



3 1293 00902 1324

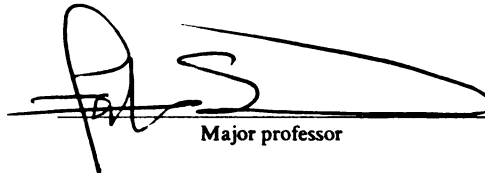
This is to certify that the
dissertation entitled
Supervised Learning of
Feedforward Artificial Neural Networks
Using Different Energy Functions

presented by

Maqbool Ahmad

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Electrical Engineering



Major professor

Date 11/15/91

LIBRARY
Michigan State
University

PLACE IN RETURN BOX to remove this checkout from your record.
TO AVOID FINES return on or before date due.

DATE DUE	DATE DUE	DATE DUE
MAY 01 2016 MAY 13 2015	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

MSU is An Affirmative Action/Equal Opportunity Institution

c:\clic\datedue.pm3-p.

**SUPERVISED LEARNING OF
FEEDFORWARD ARTIFICIAL NEURAL NETWORKS
USING DIFFERENT ENERGY FUNCTIONS**

By

Maqbool Ahmad

A DISSERTATION

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

DOCTOR OF PHILOSOPHY

Department of Electrical Engineering

1991

imp
tow
rule
con
Exp
the:
rule
con
(sur

law
gen
"ski
que
the
tion

ABSTRACT

SUPERVISED LEARNING OF FEEDFORWARD ARTIFICIAL NEURAL NETWORKS USING DIFFERENT ENERGY FUNCTIONS

By

Maqbool Ahmad

A framework, supported by analysis and computer simulations, is developed to improve the supervised learning of feedforward artificial neural networks with a view towards their software implementation. The framework encompasses various learning rules derived from carefully selected energy functions in order to speed up and enable convergence to acceptable weights. We describe the Cauchy, the Polynomial and the Exponential energy functions and characterize their properties and the performance of their associated learning rules. We use continuous-time gradient update law learning rules due to particular advantages over their discrete-time counterparts. We perform comparative investigations among these learning rules *vis-a-vis* the usual Gaussian (sum-of-the-squared) energy function.

The choice of an energy function specifies the back-propagated error in an update law. It also determines the learning performance in the sense of the speed of convergence. Carefully selected energy functions can in principle enable the neural system to "skip" undesirable minima; thus permitting convergence to acceptable minima. Consequently, a carefully selected energy function can render a learning process that enables the neural network to execute good classification/association and improved generalization capability.

C

w

en

le

th

co

w

te

.

th

en

a

tic

on

.

ou

ex

mi

to

dy

lea

.

eva

use

sep

clas

req

reco

The Gaussian energy function has been used frequently. In contrast, we use the Cauchy energy function, derived from the Cauchy distribution, in the gradient descent weight update law. We provide a statistical rational for the selection of the Cauchy energy. Analytical methods and computer simulations show that it could speed up the learning dynamics and would improve performance in a certain sense. We also show that all the stable equilibria of the learning dynamics based on the Cauchy energy are contained in the learning dynamics based on the Gaussian energy. Therefore, the network would still converge to one of the equilibrium points of the Gaussian-based system.

We denote a polynomial energy function of order r as a linear combination of all the L_p norms ($1 \leq p \leq r$). We show analytically and using computer simulations that employing the Polynomial energy function would improve the learning performance in a certain sense. The Polynomial energy function of order 2 does not introduce additional equilibria to the Gaussian system. As a consequence the network converges to one of the minima of the Gaussian system.

In order to achieve good classification and generalization results, the network ought to converge to a "good" or desirable minimum of the neural system. An exponential energy function is introduced to enable convergence to one of the desirable minima. A bound is established on the parameters of the Exponential energy function to achieve faster convergence to a desirable minimum. We also show that the learning dynamics based on the Exponential energy function possess the same minima as the learning dynamics based on the original Gaussian energy function.

The performance of each of the energy functions addressed in this work, is evaluated using the XOR and the English character recognition problems. A highly user-oriented character recognition network is developed for the evaluation, with two separate modules: a feature extraction network to be trained on a large data base and a classification network to be subsequently trained by the user according to his/her requirements. An error correction capability is incorporated to enhance the character recognition performance.

To
my mother, who always prayed for my success
but could not live to see this day.

is
hun
of
this
own

Nat
(RE

Sala
studi
Jr. fo

ence,
ings
for th

ACKNOWLEDGMENTS

Praise be to Allah *Subhanahu-wa-Taala*, who created this universe and all that is contained in it; mankind being the most superior due to its thinking faculties. The human brain inspires from its own structure numerous ideas and opens up many areas of pursuit and excellence in research to benefit the whole humanity. The more I pursue this research, the more He reveals His greatness to me. What ever I am today, I wholly owe it to Him.

I am grateful for the financial support from the Government of Pakistan, the National Science Foundation and the State of Michigan Research Excellence Fund (REF). Without this support, this research effort would not have been possible.

I would like to express my sincere gratitude to my major advisor, Prof. F. M. A. Salam, for his inspiration, guidance and support throughout the years of my graduate studies. Thanks to Dean F. C. Hoppensteadt, Prof. H. K. Khalil and Prof. J. R. Deller, Jr. for their valuable comments and suggestions.

I wish to express my sincere thanks and appreciation to my parents for their patience, continued prayers, support and encouragement. I deeply owe my heartiest feelings of gratitude to my wife, Ghazala and my children, Muhammad, Hira and Saad for their love, understanding and support.

List

List

List

Chap

1

1

1.

Chap

2.

2.2

2.3

2.4

2.5

TABLE OF CONTENTS

List of Tables	vii
List of Figures	viii
List of Symbols	x
 Chapter 1 Introduction	 1
1.1 Neurobiology and artificial neural networks	2
1.1.1 A biological inspiration	3
1.1.2 Topologies of Artificial Neural Networks	5
1.2 Why new energy functions?	5
1.3 Contribution of the thesis	7
 Chapter 2 Background	 8
2.1 The feedforward neuron model	9
2.2 The feedforward neural network connection topology	12
2.3 The back-propagation architecture	14
2.4 Supervised training of the back-propagation network	16
2.5 The weight update laws and the energy functions	17

Ch

Cha

4

4

2.6 The Gaussian energy function	18
2.6.1 The learning dynamics due to the Gaussian energy function	20
2.6.2 Software implementation	22
2.7 The problem description	23
2.8 The objectives and the outline of the thesis	24
 Chapter 3 The Cauchy energy function	 26
3.1 Motivation	26
3.2 The learning dynamics	29
3.3 Analytical comparison with the Gaussian energy function	31
3.3.1 A criterion for the learning performance	31
3.3.2 The learning dynamics for a single pattern	32
3.3.3 The learning dynamics for multiple patterns	34
3.4 Software implementation of the learning dynamics	36
3.5 Simulation example: the XOR problem	37
3.6 Discussion	40
 Chapter 4 The Polynomial energy function	 42
4.1 Motivation	42
4.2 The learning dynamics	43
4.3 Analytical comparison with the Gaussian energy function	45
4.3.1 The learning dynamics for a single pattern	46
4.3.2 The learning dynamics for multiple patterns	47
4.4 Software implementation of the learning dynamics	48
4.5 Simulation example: the XOR problem	49

4.6 Discussion	50
Chapter 5 The Exponential energy function	58
5.1 Motivation	58
5.2 The learning dynamics	59
5.3 Analytical comparison with the simple energy function	60
5.3.1 Comparison of the learning dynamics	60
5.4 Software implementation of the learning dynamics	63
5.5 Simulation example: the XOR problem	64
5.6 Discussion	71
Chapter 6 The pattern/character recognition problem	73
6.1 Problem description	73
6.2 Data base	74
6.3 The network design	74
6.3.1 The feature extraction or preprocessing network	76
6.3.2 The classification network	78
6.4 Learning evaluation with different energy functions	80
6.4.1 Training of the feature extraction network	81
6.4.2 Training of the classification network	84
6.5 Discussion	87
Chapter 7 Summary and Conclusion	89
7.1 Summary	89
7.2 Conclusion	91

Appendix A Entropy of the Gaussian and the Cauchy distribution	92
Appendix B Proof of Lemma 3.1	94
Appendix C Proof of Theorem 3.1	95
Appendix D Proof of Lemma 3.2	97
Appendix E Proof of Theorem 3.2	98
Appendix F Proof of Proposition 3.1	100
Appendix G Proof of Theorem 4.1	102
Appendix H Proof of Theorem 4.2	104
Appendix I Proof of Proposition 4.1	106
Appendix J Proof of Proposition 5.1	108
Appendix K List of the fonts used for the pattern recognition problem	109
Bibliography	111

LIST OF TABLES

2.1 The learning algorithm	22
3.1 Training results comparing the Gaussian and the Cauchy learning	39
4.1 Training results comparing the Gaussian, the Cauchy and the Polynomial learning	52
5.1 Training results comparing the Exponential ($\kappa = 1$) of the Gaussian, the Cauchy and the Polynomial learning	69
5.2 Training results comparing the Exponential ($\kappa = 1.8$) of the Gaussian, the Cauchy and the Polynomial learning	70
6.1 Error(s) and the number of steps at convergence of the learning algorithms used for the training of the feature extraction network	82
6.2 Character recognition using the feature extraction network trained with the Gaussian learning	83
6.3 Character recognition using the feature extraction network trained with the Polynomial learning	83
6.4 Character recognition using the feature extraction network trained with the Exponential of the Gaussian learning	84
6.5 Error(s) and the number of computer steps at convergence of the learning dynamics for the training of the classification network	85
6.6 Character recognition when the feature extraction and the classification networks are trained with set 1	86
6.7 Character recognition when the feature extraction and the classification networks are trained with set 1 and set 2, respectively	86

1.1

2.1

2.2

2.3

2.4

2.5

3.1

3.2

3.3

4.1

4.2

4.3

4.4

4.5

4.6

5.1

5.2

5.3

LIST OF FIGURES

1.1	Simplified structure of a real neuron	3
2.1	Neuron activation functions	10
2.2	The sigmoidal activation function with threshold and shape modification	11
2.3	The feedforward neuron model	12
2.4	A structure of a multilayer FFANN	13
2.5	The back-propagation architecture of a FFANN	14
3.1	Random error distribution	28
3.2	The network used to solve the XOR problem	37
3.3	Comparison of the Gaussian and the Cauchy learning	38
4.1	Comparison of the Gaussian, the Cauchy and the Polynomial learning	51
4.2	Trajectory showing the increase in L_1 norm while L_2 norm is decreasing	54
4.3	Trajectory showing the increase in L_2 norm while L_1 norm is decreasing	54
4.4	Trajectory showing both the L_1 and the L_2 norm decreasing	55
4.5	Energy level diagram of the Polynomial energy function	55
4.6	Training results of the XOR problem, trained with the Modified Polynomial energy function	56
5.1	Graph representing the relationship (5.7)	62
5.2	Comparison of the Exponential ($\kappa = 1,1.8$) and the Gaussian learning	66
5.3	Comparison of the Exponential ($\kappa = 1,1.8$) and the Cauchy learning	67

5.4

6.1

6.2

6.3

6.4

5.4 Comparison of the Exponential ($\kappa = 1, 1.8$) and the Polynomial learning	68
6.1 Typical examples of the fonts	75
6.2 Block diagram of the FFANN used for the character recognition problem	76
6.3 Structure of the FFANN used for feature extraction	77
6.4 Targets used in the feature extraction network	79

$u :$

$v :$

$\theta :$

net

$f(.)$

$x :$

$y :$

$t :$

$v :$

$g :$

$E :$

$F :$

$\varepsilon :$

$\xi :$

$\delta :$

$w :$

$\omega :$

$p :$

$\rho :$

$IN :$

$H_j,$

LIST OF SYMBOLS

- u : neuron's input
 v : neuron's output
 θ : neuron's threshold
 net : neuron's input excluding threshold
 $f(.)$: neuron's input/output function
 x : input to a neural network
 y : output of a neural network
 t : desired output of a neural network
 v : neural network's actual mapping
 g : neural network's desired mapping
 E : energy function
 F : function of an energy function
 ε : output error of a neural network
 ξ : error criterion
 δ : error signal to be propagated back
 w : weights
 ω : weight change stopping criterion
 p : index for patterns
 ρ : dummy index for patterns
 IN : index of input layer
 H_j, H_k, H_l, H_m : index of hidden layers

O : index of output layer
 i : index of input layer neurons
 j, k, l, m : index of hidden layer neurons
 n : index of output layer neurons
 η : dummy index of output layer neurons
 γ : learning rate
 χ : skipping factor
 $d(.)$: density function
 $L(.)$: likelihood function
 $H(.)$: entropy function
 σ : standard deviation of the Gaussian distribution
 λ : parameter of the Cauchy distribution
 a, b : parameters of the Polynomial energy function
 μ, κ : parameters of the Exponential energy function
 Δ : change
 ∇ : gradient
 ∂ : partial derivative
 α, β : general constants

hav
ther
per
tion
sequ
field

mod
by t
whic
have
the b
neur

ing e
ogy t
adapt
netwo
ing an

N
like da

CHAPTER 1

INTRODUCTION

Von Neuman machines have proven to be very useful in digital computation and have contributed significantly in the development of other spheres of life. However, there are various tasks where conventional digital computers can not compete with the performance of humans. Vision, classification, association, speech and pattern recognition are only a few examples. It is also recognized that no matter how fast digital sequential computers would become, they are unlikely to outperform humans in these fields.

Observations in neurobiology and psychology have inspired many researchers to model and design systems, which could emulate these human capabilities. Motivated by the highly interconnected networks of nerve cells referred to as *neural networks*, which form the basis of the human nervous system and the brain, many researchers have proposed various architectures that model the information processing aspect of the brain and the nervous system. These architectures are often referred to as *artificial neural networks*.

Artificial neural networks are systems made of a large number of simple processing elements operating in parallel whose function is determined primarily by the topology that connects them. These systems are capable of high level functions such as adaptation and learning, and/or lower level functions such as data processing. These networks are inspired both by neurobiology and various mathematical theories of learning and information processing.

Neurobiological research in recent years has suggested that many types of map-like data representation are used in the brain [1],[2]. Some of these are now being used

in neural network models [1],[3],[4]. Other work has shown that complex type of temporal processing and non-linear operations appear to be performed by neurons. Complex neuron models are being used in some neural network models [5]-[10].

Neuron models for the purpose of studying the functioning of the brain are quite involved and complex. These models are used as paradigms to test hypotheses and theories on the functioning of the brain. These models fall into the category of *reverse-engineering* and are of primary interest to neurobiologists, psychologists and physiologists. On the other hand, neuron models for the use of engineering technology fall under the category of *neuro-engineering*. These models are used to implement the principles of operation to produce artificial neural networks capable of emulating some of the aspects of the human brain and thus aim at improving the artificial processing of data or information.

1.1 Neurobiology and Artificial Neural Networks

There are two basic types of neural network models: (1) the neurobiological models that are intended as computational models of the biological nervous system and (2) the biologically inspired models that are intended as computational devices which are referred simply as neural network models.

The main objective of the artificial neural network research is to design new architectures and algorithms that can solve problems which are difficult to solve by the conventional methods but are easy for intelligent biological organisms. For the inspirational development of the artificial neural networks, we must understand the essence of neurobiological architecture of real neurons and their connectivity. A simplified description and a modest analogy of the real neuron is presented in the following subsections.

1.1.1 A Biological Inspiration

The human cerebral cortex is comprised of approximately 100 billion (10^{11}) neurons with each having roughly 1,000 dendrites that form some 100,000 billion (10^{14}) synapses. Given that this system operates at about 100 Hz, it functions at some 10,000,000 billion (10^{16}) interconnections per seconds. The cortex weighs approximately three pounds, covers about 0.15 square meters, and is about two millimeters thick. This is clearly beyond anything which can be reconstructed or modeled; but it is, perhaps, possible to understand how the brain performs information processing, and we hope that this understanding can be modeled and ultimately implemented in hardware.

The basic anatomical unit of the nervous system is the nerve cell or the neuron [11]-[13]. Every neuron is a tiny information processing system with thousands of connections through which signals are sent and received. No two of these cells are exactly alike, but most share similar features. Each neuron has an inside and outside separated by a plasma membrane. The inside of the cell and the fluid surrounding the cell have different concentrations of charged ions which create a potential difference. The major parts of a typical neuron are shown in Figure 1.1.

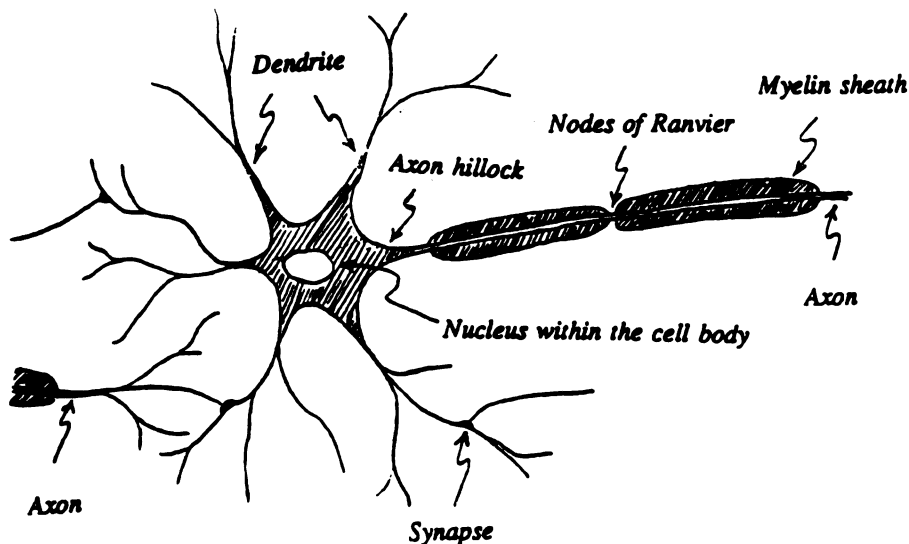


Figure 1.1: Simplified structure of a biological neuron.

•

•

•

•

• S

a

e

co

in

re

tr

ne

rec

cha

in

The major parts of a biological neuron are:

- **Cell Body** : It is composed of a *cytoplasm* and contains the *cell nucleus*, *nucleolus*, *mitochondria* and *ribosomes*. The nucleus synthesizes the information which controls the activity of the neuron.
- **Dendrites** : These are the outgrowths of the cell body through which it picks up signals from other cells.
- **Axon Hillock** : It acts as a nonlinear threshold device that produces a rapid voltage increase or decrease, called an *action potential*.
- **Axon** : It is the longest process of a cell body, which carries messages away from it. It is very resistive and carries impulses to and from the cell body. Axons are enclosed in a thick sheet made of the fatty substance *myelin*, which is a special low-dielectric-constant material. The myelinated sheath has regular indentations along its length called the *nodes of Ranvier*, which restore the pulses periodically along the axon.
- **Synapses** : These are contacts on a neuron which are the termination point for axons from other neurons. The synapse forms a gap between the slightly enlarged ending, or *synaptic knob*, of an axon terminal and the membrane of the next cell. At the synapse the electrical impulses that travel through the axon convert into chemical signals, then back to electrical impulses. When a nerve impulse reaches the synaptic knob, it causes the release of a chemical called a *neurotransmitter*, from minute vesicles in which the substance is formed or stored. The neurotransmitters diffuse across the gap and may cause *depolarization* of the receptor cell membrane, thus starting an *excitatory* signal. If the membrane is charged more negatively than its resting voltage, it is said to be *hyperpolarized*, in which case the signal is *inhibitory*.

1.1.2 Topo

Artific

form, the
amplifier, th
resistors. T
represented
amplifier, o
reach a pres

The pr

which they
tain feedback
for the natu
implications

The mo

in some deta
together, the
tem in a very

1.2 Why Ne

Artificial

Their behavior
parameters ar
(input-output)
output corresp
desired value.
with some unk
set and produc

1.1.2 Topologies of Artificial Neural Networks

Artificial neurons are analogous to their biological inspirers. In electronic circuits form, the cell body is represented by a nonlinear element such as an operational amplifier, the axons and the dendrites by electrical wires and the synapses by variable resistors. The input voltages to a neuron are weighted by the synapses which are represented by the variable resistors. The resistors are connected to an operational amplifier, on which a threshold has been set. When the sum of these input voltages reach a preset threshold, the nonlinear element fires imitating a biological neuron.

The processing nonlinear elements can interact in many ways by the manner in which they are interconnected. The interconnections may be feedforward or may contain feedback loops. The design of a neural network's feedback loops has implications for the nature of its adaptivity, while the design of network's interconnections has implications for its parallelism.

The modeling and the connection strategy of these artificial neurons are discussed in some detail in Chapter 2. When many of these processing elements are connected together, they form an artificial neural network which mimic the neurobiological system in a very restricted way.

1.2 Why New Energy Functions?

Artificial neural networks, unlike standard computers, do not execute programs. Their behavior is determined by parameters referred to as *synaptic weights*. These parameters are computed by a learning procedure, during which a set of examples (input-output pairs) are presented to the network; the weights are updated so that the output corresponding to each input from the training set is as close as possible to the desired value. When the learning phase is completed, the network can be presented with some unknown data and is anticipated to generalize from the data of the training set and produce a correct response.

o
a
.
7
o
n
u
o
ta
lo
no
m
on
lea
del
mo
ove
acce
con
tion.

The learning capability of neural networks plays a key role when we want to use them to do certain practical tasks. The automatic learning rules give artificial neural networks unique capabilities. Presently, many of the neural networks learn by using *gradient descent* methods associated with energy surface. The energy is usually defined as an error function in terms of the adjustable weights. The most commonly applied energy function gives rise to the so-called *back-propagation update rule*. This rule is discussed in detail in Chapter 2.

Pictorially, an energy surface can be thought of as consisting of hills and valleys. The surface represents a function of the output error of the network which in turn depends on the network weights. The gradient descent algorithm aims at finding the nearby lowest point called the local minimum. When the discrete-time gradient descent update law is implemented, the steps are discrete and we may observe oscillations and other undesirable behavior, which is more pronounced if the step size is large.

We use the continuous-time gradient descent update law due to specific advantages over its discrete-time counterpart. These laws ensure convergence to the nearest local minimum and do not exhibit oscillations. Also the attained local minimum may not be useful to solve the problem for which the network is trained.

The goal of this research is to make the learning faster and to reach an acceptable minimum quickly. We view the immediate domain of implementation to be software on digital computers. In that context, the learning can be increased by increasing the learning rate; but that would also increase the possibility of local oscillations and delayed convergence. In this work, energy surfaces are defined so that we do not add more minima to the already existing minima of the conventional energy surface. Moreover, starting from any initial point, the aim is to speed up the convergence to an acceptable minimum. This is achieved by using different energy functions in the continuous-time update law in contrast with the usual sum-of-the-squared energy function.

for

mu

neu

thei

- th

impr

work

tions

conti

their

their

analyt

the co

T.

Expone

cally an

the Exp

acceptab

We

update la

tion to th

are used.

1.3 Contribution of the Thesis

Properly designed and trained neural networks aims at achieving human like performance in solving association and recognition problems. Neural networks are not so much programmed as they are trained with data - thus many believe that the use of neural networks can relieve today's computer programmer of a significant portion of their programming load [14]-[16]. Moreover, neural networks improve with experience - the more data they are fed, the more accurate or complete their response becomes.

A framework, supported by analysis and computer simulations, is developed to improve the supervised learning of feedforward artificial neural networks. The framework encompasses various learning rules derived from carefully selected energy functions in order to speed up and enable convergence to acceptable weights. We use continuous-time gradient update law learning rules due to particular advantages over their discrete-time counterparts.

We describe the Cauchy and the Polynomial energy functions and characterize their properties and the performance of their associated learning rules. We show analytically and with computer simulations, that using the proposed energy functions in the continuous-time weight update law can speed up convergence to local minimum.

The local minimum may not be useful to solve the desired problem. The Exponential energy function is proposed to overcome this problem. We show analytically and with computer simulations that the continuous-time weight update law using the Exponential energy function can "skip" such minima and ensures convergence to acceptable minima.

We perform comparative investigations among these learning rules *vis-a-vis* the update law using the usual Gaussian energy function. We also demonstrate an application to the character recognition problem in which various fonts of printed characters are used.

vous
and t
mode
moder
ward
this w
multil
works

L
with th
and ex
energy
applied
around
mapping
the learn

The
unsuperv
data and
trial. Uns
Self-superv

CHAPTER 2

BACKGROUND

Neural net structures are based on our present understanding of the biological nervous systems. Neural net models are specified by the node characteristics, net topology and training or learning rules. By virtue of the type of input/output connections the models can be categorized as *the feedforward neuron model* or *the feedback neuron model*. The feedforward neuron model is discussed in section 2.1 whereas the feedforward neural network connection topology using these neurons is of primary interest in this work and is described in section 2.2. The resulting network which is reminiscent of multilayer perceptron [16],[17], will be referred to as *feedforward artificial neural networks* (FFANN).

Learning in these networks is inspired by adaptive classification networks beginning with the perceptrons convergence procedure and the least mean square (LMS) algorithm and extending to the more recent back-propagation, feature map and reduced coulomb energy (RCE) algorithms [15]. The back-propagation is currently the most widely applied neural network architecture used for learning. This popularity primarily revolves around the ability of back-propagation network to learn complicated multidimensional mappings [18]. The back-propagation architecture discussed in section 2.3, is used for all the learning strategies discussed in this work.

The back-propagation neural networks can be trained to learn using supervised, unsupervised or self-supervised training. Supervised training requires labeled training data and a teacher who provides error information through a feedback signal after each trial. Unsupervised training forms internal clusters automatically with unlabeled data. Self-supervised training uses internal monitoring and error correction to improve

perform
training

B.
forman
tion 2.5
tions. T
describe
contribu

2.1 The

Mo
McCullo
[20] by V
treated as

where x_j i
 j th input
 $f(u_k)$ is th
diagramma

It was
the neurons
was suffice
would appro
sigmoidal fun
vides a grade

performance without an external teacher. This work mainly concentrates on supervised training of the FFANN, using (continuous-time) gradient descent methods.

Back-propagation learning makes use of the gradient descent update laws. The performance of these update laws mainly depends upon the associated energy function. Section 2.5 describes the weight update laws and the influence of the associated energy functions. The Gaussian energy function is described in section 2.6. Section 2.7 is devoted to describe the current state-of-the-art in supervised training of FFANN and highlights the contributions of this work.

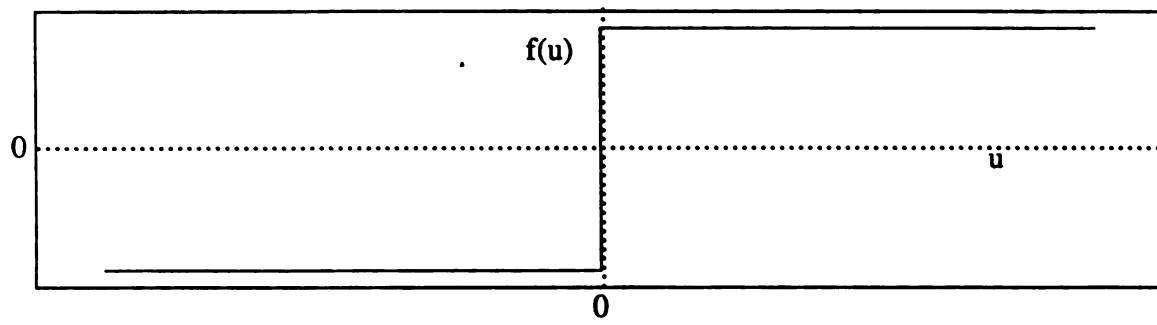
2.1 The Feedforward Neuron Model

Modeling of this class of artificial neural networks has its roots in the work of McCulloch and Pitts [19]. The feedforward model of neurons was clearly mentioned in [20] by Widrow and Hoff, where they called it *an adaptive neuron*. The neurons are treated as threshold logic units whose output v_k is given by

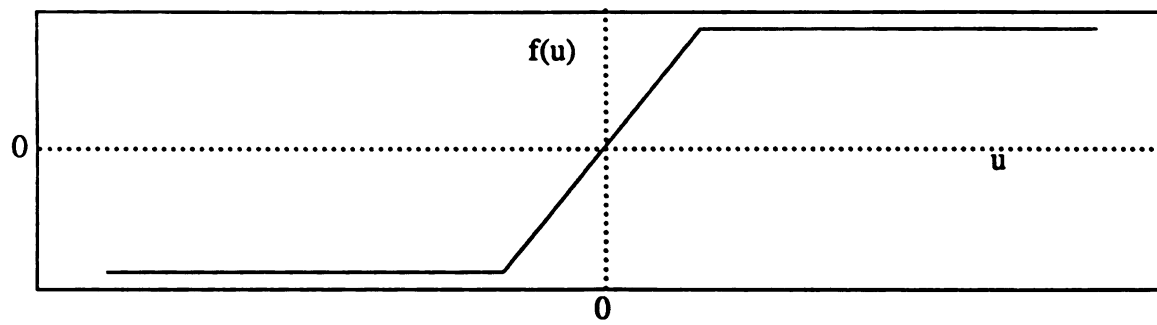
$$v_k = f(u_k) = f \left[\sum_j w_{kj} x_j + \theta_k \right] \quad (2.1)$$

where x_j is the external input, w_{kj} the *weight* or the strength of the connection from the j th input to the k th neuron and θ_k is the *threshold* or the *bias* for the neuron. Here $f(u_k)$ is the sgn function which is +1 if $u_k > 0$, 0 if $u_k = 0$ and -1 if $u_k < 0$ and is diagrammatically shown in figure 2.1a.

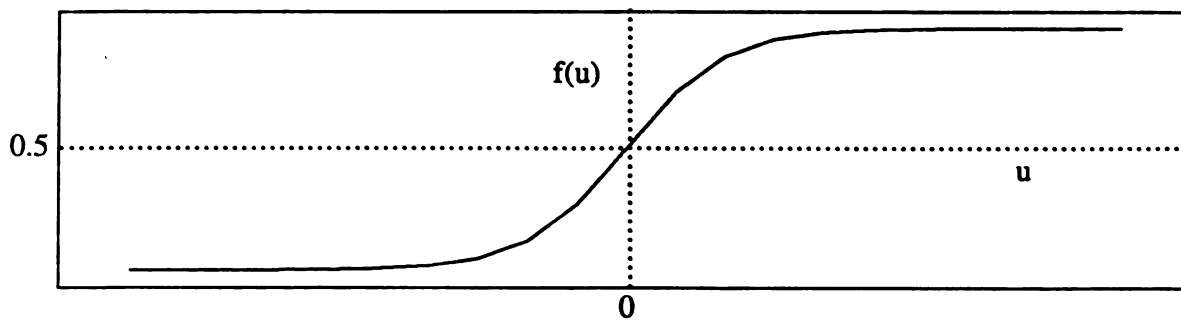
It was in 1972 that T. Kohonen [21] and J. A. Anderson [22] considered the fact that the neurons are not binary but are analog. They assumed that the activation of a neuron was sufficiently linear of the type shown in figure 2.1b, so that simple linear algebra would approximate the output of the system. It was realized later on [7],[23]-[26] that the sigmoidal function of the type shown in figure 2.1c is the most pervasive because it provides a graded and nonlinear response which is closest to a real neuron. The range of the



a. Sgn function



b. Nonlinear ramp function



c. Logistic function

Figure 2.1: Neuron activation functions

sigmoidal function is sometimes changed from $[-1,1]$ to $[0,1]$ depending upon application. The logistic function is a convenient activation relationship that represent a sigmoidal function and is given as

$$f(net_k) = \frac{1}{1 + e^{-(net_k + \theta_k)/\theta_0}} \quad (2.2)$$

Here θ_k controls the firing activity of the neuron for some given inputs. If θ_k is negative, the sigmoidal function shifts horizontally to the right and firing is delayed until a sufficient input arrives. θ_0 is used to modify the shape of the sigmoid according to the gain requirements of the neuron. This is illustrated in figure 2.2.

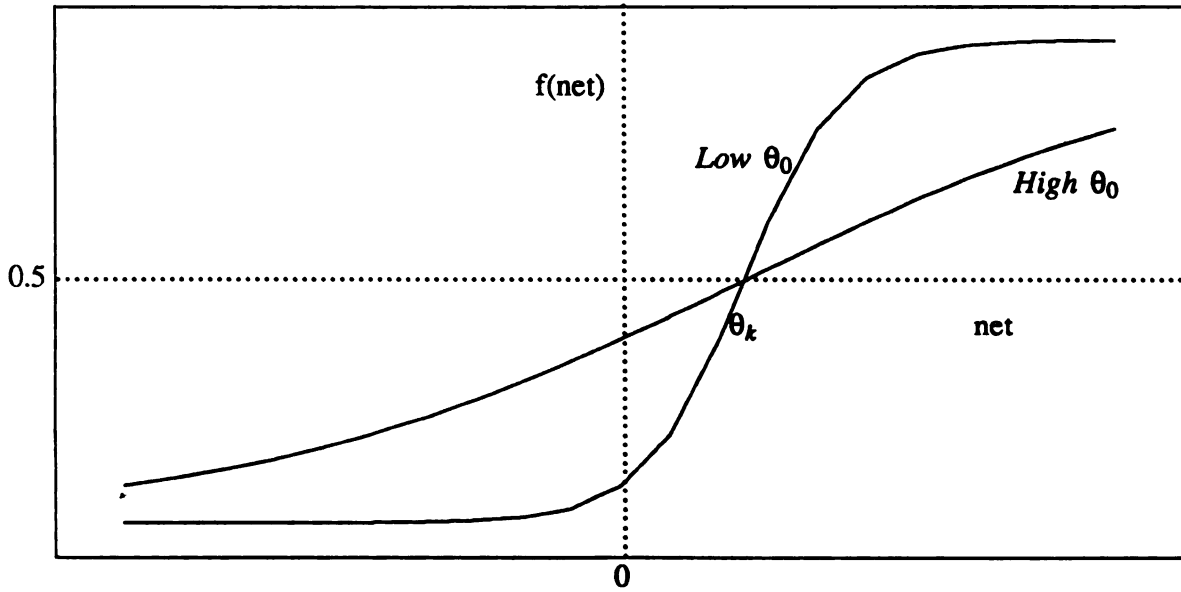


Figure 2.2: The sigmoidal activation function with threshold and shape modification

net_k represents the summation of weighted inputs to the neuron and is given by

$$net_k = \sum_j w_{kj} x_j \quad (2.3)$$

The complete feedforward model is shown in figure 2.3.

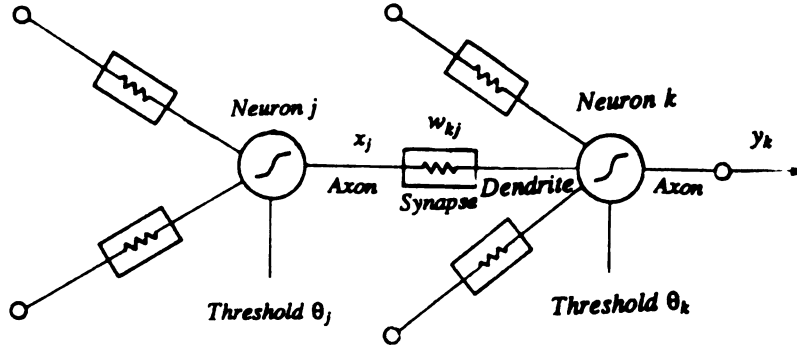


Figure 2.3: The feedforward neuron model.

2.2 The feedforward neural network connection topology

The neuron interconnection topology found in the human brain is very complicated but follows some pattern. Neurons in the retina and cortex are organized into layers with intra- and interlayer connections. Connections within a layer are referred to as short-term memory (STM) whereas connection between layers are referred as long-term memory (LTM). STM connections are usually considered to be unidirectional whereas LTM connections may propagate signals in feedforward and/or feedback direction.

We will use the feedforward connection topology throughout this work. This type of network topology makes use of feedforward neurons in different layers. In general the network consists of an input layer, m hidden layers and an output layer. These layers are indexed input through output layers by $IN, H_1, \dots, H_j, H_k, \dots, H_m, O$ consecutively. The input layer is assumed to have I passive nodes with indices $1, \dots, i, \dots, I$. The input passive nodes simply accept the individual components x_i of the input vector x and distribute them, without modification, to all of the neurons of the first hidden layer. m hidden layers may have sufficient number of neurons to perform the assigned task satisfactorily [27]-[29]. It was discovered early in the neural network research [17],[30],[31] that a sufficient

number of hidden layer(s) are required to solve a problem of normal complexity. Recently, it has been stated that one hidden layer would suffice for a large class of applications [32].

Each neuron in each layer receives the output signal of each of the neurons of the layer preceding it. This continues through all of the layers of the network until the final output layer. The output layer of the network consists of N neurons with indices $1, \dots, n, \dots, N$. The N -dimensional y output of the network is available at the output of the O th layer. It is assumed that the neurons in any particular layer, as depicted in figure 2.4, have no lateral or feedback connections.

A FFANN formally carries out a non-linear bounded mapping $v : X \subset R^I \rightarrow R^N$, from a compact subset X of I -dimensional space to the bounded subset $v(X) = Y$ of the N -dimensional space. When an input vector x is fed into the network's input layer, the passive nodes of this layer transmit all of the components of x to all of the neurons of the first hidden layer. The output of all of the neurons of the first hidden layer are then transmitted to all of the neurons of the second hidden layer and so on, until finally the N output units emit the components of the output vector Y .

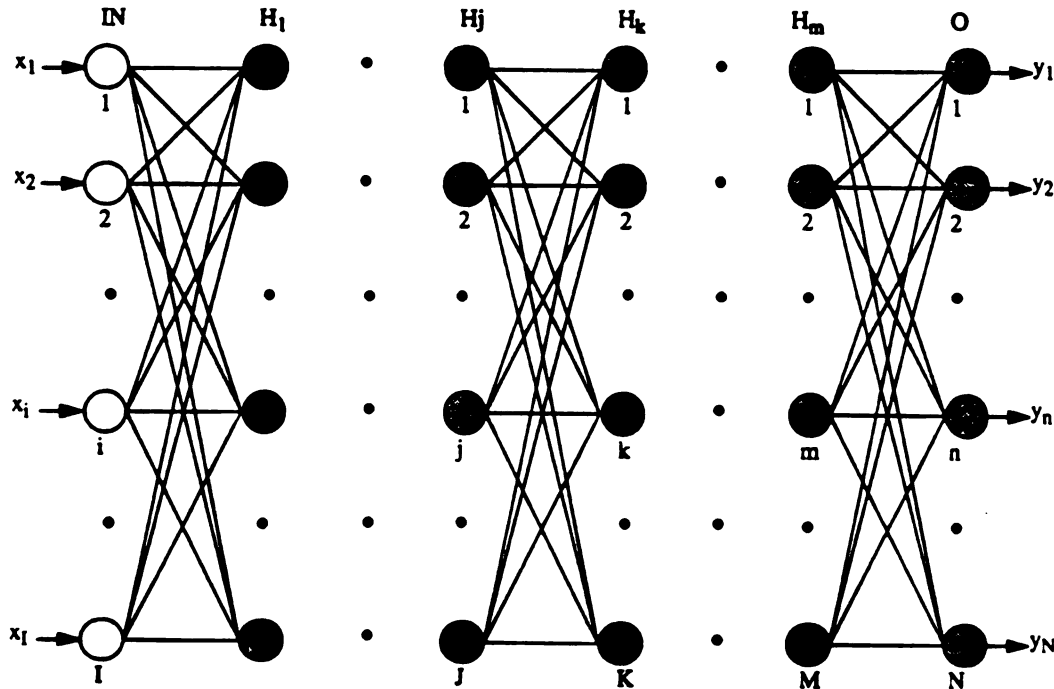


Figure 2.4: A structure of a multi-layer feedforward artificial neural network.

2.3 The Back-propagation Architecture

The back-propagation architecture as a technique was popularized by Rumelhart *et al.* in 1986 [7,8]. This architecture was earlier discovered by Bryson and Ho in 1969 [33] and independently rediscovered by Werbos in 1974 [34]. Today it is the most commonly used architecture in FFANN learning strategies [18].

Besides the feedforward connections, discussed earlier in section 2.2, each unit of the hidden layer receives an error signal from each of the units in the subsequent layer as depicted in Figure 2.5. The output units directly receive the error signal whenever the network is presented with some input.

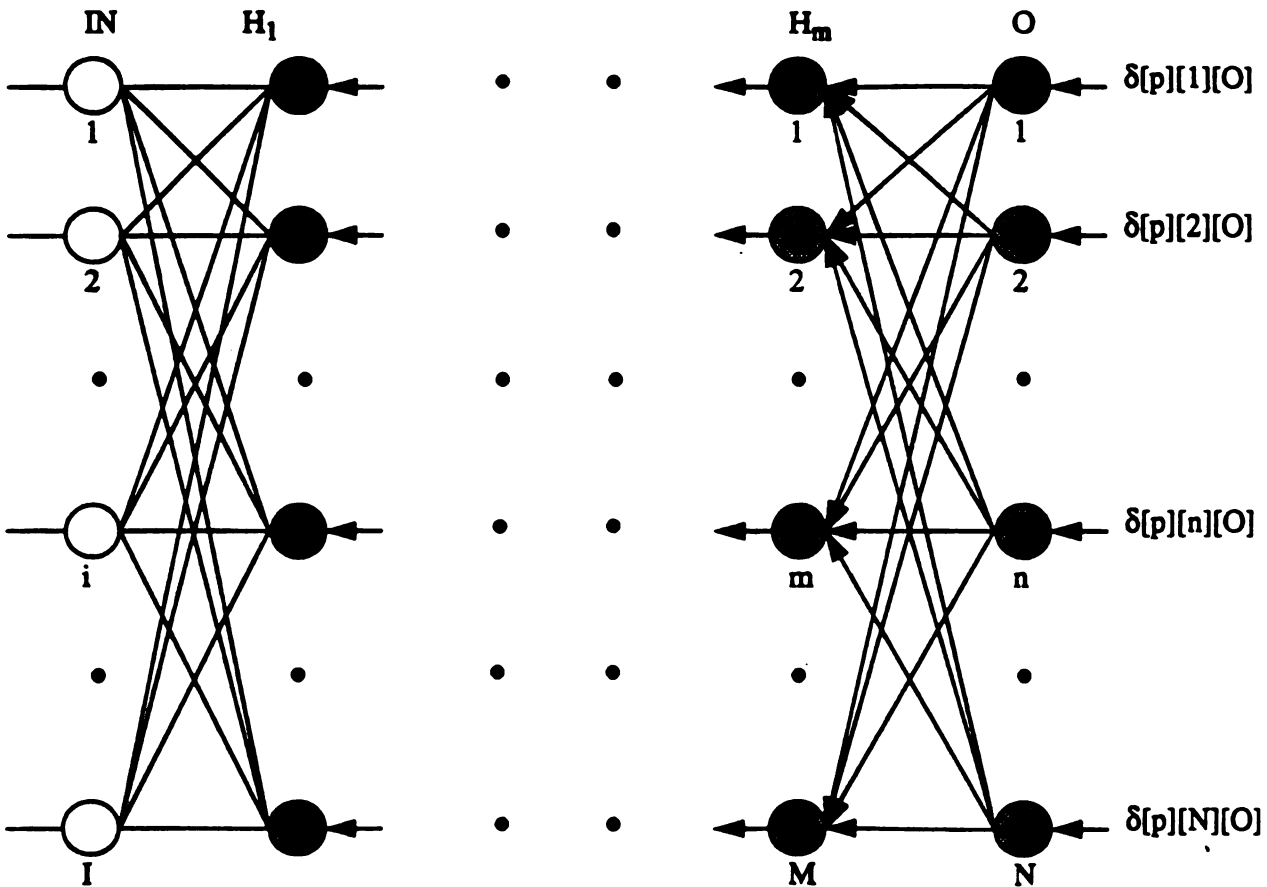


Figure 2.5: The back-propagation architecture of the FFANN.

The network functions in two stages: *a forward pass* and *a backward pass*. Consider the network's input consisting of P vectors, corresponding to P input patterns, indexed $1, \dots, p, \dots, P$, and each vector consists of I elements corresponding to I input passive nodes, indexed $1, \dots, i, \dots, I$. The forward pass starts by inserting I dimensional vectors $[x_p]_I^P$ into the network's input and the network emits out the corresponding N dimensional output vectors $[y_p = v(x_p)]_N^P$, which is the network's estimate of the desired or the target output $[t_p = g(x_p)]_N^P$. After the estimates $[y_p]_N^P$ are emitted, each of the output units is supplied with their correct or target output vector $[t_p]_N^P$. The elements of the error vectors are computed at the output of the network as

$$[\varepsilon_{pn}]_N^P = [|t_{pn} - y_{pn}|]_N^P \quad n = 1, \dots, N \quad (2.4)$$

The error elements ε_{pn} and the error signal $\delta[p][n][O]$ (to be discussed in section 2.6) are calculated and are available at each unit n of the output layer O .

In the backward pass, the output units directly receive the error signals from their outputs. The hidden units in the layer below the output layer receive their error signals from the output units to which they are connected. The error signal is thus passed back through the same links which were used for the forward pass but in the backward direction. The error signal $\delta[p][j][H_j]$ of a unit j in a hidden layer H_j is thus a function of the error signal of the units in the H_k th layer and their inter-linking weights. The error is propagated back till each of the units in the first hidden layer get their corresponding error signals. The output layer becomes the input layer in the backward pass.

To summarize, in each training cycle, the input is fed at the input layer of the network. The output of the network is determined in the forward pass. The error signal for each neuron in the network is calculated and propagated back in the backward pass. Depending upon the error signal the weight or the synaptic strength of each link in the network is adjusted in a manner so as to reduce the output error.

2.4 Supervised training of the back-propagation network

After having designed the network with the back-propagation architecture, using feedforward neurons in the feedforward connection topology as discussed in sections 2.1, 2.2 and 2.3, the most critical task for the network is to learn, i.e. achieve weights values that would realize the mapping. Learning is one of the key issues in the artificial neural network research.

With respect to learning, neural networks fall into a number of classes. Some networks are completely fixed in which the design of the network determines the weights. Other networks have synaptic weights that depend on the problem that is being addressed by the network, but the weights are pre-programmed into the network. Other networks can be updated continuously by the outside data source. Networks of this type can perform adaptive learning in response to changing environment. The final class is *in situ* learning, where the learning algorithm is incorporated directly into the network.

Most of the neural network research has focussed on direct learning to solve specific tasks instead of building an internal model. Direct learning involves learning a specific action to solve each task. Learning internal world models is opening up an important area of learning where internal models of limited environment are built through interaction and do not have to be programmed into the network. One of the approaches is to use FFANN to mimic the input/output behavior of network interacting with the environment.

This work focusses on the supervised training of FFANN with back-propagation architecture. It is a means of training adaptive neural networks which requires labeled training data and an external teacher. The teacher knows the desired correct response and provides an error signal when error is made by the network. This is sometimes called *reinforcement learning* or *learning with a critic* when the teacher only indicates whether a response was correct or incorrect but does not provide detailed error information.

During the supervised training, a training set consisting of examples (x_p, t_p) of mapping between the input x_p and corresponding desired output $t_p = g(x_p)$ for the input-output pattern $p = 1, \dots, P$ is provided. It is assumed that such examples of the mapping g are representative of the whole class of map from the input space to the desired space. When the input data set $[x_p]_1^P$ is presented to the network, it produces the estimate $[y_p = v(x_p)]_1^P$ of $[t_p = g(x_p)]_1^P$ and computes the output error vectors $\epsilon_p, p = 1, \dots, P$ as given by (2.4). The error signal is generated from the error vectors according to certain *weight update law*, which is discussed in section 2.5. The error signal when propagated back adjusts the weights of the network in a direction, so as to minimize the output error. With this training, the network thus, learns to associate an input pattern to its corresponding output pattern. If the training examples were the true representative of the class of mapping g , the network hopefully will associate all patterns (including those for which it has not been trained) correctly.

2.5 The weight update laws and the energy functions

The weight update law is very critical in determining the error signal which ultimately makes the network learn. The Generalized Delta Rule (GDR) is a gradient descent method of updating the weights. It has been proposed by Rumelhart *et al.* [7,8] and is given as

$$\Delta w_{kj} = -\gamma \nabla_{w_{kj}} E. \quad (2.5)$$

Here w_{kj} is the weight connecting the output of the j th node in the H_j th layer to the input of the k th node in the H_k th layer, γ is the learning rate, Δ represents the change and $\nabla_{w_{kj}}$ represents partial derivative with respect to w_{kj} . E is the energy function and is given as

$$E = \sum_p E_p \quad E_p = \frac{1}{2} \sum_n \epsilon_{pn}^2. \quad (2.6)$$

The weight update law (2.5) achieved a breakthrough in training the FFANN. It can be conveniently used in most cases, however the convergence cannot be guaranteed. Describing the update law as a system of differential equations [35]-[39] in the form

$$\dot{w}_{kj} = -\gamma \nabla_{w_{kj}} E, \quad (2.7)$$

has its advantages over the system of difference equations (2.5). In particular, being a continuous-time gradient descent system, (a) ensures convergence to local minima, i.e., stable equilibria, (b) prevents the existence of oscillations and complicated behaviors and (c) lends itself to natural implementation via analog VLSI/LSI silicon circuits [35]-[39].

The choice of energy function specifies the error signal to be propagated back. It also determines the learning performance in the sense of the speed of convergence and the size of domains of attraction of the stable equilibrium points in the weight space. We remark that the sum-of-the-square energy function is the L_2 (Euclidian) norm [40],[41].

2.6 The Gaussian energy function

A learning rule which achieves the desired values of weights (preferably the global minimum), is statistically equivalent to a maximum likelihood estimator (MLE). In other words, obtaining a MLE is equivalent to minimizing the errors given by some energy function. In the process of maximizing the likelihood estimate, the energy function is derived given the probability distribution of the errors from the desired values in the output of the FFANN.

A well known method of approximating a distribution on the basis of partial knowledge is to choose the density function $d(x)$ which maximizes the entropy subject to the constraint of moments [42]-[46]. It is well established that the maximum entropy distribution subject to the second moment constraint on $(-\infty, +\infty)$ is the Gaussian distribution [46]-[48].

Now suppose that we would like the errors in the output of a FFANN to be a Gaussian random variable with zero mean. In order to make the network an MLE, the criterion is to minimize the sum-of-the-squared error energy function [48]. This is specifically shown below. The Gaussian distribution of the errors is given by

$$d(\epsilon_{pn}) = \frac{1}{\sqrt{2\pi\sigma_{pn}^2}} \exp\left[-\epsilon_{pn}^2/2\sigma_{pn}^2\right]. \quad (2.11)$$

Assuming the output errors to be independently distributed for all the output nodes, the joint Gaussian distribution is then given by

$$d_j(\epsilon_p) = (2\pi)^{-N/2} |\Sigma_p|^{-1} \exp\left[-\frac{1}{2}(\epsilon_p)^T (\Sigma_p)^{-1} (\epsilon_p)\right]. \quad (2.12)$$

Here N denotes the total number of neurons in the output layer of the FFANN and

$$\epsilon_p = [\epsilon_{p1} \ \epsilon_{p2} \ \dots \ \epsilon_{pN}]^T,$$

$$\Sigma_p = \text{diag} [\sigma_{p1} \ \sigma_{p2} \ \dots \ \sigma_{pN}],$$

where σ_{pn} is the standard deviation of the error distribution of the n th output node when the p th pattern is used. Since the errors ϵ_p , $p = 1, 2, \dots, P$, are independent, the likelihood function L is

$$\begin{aligned} L(\epsilon_p) &= \prod_{p=1}^P d_j(\epsilon_p) \\ &= (2\pi)^{-PN/2} |\Sigma_p|^{-P} \exp\left[-\frac{1}{2} \sum_{p=1}^P (\epsilon_p)^T (\Sigma_p)^{-1} (\epsilon_p)\right] \end{aligned} \quad (2.13)$$

Choose $\sigma_{p1} = \sigma_{p2} = \dots = \sigma_{pN} = 1$, for all the patterns $p = 1, 2, \dots, P$. Maximizing the likelihood function, then, corresponds to minimizing the energy function in equation (2.6), which is referred to as the Gaussian energy function. Of course, in general, Σ_p in (2.13) can be any symmetric nonsingular matrix.

The continuous-time learning dynamics of the update law (2.7), using the Gaussian energy function, have been explored in [35]-[39]. In subsection 2.6.1, we describe these

learning dynamics, whereas the software implementation is outlined in subsection 2.6.2.

2.6.1 The learning dynamics due to the Gaussian energy function.

As described earlier, the FFANN under consideration uses the error back-propagation architecture. For the weight w_{nm} , connecting neuron m in the last hidden layer H_m to the output layer neuron n of a FFANN, the learning dynamics are described by the weight update law (2.7) and is given as

$$\dot{w}_{nm} = -\gamma \nabla_{w_{nm}} E, \quad (2.14i)$$

and for any hidden layer

$$\dot{w}_{kj} = -\gamma \nabla_{w_{kj}} E, \quad (2.14ii)$$

Using the chain rule, the dynamics are

$$\dot{w}_{nm} = -\gamma \cdot \frac{\partial E}{\partial \epsilon_{pn}} \cdot \frac{\partial \epsilon_{pn}}{\partial y_{pn}} \cdot \frac{\partial y_{pn}}{\partial u_{pn}} \cdot \frac{\partial u_{pn}}{\partial w_{nm}}, \quad (2.15)$$

where E is the Gaussian energy function given by (2.6). ϵ_{pn} is the error of the output y_{pn} from the target value t_{pn} for the p th pattern and the n th output node and is given by (2.4). u_{pn} is the net input to the n th node of the output layer and is given by

$$u_{pn} = \sum_m w_{nm} z_{pm} + \theta_n. \quad (2.16)$$

Here z_{pm} denotes the output of the m th node in the H_m th layer. θ_n is the threshold of the n th node in the output layer. The partial derivatives in equation (2.15) can be explicitly calculated as

$$\frac{\partial E}{\partial \epsilon_{pn}} = \epsilon_{pn},$$

$$\frac{\partial \epsilon_{pn}}{\partial y_{pn}} = -\text{sgn}(t_{pn} - y_{pn}),$$

$$\frac{\partial y_{pn}}{\partial u_{pn}} = y_{pn}(1 - y_{pn}),$$

and

$$\frac{\partial u_{pn}}{\partial w_{nm}} = z_{pm}.$$

Here we have selected the so-called *logistic function* [7,8]

$$y_{pn} = \frac{1}{1 + e^{-u_{pn}}} \quad (2.17)$$

to describe the input-output relationship of a neuron. Now the weight update law (2.15) can be written as

$$\begin{aligned} \dot{w}_{nm} &= \gamma \sum_p \epsilon_{pn} \cdot \text{sgn}(\epsilon_{pn}) \cdot y_{pn} \cdot (1 - y_{pn}) \cdot z_{pm} \\ &= \gamma \sum_p (t_{pn} - y_{pn}) \cdot y_{pn} \cdot (1 - y_{pn}) \cdot z_{pm} \\ &= \gamma \sum_p \delta[p][n][O] \cdot y[p][m][H_m], \end{aligned} \quad (2.18)$$

where $y[p][m][H_m]$ is the output z_{pm} of the m th node in the H_m th layer. $\delta[p][n][O]$ is the error signal which is completely determined by the desired and the actual outputs of node n in the output layer and is given by

$$\delta[p][n][O] = (t_{pn} - y_{pn}) \cdot y_{pn} \cdot (1 - y_{pn}). \quad (2.19)$$

Observe that the weight w_{nm} , is updated depending upon the error signal of the n th node and output of m th node. The weight update law for the weights w_{kj} , connecting the j th node of a hidden/input layer H_j to the k th node in the subsequent hidden layer H_k , is derived similarly and is given by

$$\dot{w}_{kj} = \gamma \sum_p \delta[p][k][H_k] \cdot y[p][j][H_j], \quad (2.20)$$

where

$$\delta[p][k][H_k] := \sum_l \delta[p][l][H_l] \cdot w_{lk} \cdot y_{pk} \cdot (1 - y_{pk}). \quad (2.21)$$

In (2.21), l is the index of the nodes in the H_l th layer to which node k feeds its output

y_{lk} and w_{lk} is the weight of the corresponding connection. $y[p][j][H_j]$ is the output of the node j in the H_j th layer which is fed to node k in the H_k th layer.

2.6.2 Software implementation

For the software implementation of the update law (2.7), using the Gaussian energy function, the learning algorithm is formalized in Table 2.1.

Table 2.1

The Learning Algorithm	
1	Present the input vectors $x_p, p = 1, \dots, P$, at the input of the FFANN.
2	Determine the network's output $y_p = v(x_p), p = 1, \dots, P$.
3	Calculate the error signal for the output nodes.
4	Propagate the error signal back to all the nodes in the network.
5	Update the weights.
6	If the weight change of the network is greater than the stopping criterion go to step 1, otherwise stop.

The stopping criterion ω can be defined as

$$\omega := \sum_{w_{kj}} |\Delta w_{kj}|, \quad (2.22)$$

where w_{kj} are all the weights in the network. Theoretically in order to reach a local minimum the weight change criterion should be zero, which may never be achieved. In practice we can choose a sufficiently small value of ω , which when achieved indicate that the trajectories of the system have reached a local minimum. The weight change Δw_{kj} of the system can be determined as

$$\Delta w_{kj} := w_{kj}(t + \Delta t) - w_{kj}(t). \quad (2.23)$$

The term Δt is the time step of the integration routine. Using the fourth order Runge-Kutta integration routine the learning algorithm outlined in Table 2.1 was implemented into computer software using the C-language [36].

The software is designed to initialize the weights of the network randomly, by a random number generator. The *seed*, which is an integer used to start the random number generator, is used to represent the set of initial weights. The value of the initial weights w is restricted to be $-0.9 \leq w \leq 0.9$. An error criterion ξ is defined as

$$\xi := \sum_p \sum_n \epsilon_{pn}^2, \quad (2.24)$$

is used to characterize the training: the lower the error, the better is the training. The error criterion can also prove to be useful in some cases, to terminate the training of the network.

2.7 The problem description

We consider the implementation of FFANN into software on digital computers. For learning of such FFANNs, the continuous-time weight update law (2.7) which is a set of differential equations is implemented on digital computers. For faithful implementation of the gradient descent update law (2.7), the learning rate γ and the time step Δt (2.23) must be very small. As a consequence of the small value of $\gamma \Delta t$,

- (1) the learning is slow
- (2) the algorithm converges to the nearest local minimum which in most cases may not be satisfactorily useful.

One may consider increasing value of the learning rate γ as an immediate solution to the above problems. This may not work in all the cases. With a large value of γ , while we increase the tendency of skipping the nearby local minima, we may also make it prone to

skip desirable minima and delay or prevent the convergence.

The so-called back-propagation update law in its existing form uses an almost Euler approximation to the continuous-time update law. In addition, most reported simulations use a large step size or a learning rate. Such large values prevent tractable analysis and have not been proven to ensure convergence. Thus the implementation of FFANNs with the two conflicting properties remains a challenge:

- (a) the update in speed is fast.
- (b) Gauranteed convergence to an acceptable minimum and the occurence of no oscillations. In this work, we aim at achieving this challenge.

2.8 The objectives and the outline of the thesis

In this work we deal with this problem in two steps.

- (1) We present a method to make the learning faster by using different energy functions in the update law (2.7).

For the supervised learning of FFANN we use the continuous-time weight update law (2.7) which is derived from an energy function. If we choose an appropriate energy function the network may learn faster. The energy function can be viewed as

- (i) related to the distribution of the random errors in the output of the network, or
- (ii) a mathematical norm.

We can choose a maximum entropy distribution of random errors in the output of the network, subject to the constraint of moments. Then, proper selection of the energy function for training the FFANN can make it an MLE in the statistical sense. In chapter 3, we show that in order to make the learning faster, we can choose a distribution other than the Gaussian distribution. We choose the Cauchy distribution and motivate a rational for its selection and show that learning in this case is faster

or at least does not deteriorate. This conclusion is also supported by computer simulations.

When the energy function is viewed as a mathematical norm, we have the option of selecting any L_p norm. We combine various norms to form what we call the Polynomial energy function. In chapter 4, we show the usefulness of choosing the Polynomial energy function analytically and with simulation support.

- (2) We present a method which make it possible to "skip" unacceptable local minima and at the same time ensures faster convergence to one of the acceptable minima.

The problem of skipping the unacceptable local minima is tackled in chapter 5. We propose an exponential energy function, which skips the unacceptable local minima and ensures faster convergence to one of the acceptable minima. The Exponential energy function is used in conjunction with the Gaussian, the Cauchy or the Polynomial energy functions.

The performance of the learning algorithm using weight update law (2.7) with different energy functions (Table 2.1) is evaluated in the application to the pattern recognition problem in chapter 6. After discussing the network design, various computer simulations are presented. Finally in chapter 7, summary of this work is collected along with concluding remarks.

CHAPTER 3

THE CAUCHY ENERGY FUNCTION

3.1 Motivation

This chapter addresses the issue of increasing the learning speed of the FFANNs. In what follows, we describe the intuitive rational for selecting the Cauchy energy function from the point of view of statistical signal processing. On the other hand, the real rational for this work may stem from the convergence speed of the would-be proposed weight update learning rule as compared to the usual update rule (2.6)-(2.7).

The derivation of a supervised training algorithm for a FFANN implies the selection of a norm criterion which gives a suitable measure of a particular distribution of errors in the output of the network. There exists a correspondence between the L_p norms and the error distributions [48]. Many researchers [44]-[48] consider the maximum entropy of the output errors subject to certain moments constraints to be a suitable measure to estimate the error distribution. The larger is the number of moment constraints, the lesser is the choice of error distributions.

It is well known that subject to the first and the second moment constraints, the maximum entropy distribution is the Gaussian distribution [44]-[48]. Corresponding to the Gaussian distribution, the criterion is to select the L_2 norm for the energy function in the weight update law (2.7) [49].

Restricting our attention to the first moment constraint, there are many choices of an error distribution. The Cauchy distribution [50], given as

$$d(\epsilon_{pn}) = \frac{1}{\pi} \left[\frac{\lambda_{pn}}{\lambda_{pn}^2 + \epsilon_{pn}^2} \right], \quad (3.1)$$

is chosen due to the following reasons.

- (1) The entropy of the Cauchy distribution (3.1) is about 1.78 times higher than the entropy of the Gaussian distribution (2.11), when the parameters λ_{pn} and σ_{pn} for these distributions is chosen to be unity. This is analytically shown in Appendix A.
- (2) The speed of convergence of the weight update law (2.7) improves when the energy function corresponding to the Cauchy distribution is used instead of the Gaussian energy function (2.6). This is discussed in detail in this chapter.

In order to derive the energy function corresponding to the Cauchy distribution of errors, let us consider the errors ϵ_{pn} , $1 \leq p \leq P$ and $1 \leq n \leq N$, to be independently distributed. The joint Cauchy distribution will then be

$$d_j(\epsilon_{pn}) = (\pi)^{-N} \prod_{n=1}^N \frac{\lambda_{pn}}{\lambda_{pn}^2 + \epsilon_{pn}^2}. \quad (3.2)$$

Since the errors for all the desired patterns $p = 1, 2, \dots, P$, are assumed to be independent, the likelihood function is

$$\begin{aligned} L(\epsilon_{pn}) &= \prod_{p=1}^P d_j(\epsilon_{pn}) \\ &= (\pi)^{-PN} \prod_{p=1}^P \prod_{n=1}^N \frac{\lambda_{pn}}{\lambda_{pn}^2 + \epsilon_{pn}^2}. \end{aligned} \quad (3.3)$$

We choose $\lambda_{p1} = \lambda_{p2} = \dots = \lambda_{pN} = 1$ for all the patterns $p = 1, 2, \dots, P$. Then maximizing the likelihood estimator (3.3), corresponds to minimizing the quantity

$$E^c = \frac{1}{2} \prod_p \prod_n (1 + \epsilon_{pn}^2). \quad (3.4)$$

We refer to the quantity in (3.4) as the Cauchy energy function [52].

For a qualitative comparison, we set σ_{pn} in (2.11) and λ_{pn} in (3.1) to 1 and obtain the two distributions depicted in Figure 3.1. From the graphs in Figure 3.1, observe that the Cauchy distribution of the error is more spread out than the Gaussian distribution. This "spread out" feature signifies higher entropy and more tolerance in errors of the

Cauchy distribution.

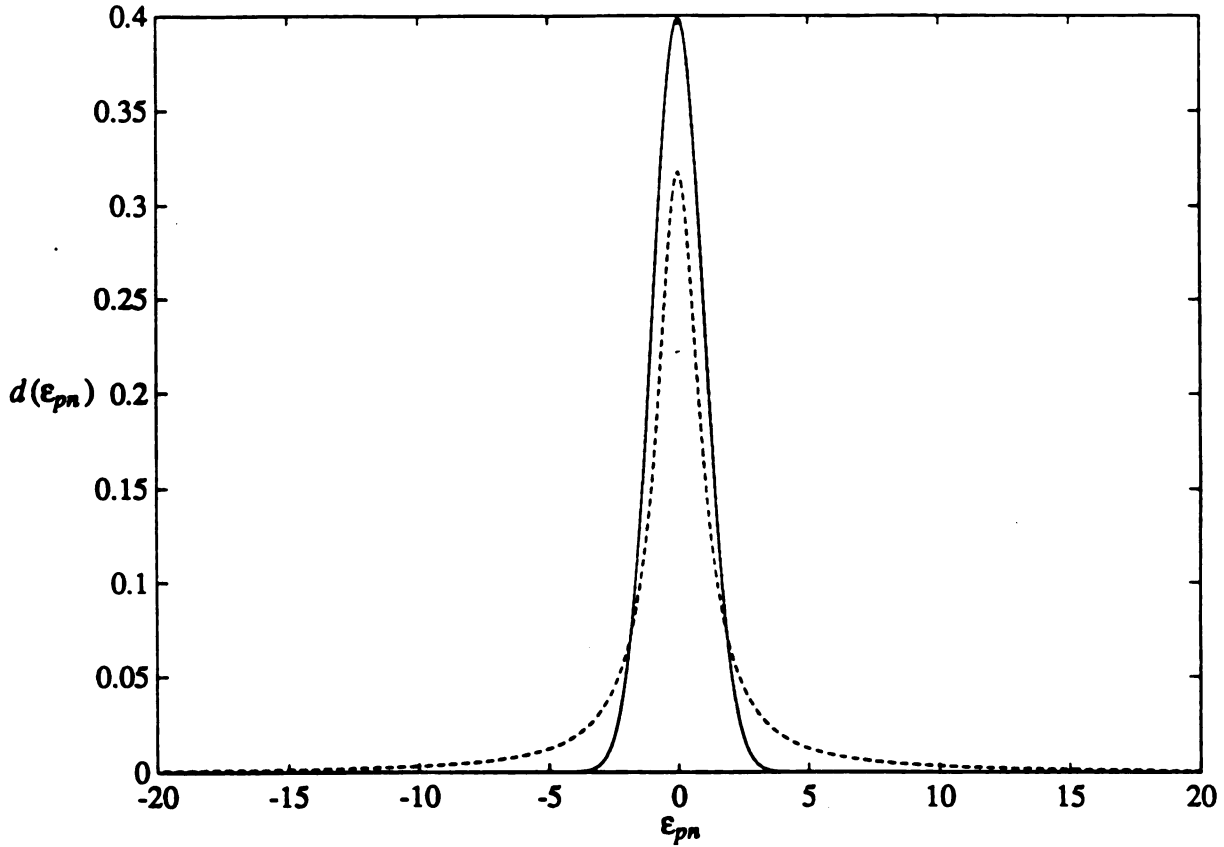


Figure 3.1: Random error distribution. The horizontal axis show the random error and the vertical axis show the density with which they are distributed. The solid line represents the Gaussian error distribution as given by (2.11) with $\sigma_{pn} = 1$ and the dotted lines represents the Cauchy error distribution as given by (3.1) with $\lambda_{pn} = 1$.

In this work, we desire to train the FFANN using a learning update law derived from the Cauchy energy function (3.4). When the update law converges to a weight value that minimizes the Cauchy energy function, the FFANN would then emulate a classifier with Cauchy distribution of errors from the target values.

The chapter is organized as follows. In section 3.2, we analyze the learning dynamics of the weight update law (2.7) using the Cauchy energy function. In section 3.3, we analytically compare the learning dynamics due to the Gaussian (2.6) and the Cauchy (3.4) energy function. Software implementation of the learning dynamics are presented in section 3.4, which are then used to train a FFANN. As an illustrative example, we specialize the software to solve the XOR problem. Simulation results are reported in section 3.5. We discuss the advantages and short-comings of using the Cauchy energy function in section 3.6.

3.2 The Learning Dynamics

In this section, the continuous-time learning dynamics of the update law (2.7), using the Cauchy energy function, are explicitly derived. As described earlier in section 2.2, the output layer is layer O , with nodes indexed by n . The $O-1$ th layer which feeds its output to the output layer is actually the last hidden layer H_m , whose nodes are indexed by m . The learning dynamics of the weights using the Cauchy energy function (3.4) in the update law (2.7); specifically for the output layer

$$\dot{w}_{nm}^c = -\gamma \nabla_{w_{nm}} E^c, \quad (3.5i)$$

and for any hidden layer

$$\dot{w}_{kj} = -\gamma \nabla_{w_{kj}} E^c. \quad (3.5ii)$$

Using the chain rule, the dynamics for the output layer are

$$\dot{w}_{nm}^c = -\gamma \cdot \frac{\partial E^c}{\partial \epsilon_{pn}} \cdot \frac{\partial \epsilon_{pn}}{\partial y_{pn}} \cdot \frac{\partial y_{pn}}{\partial u_{pn}} \cdot \frac{\partial u_{pn}}{\partial w_{nm}}, \quad (3.6)$$

where u_{pn} is the net input to the n th node of the output layer and is given by

$$u_{pn} = \sum_m w_{nm} z_{pm} + \theta_n. \quad (3.7)$$

Here z_{pm} denotes the output of the m th node in the H_m th hidden layer. θ_n is the

threshold of the n th node in the output layer. The partial derivatives in equation (3.7) can be explicitly calculated as

$$\begin{aligned} \frac{\partial E^c}{\partial \epsilon_{pn}} &= -\frac{1}{2} \left[\prod_{\rho} \prod_{\eta \neq n} (1 + \epsilon_{\rho\eta}^2) \right] \left[\sum_{\rho} \prod_{\rho \neq p} (1 + \epsilon_{\rho n}^2) \right] \cdot 2 \cdot \epsilon_{pn}, \\ &= -\sum_p \left[\left[\prod_{\rho} \prod_{\eta \neq n} (1 + \epsilon_{\rho\eta}^2) \right] \left[\prod_{\rho \neq p} (1 + \epsilon_{\rho n}^2) \right] \cdot \epsilon_{pn} \right], \end{aligned} \quad (3.8i)$$

$$\frac{\partial \epsilon_{pn}}{\partial y_{pn}} = -\text{sgn}(t_{pn} - y_{pn}), \quad (3.8ii)$$

$$\frac{\partial y_{pn}}{\partial u_{pn}} = y_{pn}(1 - y_{pn}), \quad (3.8iii)$$

and

$$\frac{\partial u_{pn}}{\partial w_{nm}} = z_{pm}. \quad (3.8iv)$$

Here ρ is a dummy variable representing patterns 1 through P and η is a dummy index for nodes 1 through N of the output layer. We have selected the so-called *logistic function* [7,8]

$$y_{pn} = \frac{1}{1 + e^{-u_{pn}}}$$

to describe the input-output relationship of a neuron. Now the weight update law can be written as

$$\dot{w}_{nm}^c = \gamma \sum_p \delta[p][n][O] \cdot y[p][m][H_m], \quad (3.9)$$

where

$$\delta[p][n][O] = \left[\prod_{\rho} \prod_{\eta \neq n} (1 + \epsilon_{\rho\eta}^2) \right] \left[\prod_{\rho \neq p} (1 + \epsilon_{\rho n}^2) \right] \cdot (t_{pn} - y_{pn}) \cdot y_{pn} \cdot (1 - y_{pn}) \quad (3.10)$$

and $y[p][m][H_m]$ is the output z_{pm} of the m th node in the H_m th layer.

The weight update law for the weights w_{kj} , connecting the j th node of the hidden/input layer H_j to the k th node in the subsequent hidden layer H_k , is derived

similarly and is given by

$$\dot{w}_{kj} = \gamma \sum_p \delta[p][k][H_k] \cdot y[p][j][H_j], \quad (3.11)$$

where

$$\delta[p][k][H_k] := \sum_l \delta[p][l][H_l] \cdot w_{lk} \cdot y_{pk} \cdot (1 - y_{pk}). \quad (3.12)$$

In (3.12) l is the index of the nodes in the H_l th layer to which node k feeds its output y_{lk} and w_{lk} is the weight of the corresponding connection. $y[p][j][H_j]$ is the output of the node j in the H_j th layer which is fed to node k in the H_k th layer.

3.3 Analytical Comparison with the Gaussian Energy Function

3.3.1 A criterion for the learning performance

To compare the dynamics of the (Gaussian) learning (2.14) and the (Cauchy) learning (3.5), we define the performance criterion delineated in the following procedure. First we consider $V := E$ as a candidate Liapunov function [53]-[54]. Then, we calculate the derivative of V along the trajectories of the (Gaussian) learning (2.14) -- denote this derivative by \dot{V} . Similarly, we calculate the derivative of V along the trajectories of the (Cauchy) learning (3.5) -- denote this derivative by \dot{V}^c . Now if $\dot{V}^c - \dot{V} \leq 0$, then we say that the *learning performance** of the second system is "faster" than or at least the same as the first system. This is motivated by the fact that $\dot{V}^c - \dot{V}$ gives the projection of the rate of change of the difference in the weight vectors, say $(W^c - W)$, along the divergence vector $\nabla_W V$. This we take as a measure of speed of updating the weight-vectors difference $(W^c - W)$, which in turn is equivalent to the comparative speed in updating the weight vectors W^c and W .

* Improvement in the learning performance of system (3.5) here means the derivative of the sum-of-the-squared error at any point on the error surface along the trajectories of system (2.14) is less in magnitude than that of system (3.5).

Formally, we observe that

$$\dot{V}^c - \dot{V} = \nabla_W V \cdot (\dot{W}^c - \dot{W}), \quad (3.13)$$

where W^c and W denote vectors containing all the corresponding weights. Equations (3.5) give the components of \dot{W}^c and equations (2.14) give the components of \dot{W} .

3.3.2 The learning dynamics for a single pattern

Consider the case when only one input-output pattern is applied to the network. For ease of notation we will drop the subscript p for patterns and define the Gaussian and the Cauchy energy functions as

$$E := \frac{1}{2} \sum_n \epsilon_n^2 \quad (3.14i)$$

and

$$E^c := \frac{1}{2} \prod_n (1 + \epsilon_n^2), \quad (3.14ii)$$

respectively. By straight forward calculations, for the output layer equation (3.5i) can be rewritten as

$$\dot{w}_{nm}^c = -\gamma \left[\nabla_{w_{nm}} E + \alpha_n \nabla_{w_{nm}} E \right]. \quad (3.15i)$$

For the hidden layer H_k the learning algorithm in (3.5ii) can be similarly rewritten as

$$\dot{w}_{kj} = -\gamma \left[\nabla_{w_{kj}} E + \frac{1}{2} \sum_n \alpha_n \nabla_{w_{kj}} \epsilon_n^2 \right]. \quad (3.15ii)$$

Here $\alpha_n \geq 0$ and is given explicitly as

$$\alpha_n = E_1^{\#} + E_2^{\#} + \dots + E_{(N-1)}^{\#},$$

where

$$E_1^{\#} := E - \epsilon_n^2,$$

$E_2^{\#}$ is the summation of all $\left[\frac{N-1}{2} \right]$ pairs not including ϵ_n^2 ,

and finally

$$E_{(N-1)} := \epsilon_1^2 \epsilon_2^2 \dots \epsilon_N^2 \quad \text{not including } \epsilon_n^2$$

Before we formally characterize the *learning performance* due to the Gaussian and the Cauchy energy function in the form of a theorem, Lemma 3.1 is in order, which uses Assumption 3.1 given below.

Assumption 3.1: All the eigenvalues of the matrix with entries $a_{ij} = \alpha_i + \alpha_j$, are non-negative.

Remark: In our simulations, we always initialize the weights in the learning dynamics so that the weight components have small values (typically within $[-0.9, 0.9]$). In doing so, the output of the individual nodes are likely to be close to 0.5. In that case, $\alpha_1 \approx \alpha_2 \approx \dots \approx \alpha_n \approx \dots \approx \alpha_N$. If the equality holds, then Assumption 3.1 is automatically satisfied. See Appendix B.

Lemma 3.1: Given $\alpha_i, x_i \in R, \alpha_i \geq 0$ and assuming that Assumption 3.1 holds.

Then,

$$\left[\sum_i x_i \right] \left[\sum_i \alpha_i x_i \right] \geq 0.$$

Proof of Lemma 3.1 is provided in Appendix B.

Theorem 3.1: Assume that Assumption 3.1 holds. Consider $V := E$ given by (3.14i), be a candidate Liapunov function. Let \dot{V} be the derivative of V along the trajectories of system (2.14) and \dot{V}^c be the derivative of V along the trajectories of system (3.15). Then the *learning performance* of system (3.15) is "faster" than or at least is the same as system (2.14).

The proof of this theorem, presented in Appendix C, analytically shows that the learning performance improves if the Cauchy energy function is used instead of the Gaussian energy function.

3.3.3 The learning dynamics for multiple patterns

Now, we consider the learning dynamics of the weights for P patterns. The (Gaussian) weight update law is given by equation (2.14) where E is given by (2.6). The (Cauchy) learning dynamics of the weights takes the form of (3.5) where E^c now is given by (3.4). By straight forward calculations, \dot{w}_{nm}^c can be written as

$$\dot{w}_{nm}^c = -\gamma \left[\nabla_{w_{nm}} E + \frac{1}{2} \sum_p \alpha_{pn} \nabla_{w_{nm}} \epsilon_{pn}^2 \right]. \quad (3.16i)$$

Similarly, for the hidden layers the learning algorithm of the Cauchy energy function (3.4) can be written as

$$\dot{w}_{kj} = -\gamma \left[\nabla_{w_{kj}} E + \frac{1}{2} \sum_p \sum_n \alpha_{pn} \nabla_{w_{kj}} \epsilon_{pn}^2 \right]. \quad (3.16ii)$$

Here $\alpha_{pn} \geq 0$ and is given by

$$\alpha_{pn} = E_1^n + E_2^n + \dots + E_{(PN-1)}^n,$$

where

$$E_1^n = E - \epsilon_{pn}^2,$$

$$E_2^n \text{ is the summation of all } \left[\begin{smallmatrix} PN-1 \\ 2 \end{smallmatrix} \right] \text{ pairs not including } \epsilon_{pn}^2,$$

and finally

$$E_{(PN-1)}^n = \epsilon_{11}^2 \epsilon_{12}^2 \dots \epsilon_{PN}^2 \quad \text{not including } \epsilon_{pn}^2.$$

Before we compare the *learning performance* due to the weight update laws (2.14) and (3.16), we state Lemma 3.2, which uses the following assumption.

Assumption 3.2: All the eigenvalues of the matrix with entries $a_{ij,k} = \alpha_{ij} + \alpha_k$, are non-negative.

Remark: Assumption 3.2 is equivalent to Assumption 3.1.

Lemma 3.2: Given $\alpha_{ij}, x_{ij} \in R$, $\alpha_{ij} \geq 0$ and assuming that Assumption 3.2 holds.

Then,

$$\left[\sum_i \sum_j x_{ij} \right] \left[\sum_i \sum_j \alpha_{ij} x_{ij} \right] \geq 0.$$

Proof of Lemma 3.2 is provided in Appendix D.

Theorem 3.2: Assume that Assumption 3.2 holds. Consider a candidate Liapunov function $V := E$ given by equation (2.6). Let \dot{V} be the derivative of V along the trajectories of (2.14) and \dot{V}^c be the derivative of V along the trajectories of the system (3.16). Then the *learning performance* of system (3.16) is "faster" than or at least is the same as system (2.14).

Again the proof of this theorem is included in Appendix E. The analysis clearly indicate that the learning performance in "speed" can improve (or at least does not deteriorate) when the Cauchy energy function is used.

Proposition 3.1: The equilibria of system (3.16) are also the equilibria of system (2.14).

The proof of Proposition 3.1 is provided in Appendix F.

Remark: Proposition 3.1 implies that when we use the Cauchy energy function (3.5) in the weight update law (2.7), we still converge to one of the equilibria of system (2.14). However, this does not mean that starting from the same initial condition, the two

systems would follow the same trajectory or that their trajectories would converge to the same equilibrium point.

3.4 Software implementation of the learning dynamics

For the software implementation of the learning dynamics of the weights connected to the output layer (3.9) and the weights connected to the hidden layers (3.11), the learning algorithm formalized in Table 2.1 is used. The weight update laws (3.9) and (3.11) are implemented into computer software using the C-language on Sun SPARC stations. Fourth order Runge-Kutta integration routine is used for the integration of these update laws.

A stopping criterion ω , was set to determine if the system has converged to a solution. The parameter ω was defined as

$$\omega := \sum_{w_{kj}} |\Delta w_{kj}|, \quad (3.17)$$

where w_{kj} are all the weights in the network and

$$\Delta w_{kj} := w_{kj}(t + \Delta t) - w_{kj}(t). \quad (3.18)$$

The term Δt is the time step chosen for the fourth order Runge-Kutta integration of the update laws. An error criterion ξ was used to characterize the training: the lower the value of the error, the better is the training. The (output) error parameter is defined as

$$\xi := \sum_p \sum_n \epsilon_{pn}^2. \quad (3.19)$$

Note that this error parameter is exactly the Gaussian energy function. The weights of the network can be initialized by giving their values or selecting randomly by a random number generator of the UNIX system.

The training simulator, using the Runge-Kutta integration routine reads the input data from an input file, which provides the values of time step Δt , learning rate γ , error criterion ξ , stopping criterion ω , the network structure, the training patterns and the

corresponding target values. The training is terminated when either the stopping criterion ω or the error criterion ξ is achieved. The simulator writes the intermediate or/and the final values of the network weights or/and other parameters into an output file.

3.5 Simulation example: the XOR problem

The training performance of the two software simulators, using the Cauchy and the Gaussian energy functions, is described and a comparison is made in this section. A prototype FFANN used to solve the XOR problem had two input passive nodes (a passive node does not include a sigmoidal transfer function), two hidden nodes and one output node as shown in Figure 3.2.

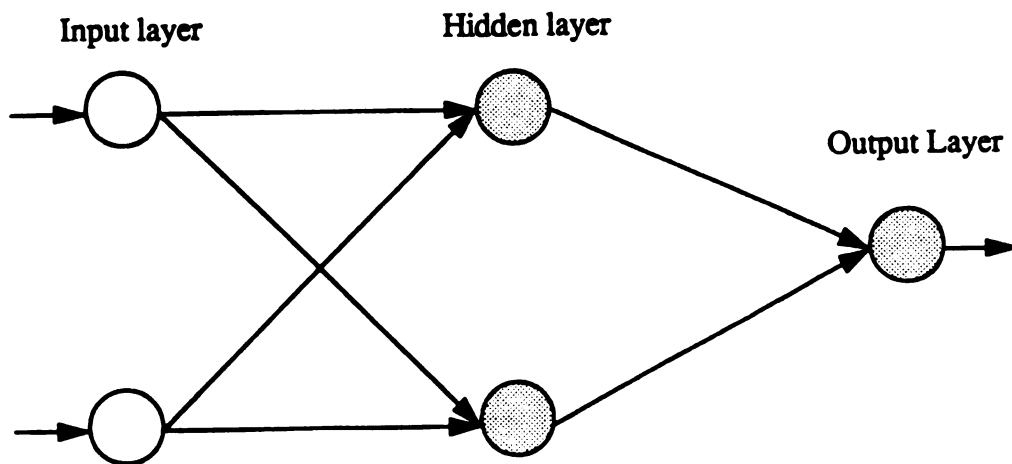


Figure 3.2: The network used to solve the XOR problem. The input layer has two passive nodes, whereas the hidden and the output layers have two and one artificial feedforward neurons, respectively.

The network was fully connected and weights were initialized randomly by a random number generator. The *seed*, which is a number used to start the random number generator, is used to represent the set of initial weights. The value of any initial weight w is restricted to be $-0.9 \leq w \leq 0.9$.

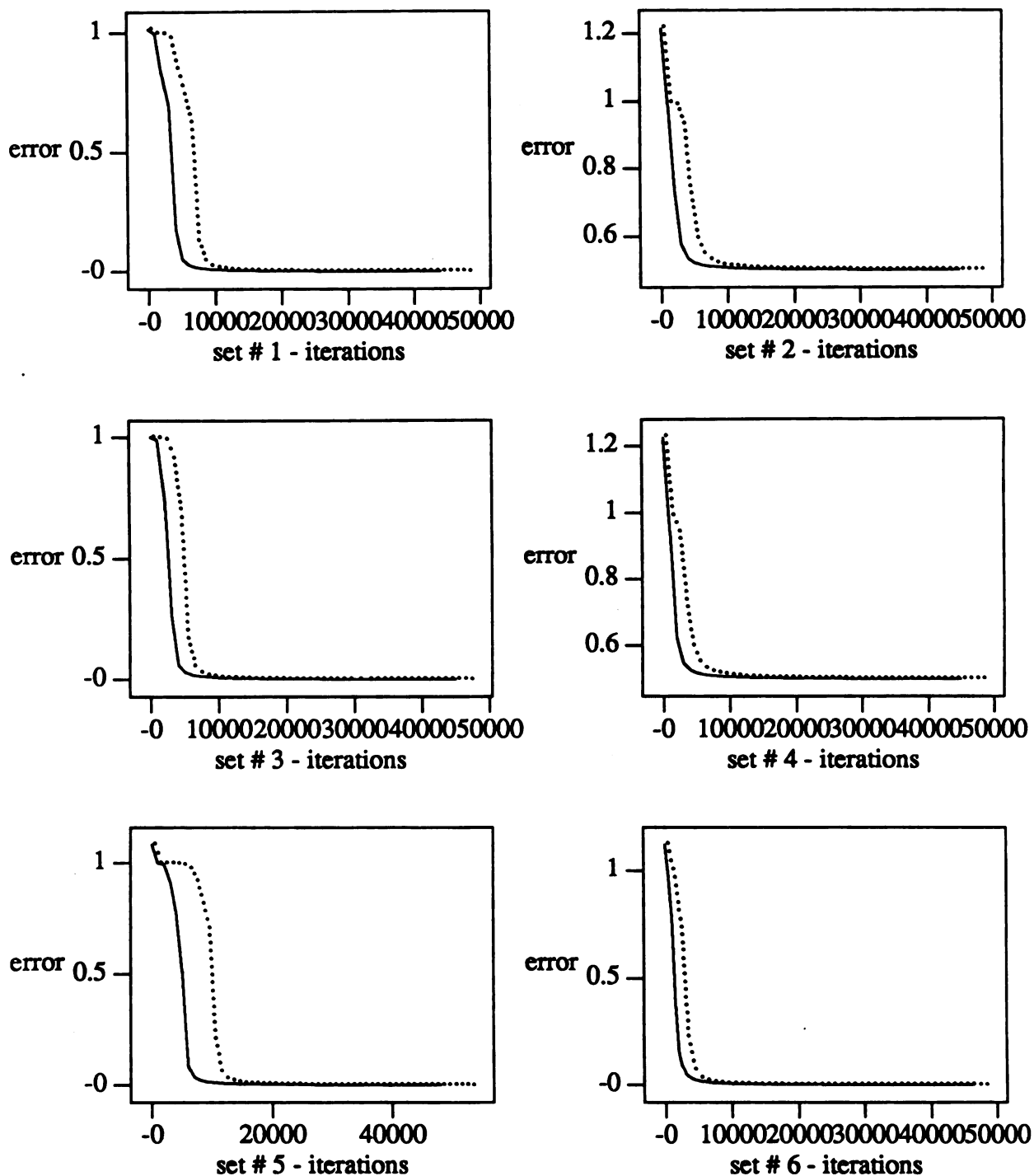


Figure 3.3: Comparison of the Gaussian (...) and the Cauchy (—) learning. In all of the above simulations, the learning algorithm using the Cauchy energy function achieves the same error with fewer iterations.

Table 3.1

set # of initial weights	energy function	intermediate data						final data		
		error	# of iterations	time(secs)	error	# of iterations	time(secs)	error	# of iterations	time(secs)
1	Gaussian Cauchy	0.005 0.005	18245 15102	105.77 101.21	0.00127 0.00127	48382 44895	281.63 301.43	0.00127 0.00127	48382 44895	281.63 301.43
2	Gaussian Cauchy	0.510 0.510	12595 8210	72.69 54.74	0.50165 0.50165	48683 34882	281.00 232.49	0.50165 0.50123	48683 45308	281.00 302.08
3	Gaussian Cauchy	0.005 0.005	16543 14356	95.99 96.24	0.00125 0.00125	47242 44988	275.17 302.00	0.00125 0.00124	47242 45016	275.17 302.50
4	Gaussian Cauchy	0.510 0.510	11833 7774	68.39 51.89	0.50164 0.50164	48265 34872	278.70 232.42	0.50164 0.50122	48265 45456	278.70 302.95
5	Gaussian Cauchy	0.005 0.005	22292 17149	129.19 114.89	0.00127 0.00127	53656 48461	312.27 325.40	0.00127 0.00125	53656 48842	312.27 328.15
6	Gaussian Cauchy	0.005 0.005	15397 13961	89.57 93.74	0.00121 0.00121	48271 46801	281.45 314.47	0.00121 0.00121	48271 46801	281.45 314.47

Training results comparing the Gaussian and the Cauchy learning. Final and intermediate data corresponding to update law (2.7) using the Gaussian and the Cauchy energy function, represents data when the error criterion ξ and the weight change criterion ω is achieved.

We assume that the network has converged when the stoppage criteria $\omega \leq 0.0001$ is satisfied. In all the simulation cases, the network converged to a local minimum. A comparison of 6 different sets of initial conditions is depicted in Figure 3.3. In all of the six simulations γ and Δt are chosen to be 1 and 0.1 seconds, respectively. For the cases when the weights were initialized by sets 1, 3, 5 and 6, the training did achieve excellent results. In the other cases when the weights were initialized by sets 2 and 4, the network was stuck in an undesirable local minimum with an error value equal to 0.5. The training results using these six different sets of initial conditions are summerized in Table 3.1.

In all six simulations the (Cauchy) learning algorithm achieves the same error performance as the Gaussian learning, in many fewer iterations. Finally, we also observe that the update law (3.5) outperforms the update law (2.14) in achieving any fixed error value in terms of the number of iterations required to realize that error value.

It may also be observed in Figure 3.3 that the bulk of the number of iterations is executed very near the local minimum. It may therefore be useful to set an error criterion ξ to a desirable value along with the stability criterion ω , for terminating the training.

3.6 Discussion

In this chapter, we derived a gradient descent weight update law using a Cauchy energy function for supervised error back-propagation learning for FFANNs. We presented a statistical rational for selecting the Cauchy energy function to be used in the weight update law. We also showed analytically that the learning performance may improve if the Cauchy energy function is used instead of the Gaussian energy function. This improvement is characterized in terms of speed of the continuous-time update law for the weights which results in fewer iterations in achieving a specified error (energy) value.

We observe that each component of the vector field of (3.16) is always greater than or equal to the amplitude of the corresponding component of the vector field of (2.14).

This means that every point in the weight state-space the dynamics of (3.16) are "faster" than the dynamics of (2.14). However this notion of "speed" is misleading: one cannot rely on the vector fields alone, one must consider the trajectories crossing into a well-defined common, constant-energy levels of the same function [40],[53]-[54]. This amounts to the trajectories of both dynamics crossing into regions defined by the same error function E .

Even though the proposed dynamics required more computations at every step, the total number of steps to converge is usually reduced. Due to the growing computations with the number of output nodes and the input patterns, it may not be suitable to use software implementation of the algorithm. The networks implemented with analog hardware can still derive full benefit from the learning speed of the proposed algorithm.

Numerous simulations focusing on the XOR problem, which support the above analysis, have been conducted. A sample of these simulations is included. The analysis indicates that the Cauchy learning dynamics are faster at each point in the weight space. On the other hand, the simulations focusing on the XOR problem seem to support a stronger conclusion: the Cauchy learning dynamics are faster along each trajectory in the weight space as compared to the Gaussian learning dynamics.

CHAPTER 4

THE POLYNOMIAL ENERGY FUNCTION

4.1 Motivation

The energy function in the weight update law (2.7) specifies the error of the output (from the desired values), to be propagated back for the adjustment of the weights. This error or the energy can be viewed as a mathematical norm. The usual sum-of-the-squared (Gaussian) energy function is the L_2 norm. Consequently we may choose other L_p norms ($1 \leq p \leq \infty$) [54].

For every error vector $\epsilon_p \in R^n, p = 1, 2, \dots, P$, we know that all the norms are equivalent in the sense that

$$K_1 \|\epsilon_p\|_\alpha \leq \|\epsilon_p\|_\beta \leq K_2 \|\epsilon_p\|_\alpha \quad (4.1)$$

Here K_1 and K_2 are any two constants and α and β are two norms. This equivalence does not imply that if, for example, the L_2 norm of an error vector ϵ_p is greater than the L_2 norm of another error vector ϵ_p^* , then the L_1 norm of ϵ_p is also greater than the L_1 norm of ϵ_p^* . This implication is a motivation to specify an energy function which when used in the weight update law (2.7), the error decreases in all the specified norms.

While error reduction in all the specified norms provide an intuitive rational for selecting the Polynomial energy function, the real rational for this work may stem from the convergence speed of the would-be-proposed weight update law as compared to the usual weight update law (2.6)-(2.7).

A learning rule which achieves the desired values of the weights (preferably the global minimum) by reducing the error in the first r norms, may use a Polynomial energy function [57] of order r given as

$$E^P = a \sum_p \sum_n (\epsilon_{pn} + b)^r, \quad (4.2)$$

where a and b are any suitable real constants. It should be noted that p used as superscript stands for polynomial and when used as a subscript stands for the pattern $p = 1, 2, \dots, P$. In this chapter we use the Polynomial energy function of order 2 and $a = b = \frac{1}{2}$ in the weight update law (2.7), to train a FFANN. The Polynomial energy function would then reduce to

$$E^P = \frac{1}{2} \sum_p \sum_n (\epsilon_{pn} + \frac{1}{2})^2. \quad (4.3)$$

It should be noted that the Polynomial energy function of order 2 (4.3) is basically a combination of L_1 and L_2 norms.

This chapter is organized as follows. In section 4.2, we analyze the learning dynamics of the weight update law (2.7), using the Polynomial energy function (4.3). In section 4.3, we analytically compare the learning dynamics due to the Gaussian (2.6) and the Polynomial (4.3) energy function. Software implementations of the learning dynamics are presented in section 4.4, which are then used to train a FFANN. As an illustrative example, we specialize the software to solve the XOR problem. Simulation results are reported in section 4.5. In section 4.6 we conclude with the discussion on various advantages and the disadvantages of using the Polynomial energy function in the weight update law (2.7).

4.2 The learning dynamics

In this section, the continuous-time learning dynamics of the update law (2.7), using the Polynomial energy function, are explicitly derived. As described earlier in section 2.2, the output layer is layer O , with nodes indexed by n . The $O-1th$ layer which feeds its output to the output layer is actually the last hidden layer H_m , whose nodes are indexed by m . The learning dynamics of the weights using the Polynomial energy

function (4.3) in the update law (2.7); specifically for the output layer

$$\dot{w}_{nm}^p = -\gamma \nabla_{w_{nm}} E^p, \quad (4.4i)$$

and for any hidden layer

$$\dot{w}_{kj}^p = -\gamma \nabla_{w_{kj}} E^p. \quad (4.4ii)$$

Using the chain rule, the dynamics for the output layer are

$$\dot{w}_{nm}^p = -\gamma \cdot \frac{\partial E^p}{\partial \epsilon_{pn}} \cdot \frac{\partial \epsilon_{pn}}{\partial y_{pn}} \cdot \frac{\partial y_{pn}}{\partial u_{pn}} \cdot \frac{\partial u_{pn}}{\partial w_{nm}}, \quad (4.5)$$

where u_{pn} and y_{pn} are respectively the net input and the output to the n th node of the output layer and are given by

$$u_{pn} = \sum_m w_{nm} z_{pm} + \theta_n \quad (4.6i)$$

and

$$y_{pn} = \frac{1}{1 + e^{-u_{pn}}}. \quad (4.6ii)$$

Here z_{pm} is the output of the m th node in the H_m th layer. θ_n is the threshold of the n th node in the output layer. The partial derivatives in equation (4.5) can be explicitly calculated as

$$\frac{\partial E^p}{\partial \epsilon_{pn}} = \sum_p \left[\epsilon_{pn} + \frac{1}{2} \right] \quad (4.7i)$$

$$\frac{\partial \epsilon_{pn}}{\partial y_{pn}} = -\text{sgn}(t_{pn} - y_{pn}) \quad (4.7ii)$$

$$\frac{\partial y_{pn}}{\partial u_{pn}} = y_{pn}(1 - y_{pn}), \quad (4.7iii)$$

and

$$\frac{\partial u_{pn}}{\partial w_{nm}} = z_{pm}. \quad (4.7iv)$$

Now the weight update law can be written as

$$\dot{w}_{hm}^p = \gamma \sum_p \delta[p][n][O] \cdot y[p][m][H_m], \quad (4.8)$$

where

$$\delta[p][n][O] := \text{sgn}(t_{pn} - y_{pn}) \cdot (\epsilon_{pn} + \frac{1}{2}) \cdot y_{pn}(1 - y_{pn}) \quad (4.9)$$

and $y[p][m][H_m]$ is the output z_{pm} of the m th node of the H_m th layer.

The weight update law for the weights w_{kj} , connecting the j th node of a hidden/input layer H_j to the k th node in the subsequent hidden layer H_k , is derived similarly and is given by

$$\dot{w}_{kj}^p = \gamma \sum_p \delta[p][k][H_k] \cdot y[p][j][H_j], \quad (4.10)$$

where

$$\delta[p][k][H_k] := \sum_l \delta[p][l][H_l] \cdot w_{lk} \cdot y_{pk} \cdot (1 - y_{pk}). \quad (4.11)$$

In (4.11), l is the index of the nodes in the H_l th layer to which node k feeds its output y_{lk} and w_{lk} is the weight of the corresponding connection. $y[p][j][H_j]$ is the output of the node j in the H_j th layer which is fed to node k in the H_k th layer.

4.3 Analytical Comparison with the Gaussian Energy Function

Error back-propagation learning with the update law (2.7) using the Gaussian energy function (2.6) is very widely accepted. In this section we compare the (Gaussian) learning (2.14) and the (Polynomial) learning (4.4). In order to do so, we use a similar criterion for the learning performance as discussed in subsection 3.3.1 and the comparison of the learning dynamics using the Polynomial and the Gaussian energy function is made in subsections 4.3.1 and 4.3.2.

4.3.1 The learning dynamics for a single pattern

In this subsection we consider a case when we only use one pattern. For ease of notation we will drop the subscript p and define the Gaussian (sum-of-the-squared) and the Polynomial energy functions as

$$E := \frac{1}{2} \sum_n \epsilon_n^2 \quad (4.12i)$$

and

$$E^P := \frac{1}{2} \sum_n \left(\epsilon_n + \frac{1}{2} \right)^2. \quad (4.12ii)$$

By straight forward calculations, equation (4.4i) can be rewritten as

$$\dot{w}_{hm}^p = -\gamma \left[\nabla_{w_{hm}} E + \nabla_{w_{hm}} E_1 \right] \quad (4.13i)$$

where

$$E_1 = \frac{1}{2} \sum_n \epsilon_n. \quad (4.13ii)$$

For the hidden layer the learning algorithm in (4.4ii) can be similarly rewritten as

$$\dot{w}_{kj}^p = -\gamma \left[\nabla_{w_{kj}} E + \nabla_{w_{kj}} E_1 \right]. \quad (4.13iii)$$

In order to formally compare the *learning performance* due to the Gaussian and the Polynomial energy functions, we state Theorem 4.1.

Theorem 4.1: Assume that Assumption 3.1 holds. Consider $V := E$ given by (4.12i), be a candidate Liapunov function. Let \dot{V} be the derivative of V along the trajectories of system (2.14) and \dot{V}^P be the derivative of V along the trajectories of system (4.13). Then the *learning performance* of system (4.13) is "faster" than or at least is the same as system (2.14).

The proof of this theorem presented in Appendix G, analytically shows that the *learning performance* improves if the Polynomial energy function is used in comparison with the usual sum-of-the-squared energy function.

4.3.2 The learning dynamics for multiple patterns

Now, we consider the learning dynamics of the weights for P patterns. The weight update law is given by equation (2.14) where E is given by (2.6). The learning dynamics of the weights due to the Polynomial energy function are in the form of (4.4) where E^P now is given by (4.3). By straight forward calculations, \dot{w}_{hm}^P can be written as

$$\dot{w}_{hm}^P = -\gamma \left[\nabla_{w_{hm}} E + \nabla_{w_{hm}} E_1 \right], \quad (4.14i)$$

where

$$E_1 = \frac{1}{2} \sum_p \sum_n \epsilon_{pn}. \quad (4.14ii)$$

Similarly, for the hidden layers the learning algorithm of the Polynomial energy function (4.3) can be written as

$$\dot{w}_{kj}^P = -\gamma \left[\nabla_{w_{kj}} E + \nabla_{w_{kj}} E_1 \right]. \quad (4.14iii)$$

When the network is trained with the multiple patterns, we formally compare the *learning performance* due to the Gaussian and the Polynomial energy function in Theorem 4.2.

Theorem 4.2: Assume that assumption 3.2 holds. Consider a candidate Liapunov function $V := E$ given by equation (2.6). Let \dot{V} be the derivative of V along the the trajectories of (2.14) and \dot{V}^P be the derivative of V along the trajectories of the system (4.14). Then the *learning performance* of system (4.14) is "faster" than or at least is the same as system (2.14).

Again the proof of this theorem is included in Appendix H. The analysis clearly indicate that the *learning performance* in "speed" can improve (or at least does not deteriorate) when the Polynomial energy function is used.

Proposition 4.1: The equilibria of system (4.14) are also the equilibria of system (2.14).

The proof of Proposition 4.1 is provided in Appendix I.

Remark: Proposition 4.1 implies that when we use the Polynomial energy function (4.3) in the weight update law (2.7), we still converge to one of the equilibria of system (2.14). However, this does not mean that starting from any initial condition, the trajectories followed by both the systems would be same or the trajectories would converge to the same equilibrium point.

4.4 Software implementation of the learning dynamics

For the software implementation of the learning dynamics of the weights connected to the output layer (4.8) and the weights connected to the hidden layers (4.10), the learning algorithm formalized in Table 2.1 is used. The weight update laws (4.8) and (4.10) are implemented into computer software using the C-language on Sun SPARC stations. Fourth order Runge-Kutta integration routine is used for the integration of these update laws.

A stopping criterion ω , was set to determine if the system has converged to a solution. The parameter ω was defined as

$$\omega := \sum_{w_{kj}} |\Delta w_{kj}|, \quad (4.15)$$

where w_{kj} are all the weights in the network and

$$\Delta w_{kj} := w_{kj}(t+\Delta t) - w_{kj}(t). \quad (4.16)$$

The term Δt is the time step chosen for the fourth order Runge-Kutta integration of the update laws. An error criterion ξ was used to characterize the training: the lower the value of the error, the better is the training. The (output) error parameter is defined as

$$\xi := \sum_p \sum_n \epsilon_{pn}^2. \quad (4.17)$$

Note that this error parameter is exactly the Gaussian energy function. The weights of the network can be initialized by giving their values or selecting randomly by a random number generator of the UNIX system.

The training simulator, using the Runge-Kutta integration routine reads the input data from an input file, which provides the values of time step Δt , learning rate γ , error criterion ξ , stopping criterion ω , the network structure, the training patterns and the corresponding target values. The training is terminated when either the stopping criterion ω or the error criterion ξ is achieved. The simulator writes the intermediate or/and the final values of the network weights or/and other parameters into an output file.

4.5 Simulation example: the XOR problem

The training performance of the three software simulators, using the Polynomial, the Cauchy and the Gaussian energy functions, is described and a comparison is made in this section. A prototype FFANN used to solve the XOR problem had two input passive nodes (a passive node does not include a sigmoidal transfer function), two hidden nodes and one output node as shown in Figure 3.2.

The network was fully connected and weights were initialized randomly by a random number generator. The *seed*, which is a number used to start the random number generator, is used to represent the set of initial weights. The value of any initial weight w is restricted to be $-0.9 \leq w \leq 0.9$.

We assume that the network has converged when the stoppage criteria $\omega \leq 0.0001$ is satisfied. In all the simulation cases, the network converged to a local minimum. A

comparison of 6 different sets of initial conditions is depicted in Figure 4.1. In all six simulations γ and Δt are chosen to be 1 and 0.1 seconds, respectively. For the cases when the weights were initialized by sets 1, 3, 5 and 6, the training did achieve excellent results. In the other cases when the weights were initialized by sets 2 and 4, the network was stuck in an undesirable local minimum with an error value equal to 0.5. The training results using these six different sets of initial conditions are summarized in Table 4.1.

It should be noted that in all six simulations the (Polynomial) learning algorithm achieves the same error performance as the Gaussian or the Cauchy learning, in many fewer iterations and computer time. Finally, we also observe that the update law (4.4) outperforms the update laws (2.14) and (3.5) in achieving any fixed error value in terms of the number of iterations required to realize that error value.

As the bulk of the number of iterations is executed very near the local minimum, it is therefore useful to set an error criterion ξ to a desirable value along with the stability criterion ω , for terminating the training. For the cases when the weights were initialized by the sets 1, 3, 5 and 6, ξ was set to a value of 0.005. In the other cases when the weights were initialized by sets 2 and 4, ξ was set to 0.51.

4.6 Discussion

Training of a FFANN with update law (2.7) using sum-of-the-squared energy function (2.6) ensures that the L_2 norm of the error vector ϵ for all the output nodes $n=1,2,\dots,N$ and all the patterns $p=1,2,\dots,P$, decreases as the training proceeds and finally converges to a local minimum value. While the L_2 norm of the error vector decreases, other L_p norms ($1 \leq p \leq \infty$), may increase or decrease.

Restricting our attention to the present case where we are using the Polynomial energy function of order 2, we only consider the L_1 and the L_2 norms. To illustrate the above fact, we consider a simple case of a trajectory in two dimensional error space in

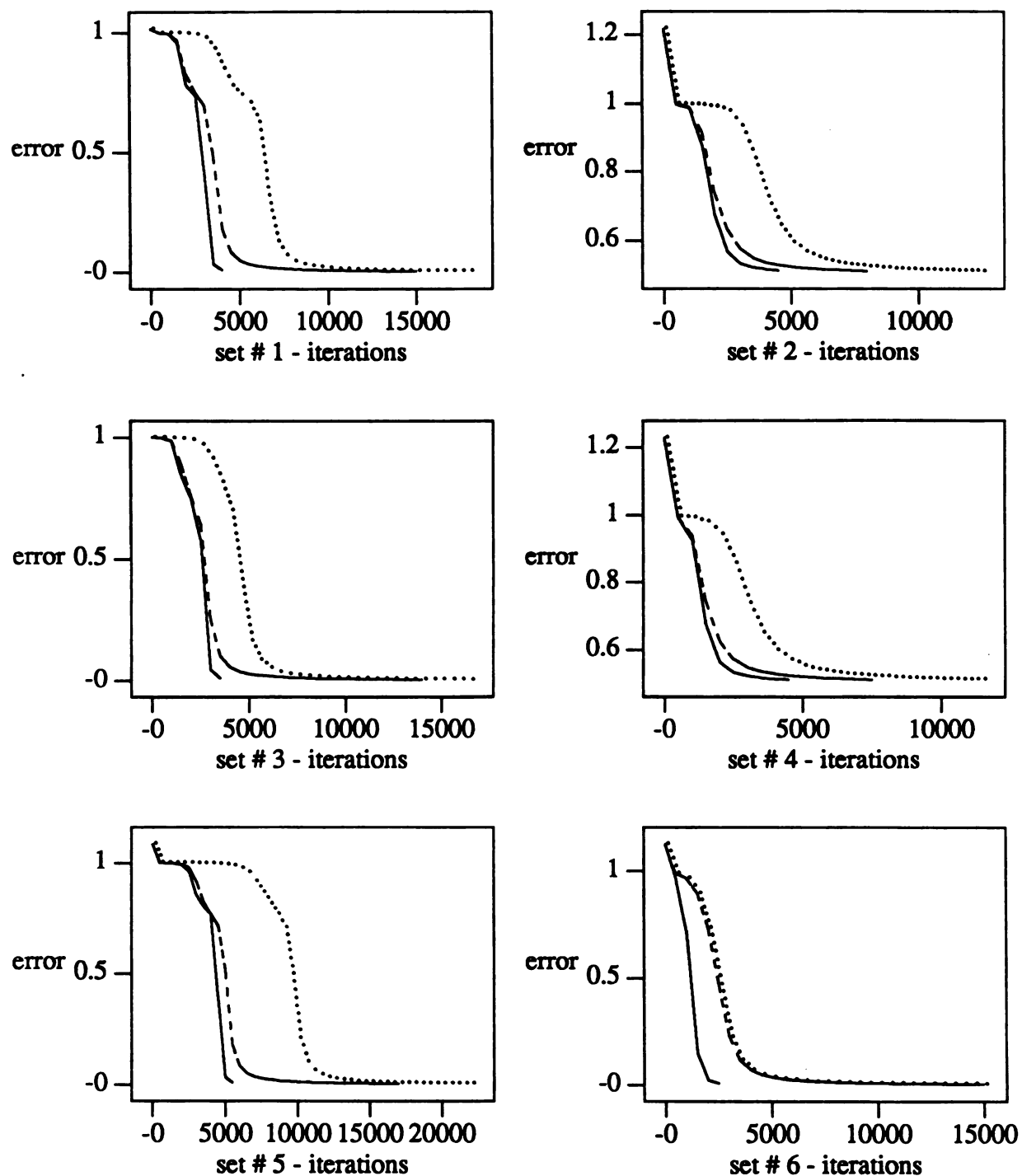


Figure 4.1: Comparison of the Gaussian (...), the Cauchy (---) and the Polynomial (___) learning. In all of the above simulations, the Polynomial learning algorithm achieves the same error with fewer iterations.

Table 4.1

set # of initial weights	energy function	intermediate data						final data		
		error	# of iterations	time(secs)	error	# of iterations	time(secs)	error	# of iterations	time(secs)
1	Gaussian	0.005	18245	105.77	0.00127	48382	281.63	0.00127	48382	281.63
	Cauchy	0.005	15102	101.21	0.00127	44895	301.43	0.00127	44895	301.43
	Polynomial	0.005	4387	25.98	0.00127	5742	34.06	0.000002	74045	442.33
2	Gaussian	0.510	12595	72.69	0.50165	48683	281.00	0.50165	48683	281.00
	Cauchy	0.510	8210	54.74	0.50165	34882	232.49	0.50123	45308	302.08
	Polynomial	0.510	4965	29.44	0.50165	18236	107.99	0.500454	58466	346.43
3	Gaussian	0.005	16543	95.99	0.00125	47242	275.17	0.00125	47242	275.17
	Cauchy	0.005	14356	96.24	0.00125	44988	302.00	0.00124	45016	302.50
	Polynomial	0.005	3938	23.34	0.00125	5302	31.48	0.000002	73312	437.78
4	Gaussian	0.510	11833	68.39	0.50164	48265	278.70	0.50164	48265	278.70
	Cauchy	0.510	7774	51.89	0.50164	34872	232.42	0.50122	45456	302.95
	Polynomial	0.510	4517	26.74	0.50164	17946	106.37	0.500456	57794	342.28
5	Gaussian	0.005	22292	129.19	0.00127	53656	312.27	0.00127	53656	312.27
	Cauchy	0.005	17149	114.89	0.00127	48461	325.40	0.00125	48842	328.15
	Polynomial	0.005	5873	34.78	0.00127	7265	43.09	0.000002	75603	451.48
6	Gaussian	0.005	15397	89.57	0.00121	48271	281.45	0.00121	48271	281.45
	Cauchy	0.005	13961	93.74	0.00121	46801	314.47	0.00121	46801	314.47
	Polynomial	0.005	2829	16.83	0.00121	4307	25.66	0.000002	75057	448.39

Training results comparing the Gaussian, the Cauchy and the Polynomial learning. Final and intermediate data corresponding to update law (2.7) using the Gaussian, the Cauchy and the Polynomial energy function, represents data when the error criterion ξ and the weight change criterion ω is achieved.

Figure 4.2. If the training starts from point ϵ_i and proceed along the trajectory and settles down at a local minimum ϵ_f , we observe that while the L_2 norm of the error vector decreases the L_1 norm increases throughout this learning.

We can quantify the possibility of increase in the L_1 norm while the L_2 norm decreases at a point, in terms of the shaded area in Figure 4.2; the larger the area, the higher is the possibility. Similarly if we train the network, by decreasing the L_1 norm of the error vector ϵ , we will notice as depicted in Figure 4.3, that the L_2 norm of the errors may increase. We can again quantify the possibility of increasing the L_2 norm while the L_1 norm decreases at a point, by measuring the shaded area in Figure 4.3; the larger the area, the higher is the possibility. If ϵ_i is a point, where the L_1 norm is tangent to the L_2 norm (i.e) the shaded area measures zero, then possibility of a norm increasing while the other norm is decreasing reduces to zero at that point. This is depicted in Figure 4.4.

This criterion can be easily extended to an n-dimensional error space, where we have an n-dimensional sphere representing the L_2 norm and n-1 dimensional surface representing the L_1 norm. The enclosed volume between the two norms in the direction of the trajectory, measures the possibility of increasing the output error in one norm while the error decreases in the other norm. This quantification is another aspect of this research which can be pursued further independently.

When a network is trained with the Polynomial energy function, we decrease the error in the L_1 as well as the L_2 norm. We observe from Figure 4.5 that we reduce the possibility of any norm increasing while the other is decreasing with the measure given above.

Giving more weight to any norm in the energy function, will enhance the tendency of decreasing that specific norm more than the other. In other words we enhance the possibility of other norm to increase while the first is decreased. This is illustrated by the following example, in which we give more weight to the L_1 norm in the energy function. We use the modified Polynomial energy function E^{mp} , which is

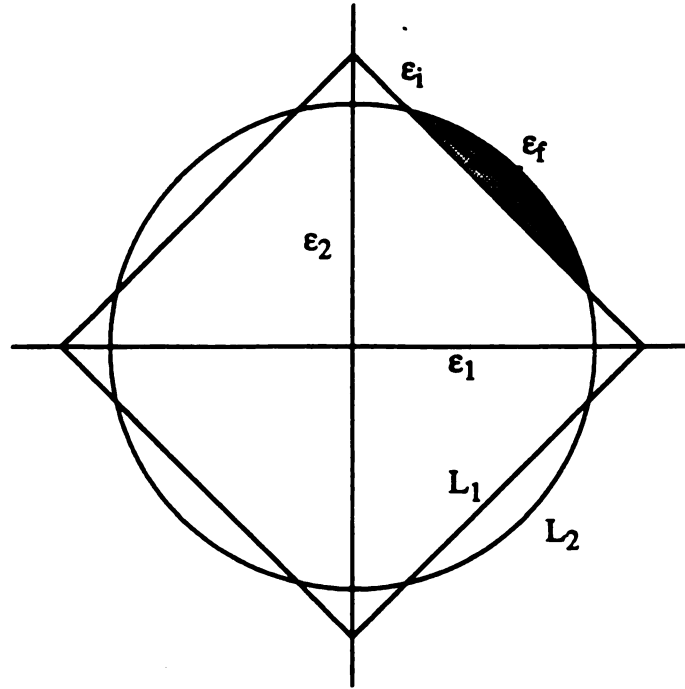


Figure 4.2: Trajectory showing the increase in L_1 norm while L_2 norm is decreasing. ϵ_i and ϵ_f represent the initial and the final points of a trajectory in the error space. The size of the shaded area represents the possibility of L_1 norm increasing while L_2 norm is decreasing.

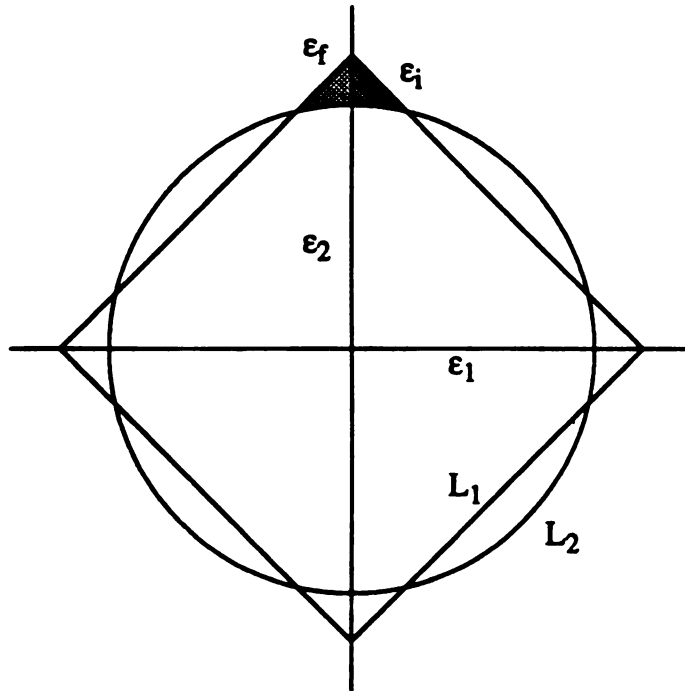


Figure 4.3: Trajectory showing the increase in L_2 norm while L_1 norm is increasing. ϵ_i and ϵ_f represent the initial and the final points of a trajectory in the error space. The size of the shaded area represent the possibility of L_2 norm increasing while L_1 norm is decreasing.

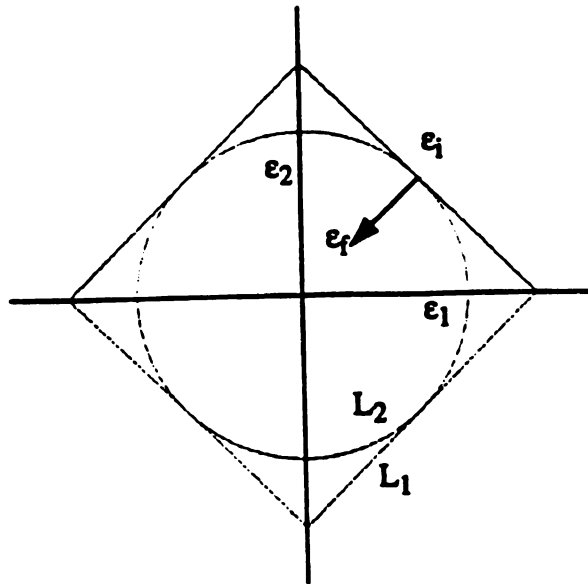


Figure 4.4: Trajectory showing both the L_1 and the L_2 norms decreasing. e_i and e_f represent the initial and the final points of a trajectory in the error space. The L_1 is tangent to the L_2 norm at the initial point of the trajectory, indicating that both the norms of the error vector decrease as the training proceeds along the trajectory.

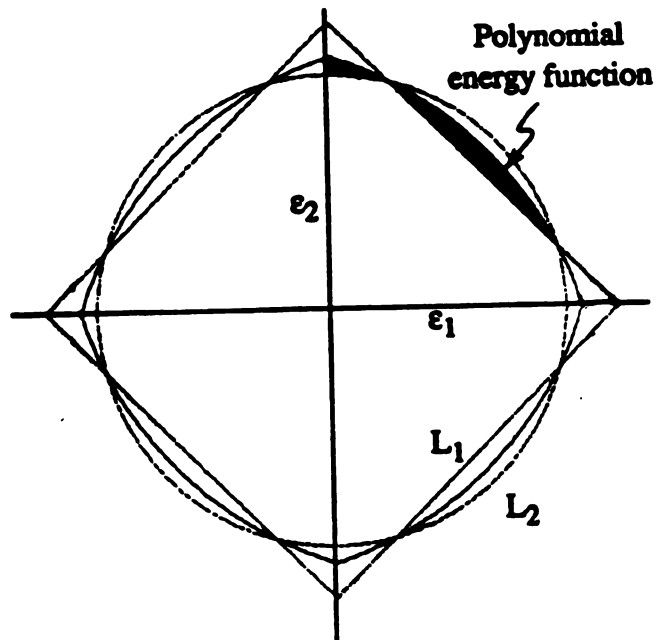


Figure 4.5: Energy level diagram of the Polynomial energy function. The network trained with the Polynomial energy function reduces the possibility of increasing the error in one norm while it is decreasing in the other (L_1 or L_2).

$$E^{mp} = \frac{1}{2} \sum_p \sum_n (\epsilon_{pn} + 1)^2. \quad (4.18)$$

When the network is trained with the weight update law (2.7) using the modified Polynomial energy function and the initial weights are given by set 5, we notice in Figure 4.6 that at some stage the error which is the L_2 norm starts to increase. Also notice that the L_1 norm and the modified energy of the errors keep decreasing throughout this training till the local minimum is achieved.

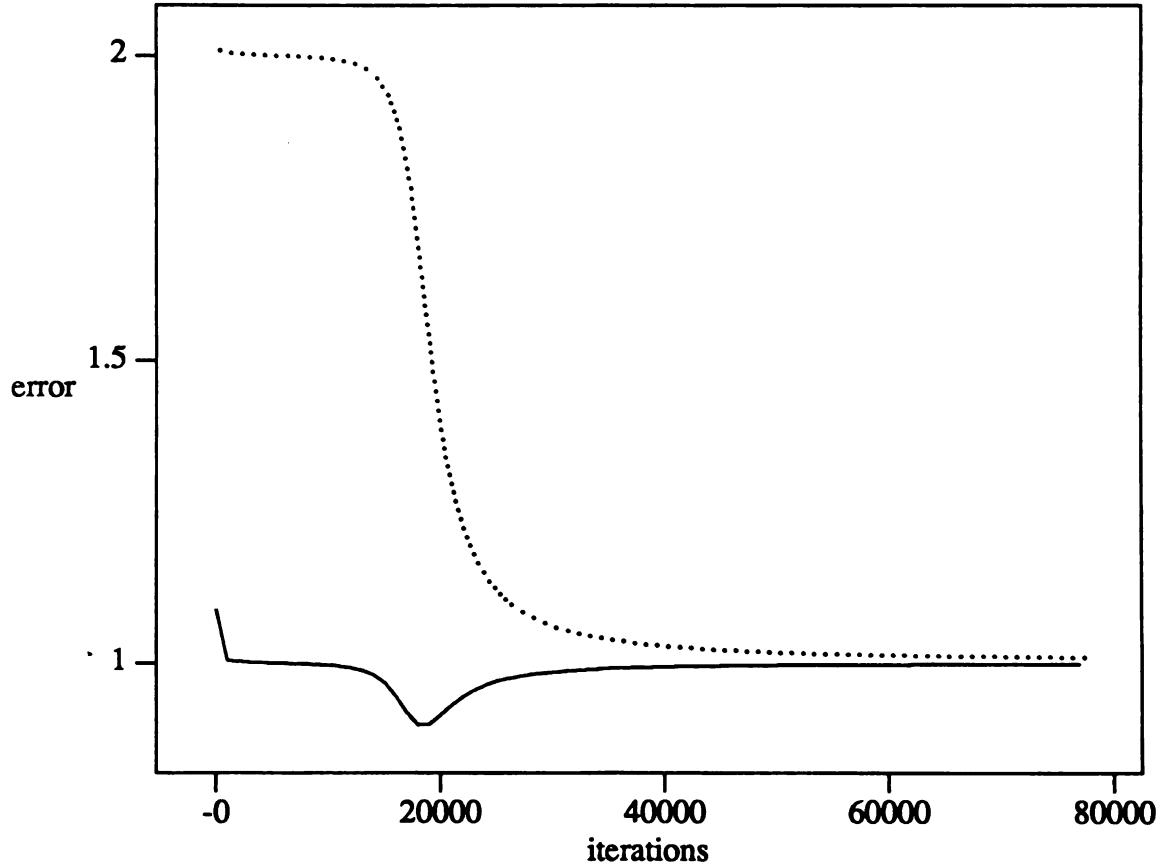


Figure 4.6: Training results of the XOR problem, trained with the modified Polynomial energy function. L_1 and L_2 norm values are represented by dotted and solid lines respectively. Note that the L_2 norm of the error increases while the L_1 norm decreases after 18000 iterations.

Besides the main advantage of increased speed of convergence in terms of the number of iterations and less time, we can pick an energy function that lowers the value of the errors in the desired norms, to suit our requirements. Using the Polynomial energy function in order to train a FFANN in comparison with the usual sum-of-the-squared energy function is thus of a threefold advantage: (1) speed of convergence in terms of computer training cycles, (2) time and (3) the choice to decrease the output error in any particular norm or a combination of norms.

CHAPTER 5

THE EXPONENTIAL ENERGY FUNCTION

5.1 Motivation

In chapter 2 and 3 respectively, we used the Cauchy (3.4) and the Polynomial (4.3) energy functions in the weight update law (2.7), to speed up convergence to a local minimum. Using the Cauchy energy function ensures convergence to a local minimum in fewer steps in comparison with the Gaussian energy function. Due to more computations in each step, the Cauchy learning may take more computer time in comparison with the Gaussian learning.

When we used the Polynomial energy function (4.3) in the update law (2.7), the algorithm not only outperformed the Gaussian and the Cauchy learning in terms of number of steps but also in computer time.

We speed up the convergence to a local minimum by using the Cauchy or the Polynomial energy function in the continuous-time update law. The local minimum that we converge to, may not be good enough to solve the problem for which the network is trained. For instance, in the case of the XOR problem when we initialize the weights with set 2 and 4, we do not converge to good minima.

In this chapter, we formalize the skipping minima that are at a higher energy value and converging to an acceptable minimum at a lower energy value. Many researchers have proposed various modifications of the discrete-time update law (2.5) based on heuristic arguments, to speed up the convergence to acceptable weights [7],[8],[58],[59]. We propose an exponential energy function [60] of the form

$$F = e^{\kappa E}, \tag{5.1}$$

to be used in the weight update law (2.7) to overcome the above mentioned problems. Here κ is any positive real number whose parameterization is discussed later in this chapter. When we use the Exponential energy function (5.1) in the continuous-time update law (2.7), the system still is a gradient descent with all of its nice properties.

In section 5.2 we include the learning dynamics of the update law (2.7) using the Exponential energy function F . We take discrete steps when we implement the update law (2.7) via software on a digital computer. This feature adds the capability of skipping the undesirable local minima. We present explicit derivations of the learning algorithm based on the Exponential energy function in this section. In section 5.3, we compare the learning dynamics of the weight update law (2.7) when it uses the Exponential energy function F and a simple energy function E . E can be the Gaussian, the Cauchy, the Polynomial or any other suitable energy function. Software implementations of these derivations are then used to train a FFANN. As an illustrative example we specialize the software to solve the XOR problem in section 5.4. Simulation results are presented in section 5.5. We conclude by finally discussing the advantages and the limitations in section 5.6.

5.2 The learning dynamics

Consider the learning dynamics of the weights using the simple energy function, which may be the Gaussian (2.6), the Cauchy (3.4) or the Polynomial (4.3) energy function; specifically

$$\dot{w} = -\gamma \nabla_w E, \quad (5.2)$$

where w represents all the weights in the network. Now consider the learning dynamics of the weights using the Exponential energy function (5.1); specifically

$$\dot{w}^e = -\gamma \nabla_w F, \quad (5.3i)$$

$$= -\gamma \kappa e^{\kappa E} \nabla_w E, \quad (5.3ii)$$

$$= -\gamma \chi \nabla_{\mathbf{w}} E, \quad (5.3\text{iii})$$

$$= \chi \dot{\mathbf{w}}, \quad (5.3\text{iv})$$

where χ is the *skipping factor* and is defined as

$$\chi := \kappa e^{\kappa E}. \quad (5.4)$$

Observe that χ is constant at any particular step of the integration and varies exponentially with the output error (energy) from one step to another. As the learning proceeds, the output error (energy) E decreases, the skipping factor χ also decreases exponentially. Therefore the likelihood of skipping the local minima which are at higher energy levels are greater than skipping the local minima at lower energy levels.

Here $\dot{\mathbf{w}}$ gives the learning dynamics for the update law (5.2). These learning dynamics have already been explored in chapters 2, 3 and 4 corresponding to the Gaussian, the Cauchy and the Polynomial energy functions.

5.3 Analytical Comparison with the Simple Energy Function

In this section, we compare the Exponential learning (5.3), with the update law (2.7) using the simple energy function (2.6), (3.4) or (4.3), already described in earlier chapters. In order to do so, we use a similar criterion for the learning performance as discussed in subsection 3.3.1 and the comparison of the learning dynamics is made in subsection 5.3.1.

5.3.1 Comparison of the learning dynamics

We analyze the *learning performance* of weight update law (2.7) using the Exponential energy function by comparing the learning dynamics of systems (5.2) and (5.3) with the criterion outlined in subsection 5.3.1. In order to formally compare the *learning performance* due to a simple and the Exponential energy functions, we state Theorem 5.1.

Theorem 5.1:

Consider a candidate Liapunov function $V := E$ given by equation (2.6), (3.4) or (4.3) corresponding to the Gaussian, the Cauchy or the Polynomial energy function. Let \dot{V} be the derivative of V along the the trajectories of (5.2) and \dot{V}^e be the derivative of V along the trajectories of the system (5.3), then the *learning performance* of system (5.3) is better than or at least the same as system (5.2), provided the skipping factor χ given by (5.4) is greater than or equal to unity.

Proof :

In order to analyze the learning performance, let's evaluate $\dot{V}^e - \dot{V}$ as given below.

$$\begin{aligned} \dot{V}^e - \dot{V} &= \nabla_w V \cdot (\dot{W}^e - \dot{W}) \\ &= [\nabla_w E] \cdot \left\{ \chi \begin{bmatrix} \dot{w}_1 \\ \vdots \\ \dot{w}_n \end{bmatrix} - \begin{bmatrix} \dot{w}_1 \\ \vdots \\ \dot{w}_n \end{bmatrix} \right\}, \end{aligned} \quad (5.5i)$$

$$= -\gamma \sum_w [\chi - 1] \left[\nabla_w E \right]^2. \quad (5.5ii)$$

In order for the update law (2.7) using the Exponential energy function (5.1) to improve (or at least does not deteriorate) the *learning performance*, $\dot{V}^e - \dot{V}$ should be non-positive. This implies

$$\chi \geq 1. \quad (5.6)$$

To improve the *learning performance*, the relationship (5.6) imposes the following restriction on the energy function.

$$E \geq -\frac{\ln \kappa}{\kappa}. \quad (5.7)$$

The above relationship is depicted in Figure 5.1. We observe that for non-positive κ the system does not behave as a gradient descent system. For positive κ the *learning performance* only improves if for a certain value of κ , E has a value that satisfies the relationship (5.7). We also observe that if $\kappa \geq 1$, the learning with the Exponential energy function always improve. If for some reason we want faster learning for some higher energy value greater than E_0 and slower learning for $E < E_0$, then we can chose appropriate value of κ such that $0 < \kappa < 1$.

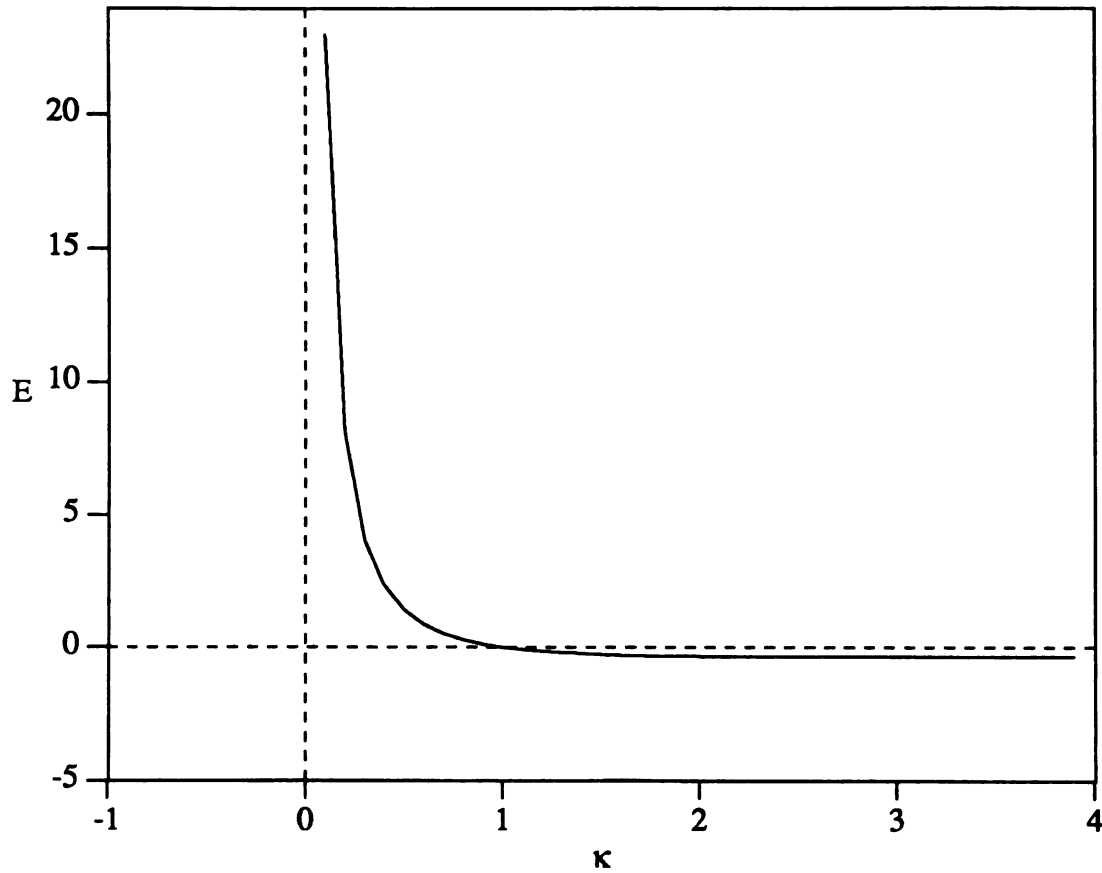


Figure 5.1: Graph representing the relationship (5.7). It is an essential condition to improve the *learning performance* of system (5.3). The *learning performance* only improves if the parameters are chosen above the graph line representing (5.7).

Proposition 5.1: The equilibria of system (5.2) and system (5.3) are same.

The proof of Proposition 5.1 is provided in Appendix J.

Remark: Proposition 5.1 implies that when we use Exponential learning (5.3) we still converge to one of the equilibria of system (5.2). However, this does not mean that starting from any initial condition, the trajectories followed by the systems would be same or the trajectories would converge to the same equilibrium point.

5.4 Software implementation of the learning dynamics

For the software implementation of the weight update law (5.3), the learning algorithm formalized in Table 2.1 is used. The weight update law (5.3) is implemented into computer software using the C-language on Sun SPARC stations. Fourth order Runge-Kutta integration routine is used for the integration of these update laws.

A stopping criterion ω , was set to determine if the system has converged to a solution. The parameter ω was defined as

$$\omega := \sum_w |\Delta w|, \quad (5.8)$$

where w are all the weights in the network and

$$\Delta w := w(t + \Delta t) - w(t). \quad (5.9)$$

The term Δt is the time step chosen for the fourth order Runge-Kutta integration of the update laws. An error criterion ξ was used to characterize the training: the lower the value of the error, the better is the training. The (output) error parameter is defined as

$$\xi := \sum_p \sum_n \epsilon_{pn}^2. \quad (5.10)$$

Note that this error parameter is exactly the Gaussian energy function. The weights of the network can be initialized by giving their values or selecting randomly by a

random number generator of the UNIX system.

The training simulator, using the Runge-Kutta integration routine reads the input data from an input file, which provides the values of time step Δt , learning rate γ , error criterion ξ , stopping criterion ω , the energy function parameter κ , the network structure, the training patterns and the corresponding target values. The training is terminated when either the stopping criterion ω or the error criterion ξ is achieved. The simulator writes the intermediate or/and the final values of the network weights or/and other parameters into an output file.

5.5 Simulation example: the XOR problem

The training performance of the software simulators, using the Exponential and the simple energy functions, is described and a comparison is made in this section. For the simple energy functions, we used the Gaussian (2.6), the Cauchy (3.4) and the Polynomial (4.3) energy functions. A prototype FFANN used to solve the XOR problem had two input passive nodes (a passive node does not include a sigmoidal transfer function), two hidden nodes and one output node as shown in Figure 3.2.

The network was fully connected and weights were initialized randomly by a random number generator. The *seed*, which is a number used to start the random number generator, is used to represent the set of initial weights. The value of any initial weight w is restricted to be $-0.9 \leq w \leq 0.9$.

We assume that the network has converged when the stopping criterion $\omega \leq 0.0001$ is satisfied. In all the simulation cases, the network converged to a local minimum, when we chose energy function parameter $\kappa = 1$. A comparison of 6 different sets of initial conditions are depicted in Figures 5.2, 5.3 and 5.4. In all six simulations γ and Δt are chosen to be 1 and 0.1 seconds, respectively. For the cases when the weights were initialized by sets 1, 3, 5 and 6, the training did achieve excellent results. In the other cases when the weights were initialized by sets 2 and 4, the network was stuck in an

undesirable local minimum with an error value equal to 0.5. The training results using these six different sets of initial conditions are summarized in Tables 4.1 and 5.1.

We increase the tendency of skipping the local minima in order to avoid the cases that we observe when the weights are initialized by sets 2 and 4, by choosing a larger skipping factor. This is achieved by selecting $\kappa = 1.8$, which subsequently increases the skipping factor χ . Comparison of the six different sets of initial conditions are made in Figures 5.2, 5.3 and 5.4. The training results of these simulations are summarized in Table 5.2.

We observe in Figure 5.2, 5.3 and 5.4 that κ was chosen to be 1.8, the update law (2.7) using the Exponential of the Polynomial skips the local minima, when the weights were initialized by sets 2 and 4. For the same two sets of initial weights, the update law using the Exponential of the Gaussian and the Cauchy energy functions, still could not skip the local minima. It can be shown easily, that by increasing the value of κ , we can skip these undesirable minima also.

In all of the simulations, observe that the Exponential (5.3) learning achieves the same error criterion as the simple (5.2) learning in many fewer iterations and computer time. Depending upon the value of κ or the skipping factor χ , the Exponential learning algorithm can be made to skip the undesirable local minima, thus allowing it to converge to one of the desirable minima.

As the bulk of the number of iterations is executed very near the local minimum, it is therefore useful to set an error criterion ξ to a desirable value along with the stability criterion ω , for terminating the training. For the XOR problem the error criterion $\xi = 0.005$ provides reasonably good results and it will save us lot of computer steps and time.

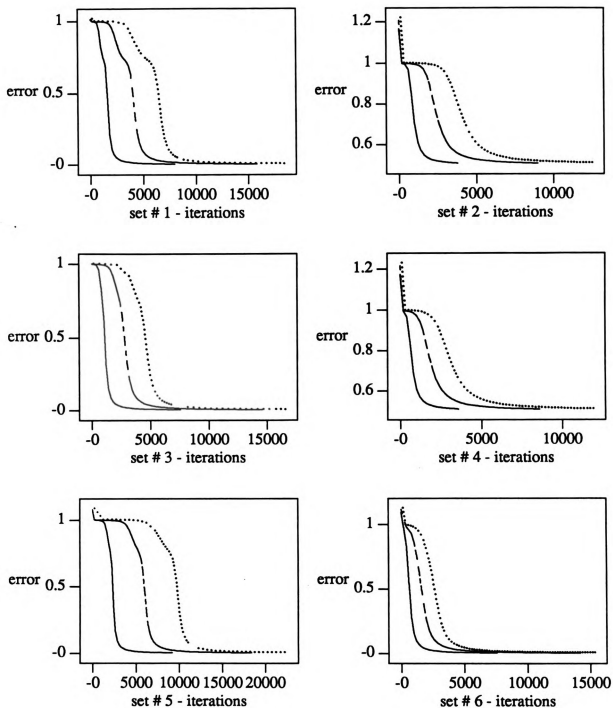


Figure 5.2: Comparison of the Exponential ($\kappa=1, 1.8$) and the Gaussian learning. The Gaussian is represented by (...) whereas the Exponential ($\kappa=1, 1.8$) is represented by (---) and (___) respectively.

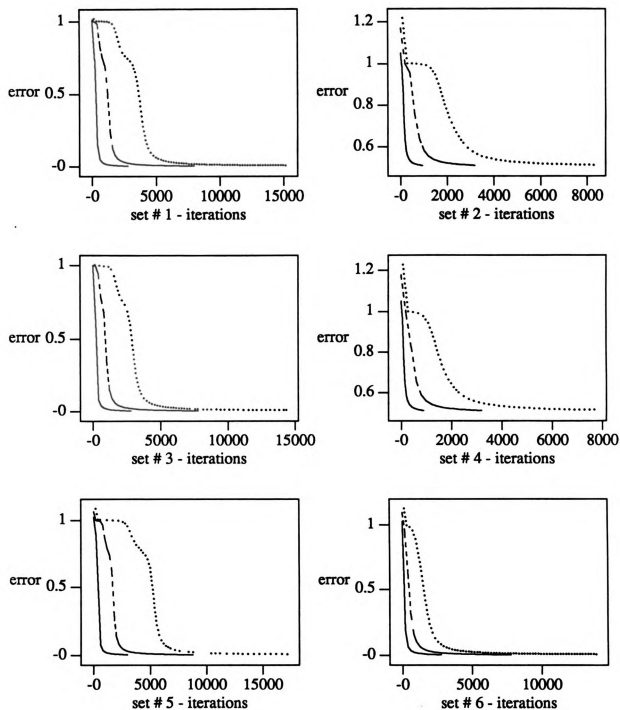


Figure 5.3: Comparison of the Exponential ($\kappa=1, 1.8$) and the Cauchy learning. The Cauchy is represented by (...) whereas the Exponential ($\kappa=1, 1.8$) is represented by (---) and (—) respectively.

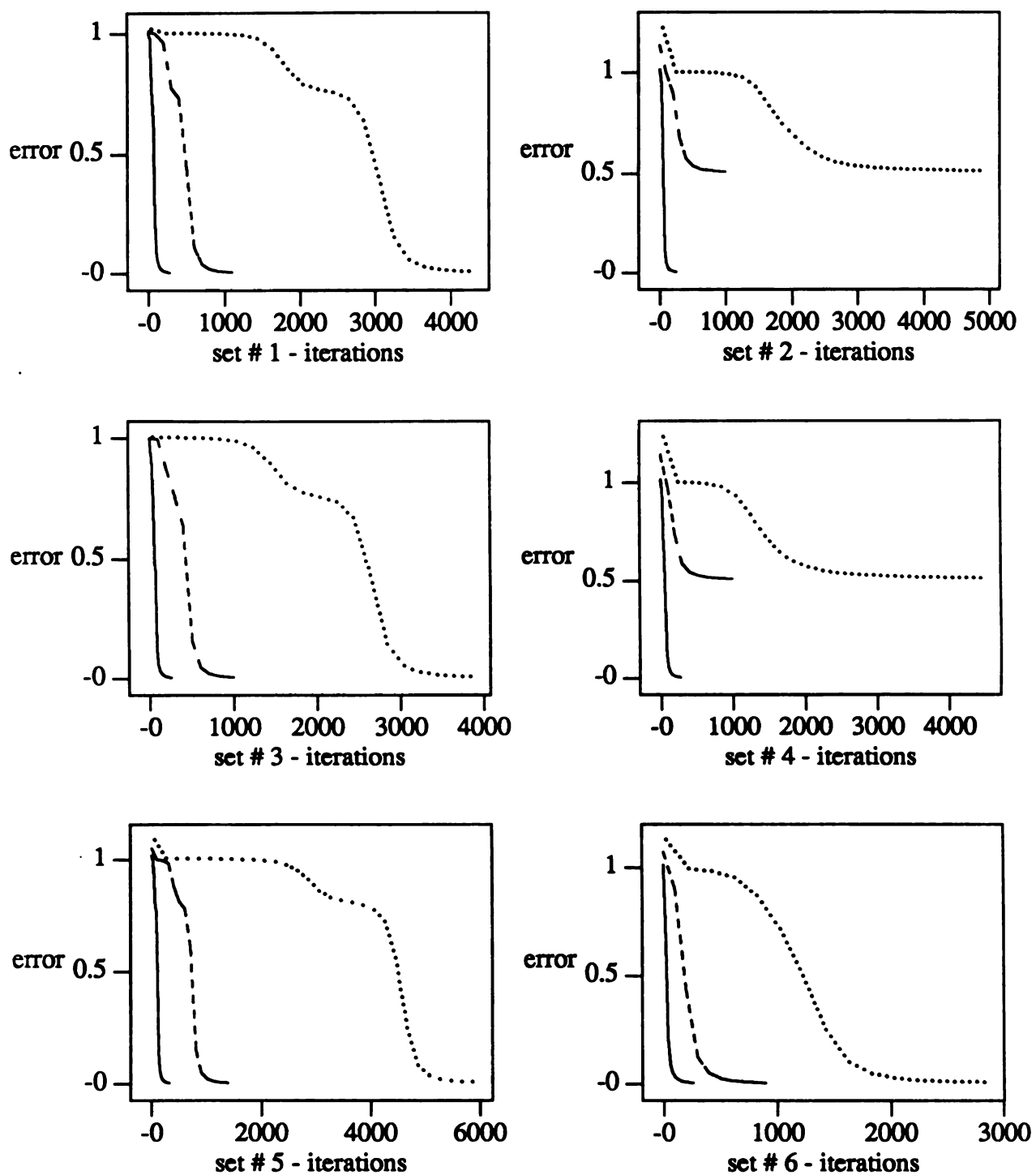


Figure 5.4: Comparison of the Exponential ($\kappa=1, 1.8$) and the Polynomial learning. The Polynomial is represented by (...) whereas the Exponential ($\kappa=1, 1.8$) is represented by (---) and (___) respectively.

Table 5.1

set # of initial weights	energy function	intermediate data						final data		
		error	# of iterations	time(secs)	error	# of iterations	time(secs)	error	# of iterations	time(secs)
1	Gaussian	0.005	15813	130.89	0.00127	45939	379.38	0.00127	45939	379.38
	Cauchy	0.005	8130	62.09	0.00127	26263	195.74	0.00076	41829	311.73
	Polynomial	0.005	1197	11.83	0.00127	1979	18.94	0.000001	70003	671.55
2	Gaussian	0.510	9045	74.53	0.50165	37226	304.40	0.50128	46600	381.26
	Cauchy	0.510	3390	25.69	0.50165	15571	115.12	0.50056	42537	314.37
	Polynomial	0.510	1092	10.54	0.50165	4864	46.29	0.50013	53741	509.72
3	Gaussian	0.005	14729	122.84	0.00125	45419	375.25	0.00125	45419	375.25
	Cauchy	0.005	7869	60.44	0.00125	26426	196.99	0.00076	41836	311.90
	Polynomial	0.005	1100	10.69	0.00125	1887	18.06	0.000001	70171	684.03
4	Gaussian	0.510	8611	72.29	0.50164	37073	303.50	0.50128	46290	378.65
	Cauchy	0.510	3263	25.28	0.50164	15637	115.66	0.50056	42530	314.28
	Polynomial	0.510	1027	10.40	0.50164	4845	46.06	0.500131	54373	515.74
5	Gaussian	0.005	18535	152.52	0.00127	49889	411.98	0.00127	49889	411.98
	Cauchy	0.005	8915	67.26	0.00127	27877	207.57	0.00076	43933	327.25
	Polynomial	0.005	1422	13.98	0.00127	2227	21.28	0.000001	71344	673.07
6	Gaussian	0.005	14291	118.44	0.00121	47151	389.83	0.00121	47151	389.83
	Cauchy	0.005	7961	60.91	0.00121	27928	208.39	0.00073	44413	331.20
	Polynomial	0.005	964	9.56	0.00121	1816	17.43	0.000001	73658	706.68

Training results comparing the Exponential ($\kappa = 1$) of the Gaussian, the Cauchy and the Polynomial learning. Final and intermediate data using the Exponential ($\kappa = 1$) of the Gaussian, the Cauchy and the Polynomial energy function, represents data when the error criterion ξ and the preset stoppage criterion ω is achieved.

Table 5.2

set # of initial weights	energy function	intermediate data						final data		
		error	# of iterations	time(secs)	error	# of iterations	time(secs)	error	# of iterations	time(secs)
1	Gaussian	0.005	8095	70.59	0.00127	24904	217.89	0.000709	42125	366.90
	Cauchy	0.005	2866	22.98	0.00127	9678	77.18	0.000291	39091	311.62
	Polynomial	0.005	297	2.99	0.00127	580	5.83	0.000000	67921	680.68
2	Gaussian	0.510	3914	33.78	0.50165	16811	144.91	0.500589	43265	372.66
	Cauchy	0.510	955	7.59	0.50165	4614	36.53	0.500170	40400	319.68
	Polynomial	0.510	60	0.61	0.50165	60	0.61	0.000000	67939	681.38
3	Gaussian	0.005	7676	66.71	0.00125	24834	216.09	0.000693	42632	371.06
	Cauchy	0.005	2803	22.34	0.00125	9774	77.93	0.000286	39696	316.33
	Polynomial	0.005	277	2.79	0.00125	562	5.64	0.000000	67463	676.33
4	Gaussian	0.510	3760	32.48	0.50164	16784	144.69	0.500585	43396	373.60
	Cauchy	0.510	929	7.39	0.50164	4646	36.76	0.500168	40932	323.92
	Polynomial	0.510	56	0.58	0.50164	57	0.59	0.000000	70274	704.65
5	Gaussian	0.005	9244	80.23	0.00127	26800	233.09	0.000705	44921	391.01
	Cauchy	0.005	3089	24.61	0.00127	10212	81.39	0.000288	41039	327.12
	Polynomial	0.005	328	3.31	0.00127	619	6.21	0.000000	69983	701.45
6	Gaussian	0.005	7630	66.38	0.00121	26054	226.80	0.000676	44523	378.51
	Cauchy	0.005	2893	23.08	0.00121	10394	82.89	0.000277	41854	333.50
	Polynomial	0.005	274	2.79	0.00121	583	5.88	0.000000	70388	704.53

Training results comparing the Exponential ($\kappa = 1.8$) of the Gaussian, the Cauchy and the Polynomial learning. Final and intermediate data using the Exponential function ($\kappa = 1.8$) of the Gaussian, the Cauchy and the Polynomial energy functions, represents data when the error criterion ξ and the weight change criterion ω is achieved.

5.6 Discussion

Weight update law (2.7) using the Exponential energy function (5.1) corresponding to the Gaussian (2.6), the Cauchy (3.4) or the Polynomial (4.3) energy function provides a variable skipping factor χ , which changes with each computer step. The learning dynamics of this system is given by (5.3iii) as

$$\dot{w}^e = -\gamma \chi \nabla_w E.$$

If we consider $\gamma \chi$ as the *cumulative learning rate*, then we can say that the learning rule (5.3iii) provides a variable learning rate, which varies with each computer step.

The changes in the *cumulative learning rate* are such that it is large at higher energy levels and is small at lower energy levels. In digital software based networks, like the one we used in this work, the concept of variable learning rate is very useful. The reason is that, we want the undesirable minima at higher energy levels to be skipped whereas it should converge to desirable minima at lower energy levels.

As the calculations in section 5.3 reveal, the weight update law (2.7) using the Exponential energy function contributes to overall better *learning performance* as compared to when the Gaussian, the Cauchy or the Polynomial energy function is used. This is observed practically in all of our simulations with the XOR problem. Some of these simulations are reflected in Figures 5.2, 5.3 and 5.4.

Apparently, we can make the learning as fast as desired, by choosing appropriate values of κ . This is misleading because if we increase the values of the skipping factor beyond certain limit (which varies from problem to problem), the weights change so abruptly that the outputs of the network clamp to saturation (0 or 1) values. As a result the partial derivatives of the outputs with respect to the corresponding inputs of the neurons are very very small. Consequently, the *error signal* (2.19), (3.10) and (4.9) which is propagated back is negligible, resulting in a smaller weight change. This makes the algorithm converge to a *false minimum*. This can be avoided to a certain extent by using

lower gain neurons as described in section 2.1, or using appropriate cost factor. Alternatively we restrict the value of κ to a certain limit.

When the weights are initialized randomly within the range $[-0.9, 0.9]$ (as is the case in all of the simulations), and value of κ is restricted to $[1, 2.5]$ normally provide good results. The exact parameterization of κ is not addressed here and can be pursued independently, depending upon the task.

CHAPTER 6

THE PATTERN/CHARACTER RECOGNITION PROBLEM

6.1 Problem description

Pattern recognition is one of the major application areas of FFANNs. In this chapter we employ FFANNs for character recognition. The characters that were chosen for this problem were the Arabic numerals of different fonts.

Character recognition using a FFANN has attracted the attention of many researchers, for it would find many applications in office automation and computer aided design. Merging the classical techniques of pattern recognition with the classification capabilities of a FFANN, the process of character recognition as viewed by many researchers [61]-[63], consists of feature extraction by classical methods and classification by a FFANN.

Many others motivated by the association and the classification capabilities of the human brain expect both of the tasks to be performed by a FFANN [64]-[66]. The FFANN reminiscent of Neocognitron [64] dedicated to solve the problem of character recognition use a large number of neurons and trainable weights. Many researchers exploit the capabilities of multilayer perceptrons [17] to solve the problem of character recognition by using a sufficient number of neurons in the hidden layers [27]-[29].

We decompose the character recognition problem into two parts: (1) feature extraction or preprocessing and (2) classification. We use two layer FFANN for feature extraction and an additional layer network for classification. The two networks are concatenated in a feedforward manner, the feature extraction network followed by the classification network.

In section 6.2, we discuss the data-base used for training and testing the network. The network design is discussed in section 6.3. The training and the learning algorithm used to train the network is discussed in section 6.4. In this section we carry out the learning evaluation with update law (2.7) using different energy functions. In section 6.5, the chapter is finally concluded with the discussion on the network design and various learning algorithms used.

6.2 The data-base

We have collected 100 different fonts from printed material [67]-[70]. Each of these printed fonts were scanned using the SC-7500 Toshiba scanner and was stored as a "Tiff" file. The files have subsequently been converted to "Hips" format which could easily handle binary data. The pixel image of various digits in different fonts were then obtained from the formatted Hips files.

The images of these fonts were stretched appropriately to fit within a pixel window of 17×15 dimension while leaving a small margin on all sides. With this pre-processing, we also achieve scaling invariance of the characters to some extent. The grey level of these pixels were set to be 0 or 1. Some of these fonts were chosen to represent this formatting, whose pixel images are depicted in Figure 6.1.

The collected fonts included roman, bold and italic versions of different classes. The network was trained with half of these fonts (which included roman, bold and italic versions of some of the classes). The second half of the data-base was used to test the character recognition and generalization capabilities of the network.

6.3 The network design

The character recognition network [71] that we used consist of the feature extraction and the classification networks as depicted in Figure 6.2. The feature extraction network is designed in a manner to extract features locally with local connections from



Augustea



Badoni bold



Banco



Breite



Choc



Consort



Francesca



Italian



Kaufmann



Stardivarius

Figure 6.1: Typical examples of the fonts used as an input to the character recognition network.

previous layer, while the classification is done with global (i.e full) connections.

Training of the network used for the character recognition takes a long time. We split the network into two, so that one part (i.e) feature extraction network, which takes major portion of the training time should be available as a commercially trained chip off-line on a very large data-base. The second part should be available to the user to train it on-line with new data to be added. We discuss the design of the feature extraction and the classification network in subsections 6.3.1 and 6.3.2 respectively.

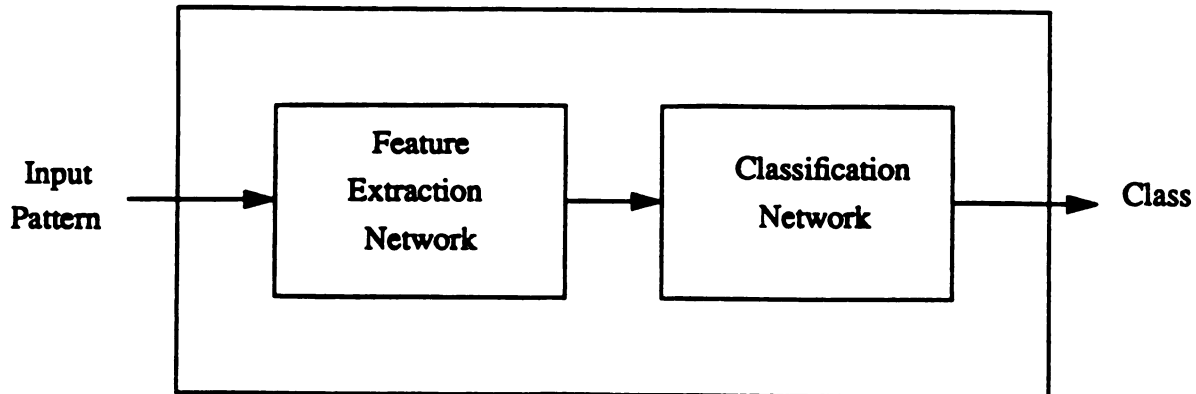


Figure 6.2: Block Diagram of the FFANN used for the Character Recognition problem.

6.3.1 The feature extraction network

A partially connected FFANN with one hidden layer is chosen to extract features of Arabic numerals of various fonts used for common printing. The input layer of this feature extraction network is a 17×15 array of passive nodes corresponding to the input patterns. These patterns are the characters 0 through 9 of different fonts. The input passive nodes simply accept the individual components of the input image and

distribute them to the specified neurons of the hidden layer. The output of each neuron is in the range (0,1) as the activation output is determined by the so called logistic function [7]-[8]

$$output = \frac{1}{1+e^{-input}}. \quad (6.1)$$

A sliding window of 7×7 array of input nodes are connected to a neuron in a hidden layer. The window slides all over the input array moving one column or one row of 7 input nodes at a time. As a result the dimension of the hidden layer reduces to 11×9 . A similar sliding window of 5×5 array of hidden layer neurons is connected to a neuron in the output layer. The resulting array-size of the output layer consequently reduces to 7×5 . The structure of the network is shown in Figure 6.3.

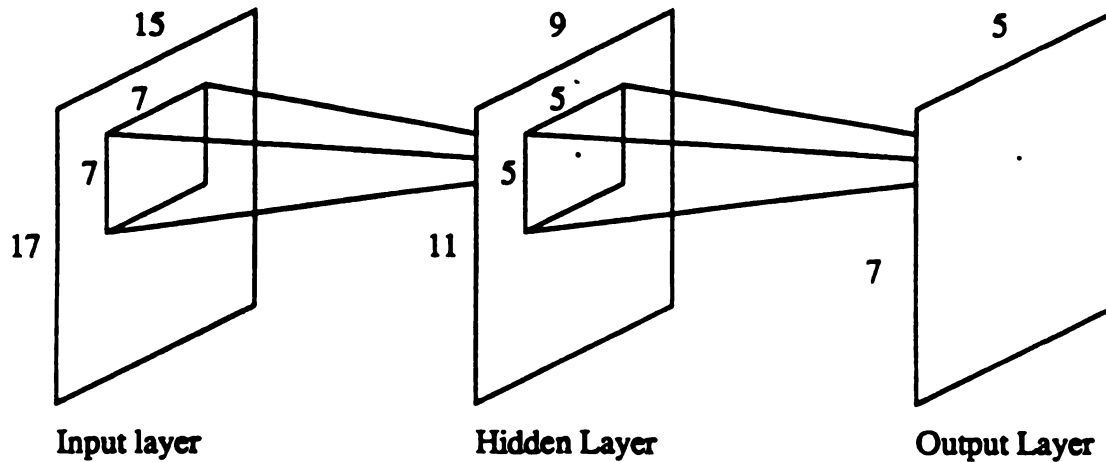


Figure 6.3: Structure of the FFANN used for feature extraction.

Ten 7×5 array targets are constructed corresponding to each input digit. The targets are constructed so as to retain the general shape of the characters and, in addition, be pseudo-orthogonal to each other. Figure 6.4 shows all the targets used in the training of this network.

The targets are considered orthogonal if the hamming distance between them is equal to the dimension of the targets, namely $7 \times 5 = 35$. In the present case, it was not possible to construct orthogonal targets while retaining the geometric shape of the digit. The minimum and the maximum hamming distance between the targets given in Figure 6.4 is 17 and 22, respectively. As a consequence, this coding has an error correction capability of up to 8 errors in the output pixels. If the pixels with similar and dissimilar activation state (specifically, a high or low state) correlates to +1 and -1, the correlation matrix C for all the targets becomes

$$C = \begin{bmatrix} 35 & -3 & -1 & -5 & -1 & -1 & +1 & -1 & -3 & -3 \\ -3 & 35 & -7 & -3 & -3 & +1 & -1 & -3 & -1 & -1 \\ -1 & -7 & 35 & -1 & -1 & -1 & -7 & -5 & +1 & +1 \\ -5 & -3 & -1 & 35 & -1 & -1 & +1 & -1 & -7 & -3 \\ -1 & -3 & -1 & -1 & 35 & -1 & -3 & -1 & +1 & +1 \\ -1 & +1 & -1 & -1 & -1 & 35 & +1 & -1 & -3 & +1 \\ +1 & -1 & -7 & +1 & -3 & +1 & 35 & -3 & -1 & -9 \\ -1 & -3 & -5 & -1 & -1 & -1 & -3 & 35 & -3 & +1 \\ -3 & -1 & +1 & -7 & +1 & -3 & -1 & -3 & 35 & -1 \\ -3 & -1 & +1 & -3 & +1 & +1 & -9 & +1 & -1 & 35 \end{bmatrix} \quad (6.2)$$

The performance of the feature extraction network can be measured by noting the correlation of the output digits with the targets. If the output is within a hamming distance of 8 of a target, it is classified to that target; otherwise we consider the network to be unable to classify the particular input.

6.3.2 The classification network

The classification network is a fully connected FFANN. There is only one layer consisting of 10 neurons corresponding to the different classes of characters, namely 0,1,...,9. The output of the feature extraction network from its 35 output nodes is

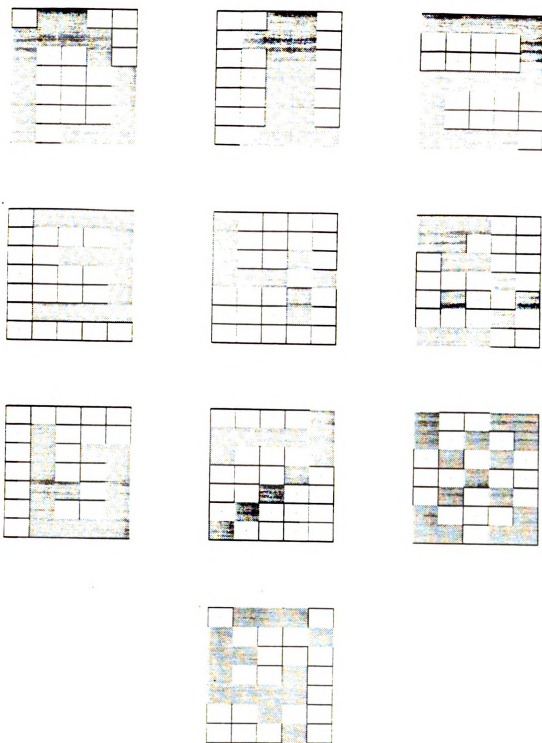


Figure 6.4: Targets used in the feature extraction network for various digits.

directly fed to all of the classification neurons. We chose orthogonal targets for all the 10 different classes. The targets are chosen so that, for any particular class, the output for the corresponding node is high and all others are low. If the output of more than one node goes high, the network is considered unable to classify.

6.4 Learning evaluation with different energy functions

We used the Gradient Descent method with the error backpropagation learning technique for the training of the network. The feature extraction and the classification networks can be trained together or separately. We trained the two networks separately with a view to achieve the following advantages:

- (1) The feature extraction and the classification networks can be viewed as two separate modules;
- (2) The feature extraction being fully trained on a large data-base off-line and the classification network can be quickly trained on-line when new fonts are added. This will enhance the character recognition capability of the network as will be demonstrated in our work.

We used a training simulator developed in [35]-[39],[57],[60],[71], coding the weight update law (2.7) in C-language on Sun SPARC station. For the integration of these update laws, the simulator uses fourth order Runge-Kutta integration routine. A stopping criterion ω , was set to determine if the system has converged to a solution. (The solution is actually the local minimum in the vicinity of the initial conditions in the weight space.) The weights were initialized randomly by a random number generator. The value of any initial weight w is such that $-0.9 \leq w \leq 0.9$.

We assume that the network has converged when the specified stopping criterion ω is achieved. The parameter ω is defined as

$$\omega := \sum_w |\Delta w|, \quad (6.3)$$

where w are all the weights in the network and

$$\Delta w := w(t + \Delta t) - w(t). \quad (6.4)$$

The term Δt is the time step of the fourth order Runge-Kutta integration which is used to integrate the update laws (2.7). We also used an error criterion to characterize the training: the lower the value of the error, the better is the training. The error parameter is defined as

$$\xi := \sum_p \sum_n \epsilon_{pn}^2. \quad (6.5)$$

Half of the data-base was used as set 1 and the other half as set 2. Each of these sets included 50 different fonts that are listed in Appendix K. We trained the network in different stages to study the response of different modules under different conditions. Training of the feature extraction and the classification network is discussed in subsections 6.4.1 and 6.4.2, respectively.

6.4.1 Training of the feature extraction network

We trained the feature extraction network with the update law (2.7) using the Gaussian (2.6), the Polynomial (4.3) and the Exponential (5.1) energy functions. The Cauchy energy function (3.4) is not used in the update law (2.7), as the computations grow exponentially with the number of patterns and the output nodes. With 50 different training fonts, the input patterns are 500 and the network has 35 output nodes. Therefore, it is not feasible to use the Cauchy energy function for the training of the pattern recognition problem.

The training simulators using the learning algorithm outlined in Table 2.1 was developed corresponding to update law (2.7) using the Gaussian (2.6), the Polynomial (4.3) and the Exponential (5.1) energy functions. In all the training simulations the value of Δt and γ were chosen to be 0.0005 and 1. κ parameter of the Exponential

energy function (5.1) was chosen to be 0.004.

We assume that the network has converged when the stopping criterion $\omega \leq 0.15$ is satisfied. The error criterion $\xi = 400$ was also chosen to terminate the training of the network. For the Gaussian and the Polynomial energy, learning was terminated when the stoppage criterion ω was achieved. On the other hand for the exponential energy, learning was stopped when the error criterion was achieved. The error(s) and the number of steps at convergence are tabulated in Table 6.1.

Table 6.1

	Gaussian	Polynomial	Exponential Gaussian
ξ	526.25	638.96	399.96
steps	16205	18442	10598

Table 6.1 shows the error(s) and the number of steps at convergence of the Gaussian, the Polynomial and the Exponential Gaussian learning used for the training of the feature extraction network.

After the network was trained with set 1, it was tested with set 2. The character recognition performance is reported in Tables 6.2, 6.3 and 6.4 corresponding to the Gaussian, the Polynomial and the Exponential Gaussian energy functions. In all of the results the network incorporates the error correction capability which is built in the targets (i.e. if the binerized output is upto a Hamming distance 8 from the targets, the corresponding input is classified to that target).

Table 6.2

patterns	correct	incorrect	reject
set 1	100%	0%	0%
set 2	92.6%	3%	4.4%

Table 6.2 shows the character recognition using the feature extraction network trained with the Gaussian learning.

Table 6.3

patterns	correct	incorrect	reject
set 1	100%	0%	0%
set 2	93%	1.8%	5.2%

Table 6.3 shows the character recognition using the feature extraction network trained with the Polynomial learning.

Table 6.4

patterns	correct	incorrect	reject
set 1	100%	0%	0%
set 2	92.4%	2.6%	5%

Table 6.4 shows the character recognition using the feature extraction network trained with the Exponential of the Gaussian learning.

On convergence we observe that the Polynomial learning achieved a better recognition performance, though the error criterion is higher than the Gaussian and the Exponential Gaussian learning. The reason is obvious if we look at the L_1 norm. When the training terminated the L_1 norm of the errors for the Polynomial learning was 882.95, in contrast with 1694.85 for the Gaussian and 1398.56 for the Exponential Gaussian learning.

We also notice that the Exponential Gaussian learning achieved almost same recognition performance as with the simple Gaussian learning, but in much less number of computer steps. In conclusion, we can use appropriate energy function in the update law (2.7) to improve learning speed and/or performance.

6.4.2 Training of the classification network

We used the Gaussian learning (2.6)-(2.7) to train the classification network. The values of Δt , ω and γ were chosen to be 0.01, 0.1 and 1 respectively for the training of the classification network. The training was terminated when the stopping criterion ω was achieved. The error(s) and the number of computer steps at convergence are tabulated in Table 6.5.

Table 6.5

	set 1	set 2
ξ	2.46	8.32
steps	296	618

Table 6.5 shows the error(s) and the number of computer steps at convergence of the learning dynamics for the training of the classification network.

The classification network was trained with two different sets. First we used the same set (set 1) as was used for the training of the feature extraction network. With this set as input to the trained feature extraction network, the output of this network was fed to the classification network and was trained with it. It is just like training the feature extraction and the classification networks together with set 1. The whole network (the feature extraction and the classification) was then tested with set 2 and the results are reported in Table 6.6.

In the second case set 2 was input to the trained feature extraction network and the output of this network was used to train the classification network. This is done assuming that the additional data available to the users to train the classification network would be different than the one on which the feature extraction network is trained. As such set 2 was used for training and set 1 was used for testing. The classification results of the network are reported in Table 6.7.

Tasble 6.6

patterns	correct	incorrect	reject
set 1	100%	0%	0%
set 2	94%	2%	4%

Table 6.6 shows the character recognition when the feature extraction and the classification networks are trained with set 1, using the Gaussian learning.

Table 6.7

patterns	correct	incorrect	reject
set 1	99.8%	0%	0.2%
set 2	100%	0%	0%

Table 6.7 shows the character recognition when the feature extraction and the classification networks are trained with set 1 and set 2, respectively, using the Gaussian learning.

For both the cases reported above, we trained the network with update law (2.7) using the Gaussian energy function. For the training of the classification network we only used one energy function (the Gaussian) as the emphasis here is to study the classification capability and not the learning speed of the network.

Observe that when the feature extraction network is trained with set 1 and the classification network is trained with set 2 (which is the output of the feature extraction network when the input is set 2), the character recognition improves considerably. This feature highlights the importance of using the modular approach. Besides improving the recognition performance, it provides training convenience to users as they have to train a smaller network (the classification network) on the new data to be added.

6.5 Discussion

We decomposed the problem of pattern recognition into two, namely, the feature extraction and the classification of the input characters. Both of these tasks are performed by the FFANN. The network performing feature extraction is partially connected whereas the classification is done with a fully connected network. The two networks are concatenated, the feature extraction followed by the classification.

The two networks performing the feature extraction and the classification are treated as two separate modules. The feature extraction is trained on a large data-base off-line whereas the user can train the classification network on-line according to their requirements, as and when the new data is added.

In this work we trained the feature extraction network on a certain data-base and tested it with another set of data. The training is supervised, in which we provide the targets, that incorporate the error correction capability due to their structure.

We illustrate that the training can be improved using appropriate energy function in the update law (2.7). When the Polynomial energy function is used, we lower the output error in L_2 as well as L_1 norm. Observe that even if the L_2 norm is higher in

value we achieve good recognition results due to the fact that the L_1 norm of the output error is also reduced.

Observe that when the classification network is trained (according to the requirements of the user) with different fonts than the training set (with which the feature extraction network was trained), the over all pattern recognition capability is enhanced. Besides improving the recognition performance, this modular approach provides training convenience to users.

The training simulators based on the update law (2.7) incorporating different energy functions can also be used to train FFANNs for various other applications. The simulators are effectively used to train multilayer feedforward artificial neural networks for handwritten numeric character recognition [72].

CHAPTER 7

SUMMARY AND CONCLUSION

Artificial neural networks are mathematical models, originally designed to mimic some of the functions of the nervous systems of higher animals. There are two main approaches to design these artificial neural networks, namely, those implemented via digital software and those manufactured via analog hardware. This research follows the former approach as applicable to the feedforward structure of these networks. Supervised learning is the key to make these networks useful for automated information processing of data. In this work we addressed some of the issues of the supervised learning as applicable to FFANNs.

7.1 Summary

In this research, theory supported by computer simulations is described, to improve the supervised learning of FFANNs. Error back-propagation learning being widely accepted is used to realize the gradient descent method of updating the weights of the network. Realizing the advantage of the continuous-time update law, we preferred it over the conventional (Euler) discrete-time update law and we used it through out this work.

The choice of an energy function specifies the back-propagated error in the update law. It also determines the learning performance in the sense of the speed of convergence and the size of the domains of attraction of the stable equilibrium points in the weight space. The Gaussian (sum-of-the-squared) energy function is commonly used due to its simplicity. Another motivating factor of its usage is that it is a derivative of the maximum-entropy distribution subject to the second moment constraint (the Gaussian

distribution). The Gaussian energy function is basically the L_2 (Euclidian) norm.

The choice of maximum-entropy distribution can be increased if the constraint is reduced to the first moment only. We chose the Cauchy distribution from the point of view that the corresponding energy function (namely the Cauchy energy function) improves the *learning performance* in a certain sense*. Though the choice of the Cauchy energy function may not be good for solving bigger problems due to the exponential growth of its computations, it still remains as a candidate for possible hardware implemented neural networks.

Reduction of the output errors of a network in the L_2 norm may not always achieve the desirable results. We may have to reduce the output errors in other norms or a combination of norms. The Polynomial energy function, not only takes care of this problem, but also improves the *learning performance*. A simple example of the Polynomial energy function is discussed in which a combination of L_1 and L_2 norms is used. Besides reducing the error in both norms, it improves the learning speed in terms of computer steps and time.

A very common problem that is inherent with the gradient descent systems is convergence to the nearest local minimum. The nearest local minimum may not be good to solve the desired problem. The requirement of skipping the undesirable local minima (at higher energy levels) and converging to the desirable minima (at lower energy levels) is achieved by using the Exponential energy function. Besides, the convergence to the desirable minimum is much faster in terms of the number of computer steps and time, when the Exponential energy function is used.

Numerous simulations on the XOR problem have been conducted to evaluate the *learning performance* of the gradient descent update law using different energy

* Improvement in the learning performance of the Cauchy system here means the derivative of the sum-of-the-squared error at any point on the error surface along the trajectories of the Gaussian system is less in magnitude than that of the Cauchy system.

functions. All simulations are in support of the analytical derivations indicating that a suitable choice of the energy function may improve the *learning performance* in terms of the learning steps and/or time.

Finally, we evaluate the use of the energy functions described in this work on the practical problem of character recognition. A highly user-oriented character recognition network is developed with two separate modules; a feature extraction network to be trained on a large data-base and a classification network to be trained by the user according to his/her requirements. It is shown that different energy functions can be used to update the weights in order to improve the *learning performance*. In addition, the proposed structure is easy to use by the user as he/she would train the network on any additional input patterns in order to improve recognition of newly acquired characters.

7.2 Conclusion

In this research, we have developed a method of exploiting the energy functions in the gradient descent weight update law to improve the learning performance of a FFANN. The improvement of learning performance aims at improving the learning speed and/or improved generalization. This methodology has been specialized to two typical problems: the prototype XOR problem and the practical character recognition problem.

We have shown that investigating various energy functions with a view towards eventual implementation onto software had produced valuable results. Analytically it is tractable to investigate the learning rules in the continuous-time setting. Yet in software implementation, we had to anticipate the step-size and the learning rate and their appropriate selections so as to achieve "faster" convergence, and in addition guarantee the absence of oscillations.

APPENDICES

Appendix A

Entropy comparison of the Gaussian and the Cauchy distribution

Consider the distribution of a random variable x , to be Gaussian with zero mean, which is given as

$$f_g(x) = \frac{1}{\sqrt{2\pi} \sigma} e^{-x^2/2\sigma^2}. \quad (\text{A.1})$$

Choosing $\sigma = 1$, the Gaussian distribution can be given as

$$f_g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}. \quad (\text{A.2})$$

The entropy $H_g(f)$ [43] of the Gaussian distribution $f_g(x)$ is calculated as

$$\begin{aligned} H_g(f) &= - \int_{-\infty}^{+\infty} f_g(x) \log f_g(x) dx \\ &= - \log e \int_{-\infty}^{+\infty} f_g(x) \ln f_g(x) dx \\ &= - \log e \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \ln \left[\frac{1}{\sqrt{2\pi}} e^{-x^2/2} \right] dx \\ &= - \log e \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \left[\ln \frac{1}{\sqrt{2\pi}} - \frac{x^2}{2} \right] dx \\ &= - \log e \left[\ln \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx - \int_{-\infty}^{+\infty} \frac{x^2}{2} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx \right] \\ &= \log e \left[\ln \sqrt{2\pi} - \frac{1}{2\sqrt{2\pi}} \int_{-\infty}^{+\infty} x \cdot -xe^{-x^2/2} dx \right] \end{aligned} \quad (\text{A.3})$$

Now solving the integration on the right hand side by parts and evaluating $xe^{-x^2/2}$ from $-\infty$ to $+\infty$ using *l'Hôpital's rule*, [51] we get

$$H_g(f) = \log e \left[\ln \sqrt{2\pi} + \frac{1}{2} \right] \quad (\text{A.4})$$

Now consider the distribution of the random variable x , to Cauchy with zero mean given as

$$f_c(x) = \frac{1}{\pi} \cdot \frac{\lambda}{\lambda^2 + x^2}. \quad (\text{A.5})$$

Choosing $\lambda = 1$, the Cauchy distribution of x can be given by

$$f_c(x) = \frac{1}{\pi} \cdot \frac{1}{1 + x^2}. \quad (\text{A.6})$$

The entropy $H_c(f)$ of the Cauchy distribution $f_c(x)$ is calculated as

$$\begin{aligned} H_c(f) &= - \int_{-\infty}^{+\infty} f_c(x) \log f_c(x) dx \\ &= - \log e \int_{-\infty}^{+\infty} \frac{1}{\pi} \cdot \frac{1}{1 + x^2} \ln \left[\frac{1}{\pi} \cdot \frac{1}{1 + x^2} \right] dx \\ &= - \log e \int_{-\infty}^{+\infty} \frac{1}{\pi} \cdot \frac{1}{1 + x^2} \left[\ln \frac{1}{\pi} + \ln \frac{1}{1 + x^2} \right] dx \\ &= - \log e \left[\ln \frac{1}{\pi} \int_{-\infty}^{+\infty} \frac{1}{\pi} \cdot \frac{1}{1 + x^2} dx + \int_{-\infty}^{+\infty} \frac{1}{\pi} \cdot \frac{1}{1 + x^2} \ln \frac{1}{1 + x^2} dx \right] \\ &= \log e \left[\ln \pi + \frac{1}{\pi} \int_{-\infty}^{+\infty} \frac{\ln (1 + x^2)}{1 + x^2} dx \right] \end{aligned} \quad (\text{A.7})$$

Now by substituting $x = \tan \theta$ and using

$$\int_0^{\pi/2} \ln \sec \theta d\theta = \frac{\pi}{2} \ln 2$$

we get

$$H_c(f) = \log e \cdot \ln(4\pi) \quad (\text{A.8})$$

Comparing the Gaussian and the Cauchy entropy function values from (A.4) and (A.8), we observe that Cauchy entropy is about 1.78 times the Gaussian entropy.

Appendix B

Proof of Lemma 3.1

Lemma 3.1 : Given $\alpha_i, x_i \in \mathbb{R}, \alpha_i \geq 0$ and assuming that Assumption 3.1 holds.

Then,

$$\left[\sum_i x_i \right] \left[\sum_i \alpha_i x_i \right] \geq 0,$$

Proof :

$$\left[\sum_i x_i \right] \left[\sum_i \alpha_i x_i \right] = [\dots x_i \dots] \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} [\alpha_1 \dots \alpha_f] \begin{bmatrix} \vdots \\ x_i \\ \vdots \end{bmatrix}$$

$$= x^T 1 \alpha^T x \tag{B.1}$$

$$= x^T A x \tag{B.2}$$

$$= x^T \left[\frac{1}{2} (A + A^T) \right] x \tag{B.3}$$

$$= x^T \hat{A} x \tag{B.4}$$

$$\geq 0. \tag{B.5}$$

In step (B.2), $A = 1 \alpha^T$ has non-negative entries. In step (B.3), A in the symmetric form $x^T A x$ is replaced by its symmetric part $\frac{1}{2}(A + A^T) =: \hat{A}$ [41],[55]. By Assumption 3.1 \hat{A} is positive semidefinite.

Remark: Note that if all the α_i s are equal, then in step (B.2), the matrix A is symmetric positive semidefinite. This is obvious since $A = \alpha_i 1 1^T$, and $x^T \alpha_i 1 1^T x = \alpha_i ||1^T x||^2$.

Appendix C

Proof of Theorem 3.1

Theorem 3.1: Assume that Assumption 3.1 holds. Consider $V := E$ given by (3.14i), be a candidate Liapunov function. Let \dot{V} be the derivative of V along the trajectories of system (2.14) and \dot{V}^c be the derivative of V along the trajectories of system (3.15). Then the *learning performance* of system (3.15) is "faster" than or at least is the same as system (2.14).

Proof : Case 1 (output nodes)

In order to analyse the *learning performance*, let's evaluate $\dot{V}^c - \dot{V}$ as

$$\dot{V}^c - \dot{V} = \nabla_W V \cdot (\dot{W}^c - \dot{W}) \quad (\text{C.1})$$

Here W is the weight vector containing all the weights. Equation (3.5i) gives the components of \dot{W}^c and equation (2.14i) gives the components of \dot{W} . Then for the output layer, we get

$$\dot{V}^c - \dot{V} = [.. \nabla_{w_{nm}} E ..] \cdot -\gamma \left\{ \begin{bmatrix} \vdots \\ \nabla_{w_{nm}} E^c \\ \vdots \end{bmatrix} - \begin{bmatrix} \vdots \\ \nabla_{w_{nm}} E \\ \vdots \end{bmatrix} \right\} \quad (\text{C.2i})$$

$$= [.. \nabla_{w_{nm}} E ..] \cdot -\gamma \left\{ \begin{bmatrix} \vdots \\ (1+\alpha_n) \nabla_{w_{nm}} E \\ \vdots \end{bmatrix} - \begin{bmatrix} \vdots \\ \nabla_{w_{nm}} E \\ \vdots \end{bmatrix} \right\} \quad (\text{C.2ii})$$

$$= -\gamma \sum_{w_{nm}} \alpha_n |\nabla_{w_{nm}} E|^2 \quad (\text{C.2iii})$$

$$\leq 0 \quad \text{as } \alpha_n \geq 0 \text{ and } \gamma > 0 \quad (\text{C.2iv})$$

Case 2 (hidden nodes)

Now consider the dynamics of weights connecting the neurons in a hidden layer. The components of \dot{W}^c and \dot{W} are given by equations (3.5ii) and (2.14ii) respectively. Plugging in the values of \dot{W}^c and \dot{W} in equation (C.1), we get

$$\dot{V}^c - \dot{V} = [.. \nabla_{w_{kj}} E ..] \cdot - \gamma \left\{ \begin{bmatrix} \vdots \\ \nabla_{w_{kj}} E + \frac{1}{2} \sum_n \alpha_n \nabla_{w_{kj}} \epsilon_n^2 \\ \vdots \end{bmatrix} - \begin{bmatrix} \vdots \\ \nabla_{w_{kj}} E \\ \vdots \end{bmatrix} \right\}, \quad (C.3)$$

where

$$\nabla_{w_{kj}} E = \frac{1}{2} \sum_n \nabla_{w_{kj}} \epsilon_n^2.$$

Therefore we get

$$\dot{V}^c - \dot{V} = - \frac{1}{4} \gamma \sum_{w_{kj}} \left[\sum_n \nabla_{w_{kj}} \epsilon_n^2 \right] \left[\sum_n \alpha_n \nabla_{w_{kj}} \epsilon_n^2 \right]. \quad (C.4)$$

For each entry of the summation $\sum_{w_{kj}}$ we use Lemma 3.1 to show

$$\left[\sum_n \nabla_{w_{kj}} \epsilon_n^2 \right] \left[\sum_n \alpha_n \nabla_{w_{kj}} \epsilon_n^2 \right] \geq 0 \quad (C.5)$$

therefore

$$\dot{V}^c - \dot{V} \leq 0.$$

Appendix D

Proof of Lemma 3.2

Lemma 3.2 : Given α_{ij} , $x_{ij} \in R$, $\alpha_{ij} \geq 0$ and assuming that Assumption 3.2 holds.

Then,

$$\left[\sum_i \sum_j x_{ij} \right] \left[\sum_i \sum_j \alpha_{ij} x_{ij} \right] \geq 0,$$

Proof :

$$\left[\sum_i \sum_j x_{ij} \right] \left[\sum_i \sum_j \alpha_{ij} x_{ij} \right] = [\dots x_{ij} \dots] \begin{bmatrix} 1 \\ \cdot \\ \cdot \\ 1 \end{bmatrix} [\alpha_{11} \dots \alpha_{IJ}] \begin{bmatrix} \cdot \\ \cdot \\ \cdot \\ x_{ij} \end{bmatrix}$$

$$= x^T 1 \alpha^T x \tag{D.1}$$

$$= x^T A x \tag{D.2}$$

$$= x^T \left[\frac{1}{2} (A + A^T) \right] x \tag{D.3}$$

$$= x^T \hat{A} x \tag{D.4}$$

$$\geq 0. \tag{D.5}$$

In step (D.2), $A = 1\alpha^T$ has non-negative entries. In step (D.3), A in the symmetric form $x^T A x$ is replaced by its symmetric part $\frac{1}{2}(A + A^T) =: \hat{A}$ [41],[56]. By Assumption 3.2 \hat{A} is positive semidefinite.

Remark: Note that if all the α_{ij} s are equal, then in step (D.2), matrix A is symmetric positive semidefinite. This is obvious since $A = \alpha_{ij} 11^T$, and $x^T \alpha_{ij} 11^T x = \alpha_{ij} ||1^T x||^2$.

Appendix E

Proof of Theorem 3.2

Theorem 3.2: Assume that Assumption 3.2 holds. Consider a candidate Liapunov function $V := E$ given by equation (2.6). Let \dot{V} be the derivative of V along the trajectories of (2.14) and \dot{V}^c be the derivative of V along the trajectories of the system (3.16). Then the *learning performance* of system (3.16) is "faster" than or at least is the same as system (2.14).

Proof : Case 1 (output nodes)

In order to analyze the *learning performance*, we calculate $\dot{V}^c - \dot{V}$ as

$$\dot{V}^c - \dot{V} = \nabla_W V \cdot (\dot{W}^c - \dot{W}) \quad (\text{E.1})$$

where W is the weight vector containing all the weights. Equation (3.5) gives the components of \dot{W}^c and equation (2.14) gives the components of \dot{W} . Then for the output layer, using equations (2.14i) and (3.5i), we get

$$\dot{V}^c - \dot{V} = [.. \nabla_{w_{nm}} E ..] \cdot -\gamma \left\{ \begin{bmatrix} \vdots \\ \nabla_{w_{nm}} E^c \\ \vdots \end{bmatrix} - \begin{bmatrix} \vdots \\ \nabla_{w_{nm}} E \\ \vdots \end{bmatrix} \right\} \quad (\text{E.2i})$$

$$= [.. \nabla_{w_{nm}} E ..] \cdot -\gamma \left\{ \begin{bmatrix} \vdots \\ \nabla_{w_{nm}} E + \frac{1}{2} \sum_p \alpha_{pn} \nabla_{w_{nm}} \epsilon_{pn}^2 \\ \vdots \end{bmatrix} - \begin{bmatrix} \vdots \\ \nabla_{w_{nm}} E \\ \vdots \end{bmatrix} \right\}, \quad (\text{E.2ii})$$

where

$$\nabla_{w_{nm}} E = \frac{1}{2} \sum_p \nabla_{w_{nm}} \epsilon_{pn}^2.$$

Therefore we get

$$\dot{V}^c - \dot{V} = -\frac{1}{4} \gamma \sum_{w_{nm}} \left[\sum_p \nabla_{w_{nm}} \epsilon_{pn}^2 \right] \left[\sum_p \alpha_{pn} \nabla_{w_{nm}} \epsilon_{pn}^2 \right]. \quad (\text{E.3})$$

Using Lemma 3.1, it follows that each of the entries of the summation over w_{nm} , on the right hand side of equation (E.3), is ≥ 0 . Subsequently $\dot{V}^c - \dot{V} \leq 0$.

Case 2 (hidden nodes)

Now consider the dynamics of weights connecting the neurons in any hidden layer. The components of \dot{W}^c and \dot{W} are given by equations (3.5ii) and (2.14ii), respectively. Again we calculate the difference $\dot{V}^c - \dot{V}$ as follows.

$$\dot{V}^c - \dot{V} = [.. \nabla_{w_{kj}} E ..] \cdot - \gamma \left\{ \begin{bmatrix} \vdots \\ \nabla_{w_{kj}} E + \frac{1}{2} \sum_p \sum_n \alpha_{pn} \nabla_{w_{kj}} \epsilon_{pn}^2 \\ \vdots \end{bmatrix} - \begin{bmatrix} \vdots \\ \nabla_{w_{kj}} E \\ \vdots \end{bmatrix} \right\}, \quad (\text{E.4})$$

where

$$\nabla_{w_{kj}} E = \frac{1}{2} \sum_p \sum_n \nabla_{w_{kj}} \epsilon_{pn}^2.$$

With this substitution, we get

$$\dot{V}^c - \dot{V} = -\frac{1}{4} \gamma \sum_{w_{kj}} \left[\sum_p \sum_n \nabla_{w_{kj}} \epsilon_{pn}^2 \right] \left[\sum_p \sum_n \alpha_{pn} \nabla_{w_{kj}} \epsilon_{pn}^2 \right] \quad (\text{E.5})$$

Considering the summation over w_{kj} , on the right hand side of (E.5), we use Lemma 3.2 on each of its entries to show

$$\left[\sum_p \sum_n \nabla_{w_{kj}} \epsilon_{pn}^2 \right] \left[\sum_p \sum_n \alpha_{pn} \nabla_{w_{kj}} \epsilon_{pn}^2 \right] \geq 0$$

As each entry of the summation over w_{kj} on the right hand side of equation (E.5) is ≥ 0 , therefore $\dot{V}^c - \dot{V} \leq 0$.

Appendix F

Proof of Proposition 3.1

Proposition 3.1: The equilibria of system (3.16) are also the equilibria of system (2.14).

Proof: First we consider the case of output nodes of a network. The equilibria of the weights connecting the output neurons of system (2.14) are given by

$$\begin{aligned}
 0 &= \dot{w}_{nm}, \\
 &= \nabla_{w_{nm}} E, \\
 &= \frac{1}{2} \sum_p \nabla_{w_{nm}} \epsilon_{pn}^2,
 \end{aligned} \tag{F.1}$$

and the equilibria of the weights connecting the output neurons of system (3.5) are given by

$$\begin{aligned}
 0 &= \dot{w}_{nm}^c \\
 &= \nabla_{w_{nm}} E + \frac{1}{2} \sum_p \alpha_{pn} \nabla_{w_{nm}} \epsilon_{pn}^2, \\
 &= \frac{1}{2} \sum_p \nabla_{w_{nm}} \epsilon_{pn}^2 + \frac{1}{2} \sum_p \alpha_{pn} \nabla_{w_{nm}} \epsilon_{pn}^2.
 \end{aligned} \tag{F.2}$$

This implies

$$\sum_p \nabla_{w_{nm}} \epsilon_{pn}^2 = - \sum_p \alpha_{pn} \nabla_{w_{nm}} \epsilon_{pn}^2. \tag{F.3}$$

Multiplying both sides by $\sum_p \alpha_{pn} \nabla_{w_{nm}} \epsilon_{pn}^2$, we get

$$\left[\sum_p \nabla_{w_{nm}} \epsilon_{pn}^2 \right] \left[\sum_p \alpha_{pn} \nabla_{w_{nm}} \epsilon_{pn}^2 \right] = - \left[\sum_p \alpha_{pn} \nabla_{w_{nm}} \epsilon_{pn}^2 \right]^2. \tag{F.4}$$

According to Lemma 3.1, the left hand side is non-negative but we notice that the right hand side is non-positive. This implies

$$\sum_p \nabla_{w_{mn}} \epsilon_{pn}^2 = \sum_p \alpha_{pn} \nabla_{w_{mn}} \epsilon_{pn}^2 = 0. \quad (\text{F.5})$$

This proves the subject claim for the weights connecting the output nodes. Following similar derivations and using Lemma 3.2 the claim for the weights connecting the hidden nodes can be easily proved.

Appendix G

Proof of Theorem 4.1

Theorem 4.1: Assume that Assumption 3.1 holds. Consider $V:=E$ given by (4.12i), be a candidate Liapunov function. Let \dot{V} be the derivative of V along the trajectories of system (2.14) and \dot{V}^P be the derivative of V along the trajectories of system (4.13), then the *learning performance* of system (4.13) is "faster" than or at least is the same as system (2.14).

Proof : Case 1 (output nodes)

In order to analyze the *learning performance*, let's evaluate $\dot{V}^P - \dot{V}$ as

$$\dot{V}^P - \dot{V} = \nabla_w V \cdot (\dot{W}^P - \dot{W}) \quad (G.1)$$

$$= [.. \nabla_{w_{nm}} E ..] \cdot -\gamma \left\{ \begin{bmatrix} \vdots \\ \nabla_{w_{nm}} E^P \\ \vdots \end{bmatrix} - \begin{bmatrix} \vdots \\ \nabla_{w_{nm}} E \\ \vdots \end{bmatrix} \right\} \quad (G.2)$$

$$= [.. \nabla_{w_{nm}} E ..] \cdot -\gamma \left\{ \begin{bmatrix} \vdots \\ \nabla_{w_{nm}} E + \nabla_{w_{nm}} E_1 \\ \vdots \end{bmatrix} - \begin{bmatrix} \vdots \\ \nabla_{w_{nm}} E \\ \vdots \end{bmatrix} \right\} \quad (G.3)$$

where $E_1 = \frac{1}{2} \sum_n \epsilon_n$. Simplifying the expression in (G.3), we get

$$\dot{V}^P - \dot{V} = -\gamma \sum_{w_{nm}} (\nabla_{w_{nm}} E) (\nabla_{w_{nm}} E_1) \quad (G.4)$$

$$= -2\gamma \sum_{w_{nm}} (\epsilon_n \nabla_{w_{nm}} E_1) (\nabla_{w_{nm}} E_1) \quad (G.5)$$

$$= -2\gamma \sum_{w_{nm}} \epsilon_n (\nabla_{w_{nm}} E_1)^2 \quad (G.6)$$

therefore,

$$\dot{V}^P - \dot{V} \leq 0. \quad (\text{G.7})$$

Case 2 (hidden nodes)

For the hidden nodes we can similarly derive the following expression

$$\dot{V}^P - \dot{V} = -\gamma \sum_{w_{kj}} (\nabla_{w_{kj}} E) (\nabla_{w_{kj}} E_1) \quad (\text{G.8})$$

As the energy functions E and E_1 are not direct functions of the weights w_{kj} , therefore we can write

$$\dot{V}^P - \dot{V} = -\frac{1}{2} \gamma \sum_{w_{kj}} \left[\sum_n \varepsilon_n \nabla_{w_{kj}} \varepsilon_n \right] \left[\sum_n \nabla_{w_{kj}} \varepsilon_n \right] \quad (\text{G.9})$$

For each entry of the summation $\sum_{w_{kj}}$ we use Lemma 3.1 to show

$$\left[\sum_n \varepsilon_n \nabla_{w_{kj}} \varepsilon_n \right] \left[\sum_n \nabla_{w_{kj}} \varepsilon_n \right] \geq 0. \quad (\text{G.10})$$

therefore,

$$\dot{V}^P - \dot{V} \leq 0. \quad (\text{G.11})$$

Appendix H

Proof of Theorem 4.2

Theorem 4.2: Assume that Assumption 3.2 holds. Consider a candidate Liapunov function $V := E$ given by equation (2.6). Let \dot{V} be the derivative of V along the trajectories of (2.14) and \dot{V}^P be the derivative of V along the trajectories of the system (4.14). Then the *learning performance* of system (4.14) is "faster" than or at least is the same as system (2.14).

Proof : Case 1 (output nodes)

In order to analyze the *learning performance*, let's evaluate $\dot{V}^P - \dot{V}$ as

$$\dot{V}^P - \dot{V} = \nabla_w V \cdot (\dot{W}^P - \dot{W}), \quad (\text{H.1})$$

$$= [.. \nabla_{w_{mm}} E ..] \cdot -\gamma \left\{ \begin{bmatrix} \vdots \\ \nabla_{w_{mm}} E^P \\ \vdots \end{bmatrix} - \begin{bmatrix} \vdots \\ \nabla_{w_{mm}} E \\ \vdots \end{bmatrix} \right\}, \quad (\text{H.2})$$

$$= [.. \nabla_{w_{mm}} E ..] \cdot -\gamma \left\{ \begin{bmatrix} \vdots \\ \nabla_{w_{mm}} E + \nabla_{w_{mm}} E_1 \\ \vdots \end{bmatrix} - \begin{bmatrix} \vdots \\ \nabla_{w_{mm}} E \\ \vdots \end{bmatrix} \right\}, \quad (\text{H.3})$$

where $E_1 = \frac{1}{2} \sum_p \sum_n \epsilon_{pn}$. Simplifying the expression in (H.3), we get

$$\dot{V}^P - \dot{V} = -\gamma \sum_{w_{mm}} (\nabla_{w_{mm}} E) (\nabla_{w_{mm}} E_1), \quad (\text{H.4})$$

$$= -2\gamma \sum_{w_{mm}} \left[\sum_p \epsilon_{pn} \nabla_{w_{mm}} \epsilon_{pn} \right] \left[\sum_p \nabla_{w_{mm}} \epsilon_{pn} \right]. \quad (\text{H.5})$$

For each entry of summation $\sum_{w_{mn}}$ we use Lemma 3.1 to show

$$\left[\sum_p \varepsilon_{pn} \nabla_{w_{mn}} \varepsilon_{pn} \right] \left[\sum_p \nabla_{w_{mn}} \varepsilon_{pn} \right] \geq 0, \quad (\text{H.6})$$

therefore

$$\dot{V}^P - \dot{V} \leq 0. \quad (\text{H.7})$$

Case 2 (hidden nodes)

For the hidden nodes we can similarly derive the following expression

$$\dot{V}^P - \dot{V} = -\gamma \sum_{w_{kj}} (\nabla_{w_{kj}} E) (\nabla_{w_{kj}} E_1). \quad (\text{H.8})$$

As the energy functions E and E_1 are not direct functions of the weights w_{kj} , therefore we can write

$$\dot{V}^P - \dot{V} = -\frac{1}{2} \gamma \sum_{w_{kj}} \left[\sum_p \sum_n \varepsilon_{pn} \nabla_{w_{kj}} \varepsilon_{pn} \right] \left[\sum_p \sum_n \nabla_{w_{kj}} \varepsilon_{pn} \right] \quad (\text{H.9})$$

For each entry of the summation $\sum_{w_{kj}}$ we use Lemma 3.2 to show

$$\left[\sum_p \sum_n \varepsilon_{pn} \nabla_{w_{kj}} \varepsilon_{pn} \right] \left[\sum_p \sum_n \nabla_{w_{kj}} \varepsilon_{pn} \right] \geq 0, \quad (\text{H.10})$$

therefore

$$\dot{V}^P - \dot{V} \leq 0. \quad (\text{H.11})$$

Appendix I

Proof of Proposition 4.1

Proposition 4.1: The equilibria of system (4.14) are also the equilibria of system (2.14).

Proof: First we consider the case of output nodes of a network. The equilibria of the weights connecting the output neurons of system (2.14) are given by

$$\begin{aligned}
 0 &= \dot{w}_{nm}, \\
 &= \nabla_{w_{nm}} E, \\
 &= \frac{1}{2} \sum_p \nabla_{w_{nm}} \epsilon_{pn}^2,
 \end{aligned} \tag{I.1}$$

and the equilibria of the weights connecting the output neurons of system (4.4) are given by

$$\begin{aligned}
 0 &= \dot{w}_{nm}^P, \\
 &= \nabla_{w_{nm}} E + \nabla_{w_{nm}} E_1, \\
 &= \frac{1}{2} \sum_p \nabla_{w_{nm}} \epsilon_{pn}^2 + \frac{1}{2} \sum_p \nabla_{w_{nm}} \epsilon_{pn}, \\
 &= \sum_p \epsilon_{pn} \nabla_{w_{nm}} \epsilon_{pn} + \frac{1}{2} \sum_p \nabla_{w_{nm}} \epsilon_{pn}.
 \end{aligned} \tag{I.2}$$

This implies

$$\frac{1}{2} \sum_p \nabla_{w_{nm}} \epsilon_{pn} = - \sum_p \epsilon_{pn} \nabla_{w_{nm}} \epsilon_{pn}. \tag{I.3}$$

Multiplying both sides by $\sum_p \nabla_{w_{nm}} \epsilon_{pn}$, we get

$$\frac{1}{2} \left[\sum_p \nabla_{w_{nm}} \epsilon_{pn} \right]^2 = - \left[\sum_p \nabla_{w_{nm}} \epsilon_{pn} \right] \left[\sum_p \epsilon_{pn} \nabla_{w_{nm}} \epsilon_{pn} \right]. \tag{I.4}$$

According to Lemma 3.1, the right hand side is non-positive but we notice that the left hand side is non-negative. This implies

$$\sum_p \nabla_{w_{pn}} \epsilon_{pn} = \sum_p \epsilon_{pn} \nabla_{w_{pn}} \epsilon_{pn} = 0. \quad (I.5)$$

This proves the subject claim for the weights connecting the output nodes. Following similar derivations and using Lemma 3.2 the claim for the weights connecting the hidden nodes can be easily proved.

Appendix J

Proof of Proposition 5.1

Proposition 5.1: The equilibria of system (5.2) and system (5.3) are same.

Proof: We consider the Exponential (5.3) and the simple (5.2) systems, which are rewritten as

$$\dot{w}^e = -\gamma \chi \nabla_w E, \quad (J.1)$$

and

$$\dot{w} = -\gamma \nabla_w E, \quad (J.2)$$

where E can be the Gaussian (2.6), the Cauchy (3.4), the Polynomial (4.3) or any other suitable energy function. Let the equilibria of system (J.1) be w^* , then

$$0 = -\gamma \chi \nabla_w E|_{w^*}. \quad (J.3)$$

This implies

$$\nabla_w E|_{w^*} = 0. \quad (J.4)$$

Consequently

$$w^* \in \text{the equilibria of system (J.2)}. \quad (J.5)$$

Now consider the equilibria of system (J.2) to be given by w^{**} , then

$$0 = -\gamma \nabla_w E|_{w^{**}}, \quad (J.6)$$

which implies

$$\nabla_w E|_{w^{**}} = 0. \quad (J.7)$$

As a consequence

$$w^{**} \in \text{the equilibria of system (J.1)}. \quad (J.8)$$

(J.5) and (J.8) prove the claim.

Appendix K

List of fonts used for the pattern recognition problem

Set 1		
Ancient	Fluidum italic	Mosaik
Athenaeum italic	Folkwang	New Clarendon
Augustea	Francesca	Nicholas
Banco	Garamond	Normandy
Breite	Grotesque	Paganini italic
Century	Horizon	Peignot
Choc	Hyperion	Petra
Corvinus	Italian	Playbill
Craw Clarendon bold	Kaufmann	Saltino
Craw Modern	Letter 9	Salut
Diotima	London	Slim Black
Diotima italic	Lydian	Stardivarius
Egyptienne italic	Matura	Stencil
Elizbeth italic	Michel Angelo	Tea Chest
Falstaff	Microgramma bold	Walbaum italic
Flash	Mistral	Weiss italic
Fluidum	Montan	

Set 2		
Albertus	Gothic bold italic	Plantin
Amanda	Goudy bold	Psitt
Badoni bold	Hercules	Quirinus bold
Baskerville	Jacno	Radiant bold
Bembo	Keyboard	Ritmo
Cartoon bold	Klang	Rockwell
Caslon italic	Letter 4	Standard
Consort bold	Letter 6	Tiemann
Craw modern bold	Letter 7	Times
Eckmann	Melior	Times bold
Editor	Mercurius	Topic bold
Egizio	Modern 20	Trump
Erbar italic	New Gothic	Universe
Folio	Noname	Vendome
Fortune	Paganini	Walbaum
Ganton	Perpetua italic	Weiss
Gill-sans	Placard	

BIBLIOGRAPHY

BIBLIOGRAPHY

- [1] D. H. Ballard, *Interpolation coding: A representation of numbers in neural models*, Biological Cybernetics, vol. 57, pp. 389-402, 1987.
- [2] E. I. Knudsen, S. du Lac and S. D. Esterley, *Computational maps in the brain*, Annual Review of Neuroscience, vol. 10, pp. 41-65, 1987.
- [3] M. Kuperstein, *An adaptive neural model for mapping invariant target position*, Behavioral Neuroscience, pp. 148-162, 1988.
- [4] D. Walters, *Selection of image primitives for general purpose visual processing*, Computer Vision, Graphics and Image Processing, vol. 37, pp. 261-298, 1987.
- [5] S. Dehaence, J. Changeux and J. Nadal, *Neural networks that learn temporal sequences by selection*, Proceedings National Academy Science, USA, Biophysics, vol. 84, pp 2713-2727, 1987.
- [6] J. A. Feldman and D. H. Ballard, *Connectionist model and their properties*, Cognitive Science, vol. 6, pp. 205-254, 1982.
- [7] D. E. Rumelhart, G. E. Hinton and R. J. Williams, *Learning internal representations by error propagation*, in Parallel Distributed Processing: Exploration in the microstructures of cognition, Cambridge, MA : MIT Press, vol. 1, pp. 318-362, 1986.
- [8] D. E. Rumelhart and J. L. McClelland, *Explorations in Parallel Distributed Processing: A Handbook of Models*, Cambridge, MA : MIT Press, pp. 121-159, 1988.
- [9] D. Tank and J. J. Hopfield, *Concentrating information in time: analog neural networks with application to speech recognition problem*, 1st International Conference on Neural Networks, IEEE, June 1987.
- [10] F. C. Hoppensteadt, *An Introduction to the Mathematics of Neurons*, Cambridge University Press, 1986.
- [11] D. Purves, *Body and Brain : a trophic theory of neural connections*, Harvard University Press, Cambridge, Mass., 1988.

- [12] R. J. MacGrager, *Neural and Brain Modeling*, Academic Press, San Diago, Calif., 1987.
- [13] M. Berry, *The Nerve Cell*, MacKeith Press, London, 1986.
- [14] B. Widrow, study director, DARPA Neural Network Study, AFCEA International Press, February 1988.
- [15] J. A. Anderson and E. Rosenfield, *Neurocomputing: Foundations of Research*, Cambridge, MA: MIT Press, 1988.
- [16] F. Rosanblatt, *Perceptrons and the theory of Brain Mechanisms*, Spartan Books, Newyork, 1962.
- [17] R. P. Lippman, *An introduction to computing with neural nets*, IEEE ASSP Magazine, vol. 4, pp. 4-22, April 1987.
- [18] R. H. Nielsen, *Theory of back-propagation neural networks*, International Joint Conference on Neural Networks, vol. 1, pp. 593-605, June 1989.
- [19] W. S. McCulloch and W. Pitts, *A logical calculus of the ideas imminent in nervous activity*, Bulletin of Mathematical Biophysics, vol. 5, pp. 115-133, 1943.
- [20] B. Widrow and M. E. Hoff, *Adaptive Switching Circuits*, IRE WESCON Convention Record, Newyork: IRE, pp. 96-104, 1960.
- [21] T. Kohonen, *Correlation matrix memories*, IEEE transactions on Computers, C-21, pp. 353-359, 1972.
- [22] J. A. Anderson, *A simple neural network generating an interactive memory*, Mathematical Bioscience, vol. 14, pp. 197-220, 1972.
- [23] S. Grossberg, *Studies of Mind and Brain*, Reidel, Boston, 1982.
- [24] R. J. Williams, *Unit activation rules for cognitive network models*, ICS Report 8303, Institute of Cognitive Sciences, University of California at San Diago, November, 1983.
- [25] G. E. Hinton, T. J. Sejonowsky and D. H. Ackley, *Boltzman machines: constraint satisfaction networks that learn*, Report CMU-CS-84-157, Carnegie-Mellon University, 1984.
- [26] J. J. Hopfield, *Neurons with graded response have collective computational properties like those of two-stage neurons*, Proc. Natl. Acad. Sci., vol. 81, pp. 3088-3092, May 1984.

- [27] M. Arai, *Mapping abilities of three layer neural networks*, International Joint Conference on Neural Networks, vol. 1, pp. 419-423, 1989.
- [28] G. Mirchandni and W. Cao, *On hidden nodes of Neural Nets*, IEEE Transactions on Circuits and Systems, vol. 36, No. 5, pp. 661-664, May 1989.
- [29] S. C. Huang and Y. H. Huang, *Bounds on the number of hidden neurons in multilayer perceptrons*, IEEE Transactions on Neural Networks, vol. 2, No. 1, Jan. 1991.
- [30] M. Minsky and S. Papert, *Perceptrons*, Cambridge, MA: MIT Press, 1969.
- [31] G. Cybenko, *Continuous valued neural networks with two hidden layers are sufficient*, Tech. Rep., Department of Computer Science, Tufts University, March 1988.
- [32] K. Funahashi, *On the approximate realization of continuous mapping by neural networks*, Neural Networks vol. 2, pp. 183-192, 1989.
- [33] A. E. Bryson and Y. C. Ho, *Applied Optimal Control*, Blaisdell, Newyork, 1969.
- [34] P. J. Werbos, *Beyond regression: New tools for prediction and analysis in the behavioral sciences*, Ph.D. Thesis, Applied Mathematics, Harvard University, November 1974.
- [35] F. M. A. Salam, *Neural Nets and Engineering Implementations*, key address at the 31st Midwest Symposium on Circuits and Systems, St. Louis, Missouri, August 10-12, 1988.
- [36] F. M. A. Salam, *Artificial Neural Nets: Basic Theory and Engineering Implementations*, Department of Electrical Engineering, Michigan State University, East Lansing, MI 48824, October 1989.
- [37] F. M. A. Salam and M. R. Choi, *An All-MOS Analog Feedforward Neural Circuit with Learning*, 1990 IEEE International Symposium on Circuits and Systems (ISCAS), New Orleans, Louisiana, May 1-3, 1990.
- [38] F. M. A. Salam, *Learning Algorithms For Artificial Neural Nets For Analog Circuit Implementation*, Interface '90 Proceedings, East Lansing, MI, May 15-16, 1990. (Proceedings will appear in 1991).
- [39] I. K. Sethi and A. K. Jain, *Artificial Neural Networks and Statistical Pattern Recognition*, North Holland, 1991.
- [40] M. W. Hirsch and S. Smale, *Differential Equations, Dynamical Systems and Linear Algebra*, Academic Press, 1974.

- [41] C. T. Chen, *Linear System Theory and Design*, Holt, Rinehart and Winston, New York, 1984.
- [42] E. T. Jaynes, *Information theory and statistical mechanics*, Phys. Rev., vol. 106, pp. 620-630, 1957.
- [43] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*, University of Illinois Press, Urbana, 1949.
- [44] A. Wragg D. C. Dawson, *Fitting continuous probability density functions over $[0, \infty)$ using information theory ideas*, IEEE Trans. Info. Theory, vol. IT-16, pp. 226-230, March 1970.
- [45] D. C. Dawson and A. Wragg, *Maximum entropy distribution having prescribed first and second moments*, IEEE Trans. Info. Theory, vol. IT-19, pp. 689-693, Sept. 1973.
- [46] J. M. Einbu, *On the existence of a class of maximum entropy probability density functions*, IEEE Trans. Info. Theory, vol. IT-23, pp. 772-775, Nov. 1977.
- [47] A. B. Carlson, *Communication Systems - An Introduction to Signals and Noise in Electrical Communication*, McGraw Hill, 1986.
- [48] P. Burrascano, *A norm selection criterion for the Generalized Delta Rule*, IEEE Transactions on Neural Networks, vol. 2, No. 1, Jan 1991.
- [49] J. Wang and B. Malakooti, *On training of artificial neural networks*, International Joint Conference on Neural Networks, Washington D. C., vol. 2, pp. 387-393, June 1989.
- [50] A. Papoulis, *Probability, Random Variables and Stochastic Processes*, McGraw-Hill, New York, 1965.
- [51] C. H. Edwards, Jr. and D. E. Penney, *Calculus and Analytic Geometry*, Prentice-Hall, inc., 1982.
- [52] M. Ahmad and F. M. A. Salam, *Error back-propagation learning using the Cauchy energy function*, under preparation.
- [53] J. K. Hale, *Ordinary Differential Equations*, Wiley-Interscience, 1969.
- [54] H. K. Khalil, *Nonlinear Systems*, Macmillan, 1991.
- [55] M. Fiedler, *Special Matrices and their Applications in Numerical Mathematics*, Martinus Nijhof Publishers, Dordrecht, The Netherlands, 1986.

- [56] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns-Hopkins University Press, 1983.
- [57] M. Ahmad and F. M. A. Salam, *Error back-propagation learning using the Polynomial energy function*, under preparation.
- [58] S. M. Ebeid, J. A. Sirat and J. R. Viala, *A rationalized error back propagation learning algorithm*, International Joint Conference on Neural Networks, vol. II, pp. 373-380, June 18-22, 1989.
- [59] T. Tollenaere, *SuperSAB: Fast adaptive back propagation with good scaling properties*, Neural Networks, vol. 3, pp. 561-573, 1990.
- [60] M. Ahmad and F. M. A. Salam, *Error back-propagation learning using the Exponential energy function*, under preparation.
- [61] W. E. Weideman, M. T. Manry and H. C. Yau, *A comparison of nearest neighbor classifier and a neural network for numeric handprint character recognition*, International Joint Conference on Neural Networks, vol. 1, pp. 117-120, 1989.
- [62] P. Smagt, *A comparative study of neural network algorithms applied to optical character recognition*, Association for Computing Machinery, pp. 1037-1044, 1990.
- [63] J. S. Denker, W. R. Gardner, H. P. Graf, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, H. S. Baird and I. Guyon, *Neural network recognizer for hand written zip code digits*, Advances in Neural Information Processing Systems 1, D. S. Touretzky, ed. M. Kaufmann, San Mateo, CA, 1989.
- [64] K. Fukushima, S. Miyake and T. Ito, *Neocognitron: a neural network model for a mechanism for visual pattern recognition*, IEEE Transactions on Systems, man and Cybernetics SMC-13, pp. 826-834, 1983.
- [65] K. Fukushima and N. Wake, *Handwritten alphanumeric character recognition by the Neocognitron*, IEEE Transactions on Neural Networks, vol. 2, No. 3, May 1991.
- [66] Y. Le Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel, *Handwritten digit recognition with a back-propagation network*, Advances in Neural Information Processing Systems 2, pp. 396-404, 1990.
- [67] A. Bastien, *Alphabet in Type*, West Drayton, Middlesex, England, 1958.
- [68] F. Lamberty, *Letter forms*, Hasting House Publishing, New York, 1964.
- [69] D. Freres, *Cent Alphabets Monotype*, Union Bibliophile de France, Paris, 1963.

- [70] W. J. Jespert, W. T. Berry and A. F. Johnson, *The Encyclopedia of Type Faces*, Blandford Press Ltd. London, 1970.
- [71] M. Ahmad and F. M. A. Salam, *Feedforward artificial neural network structure for character recognition*, to appear in the 29th Annual Allerton Conference on Communication, Control and Computing, October 1991.
- [72] S. Bebromdhane, M. Ahmad and F. M. A. Salam, *Handwritten numeric character recognition using a multilayer neural network*, to appear in the 29th Annual Allerton Conference on Communication, Control and Computing, October 1991.