On Circuits and Systems*, 38(2), Feb 1991.

[TB91] N. Toomarian and J. Barhen. Adjoint-functions and temporal learning algorithms in neural networks. In *Advance in Neural Information Processing Systems*, volume 3, pages 113–120. Morgan Kaufmann Publishing Inc., 1991.

[TH87] D. W. Tank and J. J. Hopfield. Neural computation by time compression. In *Proceedings of the National Academy of Sciences USA*, volume 84, pages 1896–1900, 1987.

[THL93] H. Chris Tseng, Victor H. Hwang, and Ling Lu. A fast supervised learning scheme for recurrent neural networks with application to associative memory design. In *Proceedings of 1993 IEEE International Conference on Neural Netw orks*, volume ii, pages 789–793. IEEE Neural Network Concel, April 1993.

[VP92] Bert De Vries and Jose C. Principe. The gamma model – a new neural model for temporal processing. *Neural Networks*, 5:565–576, 1992.

[Wai89] A. Waibel. Modular construction of time-delay neural networks for speech recognition. *Neural Computation*, 1:39–46, 1989.

[WCJ91] Yeou-Fang Wang, Jose B. Cruz Jr., and J. H. Mulligan, Jr. Guaranteed recall of all training pairs for bidirectional associative memory. *IEEE Trans. On Neural Networks*, 2(6):559–567, Nov 1991.

[WCM89] Yeou-Fang Wang, Jose B. Cruz, Jr., and J. H. Mulligan, Jr. An enhanced bidirectional associative memory. In *Proceedings of IJCNN 1989*, volume i, pages 105–110, June 1989.

[WCM90] Yeou-Fang Wang, Jose B. Cruz Jr., and J. H. Mulligan Jr. Two coding strategies for bidirectional associative memory. *IEEE Trans. On Neural Networks*, 1(1):81–92, Mar 1990.

[WHH+89] A. Waibel, T. Hanazawa, G. Hinton, K Shikano, and K Lang. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37:328–339, 1989.

[WZ89a] R. J. Williams and D. Zipser. Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, 1:87–111, 1989.

[WZ89b] R. J. Williams and D. Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280, 1989.