Université d'Ottawa · University of Ottawa

A new paradigm for the classification of patterns : The 'Race to the Attractor' neural network model

By

Guy J.M.G. Ferland, B.Eng, M.A.Sc.

A thesis submitted to the
School of Information Technology and Engineering (SITE)
in partial fulfillment of the requirements for the degree

Doctor of Philosophy

Ottawa-Carleton Institute for Electrical Engineering
Department of Electrical Engineering
Faculty of Engineering
University of Ottawa
9 September 2001

Abstract

The human brain is arguably the best known classifier around. It can learn complex classification tasks with little apparent effort. It can learn to classify new patterns without forgetting old ones. It can learn a seemingly unlimited number of pattern classes. And it displays amazing resilience through its ability to persevere with reliable classifications despite damage to itself (e.g., dying neurons). These advantages have motivated researchers from many fields in the quest to understand the brain in order to duplicate its ability in an artificial system.

And yet, little is known about the way the brain really works. But one fact which is apparent from available data is that 'TIME' is a critical component of its computational process. The brain is a dynamical system whose state evolves with time. Outside stimulus is processed and transformed repeatedly within it, with a multitude of signals interacting with each other in a complex, time-dependent manner. As a result, the process of pattern recognition inside the brain is also a time-dependent evolution of states where the initial image of the unknown pattern is progressively transformed into a form which represents the class of that pattern.

In this thesis, we seek to achieve some of the advantages of the brain as a classifier by defining a model which captures the importance of time in the recognition process. The 'race to the attractor' neural network model involves the use of dynamical systems which transform initially unknown patterns into simpler prototypes which each represent a pattern class. The time required for this transformation to occur increases as the resemblance between the unknown pattern and the class prototype decreases. This results in a race where dynamical systems compete to transform the unknown pattern as quickly as possible. The winner of this race identifies the unknown pattern as a member of the class which the prototype of that dynamical system represents.

Acknowledgements

I am grateful to my supervisor at the university, Dr. Tet Hin Yeap, for his constant support throughout this work and for many suggestions which led to significant improvements to all aspects of this thesis.

Thanks to Thomas Edison, whose words often gave me the motivation to keep pushing this thesis forward even when I was in a rut:

"Genius is one percent inspiration

and ninety-nine percent perspiration"

Thomas Alva Edison (1847-1931)

Nomenclature of Document

In this document, chapters are numbered 1 to 6 and the Appendix is labeled 'A'.

Main sections in each chapter are numbered consecutively, with the chapter number as prefix. For example, section 3.4 is the fourth main section of chapter 3.

Subsections are also numbered consecutively, with the chapter and sections above it as a prefix. For example, 3.4.5 refers to subsection 5 of section 4 of chapter 3.

Figures in each section are numbered consecutively, with the chapter, section and all subsections as a prefix. For example, Figure 3.4.5-2 refers to the second figure of subsection 5 of section 4, in chapter 3.

Equations are numbered in the same way as figures except that an additional prefix 'E_' is added to distinguish them from figures. For example, E_3.4.5-2 refers to the second equation introduced in subsection 5 of section 4 in chapter 3.

Tables are numbered in the same way as equations but with a prefix 'T_' to distinguish them from figures and equations. For example, T_3.4.5-2 refers to the second table in subsection 5 of section 4 in chapter 3.

Table of Contents

List of Abreviations and Acronyms 10

List of Symbols 12

Chapter 1 : Introduction 19

1.1 Introduction 19

1.2 Thesis Motivation 19
1.2.1 Statement of the Thesis 20

1.3 Objectives of the Thesis 20

1.4 Contribution of the Thesis 21

1.5 Thesis Organization 21

Chapter 2 : The Classification Process in the Brain 24

2.1 Introduction 24

2.2 The Freeman Neurobiological Model 25
2.2.1 Experimental Results of Freeman and Skarda 26
2.2.2 Central Aspect of the Freeman Neurobiological Model 29
2.2.3 Description of the Freeman Model 34

2.3 Key Characteristics which Make the Brain a Good and Reliable Classifier 37

2.4 Mechanisms to Implement the Key Characteristics of Learning in a Model 39
2.4.1 Learning Complex Classification Boundaries 39
2.4.1.1 Reducing Complexity by Using an Iterated Mapping 41
2.4.1.2 Reducing Complexity by Defining Output Classes as Ranges of Values 42
2.4.1.3 Reducing Complexity by Re-Using Learned Mappings for New Classes 42
2.4.2 Learning New Classes with No Memory Loss 43
2.4.3 Limiting the Complexity of the Learning Task During Incremental Learning 45
2.4.4 Repairing Neural Damage 47
2.4.5 Summary of the Four Mechanisms 48

2.5 Current Classification Paradigms based on Neural Networks 49

2.5.1 Paradigms Based on the One-Step Associative Memory 50
2.5.1.1 The Key Characteristics of Learning in the One-Step Associative Memory 51
2.5.1.2 The One-Step Associative Memory Paradigm versus the Freeman Model 52
2.5.1.3 Neural Net Architectures for the One-Step Associative Memory Paradigm 52
2.5.1.3.1 MLP and RBF Feedforward Associative Networks 53
2.5.1.3.2 Self-Organizing Feature Maps (SOFM) : The Kohonen Network 54
2.5.1.3.3 Time Delay Neural Networks (TDNN) 55
2.5.2 Paradigms Based on Neurodynamical Systems with Equilibrium States 56
2.5.2.1 The Key Characteristics of Learning in a Neurodynamical Classifier 57
2.5.2.2 The Neurodynamical Classification Paradigm versus the Freeman Model 57
2.5.2.3 Neural Net Architectures for the Neurodynamical Classification Paradigm 58
2.5.2.3.1 The Hopfield Net 58
2.5.2.3.2 Boltzmann Neural Net 60
2.5.2.3.3 Recurrent MLP or RBF Neural Networks 62

2.6 Summary and Conclusions 63

Chapter 3 : Description and Basic Theory of the RTA NN Paradigm 64

3.1 Introduction 64

3.2 Nonlinear Dynamical Systems (NDS) with Iterated Mapping Functions 64
3.2.1 Changing the Attractor of a NDS 67

3.3 The Basic Operational Concept of the RTA NN Model 69
3.3.1 Distance to the Attractor and Class Membership Probability 72

3.4 Architecture and Operation of the RTA NN Model 74

3.5 Training Aspects of the RTA NN Model 75
3.5.1 Minimum Knowledge Incremental Learning 78
3.5.1.1 Bayesian Inference Learning 79
3.5.1.2 Incremental Learning of Attractors, Trajectories and Convergence Rates 81
3.5.1.3 Learning Multiple Classes Incrementally 85
3.5.2 Maximum Knowledge Memorization Learning 86

3.6 Comparison Between the RTA NN Model and Existing Classification Paradigms 86
3.6.1 The RTA NN Paradigm versus One-Step Associative Memory Paradigms 87
3.6.2 The RTA NN Paradigm versus the Neurodynamical Memory Paradigms 88
3.6.3 The RTA NN Paradigm versus the Freeman Neurobiological Model 89

3.7 Summary and Conclusions 90

Chapter 4 : Development and Analysis of a Mathematical Model for the RTA NN 92

4.1 Introduction 92

4.2 Basic Theory of NDS with Globally and Asymptotically Stable Attractors 93
4.2.1 Basic Theoretical Aspects of Continuous-Time NDS 94
4.2.1.1 Equilibrium Point and Stability of a NDS 95
4.2.1.2 Lyapunov Function for a NDS 95
4.2.1.3 Stable Manifolds and Dissipative Systems 97
4.2.2 Discrete-Time NDS 98

4.3 Review of Basic Concepts of Statistical Pattern Recognition 100
4.3.1 Selecting Decision Boundaries to Minimize Cost 102
4.3.2 Rejection Thresholds 104

4.4 Relating T_{jk} and $P(C_j \mid X_k(0))$ at the Attractor Γ_j and on the Boundary B_{IN_J} 105
4.4.1 Monotonicity Constraint on $p(C_j|X_k(0))$ for all $X_k(0)$ on a Common Trajectory c_{jk} ... 110
4.4.2 Overcoming the Monotonicity Constraint on $p(C_j|X_k(0))$ 112

4.5 Relationship between T_j and $p(C_j \mid X_k(0))$ for All Points $X_k(0) \in R_{IN}$ 113
4.5.1 Review of Basic NDS Constraints and Constraints Relating T_{jk} to $p(C_j|X_k(0))$ 114
4.5.2 Mathematical Relationship Between $p(C_j|X_k(0))$ and $\partial X_{jk}/\partial t$ 116
4.5.2.1 Another Look at the Uncle Brian Example of Section 3.3 117
4.5.3 METHOD 1 : Defining $\partial X/\partial t$ via an Arbitrary $T(p)$ Function 120
4.5.3.1 Reconciling Valid NDS Constraints and Equation E_4.5.2-2 124
4.5.4 METHOD 2 : Defining $T(p)$ via an Arbitrary $\partial X/\partial t$ Function 129
4.5.5 METHOD 3 : Defining $T(p)$ and $\partial X/\partial t$ with a Cost Function 133

4.6 Mathematical Formulation of the RTA NN Model Paradigm 136
4.6.1 Mathematical Representation of the RTA NN Model 136
4.6.1.1 Mathematical Model for the Decision Device 139
4.6.2 One-dimensional Example of a RTA NN Classification Solution 141

4.7 Training Aspects of the RTA NN Model 144
4.7.1 The Curse of Dimensionality and the Learning Task Complexity 145
4.7.2 RTA NN Model Mechanisms Available to Reduce Learning Task Complexity 147
4.7.3 Architecture of the RTA NN Subnets and their Training 148

4.8 Summary and Conclusions 149

Chapter 5 : Performance of the RTA NN Model for a Sample Classification Task 151

5.1 Introduction 151

5.2 Neural Net Types and Training Algorithm Used for the RTA NN Model 153

5.2.1 Neural Net Types Used and Rationale for the Choices Made 153
5.2.2 Training Algorithm for the Neural Nets 156
5.2.3 NNET Software Program Description 157

5.3 Description of the Classification Problem and Effect of Orthogonalizing *H(X(n))* ... 158
5.3.1 Description of the 2D Classification Problem 159
5.3.2 Reduction in Learning Task Complexity By Orthogonalizing *H(X(n))* 172

5.4 Results of Training the Subnets with the 2D Classification Problem Data 173
5.4.1 Scope of Training Sessions and Neural Net Structures Utilized 174
5.4.2 Results of the Training Sessions 175
5.4.3 Techniques Used to Improve the Mapping Ability of NN 180
5.4.4 Overall Conclusions Based on the Results of the Training Sessions 182

5.5 Simplification of the Learning Task Complexity by Redefining the Attractor 183
5.5.1 Redefining the Attractor as a Region for the Subnets of the RTA NN 183
5.5.2 Reduction in the Learning Task Complexity 188

5.6 Re-using a Trained Subnet to Learn a New Class 192

5.7 Adding a New Class to a Fully Trained RTA NN Supernet 200

5.8 Repairing Neural Damage in the RTA NN Supernet 201
5.8.1 Definition of the Utilization Factor and Results of Neural Damage Tests 202
5.8.1.1 Summary of Neural Damage Test Results 203
5.8.2 How to Repair Neural Damage and Restore Classification Performance 208

5.9 Disadvantages and Limitations of the RTA NN Model Paradigm 214
5.9.1 Lack of Coupling Between NDS Subnets of the RTA NN Model 214
5.9.2 NDS Limited to Systems where *X(n+1) = H(X(n))* 215
5.9.3 Limited Development of the Incremental Learning Concept 216
5.9.4 Classification Regions must be Connected Regions in the Input Space 216
5.9.5 Each Subnet Must be Trained Over the Entire Input Space 218
5.9.6 Limited Ability to Repair Neural Damage 219

5.10 Summary and Conclusions 219

Chapter 6 : Conclusions and Future Work 222

6.1 Introduction 222

6.2 Summary of Thesis 222

6.3 Conclusions 224

6.4 Ideas for Future Research Work 225

Appendix A : Performance and Stability Issues for LR-RBF and CR-RBF Neurons 227

A.1 Introduction 227

A.2 Architecture and Training Equations for LR-RBF and CR-RBF Neurons 227
A.2.1 Equations for the LR-RBF Neuron 228
A.2.2 Equations for the CR-RBF Neuron 231

A.3 Stability of LR-RBF and CR-RBF Neurons 232
A.3.1 Stability of a CR-RBF Neuron 234

A.4 Prediction Ability of S-RBF, LR-RBF and CR-RBF Neural Nets 237
A.4.1 Problems in the Training of CR-RBF Neurons 237
A.4.2 Training Example with S-RBF, LR-RBF and CR-RBF NN 238

A.5 Conclusions and Summary 241

List of references 243

List of Abbreviations and Acronyms

The following abbreviations and acronyms are used throughout this document.

i.a.w.	in accordance with
CR-RBF	Center Recurrent RBF
EEG	Electroencephalogram
FIR	Finite Impulse Response
LR-RBF	Linear Recurrent RBF
MLP	Multi-Layer Perceptron
MSE	Mean Square Error
NCA	Nerve Cell Assembly
NDS	Nonlinear Dynamical System
NN	Neural Network
OOP	Object Oriented Programming
pdf	probability density function
RBF	Radial Basis Function

RTA	Race To the Attractor
SOFM	Self-Organizing Feature Map
S-RBF	Simple RBF
TDNN	Time Delay Neural Network
VC	Vapnik-Chervonenkis
vs.	versus
2D	two-dimensional
3D	three-dimensional

List of Symbols

The symbols listed below are used throughout this document. Where applicable, the page number provided indicates where the symbol is defined.

$a \Rightarrow b$	→	If a is true then b is true
$a \Leftrightarrow b$	→	If a is true then b is true and vice-versa
$a \subseteq b$	→	a is a subset of b and may include all of b
$a \subset b$	→	a is a subset of b but does NOT include all of b
$a \in b$	→	a is an element of b
$a \notin b$	→	a is NOT an element of b
$\forall$	→	Means "for all". For example, $\forall X \in [0,1]$ means "for all X in the range 0 to 1"
$\propto$	→	Proportional (e.g., $a \propto b$ means "a is proportional to b")
B_{CV_j}	→	This is the boundary of region $\mathbb{R}_{CV_j}$. The latter is a region of points near the attractor where the convergence rate function $\partial X/\partial t$ is constrained to approach 0.0 as $X_k(0) \rightarrow \Gamma_j$. See page 128.
B_{IN_j}	→	Classification boundary for a NDS with attractor Γ_j. All points on or inside the boundary are classified as members of class class $\mathbb{C}_j$, whose attractor prototype is Γ_j. See page 101.

c_{jk} → Defines the trajectory of a NDS system, from the initial point $X_k(0)$ (at time $t = 0$) to the final point Γ_j, which is the attractor of NDS-j. See page 107.

$\underline{c}_{jk}$ → The underline indicates that this is the trajectory "achieved" by trained neural net 'j' (as opposed to the "desired" trajectory c_{jk}).

$\mathbb{C}_{IN}$ or C_{IN} → Is the union of all $R+1$ classes defined for a classification problem. See page 100.

$\mathbb{C}_j$ or C_j → Class 'j'. Points X which are in the region $\mathbb{R}_{IN\_J}$ bounded by boundary B_{IN_J} are members of this class. See page 100.

$\mathbb{C}_{R+1}$ → By convention, this normally refers to the 'unknown' class for classification problems which have R basic classes.

$D_j(\Gamma_j, Y_j(n))$ → This function quantifies the distance between the output of NDS-j at time n and the attractor Γ_j. See page 75.

E_{AV_jk} → Mean Square Error (MSE) between the desired trajectory c_{jk} and the achieved trajectory $\underline{c}_{jk}$ by NN-j, with all its neurons alive. See page 202.

E_{AV_J} → Overall MSE between the M desired trajectories c_{jk} from the training data set and the achieved trajectories $\underline{c}_{jk}$ ($k = 1$ to M) for trained NN-j with all its neurons alive. See page 203.

$E_{AV\_j}(r \notin NN)$ → Overall MSE between the M desired trajectories c_{jk} from the training data set and the achieved trajectories $\underline{c}_{jk}$ ($k = 1$ to M) for trained NN-j with all neurons alive except neuron r (which is dead). See page 203.

Γ_j or $\vec{\Gamma}_j$ → This is the globally and asymptotically stable attractor of a NDS-j. The attractor has dimension P and is the prototype representing all members of class $\mathcal{C}_j$. See page 66.

Γ^R_j or $\vec{\Gamma}^R_j$ → This is the region attractor of trained NN-j, which has point attractor Γ_j within it. The boundary of Γ^R_j is approximated by line segments joining the adjacent end points $X_k(n = N_{j\_\Gamma\_REGION})$ of a set of trajectories whose $X_k(0)$ are on the boundary B_{IN_j}. See page 184.

$H(X(n))$ or $\vec{H}(\vec{X}(n))$ → Iterated mapping function for a discrete-time NDS: $X(n+1) = H(X(n))$ where both have dimension P. See page 65.

$H_j(X(n))$ → Iterated mapping function for a discrete-time NDS with point attractor Γ_j.

$\Delta H(X(n))$ or $\Delta\vec{H}(\vec{X}(n))$ → Iterated mapping function for a discrete-time NDS: $X(n+1) - X(n) = \Delta H(X(n))$ where both have dimension P. See page 66.

$M_{RACE(j)}$ → Function which indicates if subnet-j of the RTA NN model is in the race ('1') or disqualified from it ('0'). This is implicitly a function of time. Normally, a subnet is disqualified if $T_{jk} > T^{\epsilon}_j$. See page 138.

N_{jk} → Number of iterations required for a discrete-time NDS-j to converge from initial point $X_k(0)$ (at iteration $n = 0$) to the attractor Γ_j. This is the discrete-time equivalent of T_{jk}. Strictly speaking, the time required to reach the attractor for most NDS is infinite [15,18]. In this thesis, references to the NDS reaching the attractor in a finite time imply that the distance to the attractor is small. See page 110.

N^{ϵ}_j → This is the maximum number of iterations which inputs to a discrete-time NDS-j can require to reach the attractor Γ_j and still be considered members of class $\mathcal{C}_j$. Usually, only inputs $X_k(0)$ on the boundary B_{IN_j} will have $N_{jk} = N^{\epsilon}_j$. This is the discrete-time equivalent of T_j^{ϵ}.

$\underline{N}_{jk}$ or $\underline{N}^{\epsilon}_j$ → Underlined variable represents the value "achieved" by the trained NN (as opposed to the "desired" value).

$\underline{N}(B_{IN\_j})_{MIN}$ → Minimum number of iterations required by all points $X_k(0)$ on the boundary B_{IN_j} to reach the point attractor Γ_j with the trained NN-j. See page 184.

$N_{j\_\Gamma\_REGION}$ → Number of times which a set of points $X_k(0)$ on the boundary B_{IN_j} are iterated with trained NN-j in order to define the region attractor Γ^R_j. See page 184.

P → Usually denotes the dimension of the input or output vectors of a NDS. For example, $X_k(0)$, Γ_j, $H(X(n))$ all have dimension P.

$P(\mathcal{C}_j|X_k(0))$ → The posterior probability of class $\mathcal{C}_j$ given that an unknown input $X_k(0)$ has been received. See page 102.

$p(\mathbb{C}_j|X_k(0))$ → The posterior probability density function of class $\mathbb{C}_j$ given that an unknown input $X_k(0)$ has been received. See page 102.

$p(\mathbb{C}_j|T_{jk})$ → The posterior probability density function of class $\mathbb{C}_j$ given that the time to convergence of an input $X_k(0)$ is known to be T_{jk}.

p_{jk} → Value of the pdf $p(\mathbb{C}_j|X_k(0))$ for a point $X_k(0)$ on a trajectory c_{jk} for a NDS with attractor Γ_j. See page 107.

p_{j_MAX} → Maximum value of the pdf $p(\mathbb{C}_j|X_k(0))$ for a NDS with attractor Γ_j. The maximum normally occurs at $X_k(0) = \Gamma_j$. See page 107.

p_{jk_MIN} → The minimum value of the pdf $p(\mathbb{C}_j|X_k(0))$ on NDS trajectory c_{jk} which will still result in $X_k(0)$ being classified as a member of class $\mathbb{C}_j$. Therefore, this value corresponds to the point $X_k(0)$ <u>on</u> the classification boundary B_{IN_j}. See page 108.

$P(X|\mathbb{C}_j)$ → The class-conditional probability of X given that the class is known to be $\mathbb{C}_j$. See page 101.

R → Usually refers to the number of classes in a classification problem, <u>excluding</u> the unknown class (the latter is usually labeled $\mathbb{C}_{R+1}$).

$\mathbb{R}_{CV\_j}$ or R_{CV_j} → This region is a subset of the $\mathbb{R}_{IN\_j}$ and has boundary B_{CV_j}. This is a small region around the attractor Γ_j where the convergence rate function is forced towards 0.0 at the attractor so that $\partial X(\Gamma_j)/\partial t = 0.0$. See page 128.

$\mathbb{R}_{IN}$ or R_{IN} → This is the region which encompasses all points X which are in the union of all classes $\mathbb{C}_{IN}$. $\mathbb{R}_{IN}$ is a subset of the input space $\mathbb{R}^P$ where P is the dimension of X. See page 100.

$\mathbb{R}_{IN\_j}$ or R_{IN_j} → This region is a subset of $\mathbb{R}_{IN}$ and has boundary B_{IN_j}. All points X inside this region are classified as members of class $\mathbb{C}_j$ (conversely for points outside that region). See page 106.

S_R → RTA NN 'supernet' with R subnets (i.e., R NDS and therefore R classes plus the 'unknown' class $R+1$). See page 74.

T_{jk} → The time required for a continuous time NDS-j to converge from initial point $X_k(0)$ (at time $t = 0$) to the attractor Γ_j. Strictly speaking, the time required to reach the attractor for most NDS is infinite [15,18]. In this thesis, references to the NDS reaching the attractor in a finite time imply that the distance to the attractor is small". See page 106.

T^{ϵ}_j → This is the maximum amount of time which inputs to a NDS-j can require to reach the attractor Γ_j and still be considered members of class $\mathbb{C}_j$. Usually, only inputs $X_k(0)$ on the boundary B_{IN_j} will have $T_{jk} = T^{\epsilon}_j$. See page 108.

$T_j(p_j)$ → This is the inverse of function $p(\mathbb{C}_j|T_j)$. See page 116.

U_f → Utilization factor. It measures the contribution of a neuron to the mapping function implemented by a neural net. See page 203.

$U_{f\_j}(r)$ → Utilization factor of neuron r in neural net-j with attractor Γ_j. See page 203.

$V(X)$	→	This is the Lyapunov energy function of a NDS. See page 95.
$X_k(0)$ or $\vec{X}_k(0)$	→	Input vector of dimension P to a NDS at time $t = 0$. See page 74.
$X_k(p_j)$	→	This is the inverse of function $p(C_j\|X_k(0))$. See page 122.
$\partial X/\partial t$ or $\partial \vec{X} / \partial t$	→	Convergence rate function for a continuous time NDS. See page 94.
$\partial X_{jk}/\partial t$	→	Convergence rate function for a continuous time NDS-j on a trajectory from initial point $X_k(0)$ to the attractor Γ_j.
$Y_j(n)$ or $\vec{Y}_j(n)$	→	Normally used in conjunction with discrete-time NDS with attractor Γ_j : $Y_j(n+1) = H(Y_j(n))$. This is used to distinguish the output value of a NDS at any time from its input $X_k(0)$ at time $n = 0$ (i.e., $Y_j(0) = X_k(0)$). See page 74.
$Y_{jk}(n)$	→	Represents the output of NDS-j with attractor Γ_j at the n-th iteration when the initial input at time $n = 0$ was $X_k(0)$.
$\underline{Y}_{jk}(n)$	→	The underline indicates that this is the "achieved" iterated value by trained NN-j, as opposed to the "desired" value $Y_{jk}(n)$.

Chapter 1 :

Introduction

1.1 Introduction

The challenge of classifying complex patterns reliably is a problem whose solution has yet to be found. Applications involving handwriting recognition, voice recognition or the identification of persons from facial features are a few examples where classification performance remains poor even today.

And yet, the ability of the human brain to perform such tasks with little apparent effort holds the promise that reliable classification of complex patterns must be possible since we do it all the time. This promise continues to motivate researchers from the fields of neurobiology to mathematics in the quest for better classification models which can capture this amazing ability of the brain.

1.2 Thesis Motivation

It is partly this promise which motivates us to seek a better model to classify patterns. Another motivator has been the realization that there exist no classification paradigm today which duplicates one of the brain's most pervasive characteristic; *the importance of TIME in the computation process*.

The brain is a dynamical system whose state evolves with time. Stimuli from the outside world are processed, transformed and fed-back throughout the brain. These signals will in turn trigger new activity patterns in the brain and may suppress existing ones. All these signals will then interact in a complex and critically time-dependent manner to produce a suitable response to that stimuli.

As a result, the process of pattern recognition inside the brain is also a time-dependent evolution of states where the initial image of the unknown pattern is progressively transformed into a form which represents the class of that pattern. When that state is reached, recognition occurs.

1.2.1 Statement of the Thesis

There are no classification paradigms in existence today where time has such an important role in the recognition process. Although there are neurodynamical models in use where the system state evolves with time, it will be shown (in chapter 2) that time is NOT a determining factor in the classification process for these paradigms.

This thesis seeks to address this deficiency by developing a model which duplicates the brain's computational process of time-dependent state evolution and where time is one of the key parameters in determining the classification answer. The model proposed can be summarized in a few words as follows:

The classification of an unknown input pattern can be achieved by using a system where there are as many neural nets (NN) as there are possible output classes. Each NN implements a Nonlinear Dynamical System (NDS) which transforms the unknown input into a unique class prototype 'j' (an attractor) in an amount of time which is inversely proportional to the probability that the input belongs to the class C_j. All the NDS are fed the input pattern simultaneously and the winner 'r' of the 'Race To the Attractor' classifies the unknown input as a member of the class C_r which it represents.

Modeling the pattern recognition process with this approach has significant advantages over existing classification paradigms.

1.3 Objectives of the Thesis

The objectives of this dissertation are as follows:

1. Develop the 'race to the attractor' neural network (RTA NN) model.

2. Analyze the RTA NN model mathematically and describe its advantages, disadvantages, capabilities and limitations.

3. Test the RTA NN model against a sample classification problem and demonstrate its ability to perform correct classifications.

1.4 Contribution of the Thesis

The contributions of this thesis to general knowledge are as follows:

1. A new paradigm for the classification of patterns is introduced, which captures the importance of time in the computational process of the brain.

2. A mathematical framework is developed which defines the conditions under which the model can achieve reliable classification of patterns.

3. Software algorithms are produced which can be used to test the RTA NN model against practical classification problems.

1.5 Thesis Organization

In chapter 2, we start off by describing how learning and recall are believed to occur in a 'real' brain. This description is based on a neurobiological model defined by Freeman [23]. Following this, the key characteristics which make the brain such a good classifier are defined. Then, mechanisms are described through which these characteristics can be implemented in a software or mathematical model. The remainder of that chapter focuses on a review of traditional classification paradigms based on neural networks. Their mode of operation is described and their performance in terms of the key characteristics of a good classifier is assessed.

In chapter 3, the RTA NN model is introduced. The chapter starts with a basic description of nonlinear dynamical systems (NDS). This is followed by a description of the basic operational concept of the RTA NN paradigm. The fundamental axiom which defines its operation is then introduced and discussed. Following this, the architecture and operation of the RTA NN model are described in general terms. Then, the learning process in the RTA NN model is discussed. The remainder of that chapter compares the RTA NN model with the classification models introduced in chapter 2.

Chapter 4 discusses the RTA NN model in more detail. This chapter develops a formal mathematical framework for the RTA NN paradigm and analyzes its properties in a more rigorous manner than was possible in the introduction of chapter 3. The analysis encompasses the theory of nonlinear dynamical systems (NDS), statistical pattern recognition (i.e., Bayesian theory), and the fundamental axiom of the RTA NN model. Essentially, the ultimate aim of this analysis is to transform the axiom from a 'word' description into a set of equations, constraints, algorithms, etc, which can then be used to implement a 'real' classification problem with the RTA NN model.

In chapter 5, we proceed from theory to practice. This chapter describes the performance of the RTA NN model for a sample classification problem. The objective of this chapter is to demonstrate that the RTA NN model does indeed achieve the key characteristics of a good classifier and that it does so by using the mechanisms identified in chapter 2. This chapter also discusses the limitations of the RTA NN model and possible ways to overcome these.

Chapter 6 provides a summary of this dissertation and conclusions. Ideas for further research work are also proposed.

Finally, Appendix A provides an analysis of the stability properties and learning ability of recurrent Radial Basis Function (RBF) neurons. These were used extensively in the neural nets trained on the classification problem of chapter 5. This analysis is relevant to

the RTA NN performance described in that chapter but was left out in order NOT to clutter that chapter with such detailed mathematical analysis.

Chapter 2 :

The Classification Process in the Brain : Current Models and Key Aspects

2.1 Introduction

In this chapter, the classification problem and its solution are examined. Our objectives are as follows:

1. Explain how a 'real' brain implements the learning and recall of classes;

2. Identify the key characteristics which make the brain such a good and reliable classifier;

3. Identify mechanisms through which these characteristics can be implemented in a model (i.e., software, hardware and/or mathematical);

4. Examine traditional classification paradigms using Neural Networks (NN) and determine to what extent they achieve these key characteristics.

Our interest in the brain is motivated by the fact that it is arguably the best classifier system known to us at this time. Our first priority is to determine (as much as possible) how the whole classification process occurs in a 'real' brain. This is a big challenge since no one at this time can claim to know how the brain really operates. Nevertheless, a neurobiological model first proposed by Freeman [23,52] in 1987 has withstood the test of time (so far) and has seen many of its aspects confirmed by more recent research [25,60,61]. It is therefore as good a candidate as we can hope to find to explain how the brain classifies patterns. This model is described in section 2.2.

In section 2.3, the key characteristics which make the brain such a good and reliable classifier are identified and described. Once we know the attributes of the best classifier around, it is necessary to determine how these can be implemented in a software or mathematical model. Section 2.4 identifies and describes the processes, methods, etc, which will allow us to do this.

Subsequently, current classification models which seek to capture the learning process in the brain are discussed in section 2.5. The focus on this section is on determining to what extent they achieve the key characteristics of a good classifier and also how they compare to the Freeman neurobiological model from section 2.2. This information will ultimately allow us to compare our model to those currently available in terms of implementation approach and performance (this will be discussed in chapter 3).

Finally, section 2.6 summarizes the information presented in this chapter and offers some conclusions.

2.2 The Freeman Neurobiological Model

The model of the brain proposed by Freeman [23] and Skarda [52] is derived in large part from an analysis of experimental recordings of brain activity during learning and recall events. Hence, the model was developed with an heuristic approach and is not supported by hard evidence. It is described in general terms and seeks mainly to explain how learning and recall occur in the brain and how these lead to the brain activity measured in the first place.

Throughout this document, this neurobiological model will be referred to as the *Freeman model*, in recognition of the author who is most often credited in the literature with defining it [23,25,52,60,62]. The experimental results of Freeman and Skarda [23,52] are described in section 2.2.1. In section 2.2.2, the central aspect of the Freeman model is introduced and some of the terminology associated with that model is explained. Section 2.2.3 describes the Freeman model in detail.

2.2.1 Experimental Results of Freeman and Skarda

Most of the experiments performed by Freeman and Skarda [23,52] were focused on the neurons of the olfactory system (i.e., the system related to the sense of smell). The experiments usually involved rabbits which were trained to recognize certain odors and behave in a certain way when these were presented. Experimental recordings involved electrodes which were positioned in a gridlike array on the surface of the olfactory bulb. The rabbits were exposed to odorants which were either known to them or had never been encountered before. The *electroencephalogram* (EEG) tracings of the electrodes were analyzed before, during and after exposition to these odorants in an attempt to understand how the latter were learned in the first place and how they were recalled afterwards.

To better understand the results reported by the authors, it is necessary to first introduce the various parts of the olfactory system and to discuss how they interact among themselves and with the rest of the brain. This information is shown in Figure 2.2.1-1. When molecules of a scent enter the nose, they excite some receptor cells which are specialized to respond to that odorant. These excited cells fire pulses through axons and into the olfactory bulb. The bulb responds to these pulses by generating its own unique activity pattern, which is transmitted to the olfactory cortex. From there, new signals are generated which are sent to many other parts of the brain. It is believed that recognition of the odorant ultimately occurs in the frontal cortex, where activity patterns from various subsystems are integrated and interpreted as a specific odorant.

When a familiar scent enters the nose, Nerve Cell Assemblies (NCA) which are specialized in responding to that odorant excite the bulb. The number of receptors activated is proportional to the intensity of the scent while their location in the nose is related to the nature of the scent itself [23]. Recordings indicate that this excitation signal induces a burst of activity in the bulb. This burst is characterized by signals of higher amplitude, higher frequency content and more regularity than EEG tracings when the olfactory system is at rest. When the animal exhales, the burst of activity ends. The

authors [23,52] noted that when the olfactory system is at rest, EEG recordings still exhibit constant activity. But the waveforms recorded at those times are highly irregular and unpredictable, having the appearance of "freehand scrawls" [23].

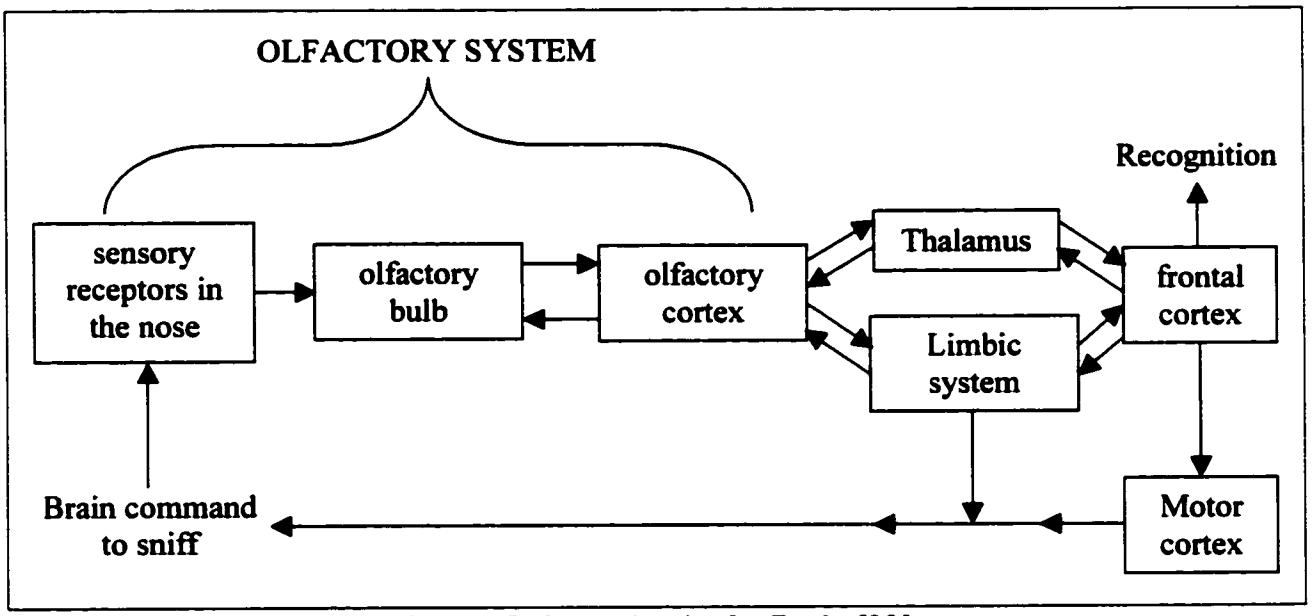


Figure 2.2.1-1 : Flow of Olfactory Information in the Brain [23]

Analysis of the bursts corresponding to the recognition of familiar odorants has revealed that they contain a common carrier wave in the 40 Hz range. An important observation made by the authors was that the shape of the EEG traces never repeat themselves. Even when the same odor is repeatedly sniffed, the shape of the EEG waveform during the burst is different each time. Hence, the authors conclude that the shape of the waveform does not correspond to any specific odorant. Instead, they have discovered that the odorant is discernible only by the bulbwide spatial pattern of average carrier-wave amplitude, which they describe with contour lines of neural activity. This average amplitude is measured over the duration of a burst. It is these specific patterns of contour lines which are unique for each odor and which reappear over and over again, when the same odor is repeatedly sniffed.

This concept is easier to understand with an illustration. In the top part of Figure 2.2.1-2, assume that the *X* and *Y* axes represent the spatial coordinates of a slice of neurons in the

olfactory bulb, relative to some reference point (at (0,0)). Assume that the *Z* axis represents the *intensity* with which these neurons generate a quasi-periodic pulse signal (e.g., with a 40 Hz component). Hence, peaks in the *Z* coordinate indicate areas in the bulb where groups of neurons fire almost continuously, thereby resulting in a composite signal of high average amplitude. Conversely, low values of *Z* in that figure indicate that the corresponding neurons are more quiet, so that the average amplitude of the composite EEG signal in that region is much lower.

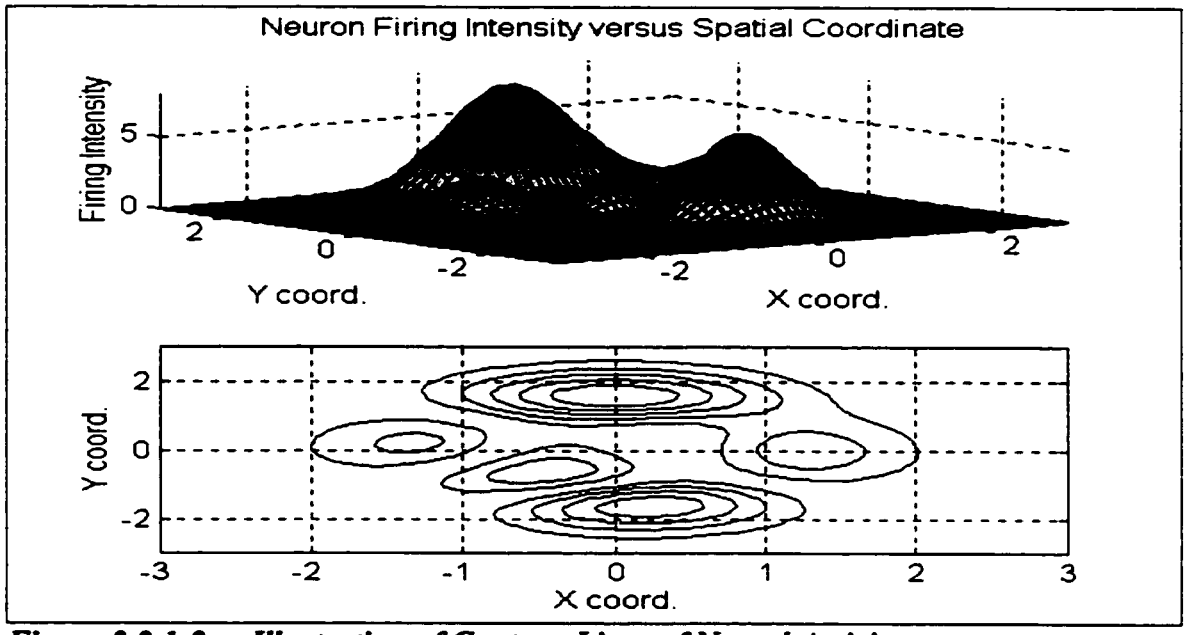


Figure 2.2.1-2 : Illustration of Contour Lines of Neural Activity

The contour lines on the *XY* axes shown in the bottom of Figure 2.2.1-2 connect neurons with a similar intensity of activity. According to the authors, it is those contour lines of average activity which reappear consistently when a specific odor is repeatedly sniffed.

The reason why the EEG trace itself looks different for each repeated sniff of the same odorant is probably related to the fact that neurons do NOT fire with perfect synchronicity or in the same order for each sniff. Since the EEG records the composite activity of thousands of neurons, the shape of that signal will look different for each sniff because the neurons fire in a different order and with different relative delays, despite the fact that the contour lines of average intensity remain roughly constant.

In summary, their experimental results indicate that odorant recognition in the brain starts with a few specialized cells in the nose (i.e., a NCA) being excited by a specific odor. This NCA will in turn feed an excitation signal into the olfactory bulb. This input signal will trigger an avalanche of activity in the bulb, which will be characterized by a composite EEG signal whose map of average amplitude versus spatial coordinates (of neurons) is always the same for a given odorant. The bulb will generate a unique map of average amplitude for each unique odorant learned. Recognition in the frontal cortex involves the perception of this pattern of activity.

2.2.2 Central Aspect of the Freeman Neurobiological Model

The central aspect of the Freeman model is that the olfactory bulb is viewed as a *nonlinear dynamical system* (NDS) which has a large number of *chaotic attractors*, and where each attractor corresponds to a learned odorant. In this section, the concepts of NDS and chaotic attractors are explained briefly.

NDS are discussed in section 2.4.1.1 and again in chapters 3 and 4. But for now, it is sufficient to view them as systems characterized by the pervasive presence of feedback. In such a system, any input signal entering the NDS is processed though the system over and over again, altering the state of the system continuously. As a crude analogy, consider a pool table without friction where all the numbered balls are initially lined up in triangular array. Then, a white ball is launched towards the numbered balls (i.e., the input to the NDS). The initial impact will send all the balls scurrying around in different directions. As they bounce against the walls of the table and against each other (i.e., feedback), their trajectories will change continuously. The pool table can be viewed as a NDS system whose 'state' (i.e., the instantaneous position of all the balls) changes continuously with time as a result of massive feedback. In a similar manner, the olfactory bulb can be viewed as a NDS whose state (e.g., EEG recording) is defined by the interaction of neurons which excite (or suppress) each other's activity in response to an input signal from the NCA in the nose.

An attractor is a specific state which the NDS has a tendency to assume after some time has elapsed. In the analogy of the pool table, if one end is tilted upwards, all the balls will tend to converge towards the other end of the table after a while. The lowest end of the table then acts as an attractor towards which all balls will tend to go. Another useful analogy is to consider a bowl in which a marble is dropped. No matter where the marble is positioned inside the bowl, it will be attracted towards the bottom. If one considers the bowl as a NDS whose 'state' is defined by the position of the marble in it, then all initial states will eventually converge to the attractor at the bottom of the bowl.

In a similar manner, the EEG patterns with constant amplitude contour lines described by Freeman can be viewed as attractors to which the olfactory bulb will tend to converge to after a while, when a suitable excitation input is present. The presence of multiple attractors can be understood with the analogy of a bowl with several depressions in it. For example, the top part of Figure 2.2.1-2 can be viewed as such a bowl, with four depressions in it (the bowl is shown upside down in that figure). In that bowl, the bottom of each depression corresponds to an attractor. Any marble dropped in such a bowl will converge to one of the attractors, depending on the initial position where it is dropped. A similar principle is at work in the Freeman model, but here it is the excitation signal from the NCA in the nose which determines which attractor the olfactory bulb 'falls' into.

A nonlinear dynamical system (NDS) which exhibits chaotic activity is characterized by two main attributes [8,40]:

1. The evolution of its 'state' with time is completely deterministic (i.e., NOT random).

2. The evolution of its state with time exhibits extreme sensitivity to minute changes in the initial conditions.

As an example of a deterministic NDS, consider again the analogy of the pool table without friction. If the position of all the balls and the angle & force with which the white

ball hits the other balls are known precisely, the trajectory of each ball on the pool table can be predicted accurately from simple laws of physics. In theory, if the initial conditions are restored exactly, each ball will follow the same trajectory each time the NDS is reset and restarted.

This is unlike a system whose state changes randomly. When true randomness is present, the state of a system in the future cannot be predicted accurately and the progression of its state cannot be repeated, even if the exact same initial conditions are restored. That is because the progression of its state is not controlled only by initial conditions but also by a probability measure. For example, consider a system which can assume a 'state' of +1 or '-1' at each time step n. In that system, the state at a time '$n+1$' will be the same as the state at time 'n' 60 times out of a 100. Hence, if its state at time $n = 0$ is '+1', it has a 60% chance of being in state '+1' at time $n = 1$ and a 40% chance of being in state '-1' at that time. For such a system, the sequence of states as time elapses can never be reproduced exactly because the state of the system depends on an uncontrollable (and unpredictable) random event. For example, if a sequence of states was {+1,+1,-1,...}, resetting the system to the initial state +1 will not necessarily result in the same sequence of states being repeated.

The extreme dependence on initial conditions for a chaotic system can also be understood from the analogy of the pool table without friction. Assume that the initial state of the system is restored almost exactly except that the angle with which the white ball hits the first numbered ball is different by a small amount (e.g., 0.00001°). As a result, the first numbered ball hit will assume a slightly different trajectory and so will the white ball when it bounces off. The next balls hit by these two balls will then also assume slightly different trajectories, and so on for subsequent collisions. Initially, the trajectories of all the balls will look roughly similar to those recorded the first time the experiment was conducted (i.e., without the 0.00001° error). But these minute trajectory errors will accumulate as time elapses and eventually, the trajectories of the balls will look very different from those of the initial experiment. This is the hallmark of chaos, where minute differences in initial conditions eventually lead to significant differences in the state of a

NDS *because these small initial differences accumulate over time to create large differences*. This is sometimes referred to as exponential divergence of nearby orbits [15].

The same chaotic NDS behavior was described by Freeman when he noted that the EEG traces had different shapes each time, even when the same odorant was sniffed repeatedly. As noted earlier, minute changes in the order in which neurons fire and in the delay between their firings result in these differences, despite the fact that the spatial contour lines of average neural activity do not change. This is a good example of a *chaotic attractor*, which is characterized by a time-plot of NDS state which is chaotic (i.e., never repeats itself exactly) but which has attributes which are always repeated nevertheless (i.e., the contour lines of average neural activity are the attractor which reappears every time).

Figures 2.2.2-1 and 2.2.2-2 illustrate an example of a chaotic NDS known as the Lorenz attractor [19,40]. This is a chaotic attractor which exhibits the exact same characteristics mentioned above. The Lorenz attractor is a NDS whose state at any time is defined by a three-dimensional vector *(x,y,z)* whose value is deterministic (i.e., defined by known equations). The traces *X(t)*, *Y(t)* and *Z(t)* are shown in Figure 2.2.2-1. Just like the EEG traces described by Freeman, they appear almost periodic but actually never repeat themselves exactly. Nevertheless, the Lorenz attractor will always exhibit a shape similar to that shown in Figure 2.2.2-2, regardless of the fact that the trajectories *X(t)*, *Y(t)* and *Z(t)* never repeat themselves.

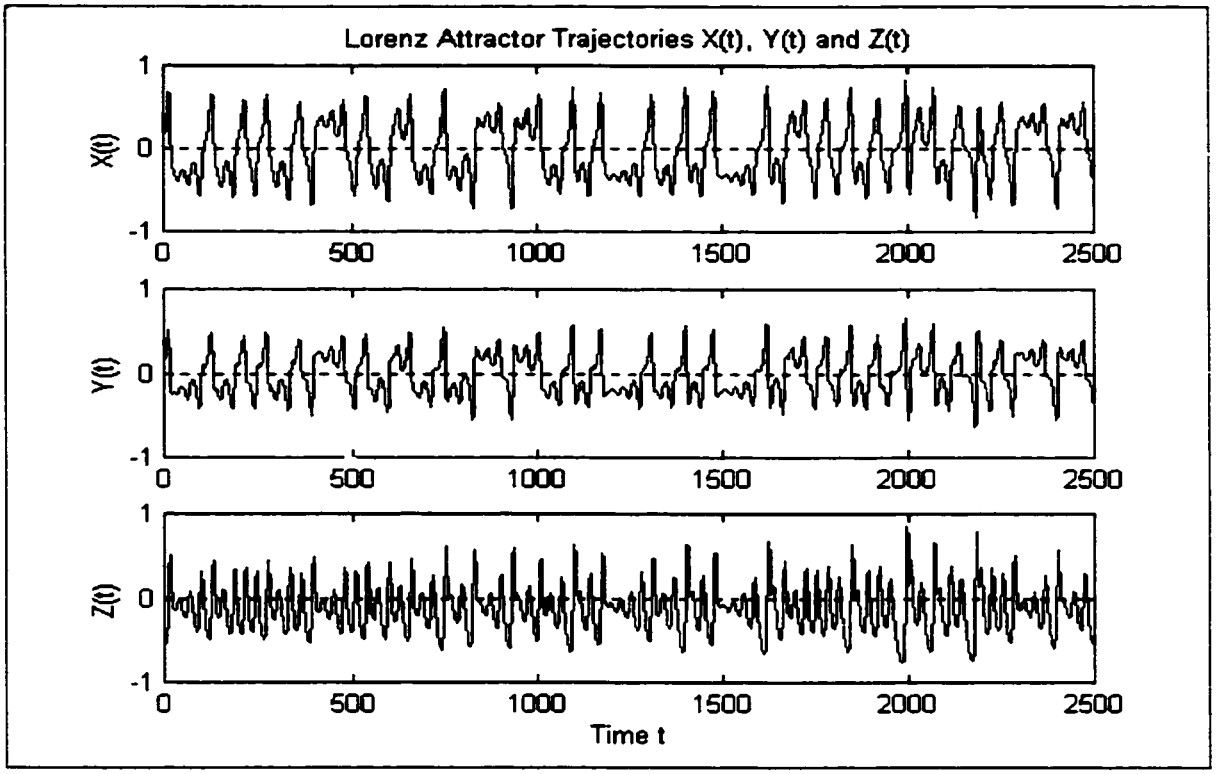


Figure 2.2.2-1 : Trajectories of the Lorenz Attractor

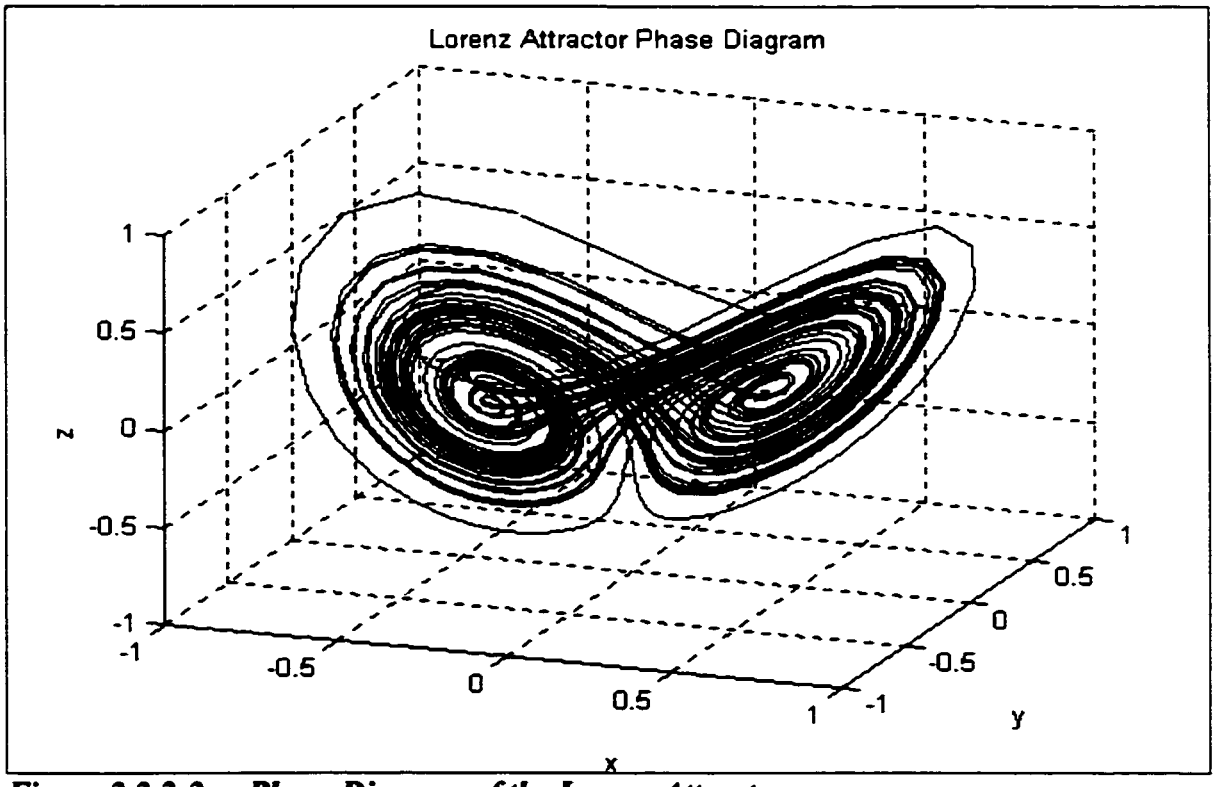


Figure 2.2.2-2 : Phase Diagram of the Lorenz Attractor

2.2.3 Description of the Freeman Model

With the information provided in the previous section, it is now possible to describe the Freeman model in more detail.

As indicated earlier, the central aspect of the Freeman model is that the olfactory bulb is viewed as a nonlinear dynamical system (NDS) which has a large number of *chaotic attractors*, where each attractor corresponds to a learned odorant.

When the olfactory bulb is at rest, the NDS is in a chaotic attractor which has little regularity apparent in it (i.e., an almost noise-like appearance). When a familiar odorant is presented to the animal, a nerve cell assembly (NCA) in the nose will be excited by the odorant and will generate pulse signals which impinge on the olfactory bulb. As indicated earlier, the location of the activated receptors in the nose define the nature of the scent. As a result, different odorants will generate different pulse signals.

As mentioned earlier, these NCA pulse signals induce a burst of activity in the bulb. In the Freeman model, different NCA pulse signals will result in the bulb converging towards different chaotic attractors during these bursts of activity. It is these attractors which result in EEG signals characterized by high amplitude and high frequency traces recorded by Freeman when recognition occurs. Once the bulb is in one of these attractors, the frontal cortex somehow senses the spatial pattern of average neural activity corresponding to that attractor and interprets it as a specific odor. Note again that these spatial contour lines of activity constitute the only common features of the bulb state which are always present for the same odor (e.g., just as Figure 2.2.2-2 is the only constant feature which the Lorenz attractor always exhibits, even though the actual trajectories of Figure 2.2.2-1 are always changing).

Freeman postulates that this model is valid not only for the olfactory system, but for most processes of pattern recognition occurring in the brain. For example, when we recognize a 'hammer', a different chaotic attractor is generated in the visual cortex than when a 'key'

is recognized. In Figure 2.2.3-1, an attempt has been made to illustrate this concept visually.

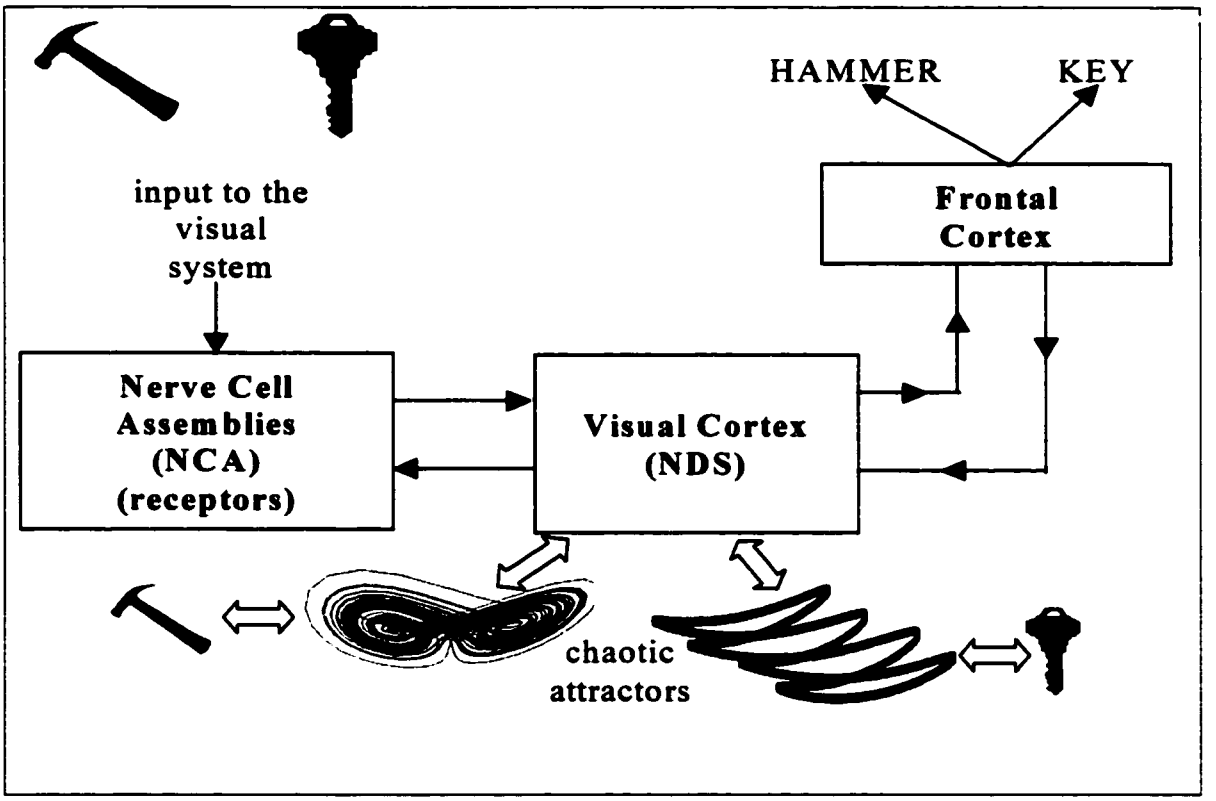


Figure 2.2.3-1 : Freeman Model of Pattern Classification

Freeman also postulates that chaos is an essential part of the brain because it continuously produces novel activity patterns, which may allow the creation of new attractors when the brain is required to learn a new pattern or class. The learning of these new attractors involves the selective strengthening of synapses between certain neurons, in a process which is best described by the classical Hebbian learning rule [26]. This selective strengthening molds both the NDS and NCA receptors so that, when the new odor is experienced again, the new attractor will recur more quickly and with higher intensity.

Another interesting aspect of the Freeman model is the presence of competition between different NCA. Recall from section 2.2.1 that the number of cells in the NCA which are excited is proportional to the intensity of the odorant itself. In addition, the location of the cells which are excited is related to the nature of the scent itself. As a result of this, when

two or more odors are present, two separate NCA will compete for the attention of the bulb, with each NCA attempting to lure the bulb towards a specific attractor. Freeman postulates that a similar competition occurs when we look at visual information which is ambiguous. For example, consider Figure 2.2.3-2, which can be interpreted as either two black faces against a white background or as a white vase against a black background. The interpretation achieved by the brain depends on which visual cues the observer focuses on. When the focus is on the faces, the NCA corresponding to these will fire in the brain with more intensity, thereby luring the visual cortex to generate the attractor corresponding to two faces. But if the attention of the observer drifts to the white background, the NCA for the white vase fires with more intensity, thereby luring the cortex towards the corresponding attractor.

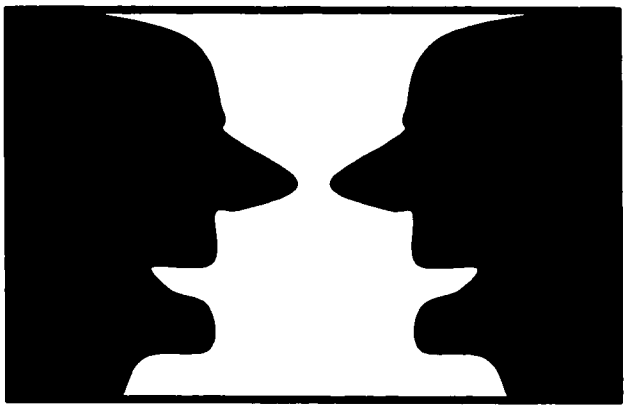


Figure 2.2.3-2 : White Vase or Two Faces

This model of the brain as a NDS with chaotic attractors has been elaborated upon by several other researchers, who have confirmed the validity of certain aspects of this model in the human brain [25,60,61]. The concept of using chaotic attractors to classify patterns has also been analyzed with a more mathematical approach by several researchers in the fields of physics, engineering, mathematics, etc [1,8-10,13,17,24, 31,34].

One advantage of the Freeman model is that it is closely linked to experimental EEG data measurements. In that respect, it seems to explain how learning and recall in a 'real' brain truly occur. Unfortunately, since the model is not rigorously defined in mathematical terms, it is not possible to implement it in a software simulation (for example). That is because the exact process through which an input gets transformed into a specific attractor is not really understood at this time. As a result, the Freeman model is of limited use to us in our endeavor to define a novel (and hopefully better) classification model.

2.3 Key Characteristics which Make the Brain a Good and Reliable Classifier

In this section, the main attributes which make the brain a good classifier system are identified.

The human brain is exceptionally good at classifying objects reliably. Think of how we can recognize familiar faces even when seen from different angles and despite irrelevant clutter such as jewelry, sunglasses, new hair styles, etc. Some of the key characteristics which make the brain such a good and reliable classifier are described below:

1. **The ability to accurately learn complex classification boundaries** using relatively simple devices (i.e., neurons). This allows us to distinguish between similar looking objects such as two identical twins (for example) by relying on small differences.

2. **The ability to learn new classes without forgetting the ones already in memory** (i.e., incremental learning without memory loss). For example, this ability makes it possible for a child to learn letters 'D', 'E', 'F', etc, without forgetting letters 'A', 'B', 'C'.

3. **The ability to learn a new class with a level of effort related mostly to the complexity of that class** (i.e., independently of how many other classes are already in memory). For example, this ability means that learning letter 'Z' is not significantly harder than learning any other letter, even though 'A' to 'Y' are already in memory.

4. **The ability to repair the damage caused by the death of neurons and continue to perform reliable classifications**. Since brains are made of live neurons, some of these are either dying every day or the strength of synaptic connections between them change continuously through life. Despite these changes, the brain continues to classify reliably. As another example, victims of

injury to the brain may suffer a temporary loss in their ability to recognize people. But often, most (if not all) of that damage is repaired by the brain even though the neurons previously involved in the classification task are dead.

The first characteristic has been a constant source of hope and motivation for researchers in neural networks (NN). The fact that the brain is a reliable classifier proves that it is possible to resolve complex classification problems by using nothing more than a set of simple devices (i.e., a network of neurons).

The second characteristic is an essential feature of all living organisms with a complex brain. For example, mammals are born with little knowledge of their environment or how to survive in it. This information must be acquired gradually and without losing previously acquired knowledge. As will be seen in section 2.5, few NN classification models paradigms are able to achieve incremental learning without memory loss.

The third characteristic is related to the capacity to learn new information independently of previously-acquired information. This characteristic has proven to be very hard to achieve in NN models, as a result of the *curse of dimensionality* [26]. This concept is introduced in section 2.4.3 but will be discussed in more detail in section 4.7.1 (page 145).

Despite the fact that the brain is a very complex system, it is NOT a fragile system prone to failure. In fact, the brain exhibits amazing resilience in the face of changes in its own makeup, as a result of trauma, natural death of neurons or frequent changes in inter-neural connections. Many examples exist of cases where persons have survived extensive damage to regions of the brain and were able to recover most of their ability afterwards [14,30]. This indicates the ability to repair damage without necessarily replacing the dead neurons. This characteristic is a necessary asset for organisms hoping to survive in a tough environment. As will be seen in section 2.5, few NN models currently available can handle the death of neurons 'gracefully' (i.e., by maintaining or restoring their classification performance).

2.4 Mechanisms to Implement the Key Characteristics of Learning in a Model

The goal of this section is to identify mathematical tools, processes, techniques, etc, through which the key characteristics of a good classifier (from section 2.3) can be implemented in a model (i.e., software, hardware or/and mathematical). Note that throughout this document, these tools, techniques, etc, are simply referred to as *mechanisms*.

In sections 2.4.1 to 2.4.4, the four key characteristics of section 2.3 will be examined again. Four mechanisms which allow these characteristics to be implemented will be described. Section 2.4.5 summarizes these mechanisms.

2.4.1 Learning Complex Classification Boundaries

The key to solve difficult problems with simple neurons is to simplify the complexity of the learning task as much as possible. In this section, three mechanisms will be identified through which this complexity can be reduced. But first, let us examine a simple example of a classification problem.

In its simplest form, a classification problem can be represented by a *mapping function* of the form $y = H_S(\boldsymbol{X})$. Here, y is a scalar value identifying the class (e.g., 1, 2, 3, etc). $\boldsymbol{X}$ is the input vector (of dimension $P \geq 1$) which is the set of parameters used to describe and classify the object. For example, to classify a person, $\boldsymbol{X}$ would include information such as hair color, color of the eyes, height, facial features, etc. H_S is the function which maps certain ranges of values of $\boldsymbol{X}$ to a specific class. The solid line in the top part of Figure 2.4.1-1 is an example of such a function, with the input vector X having dimension $P = 1$ (note that when $P > 1$, a bold symbol will be used : $\boldsymbol{X}$). Here, the four classes 'A', 'B', 'C', 'D' are represented by $y =$ 0.0148, 0.236, 0.653 and 0.990 respectively. For instance, values of X in the ranges [0.55,0.58] and [0.82, 1.0] belong to class 'D'. The task of identifying persons can understood in a similar manner, where the function H_S has an

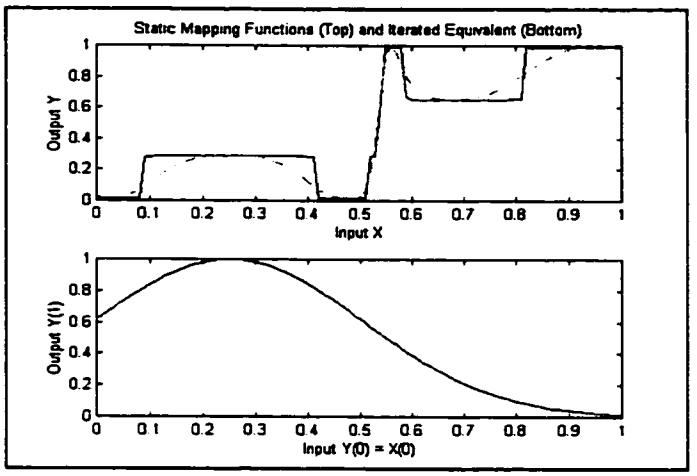


Figure 2.4.1-1 : Static versus Iterated Mappings

input vector $\boldsymbol{X}$ with a large dimension P. Note that we refer to H_S as a *static mapping function* because the answer y is obtained in one time-step only.

Any such function where the number of classes is large and/or the input vector dimension is large will necessarily be a very complex function. Complexity can be defined in many ways but for our purpose, a simple definition will do for now :

> *complexity increases as the number of peaks and valleys in the mapping function increases, it increases as the slopes of these peaks/valleys become steeper and it increases as the number of discontinuities (e.g., 'sharp corners') increases.*

An example of a function which measures complexity as indicated above is given by equation E_4.7.1-2 of chapter 4 (page 146).

The more complex a function is, the harder it is for the brain (or a neural network) to learn it. It is therefore desirable to reduce the complexity of the mapping function as much as possible in order to facilitate the learning task. The following three mechanisms will allow a reduction in the mapping function complexity:

1. Replace the static function to be learned H_S by a simpler 'iterated' mapping function H_I.

2. Redefine the output classes as ranges of values instead of discrete values.

3. Re-use previously learned mapping functions and apply them to new classification problems.

These three mechanisms are discussed in subsections 2.4.1.1, 2.4.1.2 and 2.4.1.3 respectively.

2.4.1.1 Reducing Complexity by Using an Iterated Mapping

An *iterated mapping function* H_I can be represented as follows: $Y(n) = H_I(Y(n-1))$ where the input at time n is the output of the function at time $n-1$. The input at time 0 is $Y(0) = X$, the unknown input vector to identify. In that way, the original input is transformed several times, by being fed back into the mapping function H_I. For example, consider the function H_I given below and shown in the bottom part of Figure 2.4.1-1:

$$y(n) = H_I(y(n-1)) = \exp\left(-\frac{(y(n-1)-0.25)^2}{0.13}\right) \qquad \textbf{(E\_2.4.1.1-1)}$$

If this function is iterated 20 times and the value $y(20)$ is plotted versus $y(0) = X$, the achieved mapping is identical to the static function H_S shown in the top part of Figure 2.4.1-1 as a solid line. Clearly, the function H_I in that figure is significantly smoother (i.e., less complex) than the function H_S. Hence, the complexity has been reduced by using an iterated mapping but at the cost of a longer computation time before the classification answer is available (i.e., 20 time steps for H_I versus 1 for H_S).

Systems which utilize iterated mapping functions (e.g., as per equation E_2.4.1.1-1) are often referred to as *Nonlinear Dynamical Systems* (NDS). It was seen in section 2.2 that the brain itself can be viewed as a complex NDS where such iterated mappings are common. However, unlike the NDS of section 2.2 which has chaotic attractors, the example illustrated in Figure 2.4.1-1 (and given by equation E_2.4.1.1-1) is a NDS with stable, point attractors. NDS systems will be discussed further in chapters 3 and 4.

2.4.1.2 Reducing Complexity by Defining Output Classes as Ranges of Values

The complexity of the mapping function to be learned by the NN can also be reduced by simply redefining the output classes from point values to ranges of values. For the example of Figure 2.4.1-1 (page 40), if class 'D' is redefined as values of y in the range 0.8281 to 1.0 (instead of 0.99 as before), the dashed line in the top part of that figure becomes our new static mapping function H_{S_NEW}. The same input values X in [0.55, 0.58] and [0.82, 1.0] map to class 'D' as was the case for H_S. By defining the other classes with suitable ranges of values for y, the same classification is achieved with H_{S_NEW} than with H_S but using a less complex function. Note that the map H_I (bottom of Figure 2.4.1-1) only needs to be iterated 4 times to achieve H_{S_NEW}, versus 20 iterations when the output classes are defined as points. The shorter computation time requirement is a result of the reduced complexity of the mapping function.

Note also that the reduction in the mapping function complexity has been achieved by increasing the complexity of the classification boundaries (i.e., from a point to a line). Hence, complexity has been transferred from the mapping function itself to the classification boundary.

In chapter 5, it will be shown that by redefining the attractor of a NDS from a point to a region, the complexity of the mapping function which must be learned by the NN is reduced in the same way (e.g., see Figure 5.5.2-1 on page 190).

2.4.1.3 Reducing Complexity by Re-Using Learned Mappings for New Classes

Learning a new classification mapping function can sometimes be simplified greatly by re-using a NN which has already learned a mapping function.

For example, assume that a NN has been taught the classification mapping function of Figure 2.4.1-1. Assume that it then becomes necessary to add two new classes 'E' and 'F' corresponding to input values of $X \in (1.0, 1.54]$ and $X \in (1.54, 2.0]$ respectively. The

same function H_S of Figure 2.4.1-1 can be used to achieve this. First, note that if the output classification regions are redefined as $y \in [0, 0.5]$ and $y \in (0.5, 1.0]$, the input values $X \in (0.0, 0.54]$ and $X \in (0.54, 1.0]$ will map to these two output classes respectively. These input ranges only need to be 'offset' by a constant value of 1.0 in order to implement the new classes 'E' and 'F', without any need to train a neural net.

Hence, by using simple offsets and redefining classification boundaries for an existing iterated map, a new classification mapping function can be created. As a result, one can use 'copies' of an achieved mapping function (i.e., a trained neural network) and re-use it to implement other mappings. Examples of how this process can be applied in practice will be shown in chapter 3 (e.g., section 3.2.1, starting on page 67) and in chapter 5 (e.g., section 5.6, starting on page 192).

2.4.2 Learning New Classes with No Memory Loss

The ability to learn to recognize new classes without forgetting previously learned ones can be achieved by ensuring that the different classification mapping functions are learned independently of each other. An example will help illustrate how and why this mechanism works.

Consider again the static mapping function H_S shown in the top part of Figure 2.4.1-1 (page 40). There are two ways in which a neural network (NN) can be taught to achieve this classification mapping. One way is to teach a single neural net to learn all four classification boundaries simultaneously so that, for any given input, the net will produce the desired output class. We refer to this as a *global approximation* [26] approach because all classification boundaries are learned as one, large problem. This is the approach illustrated in the top part of Figure 2.4.1-1 (solid line) where the required mapping function H_S has four output levels, one for each input class.

Another approach is to teach four separate neural nets (one for each class) in such a way that each NN specializes only in determining if a given input is inside the class it

represents or outside that class. For example, Figure 2.4.2-1 illustrates independent static mapping functions for the NN of class 'A' (H_{S_A}) and class 'B' (H_{S_B}) (for the example of section 2.4.1). With function H_{S_A}, only inputs X in the range [0,0.08] and [0.42, 0.51] will produce an output value $y = 1$. Here, it is assumed that an output value of $y = 1$ means that the input is IN the class whereas $y = 0$ implies that the input is OUT of that class.

By combining four such independent neural nets and noting which one produces a '1' for a given input, the same classification boundaries as shown in the top of Figure 2.4.1-1 (page 40) can be achieved. We refer to this as a *series expansion* [41,45] approach because the classification problem is decomposed into a sum (i.e., a series) of simpler, independent classification problems.

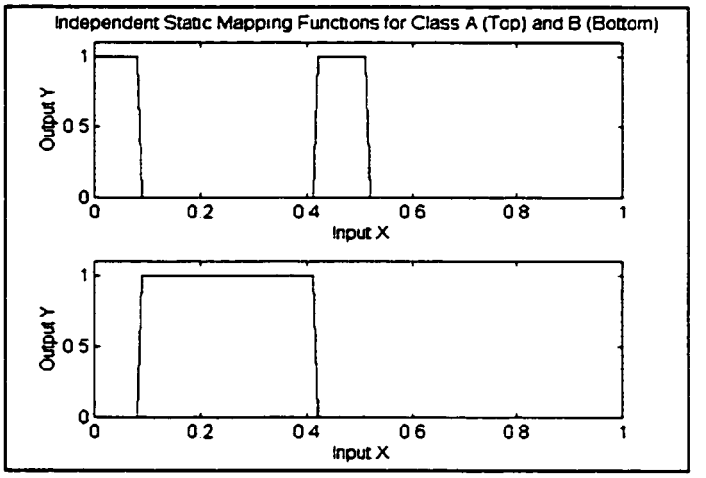


Figure 2.4.2-1 : Independent Static Mappings

Now assume that it becomes necessary to change the classification boundaries of H_S in Figure 2.4.1-1 by adding a new class (e.g., $y = 1.5$ for $X \in [0.2, 0.3]$). For the NN trained with the global approximation approach, the learned information defining the old classification boundaries is spread uniformly throughout all the neurons of the net. When this net is taught the new class boundaries, several neurons will change their parameter values in an attempt to reduce the classification error near the new boundaries. But since these neuron parameters are involved in defining all classification boundaries, these changes will likely increase the classification error at other boundaries. Since these classification errors did not exist before re-training was attempted, they constitute a loss of memory by the net.

For the case of the four NN trained by the 'series expansion' approach, the problem of memory loss due to re-training is less severe as a result of the independence of the four NN. In fact, only the NN which was taught classification boundary 'B' will be affected by the new class $y = 1.5$ for $X \in [0.2, 0.3]$ since that net must now be taught to forget that values of X in that range used to belong to class 'B'. Retraining and memory loss will still occur in that NN but at least it is confined to that input range. The other NN do not need to be retrained since they already classify inputs in the range $X \in [0.2, 0.3]$ as OUT of their respective classes.

The new class 'E' would then be implemented by adding a new independent NN and training it with a static mapping H_{S_E} where $y = 1.0$ for X in the range [0.2, 0.3] and $y = 0.0$ elsewhere in the input range [0, 1.0].

Hence, by ensuring that classification mappings are learned independently of each other, the problem of memory loss has been reduced and incremental learning is now easier to achieve. This process of specialization and independence of the learning tasks is often referred to as *orthogonalization* in mathematics [41,45]. This will be discussed in more detail in chapter 4.

2.4.3 Limiting the Complexity of the Learning Task During Incremental Learning

The ability to learn a new object class with a level of effort related mostly to the complexity of that class (i.e., independently of how many other classes are already in memory) can also achieved by ensuring that the classification mapping functions are learned independently of each other.

Consider again the example discussed in the previous section where a new class is added with $y = 1.5$ for $X \in [0.2, 0.3]$ and where a NN is taught by the global approximation approach. As a result, the static mapping H_S of Figure 2.4.1-1 (page 40) must be altered by adding a new plateau in that range of X. This necessarily increases the complexity of the mapping function, because a new peak has been added to it. In the neural network

domain, it has been shown that increasing the number of classes to be learned in such a way leads to an exponential increase in complexity [26]. For example, if $e^2 = 7.39$ is a measure of the level difficulty of learning to classify an input as one of 2 classes, the level of complexity of learning twice as many classes would be $e^4 = 54.6$ as a result of this exponential increase in complexity (i.e., 7.4 times more complex for a doubling of the number of output classes). This exponential increase in complexity is closely related to the *'curse of dimensionality'* [26] because it is more commonly encountered in problems where the input dimension is increased. This curse is discussed in section 4.7.1 (page 145).

However, with the 'series expansion' approach introduced in section 2.4.2, each NN is taught only two possible output values (e.g., as shown in Figure 2.4.2-1) : '0' (X is NOT a member of this class) or '1' (X is a member of this class). As a result, each mapping function has a limited complexity level, regardless of how many classes are learned. With this approach, if $e^2 = 7.39$ is the level of difficulty required to learn to classify any input as IN or OUT of a given class, the total complexity of learning four such classes (of similar complexity) is $4 \times e^2 = 29.6$. Even though each individual function is still suffering from the exponential complexity problem, increasing the number of classes now results in a linear increase in the overall complexity.

This reduction in complexity can be seen by comparing functions H_{S_A} and H_{S_B} of Figure 2.4.2-1 to H_S in the top part of Figure 2.4.1-1 (page 40). The latter is clearly more complex since it has more output levels, more peaks, and more transition points than the former. This reduction in complexity is also achieved in the equivalent iterated mapping functions H_{I_A} and H_{I_B}, relative to the global approximation function H_I shown in the bottom of Figure 2.4.1-1.

In summary, by using this mechanism, the complexity of learning a new class is mostly a function of the complexity of the new boundary itself. By contrast, for the NN taught with the 'global approximation' approach, each newly added class increases the learning task complexity exponentially.

2.4.4 Repairing Neural Damage

In a NN, the death of neurons (or changes in inter-neural connections) is equivalent to a change in the mapping function which the NN implements. Such changes will usually alter the mapping function implemented by the NN. This in turn will change the classification boundaries achieved by the NN, resulting in erroneous classifications (i.e., memory loss). Often, retraining the NN or adding new neurons to replace the dead ones is not a realistic option.

However, in some cases, the classification errors can be repaired by simply redefining the range of output values which defines membership in the class. Hence, the mechanism of redefining the output class from a point to a range of values (described in section 2.4.1.2, on page 42) can also be used to repair neural damage. An example will clarify how this works.

Consider again the classification example of Figure 2.4.1-1 (page 40). Assume that a NN has been taught the iterated mapping function H_I shown in the bottom plot so that, after 20 iterations, the classification boundary $y(20) = H_S$ shown in the top plot (as a solid line) is achieved. Now assume that some neural damage results in H_I being changed to H'_I so that, after 20 iterations, the mapping function achieved is the dotted line shown in the top plot (i.e., H_{S_NEW}) instead of the desired H_S. For the original H_S, an output value of $y = 0.990$ was sufficient to ensure that input values of X in [0.55, 0.58] or [0.82, 1.0] were correctly classified as members of class 'D'. But with H_{S_NEW} achieved as a result of the neural damage, the classification regions will be incorrect if $y = 0.990$ is kept as a criterion of classification.

However, if the classification boundary is redefined as a region such that any input X which produces $y \in [0.8281, 1.0]$ is a member of class 'D', the desired classification boundary has been restored. A similar approach can be used for the other classes to repair the damage.

The ability to repair the damage in such a simple way is due in part to the fact that the specific sequence of values achieved between $y(0) = X$ and $y(20)$ is of no interest to us. It is only the final value achieved $y(20)$ which matters in the classification decision.

It should be noted that a similar mechanism can be used when the classification boundaries are taught to independent NN by the 'series expansion' approach described in sections 2.4.2 and 2.4.3. However, an added advantage of that approach is that the death of neurons in one of the NN results in classification errors which are confined to the class which that NN specializes in recognizing. This is unlike the 'global approximation' learning approach, where the death of one neuron may affect several or all classification boundaries. Hence, it can be argued that by teaching classification boundaries independently of each other, the damage caused by the death of neurons is less severe (i.e., more confined) than would be the case if this mechanism was not used.

2.4.5 Summary of the Four Mechanisms

Let us summarize the four mechanisms discussed in section 2.4 and indicate how they allow the key characteristics of learning from section 2.3 (starting on page 37) to be achieved:

1. **Use an iterated mapping functions instead of a static one**. This mechanism decreases the complexity of the learning task and thereby allows a net of simple neurons to learn complex classification mappings (i.e., key characteristic #1).

2. **Redefine output classes from points to regions**. This mechanism reduces the complexity of the learning task for the NN by transferring complexity from the mapping function (which the NN must learn) to the classification boundary. Hence, it allows a net of simple neurons to learn complex classification boundaries (i.e., key characteristic #1).

This mechanism also allows neural damage to be repaired (key characteristic #4). That is because the changes in the classification mapping which result from the death of neurons can sometimes be cancelled by changing the output class to a region with a suitable shape.

3. **Re-use learned mapping functions to implement new classes**. This mechanism simplifies the complexity of the learning task since a new classification mapping function can be generated from a previously learned one (i.e., there is no need to train a new neural net to learn the new mapping). This simplification mechanism allows a net of simple neurons to learn complex classification boundaries (i.e., key characteristic #1).

4. **Teach the mapping functions independently of each other**. This mechanism limits the complexity of each mapping function regardless of how many classes are present. That is because each mapping function only has two possible output levels (e.g., 0 = "input is OUT of class-j" or 1 = "input is IN class-j"). As a result, the complexity of learning each class is mostly a function of the complexity of that class boundary (i.e., key characteristic #3).

   Furthermore, because mapping functions are independent of each other, new classes can be added with little or no memory loss (i.e., key characteristic #2).

   Finally, the independence of mapping functions means that the death of neurons in one NN will affect only the mapping function implemented by that NN. Hence, the damage caused by the death of neurons is limited (this is somewhat related to key characteristic #4).

2.5 Current Classification Paradigms based on Neural Networks

This section reviews current neural network models which seek to duplicate the classification process in the brain. For each model, emphasis is placed on describing how

the classification problem is resolved and on determining to what extent these models achieve the key characteristics of learning described in section 2.3. It is also of interest to compare these paradigms to the Freeman neurobiological model of section 2.2, in order to determine to what extent they emulate the way in which the brain classifies patterns in that model.

The various techniques to model the classification process can be partitioned into two main categories [18,26]:

a. paradigms based on the one-step associative memory; and

b. paradigms based on neurodynamical systems with equilibrium states.

These two categories are discussed in sections 2.5.1 and 2.5.2 respectively.

2.5.1 Paradigms Based on the One-Step Associative Memory

For the models based on this paradigm, a static mapping function is taught to a neural net in order to associate a suitable output Y for every input X as follows : $Y = H_S(X)$. Although the propagation of the input vector throughout the neural net requires a finite amount of time (usually small), this is considered a static net since the recognition process is completed in one discrete step. Note that time is not a significant factor in this classification paradigm; all inputs are classified in the same amount of time.

The output Y may indicate directly the class to which X belongs or it may provide a measure of the probability that X belongs to one or more classes. For example, Y could be one-dimensional but have several possible values, each one representing an output class (e.g., as shown in the top part of Figure 2.4.1-1 (page 40) as a solid line). Or, Y could be multi-dimensional, with each value representing either class membership or a probability of class membership. For example, $Y = [0,0,0,1]$ indicates that X is NOT a member of the first three classes but is a member of the fourth class. As another example, $Y = [0.05,$

0.15, 0.2, 0.6] indicates that X has a 5% probability of being in the first class, 15% probability of being in the second class, etc.

In the next two subsections, the one-step associative memory paradigm is assessed in terms of its ability to capture the key learning characteristics listed in section 2.3 and in terms of its mode of operation relative to the Freeman model of section 2.2. In section 2.5.1.3, the various neural net architectures which can be used to implement one-step associative memories are discussed briefly.

2.5.1.1 The Key Characteristics of Learning in the One-Step Associative Memory

An important limitation of nets using this paradigm is that the classification is achieved by using a static mapping function. As was seen in section 2.4.1.1, such functions are inherently more complex than iterated mapping functions. As a result, NN based on this paradigm cannot learn classification problems with complex boundaries (i.e., key characteristic #1 of section 2.3 is not achieved to a significant extent).

Another common limitation of this paradigm is that the NN is often taught the mapping function H_S as a *global approximation*. As was discussed in section 2.4.2, this results in exponentially increasing complexity as the number of classes to learn increases. This also results in a poor ability to learn new classes and in significant memory loss upon retraining. Hence, the key learning characteristics #2 and #3 are not achieved to a significant extent. Furthermore, the death of one neuron will affect all output classes as a result of the global approximation approach. To repair such damage, retraining is required, but this will result in memory loss. Therefore, the key learning characteristic #4 of section 2.3 is not achieved to a significant extent either.

However, it is possible to force neural nets trained with this paradigm to learn mapping functions with the series expansion approach described in section 2.4.2. Although this will alleviate some of the limitations described above, the limitation related to the inherent complexity of the static mapping function will remain.

2.5.1.2 The One-Step Associative Memory Paradigm versus the Freeman Model

The classification process in the one-step associative memory is unlike the process used in the Freeman model in almost every respect.

A primary difference is that the associative memory uses a one-step deterministic mapping function with no feedback present. By contrast, the Freeman model is a chaotic NDS where feedback is pervasive and where a certain amount of time is required before the answer is available (i.e., more than one time step).

The associative memory utilizes a static mapping function versus an iterated mapping function for the Freeman model. In the associative memory, output classes are represented by fixed output values. In the Freeman model, they are represented by chaotic attractors.

Another difference is that the classification process in the Freeman model may involve competition between different Nerve Cell Assemblies (NCA), where each one attempts to lure the NDS into a specific attractor state. This may occur when an ambiguous input (e.g., between two classes) is presented to the net. This was illustrated with the example of Figure 2.2.3-2 (page 36). By contrast, there is no explicit competition in the associative memory.

Hence, the one-step associative memory has little resemblance to the classification process believed to be used in a 'real' brain. However, note that this lack of resemblance is NOT necessarily a disadvantage in itself. What really matters most is the ability of the model to capture the key characteristics of learning identified in section 2.3.

2.5.1.3 Neural Net Architectures for the One-Step Associative Memory Paradigm

The static mapping function H_S in the one-step associative memory paradigm is commonly implemented using Multi Layer Perceptrons (MLP) feedforward NN or Radial

Basis Function (RBF) NN. Other popular architectures include the Kohonen network and Time-Delay Neural Nets (TDNN). These NN architectures are discussed briefly in the next subsections.

2.5.1.3.1 MLP and RBF Feedforward Associative Networks

The general architecture for MLP and RBF NN is illustrated in Figure 2.5.1.3.1-1 [4,26]. An input vector is fed to all the neurons of a hidden layer, whose outputs may be fed to another hidden layer or directly to an output layer, as shown in the figure. The hidden layer(s) of neurons in conjunction with the output layer implement the mapping function which projects an input X onto the K output classes.

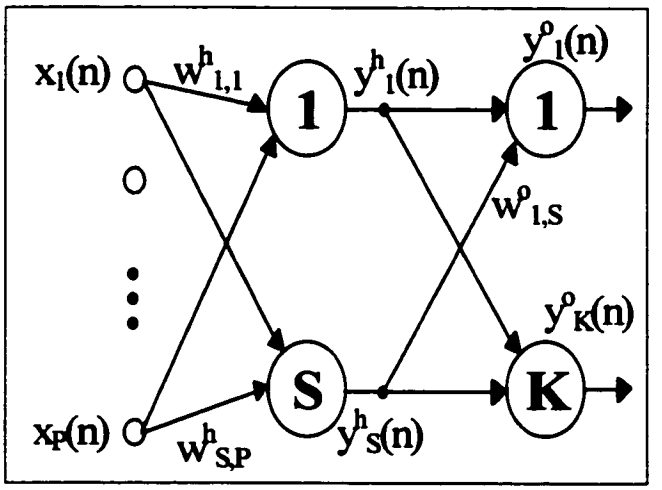


Figure 2.5.1.3.1-1 : MLP and RBF NN Models

The classification mapping function is taught to the network by presenting it with input examples and the associated output class. The backpropagation algorithm is generally used to correct errors between the actual net output and the desired net output for each training pair [4,26]. The value of weights (i.e., synapses) which interconnect the neurons are modified in such a way that the network output is brought closer to the desired output. When the network is able to generate the desired output for all training pairs within a user-defined maximum error rate, training is completed. If the training set was comprehensive enough, the NN will have learned the required mapping function for all classes and will be able to classify never-seen-before input vectors correctly. However, there is a danger of over-training the network, which may result in the neural net memorizing the data instead of learning the input-output relationship (i.e., generalizing) [26]. In such a case, never-seen-before inputs will not produce the correct output by the network.

Section 5.2.1 and [26] discuss differences between MLP and RBF nets in more detail.

2.5.1.3.2 Self-Organizing Feature Maps (SOFM) : The Kohonen Network

Figure 2.5.1.3.2-1 illustrates the basic configuration of the Kohonen net [26]. Here, an input vector X of dimension P is fed to all neurons of the network. The latter are typically arranged in an organized, two dimensional map. The output of each neuron is given by:

$$y_j(n) = \left(\sum_{i=1}^{P} w_{ji} x_i(n) \right) - \left(\sum_{\substack{k=1 \\ k \neq j}}^{R} v_{jk} y_k(n) \right) \qquad \text{(E_2.5.1.3.2-1)}$$

where R is the number of neurons which have connections to neuron j and where $y_j(n) \in [0,1]$. The output of neuron j is a function of how close vector W_j is to X and also on the current output of other neurons.

Generally, the weights interconnecting neurons (i.e., v_{jk}) are made proportional to their relative distance so that a neuron will affect only (and will only be affected by) nearby neurons. Furthermore, the feedback is made such that an active neuron will enhance the response of adjacent neurons while suppressing the response of neurons farther out. This feedback is described by a *neighborhood function* for each neuron [26].

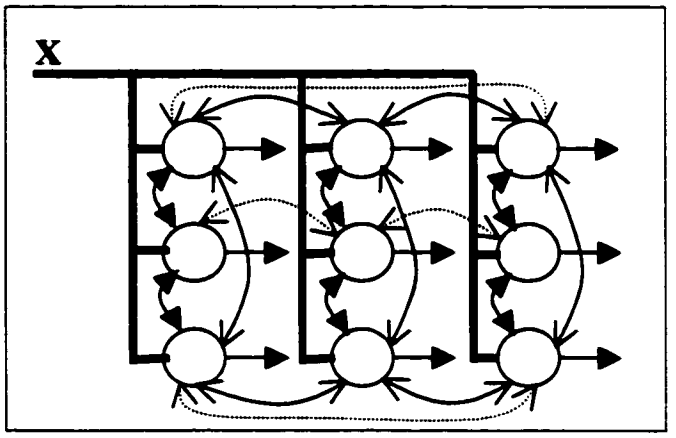


Figure 2.5.1.3.2-1 : The Kohonen Network

During the training phase, the neuron with the highest output will see its weight vector W_j brought closer to X. Its neighbors will also be updated proportionally to their vicinity to the winning neuron. At the start of the training, the neighborhood function is large but becomes gradually smaller as training progresses. This results in input *features* being mapped as an output activation potential in a specific location of the 2D output map.

The mapping function learned by the Kohonen net is considered static because the input is propagated once through the net (i.e., no feedback from output back to the input) and because an output is generated in one step (neglecting the finite time required for the settling of the net, as output neurons compete and a winner emerges). It has been shown that the Kohonen net is essentially a vector coding algorithm [26] which maps a large number of input vectors onto a smaller subset of representative vectors (i.e., lossy data compression [28]).

The outputs of the Kohonen net can therefore be viewed as a subset of coded prototypes onto which input vectors are projected. In that sense, it operates likes a pattern classifier. However, it is important to note that the user has no way to control which input vectors get lumped into what output class. That decision is made by the net and is based exclusively on the Euclidean distance between input vectors and the prototypes encoded by the weight vectors W_j of the neurons. Furthermore, the ability of the net to classify inputs is limited by the fact that only the Euclidean distance is involved in the decision making process. That distance measure can produce misleading results. For example, if a character '1' rotated by 90° is used as an input, the Euclidean distance will be large for the prototype '1' but may be lower for several other prototypes (i.e., '3', '4', '6', etc), resulting in misclassification. These limitations make it very difficult for the Kohonen net to learn complex classification boundaries.

2.5.1.3.3 Time Delay Neural Networks (TDNN)

Time delay neural networks (TDNN) [26] have the same basic architecture than the MLP shown in Figure 2.5.1.3.1-1, except that the weights are replaced by Finite Impulse Response (FIR) filters [38] (RBF neurons can also be used). This provides the feedforward net with memory, giving it dynamic properties that make it responsive to time-varying signals.

However, because there is no feedback, the net cannot be viewed as a NDS (as described in sections 2.2 or 2.4.1.1). In fact, the TDNN can be represented as a purely static net by

the process of "unfolding in time" [26] which translates the spatio-temporal net into a purely spatial one.

Hence, this net can also be considered equivalent to a static, one-step mapping function, as was the case for the MLP and RBF NN of section 2.5.1.3.1. Since the training technique for the TDNN is essentially the same as for the MLP NN, all the comments made in that section are applicable to the TDNN as well.

2.5.2 Paradigms Based on Neurodynamical Systems with Equilibrium States

For neurodynamical systems, classification is achieved through a gradual transformation process, where a nonlinear dynamical system (NDS) modeled by the neural net evolves from an initial state to a final, equilibrium state of lower energy potential. Hence, the system is dissipative [15,56]. The initial state corresponds to the presentation of the unknown vector $X(0)$ whereas the final state Γ_j represents the class which $X(0)$ is assessed to belong to. In this type of net, several iterations may be required before $X(0)$ is transformed into Γ_j. Because of this and because the NN typically has a significant amount of feedback, the recognition process is said to be dynamic. The time required to reach the final state (i.e., to classify the input) has not been considered to be significant in classic neurodynamical systems training algorithms; it is only a delay which must be endured before the final answer becomes available.

For the models based on this paradigm, an iterated mapping function H_I is taught to a neural net in order to associate a suitable output Y for every input X as follows : $Y(n) = H_I(Y(n-1))$ with $Y(0) = X$, the input pattern to classify.

In subsection 2.5.2.1, the neurodynamical classification paradigm is assessed in terms of its ability to capture the key learning characteristics described in section 2.3. In subsection 2.5.2.2, its mode of operation is compared to that of the Freeman model of section 2.2. In section 2.5.2.3, the various neural net architectures which can be used to implement the neurodynamical classification paradigm are discussed briefly.

2.5.2.1 The Key Characteristics of Learning in a Neurodynamical Classifier

Classifiers based on this paradigm make use of mechanism #1 of section 2.4 (i.e., an iterated mapping function) and therefore benefit from the reduced function complexity relative to an equivalent static function. As a result, the neurodynamical classifiers are often able to learn more complex classification problems than one-step associative memories (i.e., key learning characteristic #1).

However, a common limitation of this paradigm is that the NN is often taught the mapping function H_I as a *global approximation*. As was shown in section 2.4.2, this results in exponentially increasing complexity as the number of classes to learn increases. This also results in a poor ability to learn new classes and in significant memory loss upon retraining. Hence, the key learning characteristics #2 and #3 are not achieved to a significant extent. Furthermore, the death of one neuron will affect all output classes as a result of the global approximation approach. To repair such damage, retraining is required, but this will result in memory loss. Therefore, the key learning characteristic #4 of section 2.3 is not achieved to a significant extent either.

But here again, it is possible to teach the NN to learn mapping functions with the series expansion approach described in section 2.4.2. In theory, this will allow the neurodynamical system to achieve all the learning characteristics of section 2.3. to a significant extent. However, the neural net architectures commonly used for this paradigm (i.e., Hopfield net and Boltzmann machine) suffer from limitations which make this approach unattractive. These limitations are discussed in section 2.5.2.3.

2.5.2.2 The Neurodynamical Classification Paradigm versus the Freeman Model

The neurodynamical classification model shares many similarities with the Freeman model of section 2.2. Both of them utilize a NDS with iterated mapping functions. In both cases, classification involves the NDS converging to an attractor which represents a

specific class. In both cases, the unknown input to classify serves as the trigger which leads the NDS to converge to the appropriate attractor.

However, there are significant differences as well. One of the most important ones is that the NDS used in the neurodynamical paradigm typically involve point attractors whereas the Freeman model makes use of chaotic attractors. Furthermore, for the Freeman model, classification involves the NDS being excited into a higher energy state from a lower one[i] whereas the exact opposite is done for traditional neurodynamical models. Another difference is that there can be competition in the Freeman model when ambiguous inputs or multiple inputs are present simultaneously (e.g., the example of Figure 2.2.3-2 on page 36). By contrast, there is no such competitive mechanisms in traditional neurodynamical models.

2.5.2.3 Neural Net Architectures for the Neurodynamical Classification Paradigm

Typical models based on this paradigm are Hopfield nets, Boltzmann nets and MLP/RBF recurrent nets [4,26]. These NN architectures are examined briefly in the following subsections.

2.5.2.3.1 The Hopfield Net

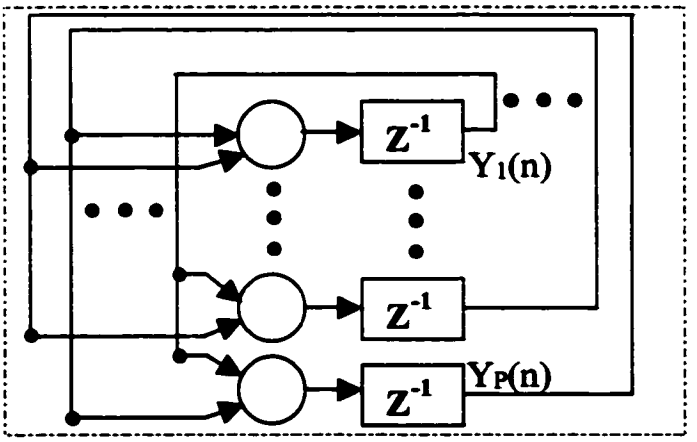


Figure 2.5.2.3.1-1 : The Hopfield Net

Figure 2.5.2.3.1-1 illustrates the basic configuration of the Hopfield net [26]. The output Y_j of each neuron is given by equation E_2.5.2.3.1-1 (shown below) and has only two possible values : $Y_j \in [-1,1]$. The Hopfield net has P neurons, with feedback between all

[i] Recall from section 2.2.1 that when a rabbit sniffs an odorant, the olfactory bulb experiences a short burst of high activity and then falls back into a state of rest when the rabbit exhales.

neurons allowed except for self-feedback (i.e., $w_{ii} = 0 \ \forall i$).

$$Y_j(n) = \text{sgn}\left(\sum_{\substack{i=1 \\ i \neq j}}^{P} w_{ji} Y_i(n-1) \right) \qquad \textbf{(E_2.5.2.3.1-1)}$$

In order to learn K prototypes (each one of dimension P), the outer product rule is used to define the weight matrix of $P \times P$ elements as follows:

$$w_{ji} = \begin{cases} \frac{1}{P} \sum_{r=1}^{K} \xi_{r,j} \xi_{r,i}, & j \neq i \\ 0, & j = i \end{cases} \qquad \textbf{(E_2.5.2.3.1-2)}$$

where ξ_j is a vector of size P, which represents a fundamental memory of the system. The above equation ensures that the fundamental memories are mapped onto fixed points Γ_j of the nonlinear dynamical system.

After learning is completed, the network is used by setting $Y(0) = X(0)$ (the unknown vector to classify) at time $n = 0$. Then, the network is iterated and the neuron's outputs are updated in accordance with equation E_2.5.2.3.1-1 until a stable equilibrium state Γ_j is reached. That equilibrium state represents the class to which $X(0)$ is assumed to belong to. Note that in the Hopfield model, one neuron (selected randomly) is updated at each iteration. The Hopfield network has a Lyapunov function [15,56], so that its convergence to a stable attractor point is guaranteed (Lyapunov functions are discussed in chapter 4).

Normally, ξ_j is learned as a stable equilibrium point Γ_j but this is not always the case. One problem of the Hopfield net is that the fixed points Γ_j defined by equation E_2.5.2.3.1-2 are not necessarily stable equilibrium points of the system. As a result of this, $X(0)$ may not converge to the correct prototype class. Furthermore, *spurious states* exist in the system [26]. These are stable states of the network but which are not fundamental memories. Hence, these represent undefined class prototypes to which an input vector $X(0)$ may converge to. The Hopfield net also has a very limited storage

capacity of approximately $0.15P$ fundamental memories [26] (e.g., 20 neurons are required to store 3 prototypes).

These limitations of the Hopfield net are always present, regardless of whether classes are taught with a global approximation approach or a series expansion approach (i.e., as discussed in sections 2.4.2 (page 43) and section 2.5.2.1 (page 57)). Hence, it is unattractive to use this type of net for practical classification problems.

2.5.2.3.2 Boltzmann Neural Net

The Boltzmann NN [26] is similar to the Hopfield net in many respects. The neurons have an output value defined by $y_j \in [-1,1]$ and all the neurons are interconnected except that there is no self-feedback. Also, individual neurons are picked randomly for updating.

However, the net structure is different in that hidden neurons exist, along with visible neurons (the latter constitute the inputs and outputs of the network). The transition of a neuron from its current state Y_j to $-Y_j$ is probabilistic, in accordance with:

$$P(Y_j \rightarrow -Y_j) = \frac{1}{1+\exp(-\Delta E_j / T)} \qquad \text{(E_2.5.2.3.2-1)}$$

By contrast, the update rule was deterministic for the Hopfield net. In the above equation, ΔE_j is the system's energy change that would occur as a result of neuron j flipping state from Y_j to $-Y_j$. Parameter T represents the temperature of the system (more on this later). The energy of the system is defined by:

$$E = -\frac{1}{2}\sum_{\substack{i=1 \\ i \neq j}}^{N}\sum_{j=1}^{N} w_{ji} Y_j Y_i \qquad \text{(E_2.5.2.3.2-2)}$$

where N is the number of neurons in the system.

Unlike the Hopfield net, the Boltzmann NN can be taught to associate inputs with specific output class prototypes. This is done by first *clamping* input neurons to $X(0)$ (the input vector) and the output neurons to Γ_j (the desired class prototype). Then, starting with a high temperature T, the network is made to relax by updating the states of the hidden neurons, using equation E_2.5.2.3.2-1. After a certain number of updates, the temperature is decreased and the process is repeated. This is repeated for several temperatures until a user-defined minimum value is reached. Decreasing the temperature decreases the probability that transitions to a higher energy state will occur. Hence, the hidden neurons gradually converge to a lower energy state as time progresses.

In a second phase of learning called the *free-running* phase, the input neurons are still forced to $X(0)$ but the output neurons are free to assume any value. The relaxation process described above is repeated, starting at a high temperature which is gradually decreased. The statistical difference between the neuron states in the *clamped* phase and *free-running* phase is used to adjust the weights of the neurons in order to decrease this difference. The clamped and free-running training phases are repeated until the network achieves all the desired states Γ_j in the free-running phase when the appropriate input vector is presented.

When the training is completed, an unknown vector $X(0)$ is presented at the input and the free-running relaxation process described above is performed. The final state Γ_j of the network (after the cooling-down process) indicates the class prototype to which $X(0)$ belongs.

In this learning paradigm, the process of training through clamping and free-running phases must be repeated in sequence for each $X_j(0)$ which belongs to the class represented by Γ_j. For the Boltzmann NN, the learning process is extremely slow, even for only one prototype because the temperature must be reduced very slowly. As a result, the NN becomes useless in practice as K (the number of prototypes to learn) increases or as the number of training samples increases.

2.5.2.3.3 Recurrent MLP or RBF Neural Networks

For recurrent MLP/RBF NN, the general architecture is similar to that shown in Figure 2.5.1.3.1-1 (page 53), except for the addition of feedback connections between some or all the neurons in the network [26]. As a result of this feedback, the network is a nonlinear dynamic system whose state evolves with time. It should be noted that recurrent nets can also involve TDNN structures, using either MLP or RBF neurons.

Training for recurrent MLP nets can also be done using the backpropagation algorithm, as was the case for the feedforward nets discussed in section 2.5.1.3.1.

However, instead of teaching a static mapping, recurrent nets must be taught a specific trajectory from the initial input point X all the way to the attractor Γ_j which represents the class to which it belongs. This must be done for every training pair (X_k, Γ_j) so that the NN will learn a NDS mapping function of the type $Y(n) = H(Y(n-1))$ (with $Y(0) = X_k$) over the entire input space where correct classifications are required. Note that by contrast, only the final point Γ_j and the initial input X_k need to be shown to the Boltzmann or Hopfield nets. For these, the trajectory followed from $Y(0) = X_k$ to Γ_j is NOT controlled by the training data.

When the training of these nets involves the global approximation approach described earlier, these nets perform poorly as the number of classes to learn increases. This is again due to the exponential increase in complexity which severely limits the learning ability of these nets. As discussed in earlier sections, this training approach also results in memory loss when incremental training is attempted. Hence, recurrent nets trained with the global approximation method do not capture the key advantages of learning in the brain to a significant extent.

2.6 Summary and Conclusions

In section 2.2, a model explaining how learning and recall works in a 'real' brain was introduced. This Freeman neurobiological model [23] was shown to utilize a nonlinear dynamical system (NDS) with several chaotic attractors as the basis for classification. In that model, inputs excite the NDS towards a specific attractor state, where each attractor corresponds to a learned pattern class. Although this model has the advantage of being a realistic representation of the classification process in a 'real' brain, it is described only in heuristic terms. Hence, it cannot be implemented in software or with mathematical models since the learning and recall processes are not defined clearly enough.
In section 2.3, four key characteristics were identified which make the brain a good and reliable classifier. Section 2.4 described four mechanisms which allow us to implement these characteristics in a software (or mathematical) model of the brain as a classifier.

In section 2.5, traditional classification paradigms were reviewed. These were partitioned into two main categories; one-step associative memories and neurodynamical systems. This review highlighted the capabilities and limitations of these paradigms in terms of the key characteristics (of section 2.3) and also compared their modes of operation relative to the Freeman neurobiological model (section 2.2). It was shown that all the neural net architectures used to implement both classification paradigms fall short of being able to capture the key characteristics required to make them good and reliable classifier systems.

We are now ready to introduce a novel classification paradigm whose advantages and differences relative to the paradigms reviewed in this chapter will be made clear in the next chapter.

Chapter 3 :

Description and Basic Theory of the RTA NN Paradigm

3.1 Introduction

This chapter describes the basic theory behind the Race to the Attractor Neural Network (RTA NN) model paradigm.

Before the RTA NN model itself can be introduced, it is necessary to review some basic concepts of Nonlinear Dynamical Systems (NDS), since these are an integral part of the RTA NN. This is done in section 3.2, with the help of specific examples. Following this, the basic theory of the RTA NN paradigm is introduced in section 3.3. A fundamental axiom is derived, which defines key aspects of the RTA NN model paradigm. Then, the architecture and operation of the RTA NN model are described in section 3.4. This is followed by a discussion of the basic approach to the training of the RTA NN model in section 3.5.

Section 3.6 compares the RTA NN model to the other neural network paradigms described earlier in chapter 2 and highlights the fundamental differences between them. Finally, section 3.7 summarizes the information presented in this chapter.

3.2 Nonlinear Dynamical Systems (NDS) with Iterated Mapping Functions

This section provides a brief description of the architecture and operation of a NDS.

A NDS is a system whose state evolves with time through a nonlinear mapping function. This evolution depends on the state of the system at earlier times. For example, if the

state at time *n-1* is ***X(n-1)***, the state at time *n* can be represented by an *iterated mapping function* ***H***, which can be defined as follows:

$$X(n) = H(X(n-1)) \qquad \textbf{(E\_3.2-1)}$$

Note that ***X*** can be multi-dimensional. When ***X(n)*** is the output of the NDS, the above equation indicates that the evolution of the NDS involves re-mapping the previous output of the system through the same function over and over again as time elapses. The architecture for a system which implements this type of mapping is shown in Figure 3.2-1. The box with a Z^{-1} in that figure represents a delay of one time step. At time $n = 0$, the input is $X(0) = X_{INIT}$ and the output is $H(X(0))$. At time $n = 1$, the switch in Figure 3.2-1 is flipped from the down to the up position so that, from that time onward, the input at time *n* is ***X(n-1)***.

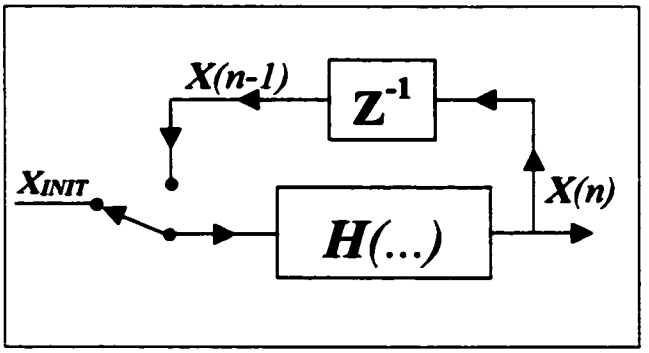


Figure 3.2-1 : Basic Discrete-Time NDS

Let us illustrate how such a system operates with an example. Consider the function given below:

$$X(n+1) = H(X(n)) = X(n) + \frac{-0.1\sin(X(n))}{1.04 - 0.4\cos(X(n))} \qquad \textbf{(E\_3.2-2)}$$

This iterated mapping function is clearly nonlinear. Figure 3.2-2 illustrates *H(n)* as a function of *X(n)* and also shows the straight line *X(n)* = *X(n+1)*. It can be noted that the function *H(X(n))* is difficult to distinguish from the dotted line. This is because the difference between *X(n)* and *X(n+1)* at any point *X(n)* is small relative to the magnitude of *X(n)* itself. Because of this, it is usually preferable to visualize the function *ΔH(X(n))*, which is defined as follows:

$$X(n+1) - X(n) = \Delta H(X(n)) = \frac{-0.1\sin(X(n))}{1.04 - 0.4\cos(X(n))} \qquad \text{(E\_3.2-3)}$$

This equation defines the magnitude of the change of the NDS output at time $n+1$ relative to its previous output. The function $\Delta H(X(n))$ is sometimes called the *convergence rate function* [56]. That function is shown in Figure 3.2-3.

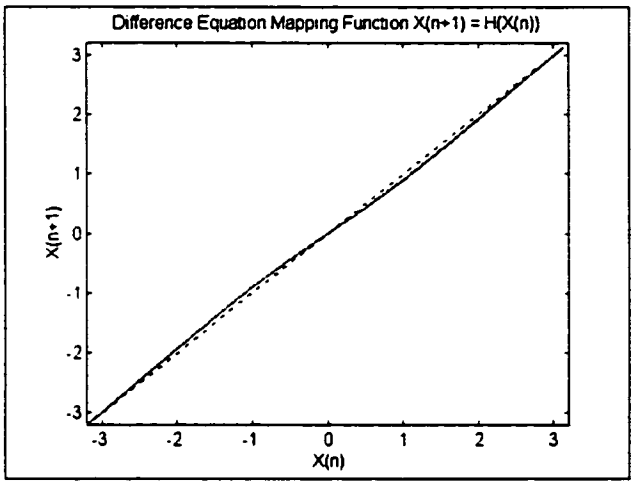


Figure 3.2-2 : H(X(n)) versus X(n)

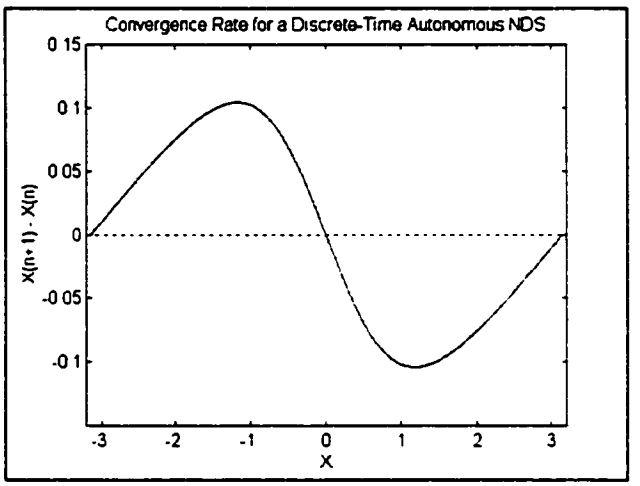


Figure 3.2-3 : ΔH(X(n)) versus X(n)

The sequence of points $\{X(0), X(1),....X(n),...\}$ is often referred to as a *trajectory* [56] of the NDS since this sequence shows the path followed by the NDS from its initial condition to its current state. Figure 3.2-4 illustrates some sample trajectories for the NDS defined by equation E_3.2-2. Note that at some point in time, all trajectories end up at the same point $X = 0.0$ for this NDS. This point is typically called the *attractor* [15,56] of the NDS, which we label as Γ. Although not all NDS have attractors, those of interest to us for the RTA NN model will always have one of them. Furthermore, all trajectories for these NDS will eventually lead to it, regardless of the initial condition (a mathematical analysis of NDS properties is presented in chapter 4).

As explained in section 2.2.2 of chapter 2, one can view a NDS with an iterated mapping function as a system which gradually transforms an initial input $X(0)$ into another value. For the NDS of equation E_3.2-2, all initial values $X(0)$ get transformed into $X(n) = 0.0$ eventually, for a large enough value of n. Figure 3.2-5 indicates how many iterations N_k

are required for a trajectory with initial input $X(0) = X_k$ to reach the attractor[ii], for different initial values $X(0)$. It will be shown in chapter 4 that the value N_k for any point X_k is determined by the magnitude of the convergence rate function $\Delta H(X(n))$ over the trajectory from $X(0) = X_k$ to the attractor Γ. For now, it is sufficient to appreciate that by reshaping $\Delta H(X(n))$, the value of N_k for any given point $X_k(0)$ can be modified. In section 3.3, it will be shown how this ability can allow us build a NDS which is capable of classifying objects reliably.

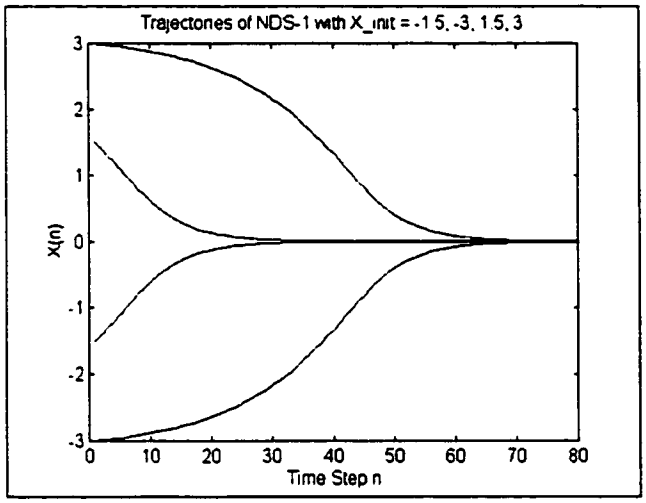


Figure 3.2-4 : Sample Trajectories of NDS-1

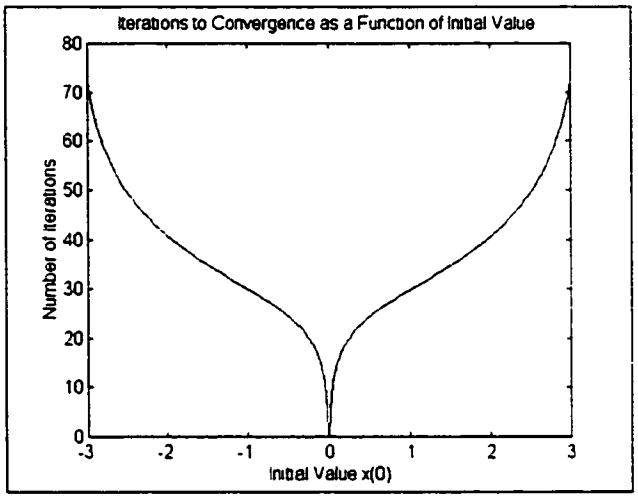


Figure 3.2-5 : Iterations to Convergence

3.2.1 Changing the Attractor of a NDS

It is possible to re-use a NDS which has an attractor Γ_1 to generate trajectories which lead to another attractor Γ_2 . In that way, the original NDS can be used to implement several other NDS, each one with its own attractor. This can be done with the help of offsets which translate the coordinates of one attractor to another one. This process is shown in Figure 3.2.1-1, where the value of the offset is defined as follows:

$$x_{offset\_in} = (\Gamma_1 - \Gamma_2) \qquad \text{(E\_3.2.1-1)}$$

[ii] Strictly speaking, the time required to reach the attractor for most NDS is infinite [15,18]. In this thesis, references to the NDS reaching the attractor in a finite time imply that the distance to the attractor is small.

In that figure, the inner box labelled $H_1(X(n))$ could be the iterated mapping function shown in Figure 3.2-2 (for example), with attractor $\Gamma_1 = 0.0$. If $\Gamma_1 = 0.0$ and $\Gamma_2 = 2$, $x_{offset\_in} = -2$. For example, if $X_2(n) = 2$, the 'real' input to $H_1(X(n))$ would be $X_1(n) = 0.0$. But since this is the attractor, $X_1(n+1) = 0.0$ also. The output of the dashed box would then be $X_2(n) = 2$, which is the attractor Γ_2. Hence, for an observer outside the dashed box in Figure 3.2.1-1, the NDS seems to be generating the trajectories shown in Figure 3.2.1-2. But in reality, the system inside the dashed box actually implements the trajectories shown in Figure 3.2-4 of the previous section. In that way, the original NDS has been re-used to generate a new NDS with a different attractor.

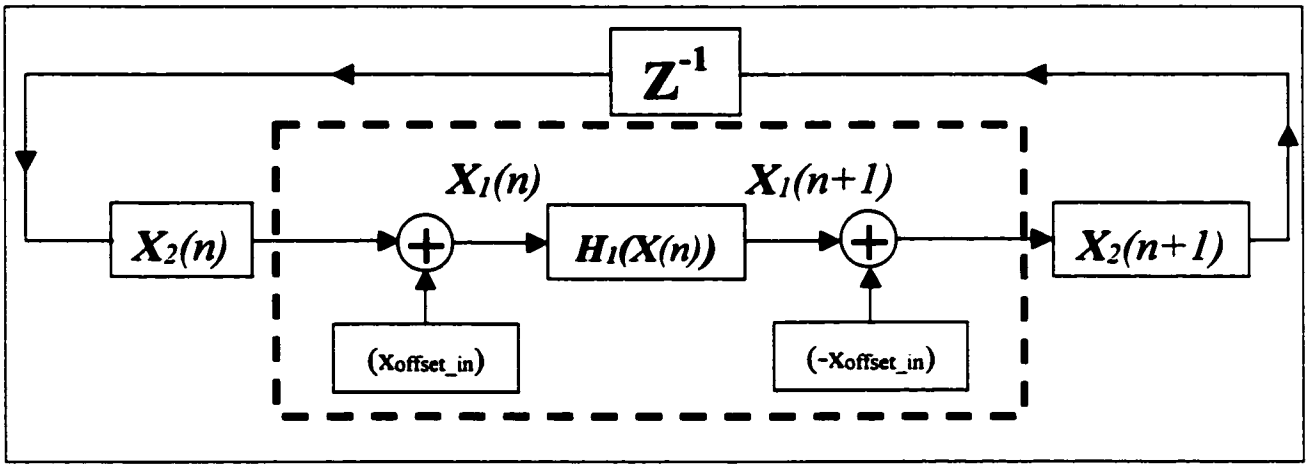


Figure 3.2.1-1 : Setup Required to Change the Attractor of a NDS from Γ_1 to Γ_2

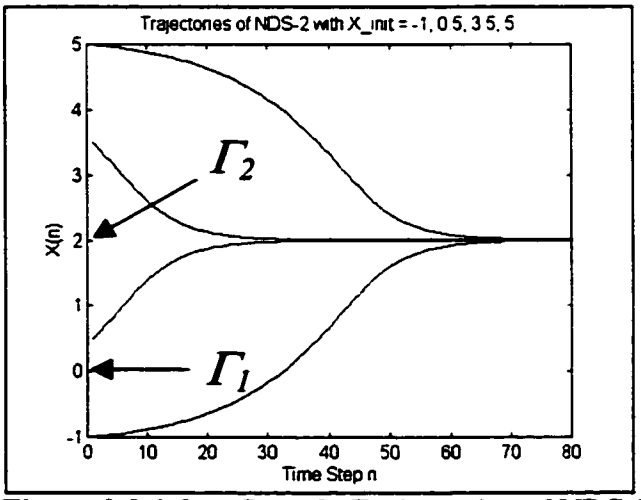


Figure 3.2.1-2 : Sample Trajectories of NDS-2

In simple terms, this is the mechanism of *re-use* which was described in section 2.4.1.3 of chapter 2 (page 42) and which allows the user to simplify the complexity of the overall learning task. The use of this mechanism will be illustrated in chapter 5 with a real classification problem.

3.3 The Basic Operational Concept of the RTA NN Model

In this section, the basic concept behind the RTA NN model paradigm is introduced with a simple example. Some of the key properties of the RTA NN paradigm are also described.

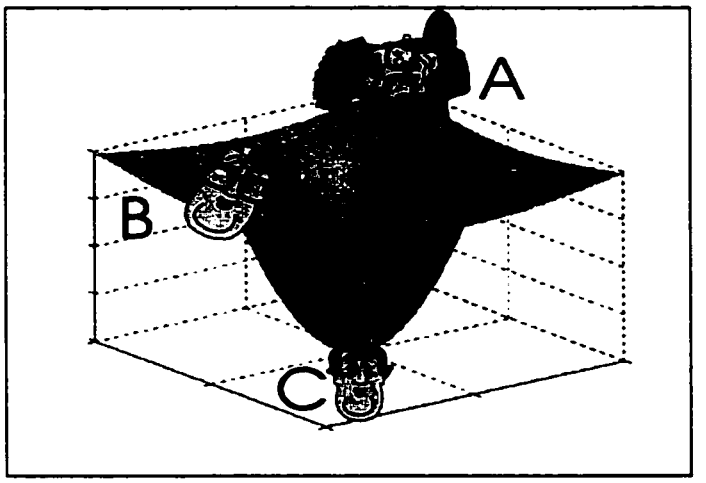


Figure 3.3-1 : RTA NN Object Recognition Process

Figure 3.3-1 illustrates the recognition process for someone we refer to as "Uncle Brian". The input shown at point A is a rotated view of Uncle Brian wearing a false beard, glasses, hat, etc. The model of the brain is a NDS which processes this input by iterating it several times through a function ***H*** which progressively transforms this input into a simpler image, by removing some 'irrelevant' information at each step (e.g., the trajectory from point A to point B). After a while, the output of the iterated function converges to a single point attractor which is a simple 'class prototype' that captures the 'essence' of Uncle Brian (i.e., point C). The less resemblance there is between the input and the prototype, the more time will be required by that input point to reach the attractor through function ***H***.

The input to the model can be visualized as a marble which is dropped a fixed distance *D* away from the center of the bowl shown in Figure 3.3-1. When Uncle Brian is well disguised, the slopes of the bowl are shallow and the marble rolls slowly to the bottom.

When Uncle Brian has a less elaborate disguise, the slopes are steeper and the marble rolls to the bottom of the bowl faster. When the marble reaches the attractor at the bottom (i.e., the prototype location), identification is complete. The time required for the marble to traverse the distance D to the attractor is determined by the slope of the bowl along the trajectory steered by the marble.

With this analogy in mind, object classification in the 'Race to the Attractor' (RTA) neural network (NN) model can be explained as follows:

To classify an object as one of R classes, there are R independent neural networks, each one implementing a different iterated mapping function H_j. The unknown input $X_k(0)$ is fed to these R NN simultaneously at time $n = 0$ and each one iterates the input towards its unique attractor prototype Γ_j. The first neural net J to reach its prototype in K iterations or less identifies the input as belonging to class J (i.e., C_J). If none of the NN reach their attractor in K iterations, the input is classified as unknown. This algorithm is illustrated in Figure 3.3-2 for $K = 20$. Referring again to Figure 3.3-1, one can think of the RTA NN model as a set of 'bowls' side by side, with different shapes, and where the bottom of each bowl represents a prototype for a person (i.e., a class). To identify an initially unknown input, copies of the same marble are dropped in each bowl simultaneously, at a position where the distance and steepness of the bowl result in a time to convergence which is proportional to how different the input is from the prototype (i.e., as the degree of difference increases, the time required to converge to the attractor increases). All marbles then start racing towards the center of their respective bowl. The first marble to reach the bottom within a given time limit

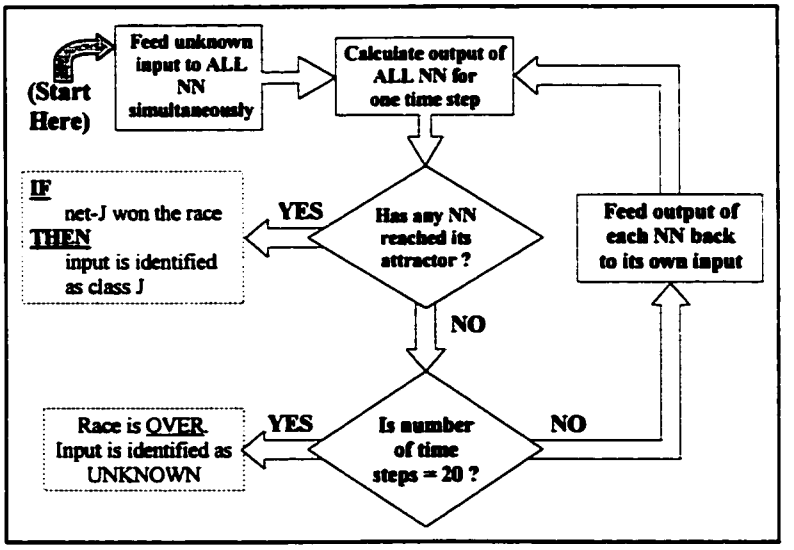


Figure 3.3-2 : RTA NN Classification Algorithm

identifies the unknown input as a member of the class which the prototype represents. Note that a time limit is used in order to allow the RTA NN model to implement the class "unknown" as a valid identification for inputs which are very different from all the classes which the RTA NN is trained to recognize.

With this approach, each NN '*j*' specializes in classifying any input as one of two possible outputs : "X is a member of class *j*" (i.e., NN-*j* won the race), or "X is NOT a member of class *j*" (i.e., NN-*j* did NOT win the race). Also, each iterated map H_j is independent of the other ones. Finally, note that each net implements the classification mapping gradually, with an iterated function H_j as opposed to a static one-step mapping. As explained in section 2.4 of chapter 2 (starting on page 39), these mechanisms result in a simpler mapping problem and allow incremental learning without significant loss of memory (this will be demonstrated in chapter 5).

For the RTA NN model, the fundamental axiom which defines its operation can be stated as follows:

For any input vector, the time required to converge to the attractor is inversely proportional to the probability that the input is a member of the class whose prototype is the attractor (E_3.3-1)

Hence, inputs which have a low probability of class membership (i.e., which have little resemblance to the class prototype) will converge very slowly to the attractor, and vice-versa for inputs which have a high probability of class membership (i.e., which resemble the class prototype in many ways). For the example of Figure 3.3-1, the input at point 'A' has a lower probability of class membership than the input at point 'B' (i.e., less resemblance to the prototype at point 'C') and will therefore require more time to reach the attractor than the latter.

3.3.1 Distance to the Attractor and Class Membership Probability

In the RTA NN paradigm, the distance to the attractor is <u>not</u> directly related to the probability of class membership. As a result of axiom E_3.3-1, this implies that the time required to converge to the attractor is NOT necessarily inversely proportional to the distance from the attractor. For example, two points at a different distance from the attractor may in fact have the same probability of class membership and would therefore converge to the attractor in the same amount of time as a result of this axiom.

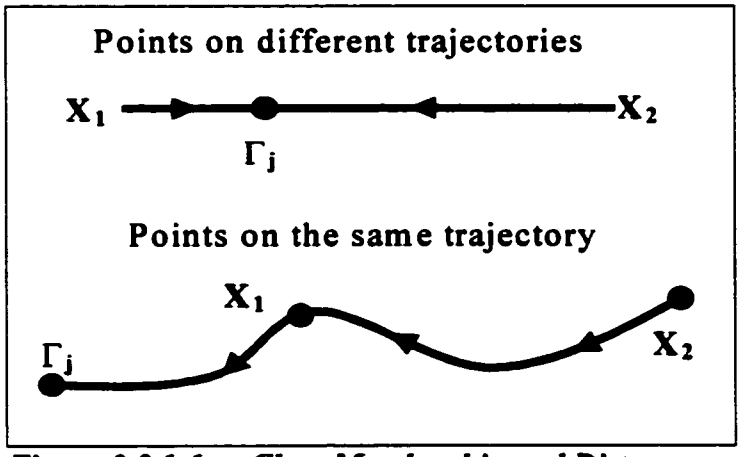


Figure 3.3.1-1 : Class Membership and Distance

An example of such a situation is illustrated in the top part of Figure 3.3.1-1. Here, assume that X_1 and X_2 have the same probability of being in class C_j. As a result of axiom E_3.3-1, they must require the <u>same</u> amount of time to converge to the attractor. Let us label these times as T_{j1} and T_{j2}. But because their respective distance to the attractor is different, the convergence rate (i.e., $\Delta H(X(n))$) on their respective trajectory must be different (i.e., X_2 must converge faster than X_1 to reach the attractor at the same time).

However, points on the <u>same trajectory</u> to the attractor must have a probability of class membership which is inversely proportional to the distance from the attractor. Hence, the probability of class membership along any trajectory must decrease monotonically as the distance to the attractor increases. Such a situation is illustrated in the bottom part of Figure 3.3.1-1. In that figure, the initial trajectory starting at point X_2 will eventually intersect point X_1 on its way to the attractor Γ_j. If T_{j1} is the time required by X_1 to reach the attractor, then $T_{j2} > T_{j1}$ is necessary, which means that the probability that X_1 is a

member of class C_j must be larger than the probability that X_2 is a member of class C_j (because of axiom E_3.3-1). This will be proven mathematically in chapter 4.

The discussion presented above can be summarized as follows:

> *In the RTA NN model, the probability of class membership for points on different trajectories is NOT necessarily inversely proportional to their relative Euclidean distance from the attractor (i.e., longer distance does NOT necessarily imply lower probability).*
>
> *But for points on the same trajectory, the probability of class membership must be inversely proportional to the Euclidean distance from the attractor (i.e., probability must decrease as distance increases).*

As a result of this and axiom E_3.3-1, the relationship between 'time to convergence' and distance from the attractor can be summarized as follows:

> *In the RTA NN model, the time required to converge to the attractor for points on different trajectories is NOT necessarily proportional to their relative distance from the attractor (i.e., longer distance does NOT necessarily imply longer time to convergence).*
>
> *But for points on the same trajectory, the time to convergence must be proportional to the distance from the attractor (i.e., time to convergence must increase as distance increases).*

These statements and axiom E_3.3-1 of the previous subsection affect the operation of the RTA NN model in many ways and impose constraints on the shape of the class membership probability density function. These issues will be discussed further in chapter 4.

Note that by allowing points with different distances from the attractor to have the same probability of class membership, the RTA NN model is able to implement complex, highly nonlinear classification boundaries. An example in two dimensions is shown in Figure 3.3.1-2 with three attractors and three classification boundaries B_{IN_j} . In that figure, all points on or inside a boundary B_{IN_j} are IN the class C_j (represented by the attractor Γ_j) wheras points outside the boundary are OUT of that class. Such complex boundaries could not be learned by the RTA NN model if the probability of class membership had to be directly related to the distance from the attractor.

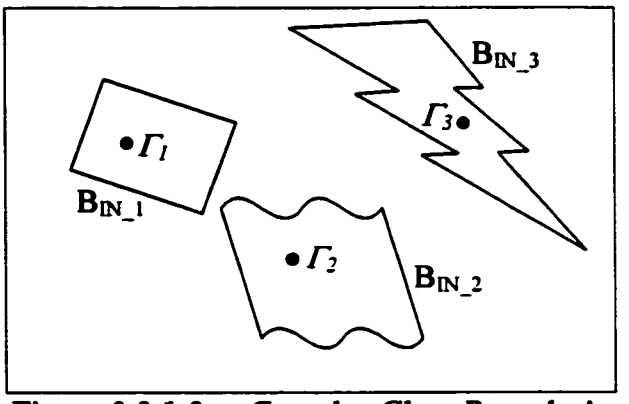


Figure 3.3.1-2 : Complex Class Boundaries

3.4 Architecture and Operation of the RTA NN Model

Figure 3.4-1 shows the main components of the RTA NN model. The overall system (within the dotted lines) is referred to as a *supernet* S_R. It consists of R subnets, where each subnet is a neural network which models a discrete-time Nonlinear Dynamical System (NDS) with a single, globally and asymptotically stable attractor Γ_j. Just like the Uncle Brian example of section 3.3, each attractor represents a different class prototype and all inputs to a subnet-j will eventually converge to the attractor Γ_j in an amount of time which is inversely proportional to the class membership probability.

To implement the various NDS, each subnet is taught an iterated mapping function of the form $Y_j(n) = H_j(Y_j(n-1))$, as described earlier in section 3.2. The boxes with the Z^{-1} factor in Figure 3.4-1 represent delay devices which implement the iteration of the NDS output back to its input, with a one-step lag. At time $n < 0$, $Y_j(n) = 0$. At $n = 0$, the input to each subnet NN-j is the vector to be classified $X_k(0)$, such that $Y_j(0) = X_k(0)$. The switches in the figure are flipped from the down to the up position at time $n = 1$, so that $Y_{jk}(1) = H_j(X_k(0))$. All the subnets are then iterated simultaneously and the first one to

reach its attractor within a given time limit wins the race (i.e., identifies the input). So far, this is similar to the concept introduced in section 3.2 where classification was described with the simple analogy of dropping identical copies of the same 'marble' into R independent 'bowls' simultaneously and observing which one reaches the bottom first.

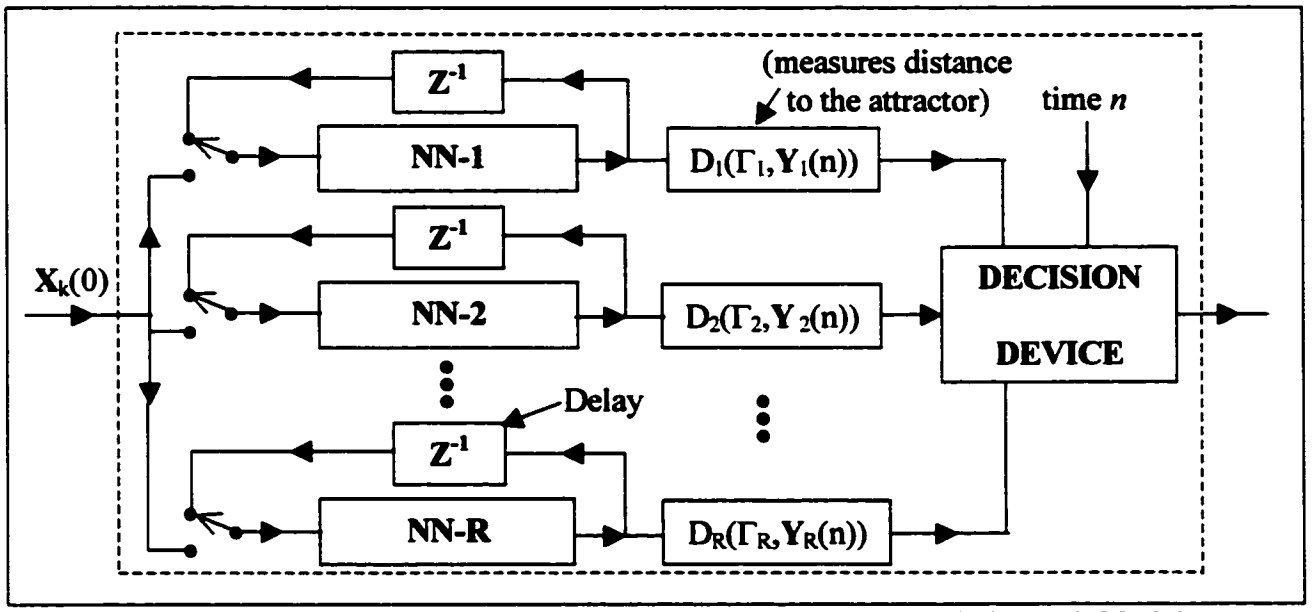


Figure 3.4-1 : Basic Architecture of the Race To the Attractor Neural Network Model

At the output of each subnet, a function $D_j(\Gamma_j, Y_j(n))$ is used to quantify the distance between the output $Y_j(n)$ of NN-j at time n and the attractor Γ_j. Note that the attractor itself can be defined as a region in the distance objects D_j of Figure 3.4-1 (this process is described in chapter 5). By doing so, the distance object implements one of the mechanisms described earlier in section 2.4.1.2 of chapter 2 (page 42).

The output of all these distance functions is fed to a decision device which essentially acts as the referee for the race in order to determine the winner.

3.5 Training Aspects of the RTA NN Model

In this section, some of the main aspects of learning in the RTA NN model are introduced. Our aim is to provide a basic appreciation of how the RTA NN model can be

taught to discriminate between inputs belonging to different classes.

In order to classify inputs correctly, each subnet of the RTA NN supernet must learn the following information:

1. An attractor Γ_j, which is a 'suitable' class prototype. This means that the attractor must be as simple as possible but must still capture all the essential characteristics of that class (e.g., just as point 'C' of Figure 3.3-1 (page 69) represents the simplest representation of what constitutes 'Uncle Brian').

2. Suitable trajectories from initial points $X_k(0)$ to the attractor Γ_j. This means that the path of transformation from one to the other must be logical in some sense.

3. A suitable convergence rate function $\Delta H(X(n))$ (e.g., as shown in Figure 3.2-3 on page 66) on each trajectory. This means that the speed at which points on a trajectory converge to the attractor must be adjusted to ensure that the number of iterations to convergence for each point $X_k(0)$ on each trajectory is inversely proportional to the probability of class membership (i.e., in accordance with axiom E_3.3-1).

How the learning process actually evolves depends in large part on how much of that information is available at the start of training. Let us consider the two extremes of the range of possibilities:

a. ***Minimum knowledge incremental learning*** : Here, nothing is known at the start of training, not even the class prototypes. Learning is purely incremental, based on each training example as it arrives. This is somewhat of a trial-and-error process, where estimates of class prototypes, trajectories, etc, must be constantly revised/refined as new data arrives.

b. ***Maximum knowledge memorization learning*** : Here, everything is known prior to the start of training. All that is required is to teach the predefined convergence rate function $\Delta H(X(n))$ on each predefined training trajectory from $X_k(0)$ to the attractor Γ_j to each of the subnets of the RTA NN model. Because the complete solution is known before training starts, the NN simply needs to 'memorize' it (i.e., there is no trial-and-error process as in incremental learning).

The case of minimum knowledge incremental learning is of interest to us because it is a realistic model of how learning typically occurs in the brain. Since the RTA NN paradigm seeks to model the learning process in the brain to a certain extent, it is relevant to this thesis to discuss how the RTA NN model would operate with this learning approach. This subject is discussed in section 3.5.1.

The case of maximum knowledge memorization learning is somewhat more artificial since the classification problem is essentially solved completely before training of the subnets of the RTA NN even begins. However, this approach is of interest because it allows us to quickly teach a critical mass of information to a NN without having to go through the much longer trial-and-error process of incremental learning. This ability is particularly important in our case since the RTA NN is implemented in software on a sequential-instruction computer. Unlike hardware implementations which benefit from the massively parallel structure of NN to achieve fast learning [26], software implementations are inherently slower because parallel neurons are actually taught sequentially. Hence, it is advantageous for us to be able to quickly teach the RTA NN a large quantity of information in a relatively short time interval[iii]. This learning approach will be discussed briefly in section 3.5.2.

[iii] Even then, training can still require several hours of non-stop computing (e.g., 5 to 20) on a Pentium II-300 MHz computer to teach a NN of a few hundred neurons to learn trajectories with 6000-15000 training points.

3.5.1 Minimum Knowledge Incremental Learning

To better explain the concept of incremental learning, it is convenient to consider the analogy of a child who is being taught the alphabet. Such a learning task would evolve as follows:

1. Initially, the child has no concept of what the alphabet is or is not (i.e., the "*clean slate*" starting point);

2. As packets of information arrive, a crude model starts to emerge, which describes not only what the letters of the alphabet are but also what they are not (e.g., 'A' is not 'B' or number '4' or an animal, etc);

3. As additional bits of information arrive, they are integrated and reconciled with previously acquired knowledge to refine the model (e.g., **A**, *A*, *A*, **A**, are all instances of class C_A but are not instances of classes C_B, C_C, C_D, etc).

A key aspect of incremental learning which is apparent from this process is that new information does not involve the loss (or discarding) of previously acquired information. Instead, new information is integrated, correlated and reconciled with existing information in order to refine the model, class prototype or concept being learned. Thus, incremental learning involves the modification of assumptions made earlier to a certain extent. This is related to the key characteristic of 'learning without memory loss' which was introduced in section 2.3 of chapter 2 (page 37).

The type of incremental learning described above is well rooted in mathematics, through the concept of *Bayesian inference learning* [4]. This concept is introduced in section 3.5.1.1. In section 3.5.1.2, the manner in which the subnets of the RTA NN utilize Bayesian inference learning to learn attractors, trajectories and convergence rates for one class is discussed. In section 3.5.1.3, the process of learning several classes concurrently is discussed.

3.5.1.1 Bayesian Inference Learning

In this section the concept of Bayesian inference learning is introduced. The goal of this section is simply to provide the reader with an appreciation of how this concept can model incremental learning and how it is related to the discussion of class membership probabilities. A thorough, theoretical discussion of this concept is not required to understand the RTA NN model paradigm and is therefore beyond the scope of this dissertation. However, such a discussion is available in reference [4] for the interested reader.

Consider first the initial condition of zero-knowledge about any specific subject (i.e., the "clean slate" condition described in item '1' of section 3.5.1). In Bayesian inference learning, this condition is modeled with a *prior probability density function* (pdf) which we label $p(Y)$. In simple terms, a pdf is a function which provides an indication of the probability of an event Y. Peaks in $p(Y)$ indicate that the corresponding values of Y are very likely to occur, whereas troughs in $p(Y)$ indicate that the corresponding values of Y are unlikely to occur. A *prior* pdf is a function which represents the probable outcomes of an experiment, before results from the outcome of the experiment become available. Chapter 4 provides a more formal description of a pdf.

In Figure 3.5.1.1-1, a pdf $p(Y)$ is illustrated for a one-dimensional (1D) example. In this example, assume that the 'learner' is told that he/she must guess a number Y. In the absence of any other information, any value between $\pm\infty$ (i.e., infinity) is feasible. This is represented by a uniform *prior* pdf $p(Y)$ in that figure. In the analogy of the child learning the alphabet, the "clean slate" condition represents the condition where the child has absolutely no idea of what an alphabet is. For all he/she knows, it could be an animal, a number, a person, etc. In the

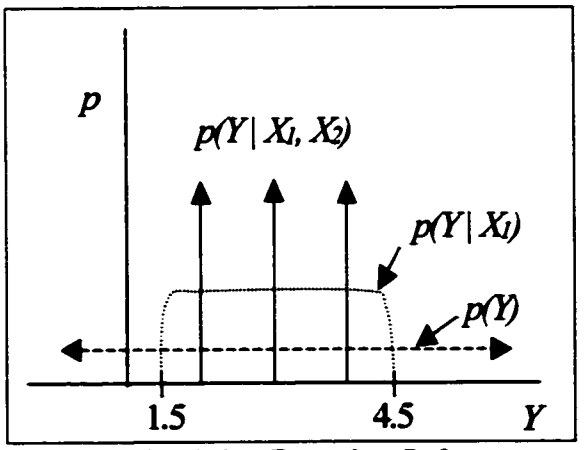


Figure 3.5.1.1-1 : Bayesian Inference

absence of knowledge, the alphabet is equally likely to be any of these. Hence, a uniform pdf is adequate since it indicates an equal probability throughout the space of possibilities.

As soon as some information arrives, the Bayesian inference learning process seeks to develop a *posterior* probability function $p(Y \mid X_1)$. The notation $p(Y \mid X_1)$ should be interpreted as follows : "the probability (or pdf) of event Y given that information X_1 has been received". The posterior pdf refines the knowledge about what Y is (and what it is not) based on the information provided. The posterior pdf $p(Y \mid X_1)$ is related to the prior pdf via Bayes' rule (this will be discussed in chapter 4).

Assuming this information X_1 is relevant to Y, the posterior pdf should become more peaked. For the example of Figure 3.5.1.1-1, assume that $X_1 = \{$*"the number is between 1.5 and 4.5"*$\}$. Although this information does not tell us what the number is exactly, it does tell us what it is not. Hence, the posterior pdf $p(Y \mid X_1)$ now consists of a plateau which extends over the range 1.5 to 4.5. If the learner is then told $X_2 = \{$*"the number is an integer"*$\}$, the pdf becomes a discrete function with three peaks of height 0.3333, indicating that only 2, 3, and 4 are possible. This is shown as $p(Y \mid X_1, X_2)$ in Figure 4.8.1.1-1. Here, $p(Y \mid X_1, X_2)$ should be interpreted as follows : "the probability (or pdf) of event Y given that information X_1 and X_2 has been received".

Note that X_2 by itself only indicates that integers in the range $-\infty$ to $+\infty$ are possible whereas X_1 by itself only indicates that it can be a real number in the range [1.5, 4.5]. But when the two are integrated and reconciled with each other (i.e., no memory loss), the model of the number being guessed becomes more accurate.

Bayesian Inference learning provides a systematic approach through which new information can be merged with previously acquired information to refine the model of what is being learned. This learning is represented as the reshaping of a posterior pdf which captures what the model (or class prototype, or concept) probably is and also what it probably is not.

The concepts of prior and posterior pdf will be clarified more rigorously in chapter 4 (section 4.3). But the above description is sufficient at this stage to allow us to explain how the subnets of the RTA NN model can learn attractors, trajectories and NDS convergence rates incrementally.

3.5.1.2 Incremental Learning of Attractors, Trajectories and Convergence Rates

Just like a child has initially no conception of what the letters of the alphabet represent, the RTA NN must start life with a "clean slate". In practical terms, this corresponds to having random initial parameters for the neurons of each subnet. This means that the convergence rate function $\Delta H(X(n))$ is initially random so that trajectories from any inputs are random (or chaotic), with no attractor defined[iv].

Training involves showing examples of different classes to the student. As training data is shown to the NN, an estimate of a suitable class prototype begins to emerge. Initially, the first such prototype is simply the first class sample shown (e.g., (A, C_A) results in $\Gamma_A \approx A$). But as more and more samples are shown (e.g., (A, C_A), $(\alpha A, C_A)$, (A, C_A), etc), this estimate is refined until the simplest possible prototype emerges which captures what all these samples have in common. Hence, throughout the training process, the attractor itself can be redefined. This process of refining the estimate of the attractor as new training samples arrive is based on the Bayesian learning concept described in the previous section.

New training pairs do not only define the attractor, but also force the NN to postulate a 'reasonable' trajectory from each sample point to the attractor. The process of selecting a suitable trajectory is also based on Bayesian inference learning. In the absence of knowledge, a possible trajectory is initially guessed at. As new training data arrives, these trajectories may be modified to agree with the new information.

[iv] It is interesting to note that this initial chaotic NDS state with no fixed attractor is very similar to the initial state of the neurobiological brain model of Freeman [23] (discussed in chapter 2) before recognition or training takes place.

If several trajectories are equally likely, a suitable one must be selected randomly. This latter concept is illustrated in the example Figure 3.5.1.2-1 where a noisy number '4' input can follow several equally likely trajectories to reach the class prototype. In that figure, hatched squares represent pixels which are in error (i.e., a 0 instead of a 1 or vice-versa). For that particular example, each iteration results in two such errors being corrected.

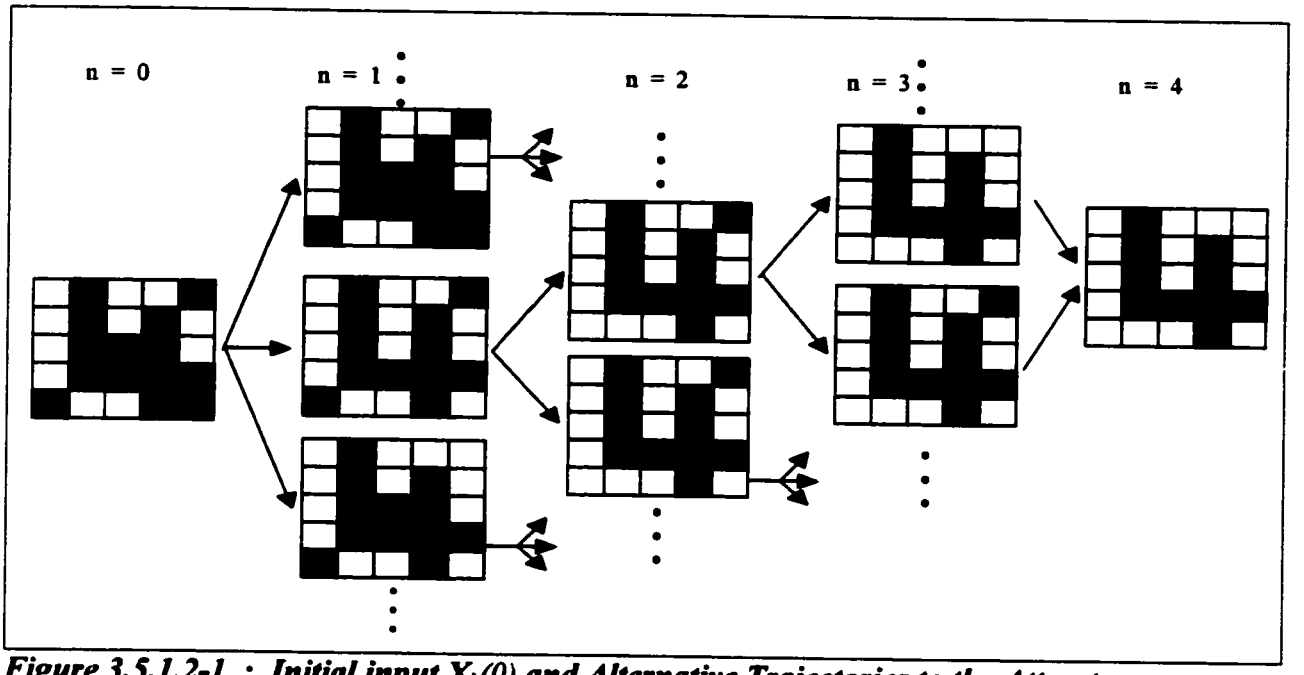


Figure 3.5.1.2-1 : Initial input $X_k(0)$ and Alternative Trajectories to the Attractor

In section 2.4.1 (page 39), it was argued that the brain learns complex classification tasks by simplifying them as much as possible, through various mechanisms. One way to do this is to minimize the complexity of the iterated mapping function by defining each trajectory such that each point *Y(n)* iterates to a point *Y(n+1)* which is:

a. closer to the attractor than *Y(n)*; and

b. has a probability of class membership which is as close as possible to that of *Y(n)* but is slightly higher (recall from section 3.3.1 (page 72) that this is necessary for points on the same trajectory).

The first rule ensures that all trajectories converge towards the attractor. The second rule ensures that the convergence rate ΔH is maximum between points $Y(n)$ and $Y(n+1)$ (this will be proven in section 4.5.2.1). The rationale for maximizing ΔH on all trajectories can be understood by considering the situation where a marble is rolling down the sides of a bowl. The marble will naturally seek the path of least resistance in order to reach the attractor (i.e., the bottom of the bowl) as quickly as possible, through the simplest route available. This path has the maximum possible convergence rate. The same approach is used in the RTA NN.

Initially, trajectories may need to be changed often. But eventually, as more and more sample points get correctly assigned to suitable trajectories, regions will emerge around the attractor where trajectories become relatively stable. These regions can be defined by boundaries which connect all points which reach the attractor in the same amount of time. Then, when a new point far away from the attractor are introduced, the learning process becomes simpler since the NDS now only needs to find a reasonable trajectory from that point to the closest stable boundary.

The magnitude of the convergence rate function will also have to be adjusted to ensure that the number of iterations to convergence is inversely proportional to the class membership probability of each point $X_k(0)$ on each trajectory. Recall that this is necessary to ensure that only the correct subnet wins the race. For example, when a child makes a false assumption such as "B is a member of class C_A", the tutor will correct him/her so that a boundary is set in the brain to separate 'B' from class C_A. Similarly, if subnet-B in the RTA NN model incorrectly wins the race, the tutor will 'punish' subnet-B (i.e., force it to decrease the convergence rate on that trajectory) and will also 'reward' subnet-A (i.e., force it to increase the convergence rate on that trajectory). In that way, the 'correct' subnet will eventually learn to win the race every time for inputs which are IN its class (and to lose it every time for inputs OUT of that class). Note that, here again, an initial estimate of the convergence rate is made but gets refined as new information is provided (i.e., Bayesian inference learning).

In Figure 3.5.1.2-2 an attempt has been made to illustrate how the basin of attraction and trajectories of a NDS for character 'A' might be visualized. Here, the class prototype 'A' is shown in the center and is the attractor of the NDS. Nearby, a few training examples which are rotated, scaled and sheared versions of that prototype are shown, as well as trajectories leading to the attractor. That figure also shows more complex-looking members of that class and their trajectories to the attractor.

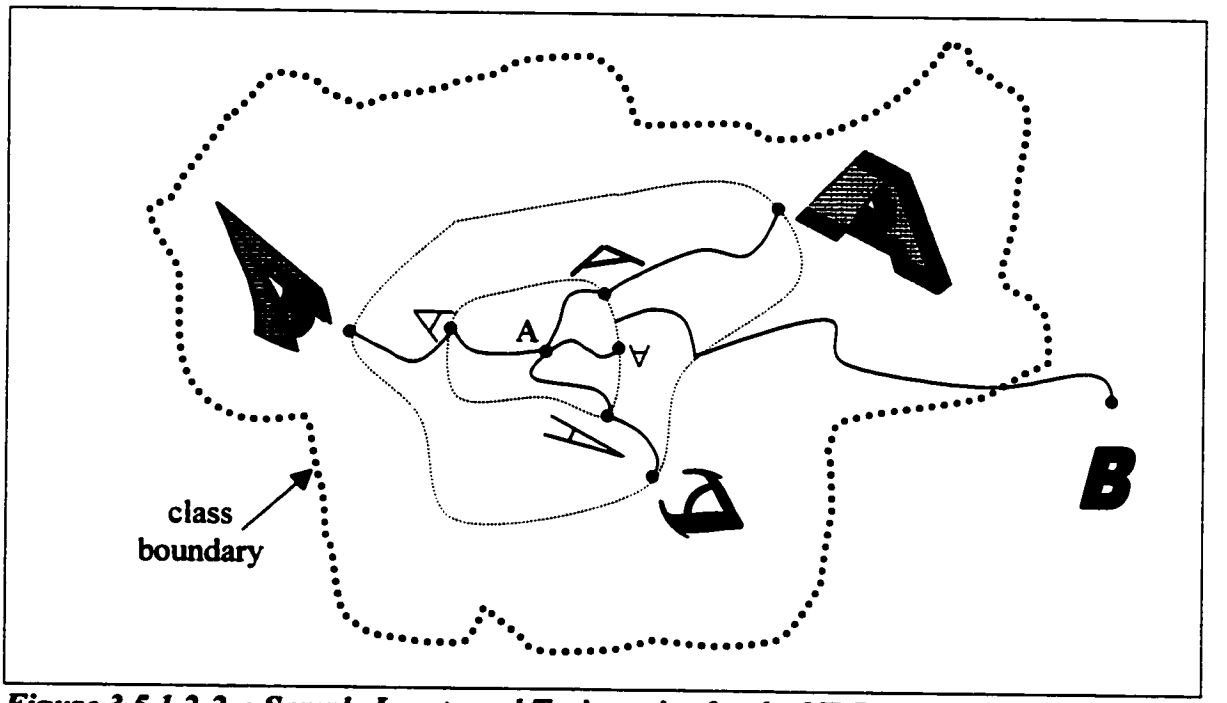


Figure 3.5.1.2-2 : Sample Inputs and Trajectories for the NDS with Attractor Γ_j = 'A'

Each boundary represents inputs which have the same probability of class membership (i.e., same time to convergence). The outer line represents the class boundary B_{IN_A} (i.e., points on or inside it are members of class C_A while points beyond it are OUT of that class). Note that an input 'B' has also been included, to emphasize the fact the NDS which specializes in recognizing letter 'A' must still be taught the inputs which are NOT members of that class. Finally, keep in mind that this figure is intended only to illustrate a concept; trajectories, boundaries, etc, should not be interpreted too literally.

3.5.1.3 Learning Multiple Classes Incrementally

Consider again the example of a child learning the alphabet. It does not seem sensible to teach that child every conceivable shape of letter 'A' before introducing letters 'B', 'C', etc. Instead, it seems more effective to teach him/her the 'basic' shapes of letter 'A' and then move on to letters 'B', 'C', etc. The reason why this seems more sensible is that learning the class for letter 'A' does not involve learning only what is a letter 'A' but also what is not a letter 'A'. Without presenting such negative examples of class membership (e.g., "B is NOT in class C_A"), the child would soon assume that any shape can be viewed as a member of class C_A if it is transformed sufficiently. Thus, effective learning of the alphabet requires that characters be learned concurrently so that the boundaries between different classes are defined gradually (i.e., through a Bayesian inference learning process).

In terms of the RTA NN model, the learning process would first involve teaching one subnet some basic examples of letter 'A'. This would include some basic forms of letter 'A' (e.g., A, *A*, $\mathcal{A}$, **A**) so that a reasonably accurate attractor prototype is estimated. Then, rotated, scaled and sheared versions near the attractor would be presented (e.g., as shown in Figure 3.5.1.2-2). When a new character class is introduced, the subnet-A will then postulate trajectories from samples of that class to Γ_A. The subnet will then adjust the magnitude of its convergence rate function along the most suitable trajectory in response to 'punishment' from the decision device so that the time to convergence will ensure that subnet-A does NOT win the race. In that way, the class boundary is also learned incrementally.

When a new class appears, a new subnet is added to the RTA NN. This new subnet must then learn an attractor, trajectories and convergence rates as described in the previous section. But when a new subnet is added, it is henceforth trained on all training vectors which arrive, regardless of whether or not they belong to its class.

3.5.2 Maximum Knowledge Memorization Learning

In this section, an alternative training approach is discussed briefly. In this approach, it is assumed that the 'final solution' is known before the training of the subnets of the RTA NN is even started. This means that the final trajectories which iterate all input points to the attractor in a time consistent with the class membership probabilities are already known. Furthermore, the optimal class prototypes Γ_j and the optimal decision boundaries are all assumed known.

In such a case, it is only necessary to teach each subnet of the RTA NN to 'memorize' the convergence rate function $\Delta H(X(n))$ for all training examples and the training task is complete.

This approach will be used initially in the training example discussed in chapter 5. Note that despite the artificiality of this learning method, it will provide us with a fully trained RTA NN which already has a basic, critical mass of acquired knowledge. With that knowledge in place, it will be possible to demonstrate that the RTA NN model achieves the key learning characteristics described in chapter 2, and that it implements the mechanisms described in that chapter.

With this approach, the learning process can be implemented with traditional algorithms (e.g., such as backpropagation). The implementation of this learning process will be discussed in more detail in chapter 5.

3.6 Comparison Between the RTA NN Model and Existing Classification Paradigms

In this section, the RTA NN model paradigm is compared briefly to the existing neural network paradigms introduced in sections 2.2 (page 25) and 2.5 (page 49) of chapter 2. The purpose of this comparison is to highlight the differences, advantages and disadvantages of the RTA NN model relative to those other classification paradigms.

In section 3.6.1, the RTA NN is compared to one-step associative memory classification paradigms. Section 3.6.2 compares the RTA NN model to neurodynamical classification paradigms. Section 3.6.3 compares the RTA NN model to the Freeman [23] neurobiological model introduced in chapter 2.

3.6.1 The RTA NN Paradigm versus One-Step Associative Memory Paradigms

This paradigm was introduced in section 2.5.1 of chapter 2 (page 50). One-step associative memories classify inputs through the use of static mapping functions H_S which transform an input X into an output class in one time step. For these nets, time is irrelevant to the classification process. By contrast, the RTA NN model utilizes time to convergence as the key parameter through which classification is achieved. This constitutes a significant difference in the classification paradigm.

Furthermore, one-step associative memories do not achieve the key characteristics of learning discussed in chapter 2 to a significant extent. This is due in large part to the fact that H_S is complex because it is a static function whereas the RTA NN model uses simpler iterated functions. The additional complexity of H_S limits the complexity of problems which such nets can learn. Furthermore, one-step associative memories often attempt to learn all classes as one, global mapping function H_S. This makes the function more complex (i.e., more difficult to learn). In addition, this global mapping approach results in significant memory loss if the addition of a new class is attempted. Finally, the fact that each neuron participates in the global mapping of all classes means that the death of one neuron may alter the mapping for all classes. Repairing such damage can only be done by attempting to retrain the whole net but this will usually lead to memory loss.

By contrast, the RTA NN utilizes iterated maps H_I which are inherently less complex than their static equivalent. In addition, each subnet specializes in the classification of one class only (e.g., "X is IN class C_j" or "X is OUT of class C_j"). This limits the inherent complexity of each mapping function, regardless of how many classes are learned. The

RTA NN model also has the ability to redefine the attractor as a region. This latter technique allows a further reduction in the complexity of the mapping function and can be used to repair neural damage. Finally, the RTA NN model has the ability to re-use one subnet trained for one NDS-j to implement a different NDS-k by using a process similar to that discussed in section 3.2.1 (page 67). Hence, the RTA NN model implements the four mechanisms discussed in chapter 2 and is therefore able to achieve the key characteristics of learning to a significant extent (this will be demonstrated in chapters 4 and 5).

3.6.2 The RTA NN Paradigm versus the Neurodynamical Memory Paradigms

The neurodynamical paradigm discussed in section 2.5.2 of chapter 2 (page 56) are similar to the RTA NN in some respects. They all utilize feedback in a NDS to implement trajectories which lead to specific attractors. However, in traditional neurodynamical paradigms, time to convergence is NOT a significant factor in the classification problem; it is simply a delay which must be endured before the classification answer becomes available. In that respect, the RTA NN model paradigm is significantly different since time to convergence is the key classification parameter for our model.

Furthermore, most neurodynamical paradigms discussed in chapter 2 usually implement a classification mapping as a *global* function, which captures all classes in one iterated map H_I. This was shown to result in a function whose complexity increases rapidly as more and more classes are added. Such nets therefore have a limited learning capacity and are unable to learn new classes without loss of memory. Finally, the death of individual neurons may affect all classes. Here again, there is no simple way to repair such neural damage without retraining (and memory loss).

Other significant problems were identified for specific nets. For example, the Hopfield net was shown to have a very limited learning capacity and a tendency to create false

attractors. As another example, the Boltzmann machine has a very slow learning rate. As a result, even modestly complex problems are not achievable in practice.

Overall, these conventional neurodynamical models achieve few (if any) of the key learning characteristics identified in chapter 2. By contrast, the RTA NN model can achieve all of these to a much higher degree.

3.6.3 The RTA NN Paradigm versus the Freeman Neurobiological Model

In many ways, the Freeman [23] neurobiological model discussed in section 2.2 of chapter 2 (page 25) has the most similarities with the RTA NN model paradigm. Both utilize the concept of attractors in a NDS to encode classes. In both cases, there are as many attractors as there are classes. In both cases, the attractor which emerges at the end of the classification process identifies the unknown input as the class associated with that attractor. In both cases, a competition takes place to determine which attractor will prevail.

But there are also several differences between these models. For the RTA NN model, the attractors are defined as static points (or regions) whereas the Freeman model involves attractors which are chaotic, time-varying waveforms with quasi-periodic components. The RTA NN model utilizes one simple NDS with one attractor for each class, versus one large NDS with many chaotic attractors for the Freeman model. Furthermore, time to convergence is largely irrelevant in the Freeman model since it is only the final attractor achieved that matters, not how long it took the NDS to fall into it. By contrast, time to convergence is crucial to the classification process in the RTA NN.

There is also a significant difference in the competition process to determine the winning attractor. For the RTA NN model, the competition involves several NDS which race independently of each other towards the finish line. This is analogous to a horse race where each horse runs in its own track so that there is no interference between them. By contrast, the Freeman model involves independent Neural Cell Assemblies (NCA) which each attempt to lure the global NDS (i.e., the olfactory bulb) to converge to their specific

attractor. The NCA which succeeds at this task identifies the input. Hence, this is not really a race, but more like a contest to capture the attention of the olfactory bulb. This is somewhat analogous to having several persons in an arena attempting to lure a bull towards them. The person that "wins" the attention of the bull wins the classification contest. Note that in this type of contest, there seems to be some degree of interference between the various NCA competing for the attention of the olfactory bulb. By contrast, there is no interference between the competitors in the RTA NN model.

The Freeman model appears to have all the key learning characteristics identified in chapter 2 (as far as we can tell from the experimental results reported). However, the exact process through which input excitation gets transformed into a learned attractor response is not known at this time. As a result, this model cannot be used to implement classification problems since no algorithm yet exists to explain precisely how it learns or recalls in practice.

In that respect, the RTA NN model has an advantage since the theory to teach it to learn classification problems exists (chapter 4) and can be shown to work in practice (chapter 5).

3.7 Summary and Conclusions

In section 3.2, it was shown how Nonlinear Dynamical Systems (NDS) could be used to define trajectories from arbitrary initial points which converge to specific attractors as time elapses. It was shown how the convergence rate of the iterated mapping function determines the time required for an input point to reach that attractor. That section also demonstrated how a NDS could be re-used to map a different attractor.

Section 3.3 introduced the concept of the RTA NN model. It was shown how several NDS could be used to classify inputs by using attractors which each represent a different class prototype. It was argued that the key to the correct classification of inputs by each NDS is to ensure that the time required to reach the attractor is inversely proportional to

the probability that those inputs are members of that class (i.e., axiom E_3.3-1 on page 71)[v]. That section also clarified the important fact that the time required to converge to an attractor is not always related to the distance from that attractor.

Section 3.4 described the architecture and basic operation of the RTA NN model. It was indicated that the 'distance objects' at the output of each subnet could be used to redefine the attractor from a point to a region. Following this, fundamental aspects of training were introduced in section 3.5. It was argued that the RTA NN would normally learn classes and prototypes incrementally (i.e., "minimum knowledge incremental learning"), through a process which is based on Bayesian inference learning. A more artificial learning approach was also introduced (i.e., "maximum knowledge memorization learning"). This approach allows the RTA NN to quickly acquire a critical mass of knowledge and is therefore useful for software implementations of the RTA NN model (which is the case for this thesis).

In section 3.6, the RTA NN model paradigm was compared to existing classification paradigms using neural networks (NN). It was argued that the RTA NN is significantly different from all these paradigms because of the novel use of 'time to convergence' as the key parameter defining the classification process. It was also argued that the RTA NN model achieves the four key characteristics of learning (described in chapter 2) to a higher degree than these other paradigms. This is possible for the RTA NN model because it implements the four mechanisms (described in chapter 2) which allow these characteristics to be achieved to a significant degree. This ability will be demonstrated in chapters 4 and 5.

[v] Recall again that the time required to reach the attractor for most NDS is infinite [15,18]. In this thesis, references to the NDS reaching the attractor in a finite time imply that the distance to the attractor is small.

Chapter 4 :

Development and Analysis of a Mathematical Model for the RTA NN Paradigm

4.1 Introduction

This chapter expands upon the basic RTA NN theory introduced in the previous chapter. Our objective is to develop a mathematical model which will capture the relationship required between 'time to convergence' and class probabilities in order to define a NDS whose convergence rate, attractor and trajectories will allow the correct classification of inputs. The ultimate aim of this development is to allow us to progress from theory to practice, so that a 'real' classification problem can be implemented with the RTA NN model (this will be discussed in chapter 5).

In section 4.2, the basic theory of Nonlinear Dynamical Systems (NDS) is examined and the conditions required to ensure that an attractor is globally and asymptotically stable are defined. In section 4.3, basic concepts of statistical pattern recognition are reviewed. Section 4.4 develops the relationship between the probability of class membership and 'time to convergence' for input points at the class boundary and at the attractor. The fundamental constraints which result from this relationship are also discussed. In section 4.5, this relationship is further expanded upon to include all input points to be classified. This section also describes how the relationship between 'time to convergence' and posterior probabilities of class memberships is reconciled with the NDS requirements for a globally and asymptotically stable attractor. Different approaches to assign 'time to convergence' are also discussed and the significance, advantages and disadvantages of each one is assessed.

Subsequently, the architecture and operation of the RTA NN model is formalized in mathematical terms in section 4.6. In section 4.7 practical training aspects of the RTA NN model are examined. This section focuses primarily on the 'curse of dimensionality' and its effect on the complexity of a learning task for a neural network. This is followed by a review of the mechanisms through which this learning task complexity can be reduced in the RTA NN model. The section concludes with a discussion of the neural net architectures and training algorithms which can be used for the subnets of the RTA NN.

Finally, section 4.8 summarizes briefly the key concepts presented in this chapter.

4.2 Basic Theory of NDS with Globally and Asymptotically Stable Attractors

In this section, some basic theoretical aspects of NDS are reviewed and mathematical terminology is introduced in order to describe and define NDS characteristics. For the sake of brevity, only aspects of NDS theory which are essential to the operation of the RTA NN model are introduced. A detailed description of NDS theory can be found in classical texts such as references [15,56].

A dynamical system is characterized by a set of related variables which can change with time. The evolution of the system variables is normally described by a set of differential equations. When these equations are nonlinear, the dynamical system is nonlinear (i.e., a NDS). Such systems can be continuous in time or discrete-time. The latter type are more relevant to the RTA NN model because the neural networks used to model the NDS in this dissertation operate in discrete time steps (neural networks were implemented in software). However, most of the analysis on NDS will focus on continuous-time NDS because the mathematical theory is easier to develop for such systems. Section 4.2.1 provides this analysis whereas section 4.2.2 discusses briefly the relationship between continuous-time NDS and discrete-time NDS.

4.2.1 Basic Theoretical Aspects of Continuous-Time NDS

A NDS is typically characterized by a differential equation as shown below [26]:

$$\dot{X} = \partial\vec{X} / \partial t = \vec{F}(\vec{X}(t)) = [f_1(\vec{X}), f_2(\vec{X}), \cdots, f_Q(\vec{X})] \qquad \text{(E_4.2.1-1)}$$

where $X(t)$ = [$X_1(t)$, $X_2(t)$,...,$X_Q(t)$] is the state of the system at time t. This is known as a Q-th order system. $X(t)$ is often referred to as the set of *state variables*.

The function $\partial X/\partial t$ is sometimes referred to as a velocity vector [26]. For the Uncle Brian example of Figure 3.3-1 (page 69), this function actually defines the *convergence rate* of the input towards the attractor along the path shown.

When $\partial X/\partial t$ does not depend explicitly on time, the NDS is known as an *autonomous system* (or time-invariant) [56]. For example, a first-order system $\partial x/\partial t = -2x$ is autonomous but $2xt$ is not. In this thesis, only autonomous systems are considered since the classification problems of interest to us are always independent of time (e.g., '$\mathcal{A}$' is a member of character class C_A regardless of the time when it is classified by the NDS).

The path followed by the state variables from the initial point $X(0)$ to some arbitrary time T is often referred to as a *trajectory* of the system. For example, the line shown in Figure 3.3-1 (page 69) represents the trajectory from the input to the attractor. Similarly, the paths from X_1 to Γ_j or X_2 to Γ_j shown in Figure 3.3.1-1 (page 72) are trajectories of the NDS.

$X(t)$ can be related to $\partial X/\partial t$ along any trajectory c_j through this equation:

$$\vec{X}(T) = \vec{X}(0) + \int_{\substack{t=0 \\ c_j}}^{T} \frac{\partial X_j}{\partial t} dt \qquad \text{(E_4.2.1-2)}$$

where $X(0)$ is the initial point and the integral is evaluated over trajectory c_j.

4.2.1.1 Equilibrium Point and Stability of a NDS

Γ_j is an *equilibrium point* of NDS-j if $\partial \vec{X}(\Gamma_j)/\partial t = 0.0$ [15].

Γ_j is a uniformly stable equilibrium point if, for any $\delta > 0$, there exists a $\alpha > 0$ such that [26]:

$$\left\|\vec{X}(0) - \vec{\Gamma}_j\right\| < \delta \Rightarrow \left\|\vec{X}(t) - \vec{\Gamma}_j\right\| < \alpha \quad \forall t > 0 \qquad \text{(E_4.2.1.1-1)}$$

(i.e., trajectories which start within a certain distance from the equilibrium point always remain within a certain distance of it).

Γ_j is convergent if there exists a $\delta > 0$ such that [26]:

$$\left\|\vec{X}(0) - \vec{\Gamma}_j\right\| < \delta \Rightarrow \vec{X}(t) \to \vec{\Gamma}_j \quad as \quad t \to \infty \qquad \text{(E_4.2.1.1-2)}$$

Γ_j is *asymptotically stable* if it is both convergent and stable. It is said to be globally and asymptotically stable over an input region $R_{IN} \subset \mathbb{R}^Q$ if all trajectories with initial values $X(0) \in R_{IN}$ are stable and convergent. By definition, such NDS have only one equilibrium point to which all trajectories in the input region converge. This makes them well suited to model the classification approach which was introduced in section 3.3 and is illustrated in Figure 3.3-1 (page 69).

4.2.1.2 Lyapunov Function for a NDS

A scalar function $V(X)$ is a *Lyapunov function* of a NDS with state vector $X(t)$ if [15]:

a. it is defined over the entire input space R_{IN};

b. If R_{IN} is unbounded, $V(X) \to \infty$ as $X \to \infty$;

c. $V(X) > V(\Gamma)\ \forall X \neq \Gamma, X \in R_{IN}$;

d. $V(\Gamma) = 0$; and

e. $dV(X)/dt \leq 0.0\ \forall X \in R_{IN}$.

The existence of a Lyapunov function guarantees that a NDS is globally and asymptotically stable over the entire input space R_{IN} over which the above conditions are met [15,56]. Conversely, any globally and asymptotically stable NDS has Lyapunov functions. The time derivative of a Lyapunov function $V(X)$ can be expressed as follows [15,56]:

$$\frac{dV(\vec{X})}{dt} = \nabla V(\vec{X}) \bullet \vec{F}(\vec{X}) = \sum_{j=1}^{Q} \frac{\partial V}{\partial X_j} \frac{\partial X_j}{\partial t} = \sum_{j=1}^{Q} \frac{\partial V}{\partial X_j} f_j(\vec{X}) \qquad \textbf{(E_4.2.1.2-1)}$$

where $F(X)$ is given by equation E_4.2.1-1. Lyapunov functions can often take the form of an "energy function", which relates the rate of dissipation of energy of the NDS system to the convergence rate $\partial X/\partial t$ along a trajectory. For example, if $\partial x/\partial t = -2x$, $x \geq 0$ and $V(x) = mgx$, the five conditions listed above are met for equilibrium point $\Gamma = 0.0$. Here, $V(x)$ can be viewed as the *potential energy* [49] of a falling particle with mass m and applied force g. The rate of fall is twice the height, in meters per second. For the Uncle Brian example of Figure 3.3-1 (page 69), the distance of the initial point from the attractor measured on the trajectory can also be viewed as a certain quantity of energy which must be dissipated in a fixed amount of time.

Lyapunov functions are relevant to the RTA NN model because they allow us to relate 'distance to the attractor' (which can be viewed as a quantity of energy) to the convergence rate along a trajectory (which can be viewed as the rate of dissipation of energy).

4.2.1.3 Stable Manifolds and Dissipative Systems

NDS which have a globally and asymptotically stable attractor are sometimes referred to as *dissipative systems* [26], because the energy of the system (represented by a Lyapunov function $V(\vec{X})$) is dissipated as time evolves (i.e., $dV(\vec{X})/dt \leq 0 \ \forall \vec{X}, t$). For such systems, the divergence of the convergence rate $\partial\vec{X}/\partial t$ is always negative [26]:

$$\nabla \cdot \vec{F}(\vec{X}) = \frac{\partial f_1(\vec{X})}{\partial X_1} + \frac{\partial f_2(\vec{X})}{\partial X_2} + \cdots + \frac{\partial f_Q(\vec{X})}{\partial X_Q} < 0 \Rightarrow \textit{the NDS is dissipative} \qquad \textbf{(E_4.2.1.3-1)}$$

Dissipative systems are generally characterized by the presence of attracting sets or manifolds of lower dimensionality than the state space of the NDS. These are commonly called *attractors* to which regions of initial conditions will converge to as time increases.

For example, in Figure 3.3-1 (page 69), the prototype at the bottom of the bowl is the attractor of the NDS system. It is of lower dimensionality than initial inputs in the sense that the false beard, hat and glasses can be viewed as additional input dimensions which eventually get discarded (i.e., as irrelevant detail) as the attractor is approached.

The input region over which trajectories of a NDS will converge to the attractor is often referred to as the *basin of attraction* of the NDS [26]. In Figure 3.3-1, the 'bowl' shown is the equivalent to this basin of attraction. Any 'marble' dropped inside the bowl will converge to the attractor while marbles dropped beyond the edges of the bowl will not. In a NDS with more than one attractor, the boundary between basins of attraction is called the *separatrix*.

Note that, since only NDS with one attractor which is globally and asymptotically stable are considered for the RTA NN, the basin of attraction is the entire input space $\mathbb{R}^P$ for each NDS. However, for specific classification problems, inputs will normally span only a subset of that space, which we identify as $\mathbb{R}_{IN} \subseteq \mathbb{R}^P$ in this document. Hence, in practice,

it will only be necessary to ensure that the attractor of the NDS is globally and asymptotically stable over the region $\mathbb{R}_{IN}$.

4.2.2 Discrete-Time NDS

As mentioned before, discrete-time NDS are of primary interest for the RTA NN model because the neural nets used in this research work operate in discrete-time.

The evolution of a discrete-time NDS is usually represented by a *difference equation* of this form:

$$\vec{X}(n+1) = \vec{H}(\vec{X}(n)) = [H_1(\vec{X}(n)), H_2(\vec{X}(n)), \cdots, H_Q(\vec{X}(n))] \quad n \geq 0 \qquad \textbf{(E_4.2.2-1)}$$

This is a NDS whose state at time n is given by $X(n) = [X_1(n), X_2(n), \ldots, X_Q(n)]$. The system is of order Q and is related to the continuous-time NDS of section 4.2.1 through the sampling operation described below [38]:

$$\vec{X}_s(t) = \vec{X}(t) \sum_{n=-\infty}^{\infty} \delta(t - n\Delta t) = \sum_{n=-\infty}^{\infty} \vec{X}(n\Delta t)\delta(t - n\Delta t) \qquad \textbf{(E_4.2.2-2)}$$

where Δt is the sampling interval (discussed later) and where $X(n)$ is the sampled signal $X_s(t)$ evaluated at $n\Delta t$. Figure 3.2-2 (page 66) illustrates a sample mapping function $H(X(n))$ for a one dimensional NDS with $H(X(n))$ given by equation E_3.2-3 (page 66).

The function $\Delta H(X(n))$ is defined as follows:

$$\Delta\vec{H}(\vec{X}(n)) = \vec{X}(n+1) - \vec{X}(n) = \vec{H}(\vec{X}(n)) - \vec{X}(n) \qquad \textbf{(E_4.2.2-3)}$$

This function is really the discrete-time equivalent of the convergence rate function $\partial X/\partial t$ discussed in section 4.2.1. For the example of Figure 3.2-2, the equivalent $\Delta H(X(n))$ is shown in Figure 3.2-3 (page 66). The impact of this convergence rate function on the

required number of iterations to converge (N_{jk}) as a function of initial value $X(0)$ is shown in Figure 3.2-5 (page 67). Figure 3.2-4 illustrates some sample trajectories for this NDS.

Figures 4.2.2-1 and 4.2.2-2 show the Lyapunov energy function $V(X) = X^2$ and the energy dissipation rate function $dV(X)/dt$ respectively for the one-dimensional NDS of equation E_3.2-3 (page 66). The function $dV(X)/dt$ is simply $(2X) \times \partial X/\partial t$, in accordance with equation E_4.2.1.2-1 (with $\partial X/\partial t$ given by equation E_3.2-3). These functions meet the 5 requirements listed in section 4.2.1.2 for a Lyapunov function.

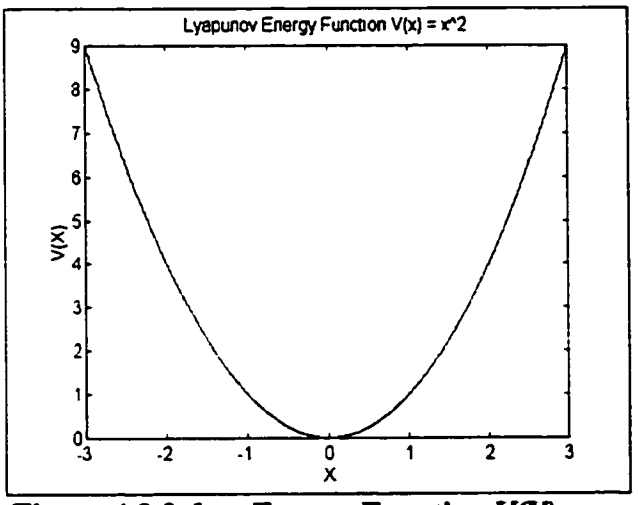


Figure 4.2.2-1 : Energy Function V(X)

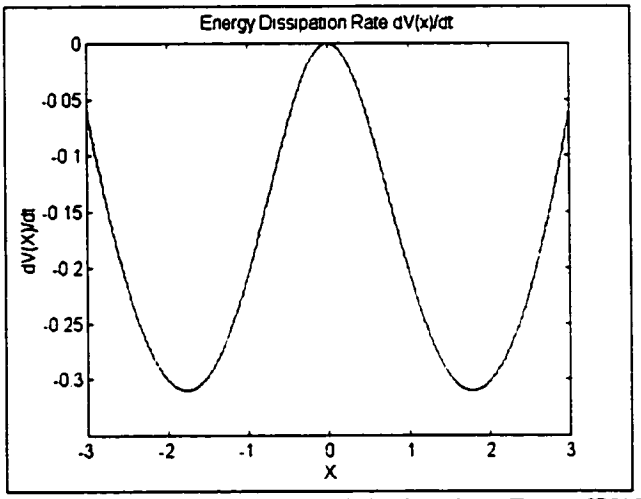


Figure 4.2.2-2 : Energy Dissipation Rate dV/dt

Finally, note that the optimal sampling period Δt to generate a discrete-time NDS from a continuous-time NDS can be related to the Fourier transform of the convergence rate function $\partial X/\partial t$. According to the Nyquist sampling theorem [38], the sampling period is lower-bounded by:

$$\Delta t < \frac{1}{2\Omega_{MAX}} \qquad \text{(E_4.2.2-4)}$$

where Ω_{MAX} is the maximum frequency of the Fourier transform of $\partial X/\partial t$. In simpler terms, the more complex the function $\partial X/\partial t$ is (i.e., the less smooth it is), the higher will be its maximum frequency content in the Fourier domain and the smaller Δt must be in order to capture this complexity [38]. If Δt is longer than the Nyquist period, *aliasing*

[38] will occur and the complexity of $\partial X/\partial t$ will not be captured properly[vi]. But if Δt is much smaller than the required period, sample trajectories generated from the discrete-time NDS will have more points than is required to adequately describe the NDS. This excess redundancy can slow down training time for a neural network (this is discussed in section 4.7.1). The optimal sampling period is close to $1/2\Omega_{MAX}$ so that excess redundancy and aliasing are both avoided.

4.3 Review of Basic Concepts of Statistical Pattern Recognition

In this section, the fundamental concepts of Bayesian theory and statistical pattern recognition are discussed. These will be useful in subsequent sections when 'time to convergence' is related to the probability of class membership.

A statistical approach to pattern recognition requires that the following problem be solved [4]:

> Given a large number of sample patterns X and their class membership C_j, define decision boundaries which minimize the probability of misclassifications.

Let us define the training data set of observations as follows:

$$\chi = \{(\vec{X}_1, C_i), (\vec{X}_2, C_j), \ldots\} \qquad \text{(E_4.3-1)}$$

where $\chi, \vec{X}_k \in \mathbb{R}_{IN} \subseteq \mathbb{R}^P$, $\chi, \vec{X}_k \in \mathcal{C}_{IN}$ and where $\bigcup_{i=1}^{R} C_i = C_{IN}$. Note that $\vec{X}_k$ and X_k are used interchangeably to identify a vector of dimension $P \geq 1$. Similarly, C_i and $\mathcal{C}_i$ both represent class 'i'. $\mathbb{R}_{IN}$ is the subset of the input space $\mathbb{R}^P$ which includes all inputs $X_k(0) \in \mathcal{C}_{IN}$. There are R classes, with $P(\mathcal{C}_k)$ being the prior probability that an input belongs to class k. In accordance with the axioms of probability theory, it is necessary that [41]:

[vi] In simple terms, aliasing occurs when a signal is sampled with an interval which is too large to capture all its 'hills' and 'valleys'. For example, sampling the signal of Figure 4.2.2-2 at $X = -3$, 0 and 3 will produce aliasing because these samples give us no indication that peaks exist at around ± 1.8.

$$\sum_{i=1}^{R} P(C_i) = 1 \qquad \text{(E_4.3-2)}$$

From the set of training samples χ, the *class-conditional probabilities* $P(X|C_k)$ can be determined from Bayes theorem [4]:

$$P(\bar{X} \mid C_k) = \frac{P(\bar{X}, C_k)}{P(C_k)} \qquad \text{(E_4.3-3)}$$

Note that because of class overlaps, there is some ambiguity to be resolved about the position of the decision boundaries between classes. This ambiguity is illustrated in Figure 4.3-1 for a one-dimensional example with two classes. Regions marked *E1* and *E2* correspond to areas where the probability density functions (pdf) [41] overlap so that classification errors may occur there. The boundary B_{IN_J} is the threshold which is normally selected to minimize the probability of classification errors. For the example of Figure 4.3-1, any input $X < B_{IN\_1}$ is classified as $X \in C_1$ while any $X \geq B_{IN\_2}$ is classified as $X \in C_2$. The selection of decision boundaries to minimize the cost of making a decision is discussed in more detail in section 4.3.1.

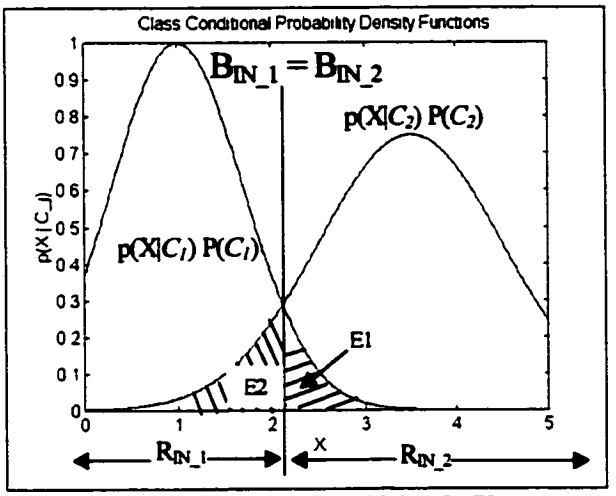


Figure 4.3-1 : Sample pdf for 2 Classes

When a neural net (or any other system) is trained to learn the optimal decision boundaries to minimize cost, the class-conditional probabilities $P(X|C_j)$ and the pdf $p(X|C_j)$ can be derived from the training data set. These functions describe the probability distribution of X, given that the class to which each sample belongs is already known (i.e., conditional probabilities [4]). However, when a NN is used to classify unknown inputs, the converse problem must be considered; the problem of determining the class

with the highest probability, given an unknown input X. This is the *posterior* probability $P(C_j|X)$, which is given by Bayes's rule [4,41]:

$$P(C_j \mid \vec{X}) = \frac{p(\vec{X} \mid C_j)P(C_j)}{p(\vec{X})} \qquad \text{(E_4.3-4)}$$

Note that 'p' refers to a probability density function (pdf) while 'P' is a probability. The denominator in the above equation is a normalizing factor which ensures that

$$\sum_{i=1}^{R} P(C_i \mid \vec{X}) = 1 \qquad \text{(E_4.3-5)}$$

Hence, Bayes' rule provides us with a mechanism through which class conditional probabilities (derived from training data) can be related to unknown posterior probabilities.

Reference [4] provides a detailed description of how neural networks can be made to learn posterior probabilities from the training data through the use of the *maximum likelihood* technique. However, this approach will not be discussed in detail in this dissertation since the RTA NN model does not learn these probabilities directly.

4.3.1 Selecting Decision Boundaries to Minimize Cost

From classical Bayesian theory, it can be shown [4,41] that the cost of making a classification error is minimized by selecting the class C_j having the largest posterior probability so that $X_k \in C_j$ is the optimal decision 'D' made when

$$P(C_j \mid \vec{X}_k) > P(C_r \mid \vec{X}_k) \quad \forall r \neq j \qquad \text{(E_4.3.1-1)}$$

Using equation E_4.3-4 and dropping the pdf $p(X)$ from the denominator (since it is independent of the class membership), the following rule can be derived:

$$p(\vec{X}_k \mid C_j)P(C_j) > p(\vec{X}_k \mid C_r)P(C_r) \;\; \forall r \neq j \Leftrightarrow D = X_k \in C_j \qquad \text{(E_4.3.1-2)}$$

With R classes, the input space is divided into R regions R_{IN_1}, R_{IN_2}, etc, such that

$$\vec{X}_k \in R_{IN\_s} \Leftrightarrow \vec{X}_k \in C_s \qquad \text{(E_4.3.1-3)}$$

Figure 4.3-1 (previous section) illustrates an example with two regions separated by a boundary B_{IN_j}. In general terms, the optimal position of the decision boundaries is based on minimizing an overall *risk function* Ω defined as follows [4]:

$$\Omega = \sum_{j=1}^{R} \omega_j P(C_j) \qquad \text{(E_4.3.1-4)}$$

where ω_j is the *class risk* associated with making a decision in favor of class C_j. The class risk is given by [41]:

$$\omega_j = \sum_{i=1}^{R} L_{j,i} \int_{R_{IN\_i}} p(\vec{X} \mid C_j) d\vec{X} \qquad \text{(E_4.3.1-5)}$$

with $L_{j,i}$ defined as the penalty associated with assigning a pattern X to class C_i when in fact it belongs to class C_j. For the 2-class example of Figure 4.3-1, the overall risk can be expressed as follows:

$$\begin{aligned} \Omega = {} & L_{1,1}P(C_1)\int_{R_{IN\_1}} p(X \mid C_1)dX + L_{1,2}P(C_1)\int_{R_{IN\_2}} p(X \mid C_1)dX + \\ & L_{2,1}P(C_2)\int_{R_{IN\_1}} p(X \mid C_2)dX + L_{2,2}P(C_2)\int_{R_{IN\_2}} p(X \mid C_2)dX \end{aligned} \qquad \text{(E_4.3.1-6)}$$

Typically, $L_{1,1} = L_{2,2} = 0$ (i.e., there is no penalty when making a correct classification) so that the risk is minimized by minimizing incorrect classifications. In Figure 4.3-1, this corresponds to minimizing the combined area *E1* and *E2*. For that example, it can be shown that this corresponds to evaluating the *likelihood ratio* [4]:

$$y(X) = \ln\left(\frac{p(X \mid C_1)}{p(X \mid C_2)}\right) + \ln\left(\frac{P(C_1)}{P(C_2)}\right) \qquad \text{(E\_4.3.1-7)}$$

and assigning X to class C_1 if $y(X) > 0$ or to C_2 otherwise. Note that in this particular case, there is no need to evaluate the posterior probabilities.

How does this relate to the RTA NN model ? What is important to remember from the above development is that optimal decision boundaries are made to correspond to values of the posterior pdf which result in a minimum cost (i.a.w. equation E_4.3.1-4). For the RTA NN to achieve the performance of an optimal Bayesian classifier, it is therefore essential to ensure that the 'time to convergence' assigned to input vectors X must be made a function of the posterior class membership probability in such a way that the classification boundaries established are the optimal boundaries.

4.3.2 Rejection Thresholds

In general, most classification errors will occur where the largest posterior probability is relatively small [4]. In such a case, it is often preferable to avoid making a classification decision. This is equivalent to assigning such inputs to a default class C_{R+1} which is labelled as 'unknown'. For the 2 class example of Figure 4.3-1, the new decision boundaries might take the form shown in Figure 4.3.2-1, with the following classification rules now in effect:

$$\begin{aligned} &\gamma_2 \le y(X) \le \gamma_1 \;\Rightarrow X \in C_3 \\ &y(X) > \gamma_1 \;\Rightarrow X \in C_1 \\ &y(X) < \gamma_2 \;\Rightarrow X \in C_2 \end{aligned} \qquad \text{(E\_4.3.2-1)}$$

where $y(X)$ is a likelihood ratio function of the type shown in equation E_4.3.1-7 and C_3 is the unknown class. The new boundaries B_{IN_1} and B_{IN_2} are sometimes referred to as *rejection thresholds* [4].

This concept of rejection thresholds for the RTA NN model has already been introduced in section 3.3. In the example of Figure 3.3-2 (page 70), any input which had <u>not</u> reached any one of the attractors in 20 iterations was classified as unknown. As another example, consider the two-dimensional classification boundaries shown in Figure 3.3.1-2 (page 74). Assume that points on boundaries B_{IN_j} (with j = 1,2,3) converge to their respective attractor Γ_j in 20 iterations. In that figure, the points <u>outside</u> the shown boundaries will converge to that attractor in more than 20 iterations, resulting in all such points being classified as "unknown" by the corresponding NDS. The region inside these boundaries contains the points which will be classified as members of that class. Note that the fact that these regions do not overlap does <u>NOT</u> imply that the classes themselves do not overlap. This can be seen from the example of Figure 4.3.2-1 where the classification regions R_{IN_1} and R_{IN_2} don't overlap despite the fact that the pdf of classes C_1 and C_2 do overlap.

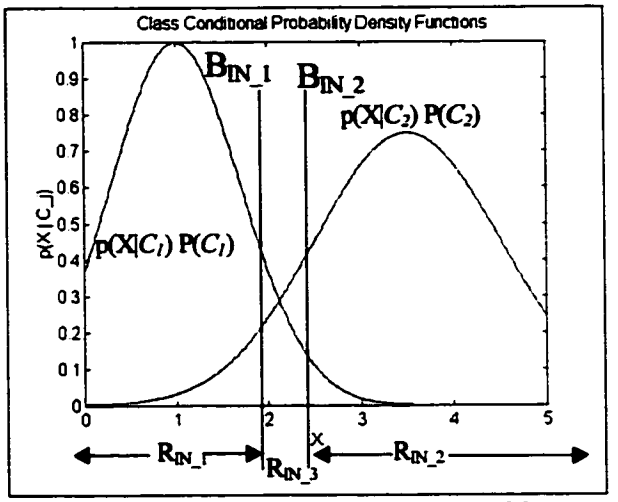


Figure 4.3.2-1 : Rejection Thresholds

4.4 Relating T_{jk} and $P(C_j \mid X_k(0))$ at the Attractor Γ_j and on the Boundary B_{IN_j}

In this section, the mathematical relationship between the 'time to convergence' and the posterior probability of class membership is developed for points on the classification boundary and for the attractor. The main constraints which emerge as a result of this relationship are also discussed.

In general terms, a subset $\mathbb{R}_{IN}$ of an input space $\mathbb{R}^P$ of dimension P is considered, from which the initial vectors to a NDS at time $t = 0$ are selected : $X_k(0) \in \mathbb{R}_{IN} \subset \mathbb{R}^P$. All vectors inside $\mathbb{R}_{IN}$ are classified into one of $R + 1$ classes C_j where:

$$\bigcup_{j=1}^{R+1} C_j = C_{IN} \qquad \text{(E_4.4-1)}$$

Note that by convention, class C_{R+1} will be considered as the "unknown" class in this dissertation. Vectors $X_k(0)$ are relegated to that class when their posterior probability $p(C_j|X_k(0))$ falls below a given rejection threshold for all the other R classes.

By definition, points $X_k(0) \in R_{IN\_j}$ are classified as members of class C_j. Here, $R_{IN\_j} \subseteq R_{IN}$ and B_{IN_j} is the boundary of region R_{IN_j} (see Figure 4.4-1 for examples of these). Points outside that boundary are NOT members of class C_j. In mathematical terms, the classification region R_{IN_j} for NDS-j with attractor Γ_j of class C_j can be defined as follows:

$$R_{IN\_j} \subseteq R_{IN} \subseteq R^P \mid \vec{X}_k(0) \in R_{IN\_j} \Leftrightarrow \vec{X}_k(0) \in C_j \qquad \text{(E_4.4-2)}$$

Let T_{jk} be the time required for an initial point $X_k(0)$ to converge to the class-prototype-attractor Γ_j of a continuous-time NDS labelled 'j'. In accordance with the axiom described by E_3.3-1 (page 71), T_{jk} must be related to $P(C_j|X_k(0))$ as follows:

$$\boxed{T_{jk} \propto \frac{1}{P(C_j \mid \vec{X}_k(0))}} \qquad \text{(E_4.4-3)}$$

so that T_{jk} increases as the class membership probability decreases (and vice-versa). As a result of this equation and axiom E_3.3-1, the following relationship must always be true:

$$T_{jk} > T_{rk} \Leftrightarrow P(C_j \mid X_k(0)) < P(C_r \mid X_k(0)) \qquad \text{(E_4.4-4)}$$

In this dissertation, it will often be more convenient to work directly with the probability density function (pdf) $p(C_j|X_k(0))$ rather than the actual probability $P(C_j|X_k(0))$ itself. The two are closely related through this equation [41]:

$$P(C_j \mid \vec{X}_k(0)) \cong \int_{\vec{X}_k(0)-\delta}^{\vec{X}_k(0)+\delta} p(C_j \mid \vec{X}_k(0))d\vec{X} = P(\vec{X}_k(0) - \delta \le \vec{X} \le \vec{X}_k(0) + \delta) \qquad \text{(E_4.4-5)}$$

In effect, the posterior probability of class membership $P(C_j|X_k(0))$ is directly proportional to the value of the pdf at $X_k(0)$: $p(C_j|X_k(0)) = p_{jk}$ so that the two are both representative of the class membership probability. It is therefore justifiable to use the pdf value as an equivalent measure of the actual probability. Hence, equations E_4.4-3 and E_4.4-4 are still valid when $P(C_j|X_k(0))$ is replaced with the pdf $p(C_j|X_k(0))$.

Let us now determine how T_{jk} must be defined at the attractor, as a result of these equations. First, recall that it was shown in section 4.2.1.1 that $\partial X(\Gamma_j)/\partial t = 0.0$ is always true for a NDS where Γ_j is a globally and asymptotically stable attractor. In effect, this says that if $X_k(0) = \Gamma_j$, then $X_k(t) = \Gamma_j$, $\forall t \geq 0$. Hence, if the input to a NDS at time $t = 0$ is $X_k(0) = \Gamma_j$, the time to convergence must be $T_{jk} = 0$. Furthermore, since no input to the NDS is more likely to be a member of class j than the class prototype Γ_j itself, the pdf $p(C_j|X_k(0))$ must have a maximum value p_{j_MAX} at the attractor. Mathematically, these statements can be expressed as follows:

$$\begin{aligned} \vec{X}_k(t=0) = \vec{\Gamma}_j &\Leftrightarrow p(C_j \mid \vec{X}_k(0)) \equiv p_{j\_MAX},\ T_{jk} \equiv 0 \\ \vec{X}_k(t=0) \neq \vec{\Gamma}_j &\Leftrightarrow p(C_j \mid \vec{X}_k(0)) < p_{j\_MAX},\ T_{jk} > 0 \end{aligned} \qquad \textbf{(E_4.4-6)}$$

Let us now define a trajectory c_{jk} as the set of points iterated by the NDS-j from $X_k(0)$ at $t = 0$ to Γ_j at $t = T_{jk}$ as follows:

$$c_{jk} = \{\vec{X}(t)\} \mid \vec{X}(0) = \vec{X}_k(0), \vec{X}(T_{jk}) = \vec{\Gamma}_j, \forall t \in [0, T_{jk}], \vec{X}(t) = \vec{X}(0) + \int_0^{T_{jk}} \frac{\partial \vec{X}_{jk}}{\partial \tau} d\tau \quad \textbf{(E_4.4-7)}$$

where $\partial X_{jk}/\partial t$ is the convergence rate function on trajectory c_{jk} (this is similar to equation E_4.2.1-2 on page 94).

It was argued in section 3.3.1 (page 72) that on any given trajectory c_{jk}, the pdf must be a monotonically decreasing function (this statement is proved in section 4.4.1). In addition, it was noted earlier in this section that points $X_k(0)$ on the decision boundary B_{IN_j} define the limit of the classification region. As a result of these two statements, all points $X_k(0)$

on the boundary B_{IN_j} for all trajectories c_{jk} must correspond to minimum values of the pdf : p_{jk_MIN}. This value is minimum in the sense that all points $X_k(0)$ with $p(C_j|X_k(0)) < p_{jk\_MIN}$ on trajectory c_j are outside the class C_j . But in accordance with equation E_4.4-3 and axiom E_3.3-1 (page 71), the p_{jk_MIN} on any given trajectory must correspond to a maximum 'time to convergence', which we label T^{ϵ}_j. This time is maximum in the sense that any point $X_k(0)$ such that $T_{jk} > T^{\epsilon}_j$ must be outside the class C_j. In mathematical terms, it can therefore be said that points inside, on or outside the boundary B_{IN_j} on any given trajectory have the following attributes:

$$\vec{X}_k(0)\begin{Bmatrix}\notin B_{IN\_j}\\ \in R_{IN\_j}\end{Bmatrix} \Leftrightarrow \begin{cases}\vec{X}_k(0)\in C_j\\ p_{j\_MAX} \le p(C_j \mid \vec{X}_k(0)) < p_{jk\_MIN}\\ 0 \le T_{jk} < T^{\epsilon}_j\end{cases}$$

$$\vec{X}_k(0)\begin{Bmatrix}\in B_{IN\_j}\\ \in R_{IN\_j}\end{Bmatrix} \Leftrightarrow \begin{cases}\vec{X}_k(0)\in C_j\\ p(C_j \mid \vec{X}_k(0)) \equiv p_{jk\_MIN}\\ T_{jk} \equiv T^{\epsilon}_j\end{cases} \qquad \textbf{(E\_4.4-8)}$$

$$\vec{X}_k(0)\begin{Bmatrix}\notin B_{IN\_j}\\ \notin R_{IN\_j}\end{Bmatrix} \Leftrightarrow \begin{cases}\vec{X}_k(0)\notin C_j\\ p(C_j \mid \vec{X}_k(0)) < p_{jk\_MIN}\\ T_{jk} > T^{\epsilon}_j\end{cases}$$

Hence, in the RTA NN model, T^{ϵ}_j is the equivalent of the Bayesian classification boundary and is used to determine class membership (e.g., as shown in the algorithm of Figure 3.3-2 (page 70) with time limit $K = 20$). Figure 4.4-1 illustrates two such boundaries in a case where there are three classes (excluding the unknown class) and three class prototypes at 1.0, 2.5 and 4.0. Note that the trajectory from X_1 to Γ_2 is different from the trajectory X_2 to Γ_2 and that the minimum pdf values p_{jk_MIN} are different for the two trajectories. For a two-dimensional problem, examples of classification boundaries were shown in Figure 3.3.1-2 (page 74).

Let us now determine the constraints on the convergence rate function $\partial X/\partial t$. For a NDS with a globally and asymptotically stable attractor Γ_j, it is known that the initial input

$X_k(0)$ at $t = 0$ must eventually reach Γ_j on trajectory c_{jk} in an amount of time equal to T_{jk}. Hence, the following relationship must be true:

$$\int_{t=0}^{T_{jk}} \frac{\partial \vec{X}_{jk}}{\partial t} dt = -(\vec{X}_k(0) - \vec{\Gamma}_j) \qquad \text{(E_4.4-9)}$$

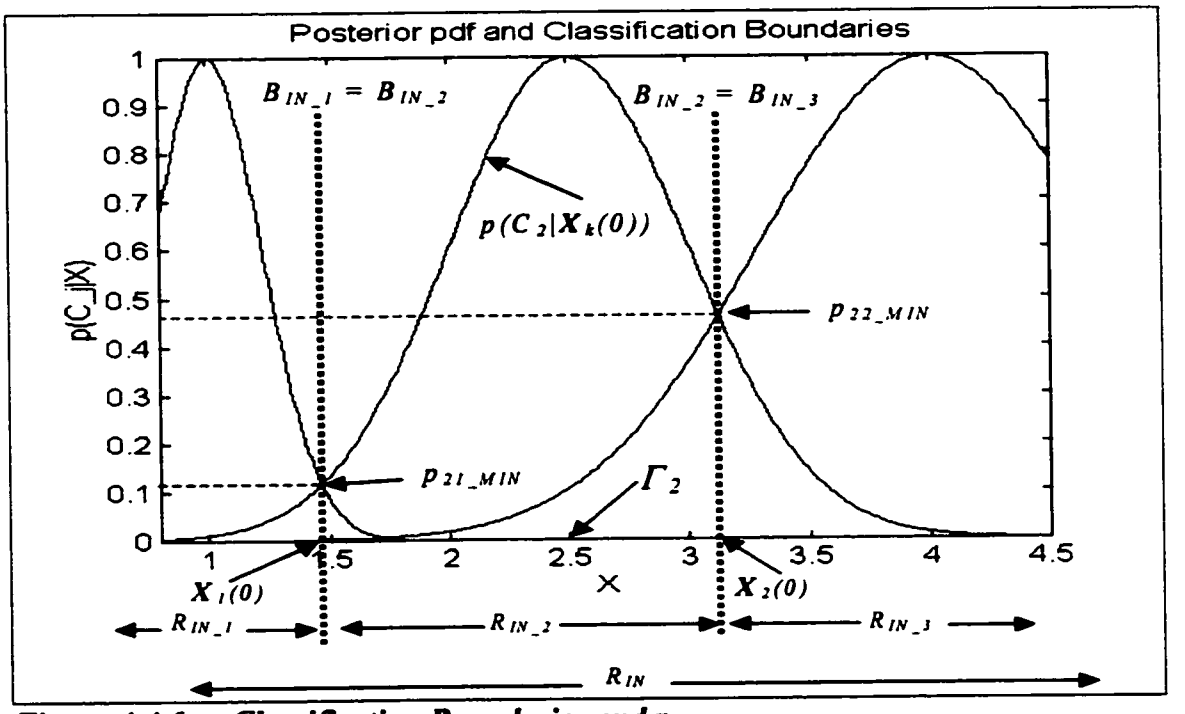


Figure 4.4-1 : Classification Boundaries and p_{jk_MIN}

This is derived simply by evaluating equation E_4.2.1-2 (page 94) over the entire trajectory. The average convergence rate over trajectory c_{jk} is fixed as a result of equation E_4.4-9 and must be[vii]:

$$\tilde{E}\left[\frac{\partial \vec{X}_{jk}}{\partial t}\right] = -\frac{(\vec{X}_k(0) - \vec{\Gamma}_j)}{T_{jk}} \qquad \text{(E_4.4-10)}$$

where $\tilde{E}[]$ represents a time average. Note that these equations do NOT constrain the value of $\partial X/\partial t$ at any given point on the trajectory. However, it will be shown in section

[vii] Note that, as $T_{jk} \rightarrow 0$ and $X_k(t) \rightarrow \Gamma_j$, l'Hôpital's rule [45] can be used to evaluate this expression. This can be done by expresssing both the numerator and denominator as a function of pdf value p_{jk}, and using l'Hôpital's rule as both approach p_{j_MAX}.

4.5 that the pdf $p(C_j|X_k(0))$ does indeed impose a strict relationship between class membership probabilities, convergence rate $\partial X/\partial t$ and time to convergence T_{jk}.
Finally, note that for a discrete-time NDS, T_{jk} is replaced with:

$$N_{jk} = \frac{T_{jk}}{\Delta t} \qquad \textbf{(E\_4.4-11)}$$

where Δt is the sampling period and N_{jk} is the number of iterations to convergence. Most of the equations described in this section remain unchanged. However, equation E_4.4-9 now becomes a discrete sum, as follows:

$$-(\vec{X}_k(0) - \vec{\Gamma}_j) = \sum_{n=0}^{N_{jk}-1} \Delta\vec{H}_{jk}(n) \qquad \textbf{(E\_4.4-12)}$$

where ΔH is as defined in equation E_4.2.2-3 (page 98).

4.4.1 Monotonicity Constraint on $p(C_j|X_k(0))$ for all $X_k(0)$ on a Common Trajectory c_{jk}

By assigning class membership probabilities to points $X_k(0)$ on trajectories c_{jk} as described in the previous section, the constraint of monotonicity is imposed on the pdf itself. This was first mentioned in section 3.3.1 (page 72) where it was argued that points on the same trajectory c_{jk} must have a probability of class membership which decreases monotonically as distance to the attractor increases. In this section, this is demonstrated mathematically.

Consider the one-dimensional trajectory shown in Figure 4.4.1-1. Point $X_2(0)$ is a distance ΔX further away from the attractor Γ_j than point $X_1(0)$. Assume that the time to convergence $T_{j2} = T_{j1} + \beta$, where $\beta > 0$ is a small number.

From equation E_4.4-9, the following relationships can be derived:

$$\int_{t=0}^{T_{j1}} \frac{\partial X_{j1}}{\partial t} dt = \Gamma_j - X_1(0) \qquad \text{(E_4.4.1-1)}$$

and

$$\int_{t=0}^{T_{j1}+\beta} \frac{\partial X_{j2}}{\partial t} dt = \Gamma_j - X_1(0) - \Delta X \qquad \text{(E_4.4.1-2)}$$

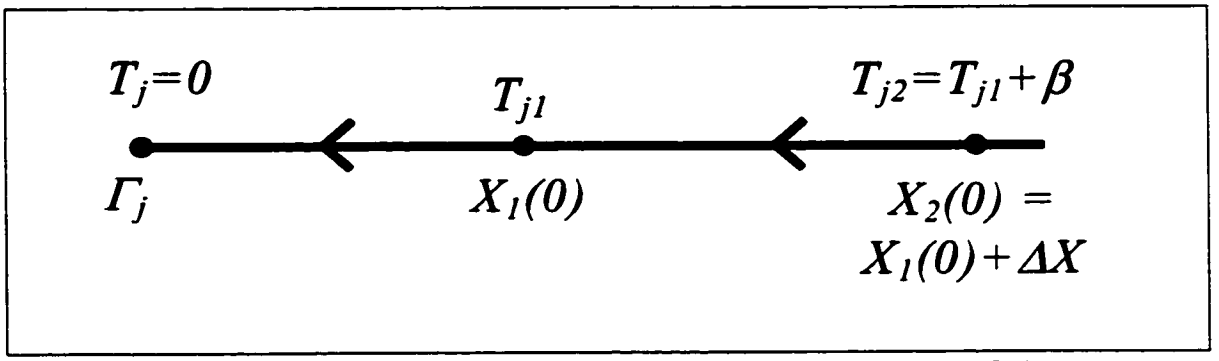


Figure 4.4.1-1 : 1D Example of T_{jk} versus Distance to Γ_j for Different Points on c_{jk}

Since the convergence rate $\partial X_{j2}/\partial t = \partial X_{j1}/\partial t$ between the attractor and point $X_1(0)$, the difference between the above two equations can be expressed as follows:

$$\int_{t=T_{j1}}^{T_{j1}+\beta} \frac{\partial X_{j2}}{\partial t} dt = -\Delta X \qquad \text{(E_4.4.1-3)}$$

In the limit $\beta \rightarrow 0$, the velocity vector $\partial X_{j2}/\partial t$ must increase without bounds since a non-zero distance ΔX must be traversed in an ever-decreasing amount of time. In short, the convergence rate must become a *Dirac delta* unit impulse function [29]. However, this is not normally permitted in a NDS since $\partial X/\partial t$ must be a continuous function over its state space, as discussed in section 4.2. But continuity in $\partial X/\partial t$ implies that time to convergence is also continuous and increases monotonically as distance to the attractor increases. Hence, through axiom E_3.3-1 (page 71) and equation E_4.4-3 (page 106), continuity in $\partial X/\partial t$ implies that $p(C_j|X_k(0))$ is also continuous and decreases monotonically as distance to the attractor increases.

4.4.2 Overcoming the Monotonicity Constraint on $p(C_j|X_k(0))$

It is possible to overcome the constraint that $p(C_j|X_k(0))$ be monotonically decreasing for all points on the same trajectory. This can be done by using a '*one-dimension-up*' approach which is discussed briefly in this section.

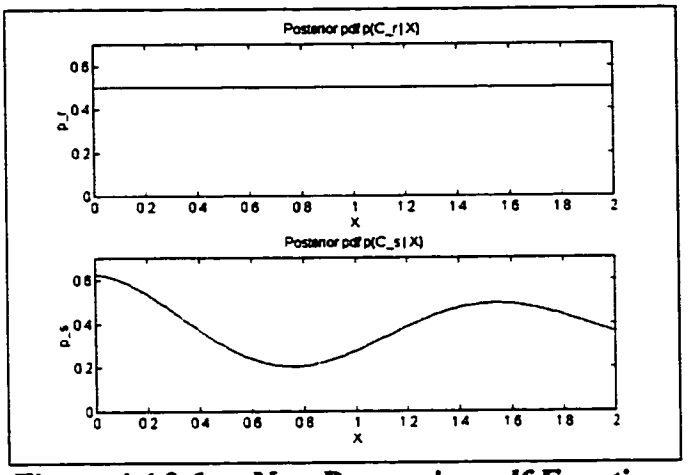


Figure 4.4.2-1 : Non-Decreasing pdf Functions

Consider the examples shown in Figure 4.4.2-1 where two hypothetical pdf which violate the monotonicity constraint discussed earlier are shown. Here, it is not possible to define a NDS with an attractor at 0.0 which can implement these pdf because $\partial X/\partial t$ would need to be discontinuous, as explained in section 4.4.1.

However, there is a way to implement a NDS which can achieve these one-dimensional pdf. This requires that a two-dimensional NDS as shown in Figure 4.4.2-2 be defined. In that NDS, all points $X \in [0,2.0]$ converge to a common *substitute attractor* along unique, non-intersecting trajectories in 2D space.

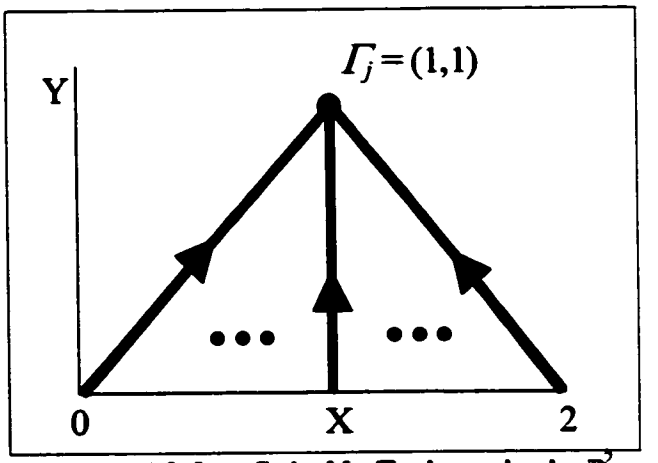


Figure 4.4.2-2 : Suitable Trajectories in $\mathbb{R}^2$

Here, the point $X_k(0) = (0.0, 0.0)$ will converge to the attractor in $T_{jk} = T_{OFFSET}$ instead of in time 0.0. Similarly, every point $X_u(0)$ with $p_{ju} < p_{j\_MAX}$ will converge to the attractor in time $T_{ju} > T_{OFFSET}$, as required by axiom E_3.3-1 (page 71).

With this approach, the 1D pdf shown in Figure 4.4.2-1 can be viewed as being a 2D pdf but where integration over one variable has been performed in accordance with the theory of joint pdf [41]:

$$p(C_j \mid X) = \int_{-\infty}^{\infty} p(C_j \mid X, Y)dY \qquad \text{(E_4.4.2-1)}$$

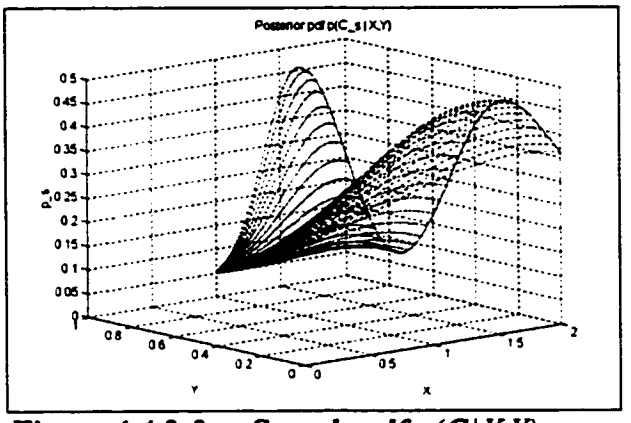


Figure 4.4.2-3 : Sample pdf $p(C_s|X,Y)$

For example, the 2D pdf corresponding to the 1D pdf for class C_s (in Figure 4.4.2-1) might look as shown in Figure 4.4.2-3. Note that in the new dimension Y, the pdf is monotonically decreasing for every trajectory so that the 2D convergence rate function $\partial X/\partial t$ will be continuous throughout the new input space.

Therefore, by adding one degree of freedom to the NDS, it becomes possible to overcome the constraint whereby the pdf must be monotonically decreasing. Hence, any pdf of any arbitrary shape can be modeled by using this approach. The fact that the 'true' attractor X at 0.0 now requires T_{OFFSET} time to converge instead of time = 0.0 does not change anything in the operation of the NDS. This offset can be viewed as just an additional delay before the classification answer is provided by the NDS.

4.5 Relationship between T_j and $p(C_j \mid X_k(0))$ for All Points $X_k(0) \in R_{IN}$

In the previous section, a formal relationship was developed between T_{jk} and $p(C_j \mid X_k(0))$ for the end-points of the pdf. In this section, a formal relationship is developed for the other points on the pdf.

In section 4.5.1, the basic constraints relating T_{jk} and $P(C_j|X_k(0))$ are reviewed. The constraints necessary to ensure that a NDS has globally and asymptotically stable

attractors are also reviewed. Following this, a mathematical relationship between $\partial X_{jk}/\partial t$ and $P(C_j|X_k(0))$ is developed and analyzed in section 4.5.2. As a result of this relationship and the constraints discussed in section 4.5.1, it will become apparent that there are several approaches through which a function $\partial X/\partial t$ can be defined. These approaches will be discussed in sections 4.5.3, 4.5.4 and 4.5.5.

4.5.1 Review of Basic NDS Constraints and Constraints Relating T_{jk} to $p(C_j|X_k(0))$

In section 4.4, constraints relating T_{jk} to $P(C_j|X_k(0))$ were developed and were summarized in equation box E_4.4-8 (page 108). In essence, these constraints enforce specific values of T_{jk} for the attractor Γ_j and for points on the decision boundary as follows:

$$\begin{aligned} T_{jk}(\vec{X}_k(0) = \vec{\Gamma}_j) &= T_{jk}(p = p_{j\_MAX}) = 0.0 \\ T_{jk}(\vec{X}_k(0) \in B_{IN\_j}) &= T_{jk}(p = p_{jk\_MIN}) = T_j^{\epsilon} \end{aligned} \qquad \textbf{(E\_4.5.1-1)}$$

where

$$\begin{aligned} p(C_j \mid \vec{X}_k(0) = \vec{\Gamma}_j) &= p_{j\_MAX} \\ p(C_j \mid \vec{X}_k(0) \in B_{IN\_j}) &= p_{jk\_MIN} \end{aligned} \qquad \textbf{(E\_4.5.1-2)}$$

Although constraints are now assigned to the endpoints of function $T_j(p)$, it is still necessary to assign values of T_j for points of the pdf between these endpoints. These values should be consistent with the fundamental axiom of the RTA NN which states that T_{jk} must be inversely proportional to class membership probability (i.e., axiom E_3.3-1 (page 71) and equation E_4.4-3 on page 106).

However, there are additional constraints to worry about in order to ensure that the NDS modeled by the subnets of the RTA NN are suitable. As indicated in section 3.4 (page 74) of the previous chapter, the subnets of the supernet must ultimately learn the convergence rate function $\partial X/\partial t$ in order to produce the required time to convergence. As explained in section 4.2 (page 93), $\partial X/\partial t$ must be continuous and defined over the entire input space.

Furthermore, since the NDS must have a globally and asymptotically stable attractor, the constraints necessary to achieve this must be met as well.

There is an additional constraint on $\partial X/\partial t$, which is related to the training of neural networks. As will be discussed in section 4.7.1 (page 145), the more complex $\partial X/\partial t$ is, the more difficult it will be to teach a NN to learn it. This is related to the *curse of dimensionality* [4,26] and is explained in section 4.7.1. Complexity is also defined more rigorously in that section. But for now, it suffices to appreciate that the complexity of a function increases as the number of peaks or valleys increases and as the steepness of their slopes increases. Hence, to reduce the learning task complexity, it is desirable to make $\partial X/\partial t$ as simple as possible.

It is therefore apparent that in order to implement the RTA NN correctly, the following constraints should be met:

1. Constraints on the endpoints of $T_j(p)$, as described by equations E_4.5.1-1 and E_4.5.1-2;

2. Constrains based on the RTA NN axiom, as described by equation E_3.3-1 (page 71) or E_4.4-3 (i.e., T_{jk} inversely proportional to class membership probability);

3. Meet the NDS requirements of continuity as well as global and asymptotic stability constraints (as defined in section 4.2, starting on page 93);

4. Minimize the complexity of the convergence rate function $\partial X/\partial t$ in order to make the training of neural nets simpler.

There are basically two ways in which values to $T_{jk}(p)$ can be assigned to input vectors $X_k(0)$:

a. **METHOD 1** : Given a posterior probability pdf $p(C_j|X_k(0))$ define an arbitrary relationship $T_{jk}(p)$ and determine the required $\partial X/\partial t$ to achieve it; or

b. **METHOD 2**: Given a posterior probability pdf $p(C_j|X_k(0))$ define a desirable $\partial X/\partial t$ (e.g., of limited complexity) and determine the required $T_{jk}(p)$ to achieve it.

As will be seen in section 4.5.3, the first approach leads to difficulties in achieving constraints 3 and 4 above, so that the second approach must be considered (this one will be discussed in section 4.5.4). Conversely, the second approach results in constraints 1 and 2 being violated to a certain extent, in some cases. This leads us to consider a compromise solution which is a composite of these two approaches. This method is discussed in section 4.5.5 (i.e., **METHOD 3**).

But before these methods can be discussed, it is necessary to derive a formal relationship between the posterior probability of class membership (represented by the pdf $p(C_j|X_k(0))$) and the convergence rate function $\partial X/\partial t$. This is the subject of the next section.

4.5.2 Mathematical Relationship Between $p(C_j|X_k(0))$ and $\partial X_{jk}/\partial t$

In a typical pattern recognition problem, a training data set is given, from which class-conditional probabilities can be derived. From these, the posterior probability function can be determined via Bayes' rule (equation E_4.3-4, page 102). Hence, the function $p_j(X(0)) = p(C_j|X(0))$ is already defined by the training data itself.

From $p_j(X(0))$ the inverse function $X(p_j) = p^{-1}(C_j|X(0))$ can be determined for each point $X_k(0)$ (our motivation for doing this will soon be made clear).

At some point, a relationship between time to convergence T_j and posterior probability will be defined for all trajectories in the input space (e.g., via methods 1, 2 or 3 as described in sections 4.5.3 to 4.5.5). Let us label this function as $T_j(p_j)$ where

$T_{jk} = T_j(p_j = p_{jk})$ is the time to convergence for a specific point $X_k(0)$ on trajectory c_{jk} with posterior probability measure p_{jk}. Note that $T_j(p_j) = p^{-1}(C_j|T_j)$.

Our motivation for expressing these functions in terms of the values 'p' of the posterior pdf is to exploit a rule of differentiation linking two separate functions which are defined in terms of the same parameter. For functions $x = f(t)$ and $y = g(t)$, it is known that [46]:

$$\frac{dy}{dx} = \frac{dy}{dt} \cdot \frac{dt}{dx} = \frac{dy/dt}{dx/dt} \qquad \textbf{(E\_4.5.2-1)}$$

Hence, with $X(p_j)$ and $T_j(p_j)$ defined in terms of p_j, it can be said that:

$$\boxed{\frac{\partial \vec{X}_j}{\partial T_j} = \frac{\partial \vec{X}/\partial p_j}{\partial T_j/\partial p_j} = \frac{\partial p^{-1}(C_j \mid \vec{X}(0))/\partial p_j}{\partial p^{-1}(C_j \mid T_j)/\partial p_j}} \qquad \textbf{(E\_4.5.2-2)}$$

This equation emerges as a fundamental relationship which clearly defines how the convergence rate $\partial X/\partial t$ of a NDS is related to the posterior probability of class membership 'p' when time to convergence is made a function of that posterior probability. Note that the above equation produces a $\partial X/\partial t$ which is a function of the pdf value 'p'. As explained in section 4.2.1 (page 94), it is more common for an autonomous NDS to express the convergence rate as a function of $X_k(0)$. This is usually easy to do, with a simple substitution of variables.

The significance of the relationship expressed in equation E_4.5.2-2 is explored in the next subsection and also in sections 4.5.3 and 4.5.4.

4.5.2.1 Another Look at the Uncle Brian Example of Section 3.3

In section 3.3, a simple RTA NN model was described using the 'Uncle Brian' example shown in Figure 3.3-1 (page 69). In accordance with axiom E_3.3-1 (page 71), it is

known that T_j increases as the probability of class membership p_j decreases (and vice-versa).

Let us consider a simple example where:

$$\frac{\partial T_j}{\partial p_j} = K_1 \qquad \textbf{(E\_4.5.2.1-1)}$$

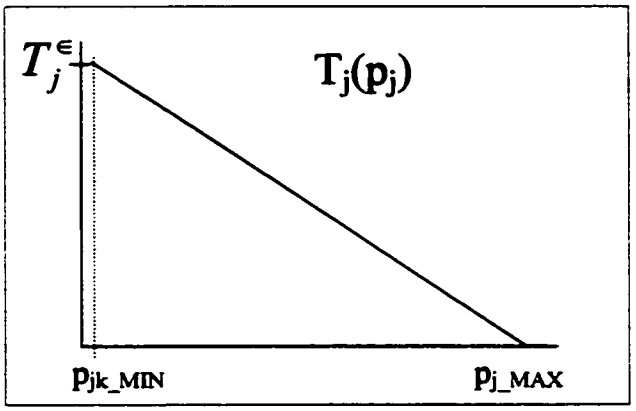


Figure 4.5.2.1-1 : Linear Function $T_j(p_j)$

with K_1 a constant. Hence, $T_j(p_j)$ must be a linear function of the form: $T_j(p_j) = K_1 p + b$. A generic example of such a function is shown in Figure 4.5.2.1-1, for a specific value of p_{jk_MIN} related to a point $X_k(0)$ on the decision boundary B_{IN_j}. As required by axiom E_3.3-1, time to convergence is inversely proportional to the probability of class membership.

With this simple relationship, equation E_4.5.2-2 is reduced to this form:

$$\frac{\partial \vec{X}_{jk}}{\partial t} = K_2 \frac{\partial \vec{X}_{jk}}{\partial p_j} = K_2 \frac{\partial p^{-1}(C_j \mid \vec{X}_k(0))}{\partial p_j} \qquad \textbf{(E\_4.5.2.1-2)}$$

with $K_2 = 1/K_1$. Now, consider three points close to each other and on the same trajectory c_{jk} . These are shown in Figure 4.5.2.1-2 as $X_1 - \delta$, X_1 , and $X_1 + \delta$, with the first point being nearest the attractor Γ_j ($\delta > 0$ is small). This figure illustrates the pdf

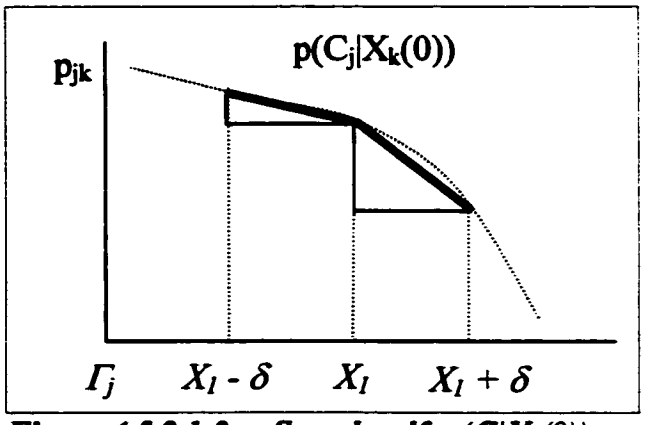


Figure 4.5.2.1-2 : Sample pdf $p(C_j|X_k(0))$

$p(C_j|X_k(0))$ for a segment of the trajectory c_{jk}. The slope $\partial p/\partial X$ between these points is shown approximated as straight line segments. It is assumed that δ is small enough so that these are close to the real values of the slopes.

In accordance with E_3.3-1 (page 71), a large change in p_{jk} from $X_1 + \delta$ to X_1 implies that $P(C_j|X_1(0)+\delta) << P(C_j|X_1(0))$. Even though these two points are roughly the same distance from the attractor, $X_1 + \delta$ is less likely to be in that class than X_1. But $\partial p/\partial X$ large implies that $\partial X/\partial p$ is <u>small</u> because $X(p)$ is the inverse of $p(X)$. Therefore, $\partial X/\partial t$ is <u>small</u>, as defined by equation E_4.5.2.1-2:

> *In other words, when $X_1 + \delta$ is much less likely to be in the class than X_1, the convergence rate between the two is <u>slow</u>.*

Similarly, a small $\partial p/\partial X$ between points X_1 and $X_1 - \delta$ indicates that these two points have roughly the same probability of class membership. Here, a small $\partial p/\partial X$ implies a <u>large</u> $\partial X/\partial p$ so that $\partial X/\partial t$ is also <u>large</u>:

> *In other words, when X_1 and $X_1 - \delta$ have roughly the same probability of class membership, the convergence rate between the two is <u>fast</u>.*

This is consistent with the Uncle Brian analogy shown in Figure 3.3-1 (page 69) where the convergence from point 'A' to 'B' was described as slow in section 3.3 because 'A' is very dissimilar to 'B' and to the attractor at 'C' (i.e., it has a much lower probability of class membership). Hence, the slope of the 'bowl' is shallow from 'A' to 'B'. By contrast, convergence from 'B' to 'C' is faster because the image at 'B' is now more similar to the attractor at 'C' (i.e., both have a roughly similar probability of class membership). Hence, the slope of the bowl from 'B' to 'C' is steep.

Therefore, the intuitive description of the relationship between probability of class membership and convergence rate given in section 3.3 is seen to be consistent with the mathematical theory developed in section 4.5.2. However, note that this consistency is

based on the initial assumption of equation E_4.5.2.1-1. As will be seen in section 4.5.4, some forms of the function $T_j(p_j)$ can actually lead to the exact opposite relationship between convergence rate and class membership probability.

In sections 4.5.3 to 4.5.5, different approaches to define $T(p)$ (and $\partial X/\partial t$) are described. The consequences of using these approaches on the relationship between $\partial X/\partial t$ and $P(C_j|X_k(0))$ (via equation E_4.5.2-2) are then discussed.

4.5.3 METHOD 1 : Defining $\partial X/\partial t$ via an Arbitrary $T(p)$ Function

As mentioned at the beginning of section 4.5.2, the posterior pdf for a typical classification problem can be derived from the training data itself. Suppose that this pdf and the classification boundaries are known (or at least, initial estimates of these are available). One way to proceed in the RTA NN model development is to assign an arbitrary $T_j(p_j)$ function and then determine the required convergence rate $\partial X_{jk}/\partial t$ for all the trajectories c_{jk} of a given NDS i.a.w. equation E_4.5.2-2 (page 117). In this section, this approach is analyzed in order to determine its advantages and disadvantages.

In Figures 4.5.3-1 to 4.5.3-8, various plots related to a NDS-1 are shown, where $\mathbb{R}_{IN\_1} \subset \mathbb{R}^1$ (i.e., one-dimensional). The posterior pdf for NDS-1 is a typical Gaussian [41] defined as follows:

$$p(C_1 \mid X_k(0)) = \begin{cases} p_{1\_MAX} \exp(-X_k^2(0)/3.0), & X \le 0.0 \\ p_{1\_MAX} \exp(-X_k^2(0)/0.4798), & X \ge 0.0 \end{cases} \qquad \textbf{(E_4.5.3-1)}$$

with $p_{1\_MAX} = 0.5$, $\Gamma_1 = 0.0$, and $p_{1k\_MIN} = 0.0622$ as the minimum pdf value associated with the points $X_k(0)$ on the decision boundaries. With this value of p_{jk_MIN}, it can be verified that the two boundary points are $X_k(0) \in B_{IN\_1} = \{-2.5, 1.0\}$. Note that in this particular example p_{jk_MIN} is the <u>same</u> for both points $X_k(0)$ on the two different trajectories. This is <u>NOT</u> necessarily true in general (e.g., see Figure 4.4-1 on page 109).

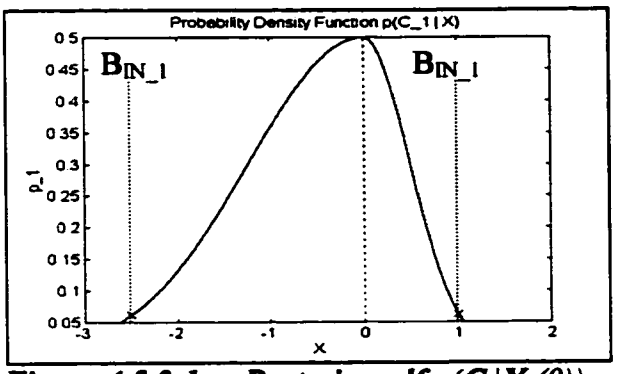


Figure 4.5.3-1 : Posterior pdf $p(C_l|X_k(0))$

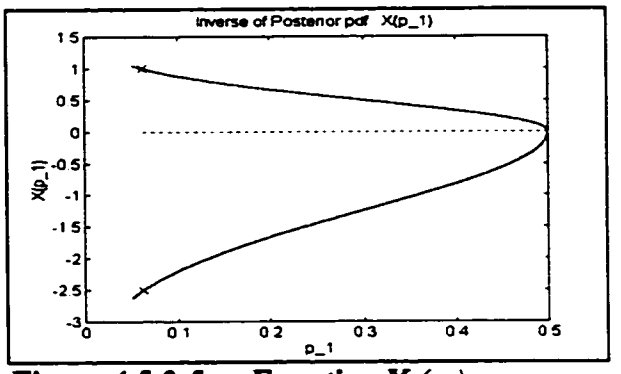


Figure 4.5.3-5 : Function $X_k(p_l)$

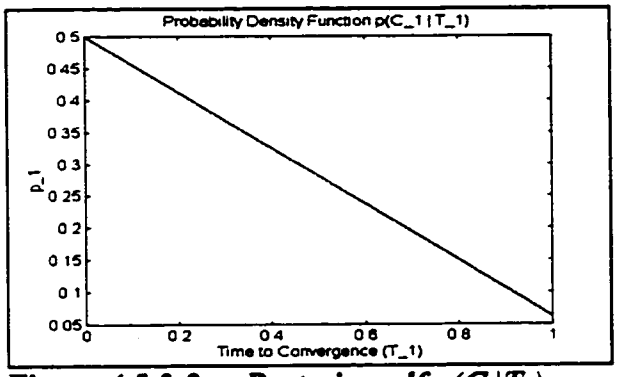


Figure 4.5.3-2 : Posterior pdf $p(C_l|T_l)$

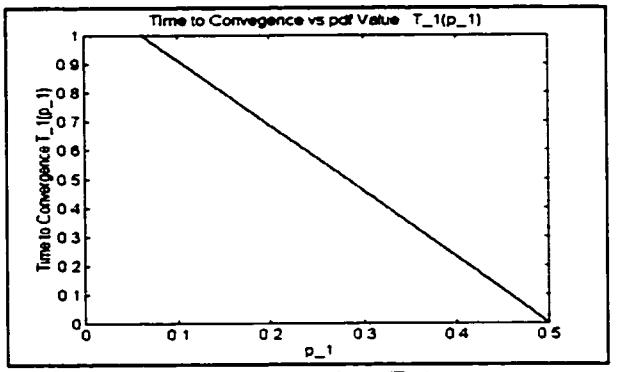


Figure 4.5.3-6 : Function $T_l(p_l)$

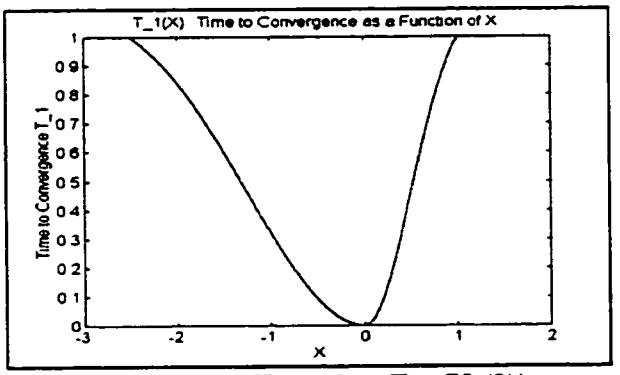


Figure 4.5.3-3 : Function $T_{lk}(X_k(0))$

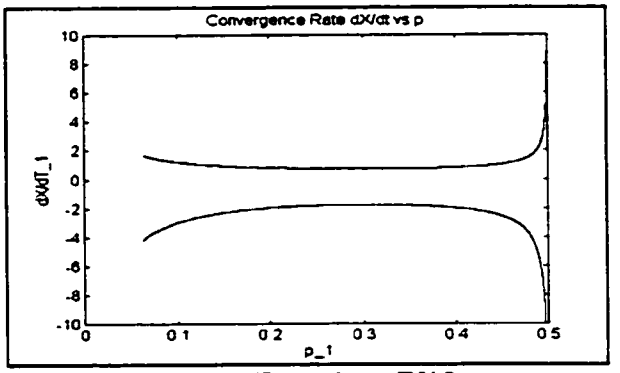


Figure 4.5.3-7 : Function $\partial X/\partial t$ versus p

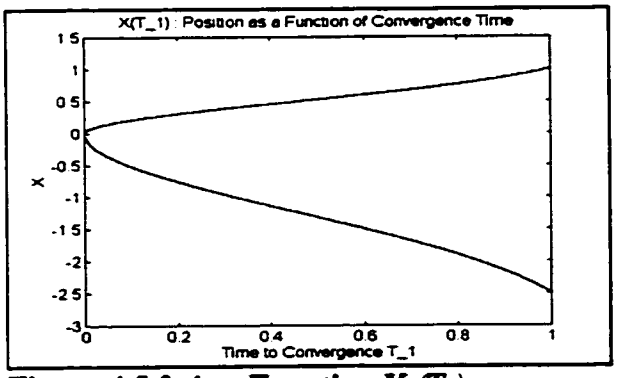


Figure 4.5.3-4 : Function $X_k(T_l)$

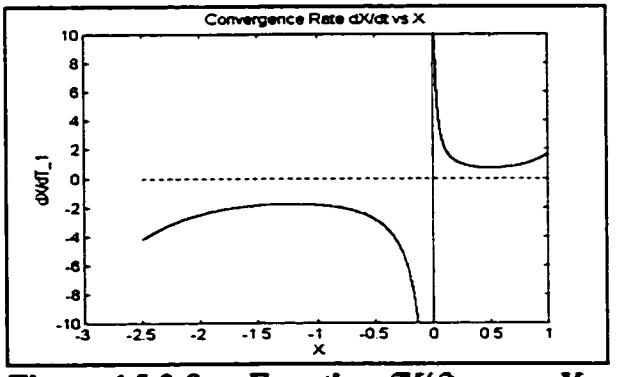


Figure 4.5.3-8 : Function $\partial X/\partial t$ versus X

The pdf for this example is shown in Figure 4.5.3-1[viii]. Figure 4.5.3-5 shows the function $X_k(p_1)$, which is the inverse of the pdf in Figure 4.5.3-1. Note that the inverse is a function on each separate trajectory c_{1k}.

For this example, $T^{\epsilon}_j = 1.0$ is used as the maximum time to convergence for the points on the boundary. The function $T_1(p_1)$ is arbitrarily made linear as follows:

$$T_1(p_1) = m_T(p - p_{1k\_MIN}) + T^{\epsilon}_1 = -2.28(p - 0.0622) + 1 \qquad \textbf{(E\_4.5.3-2)}$$

with slope $m_T = -T^{\epsilon}_1 / (p_{1\_MAX} - p_{1k\_MIN})$. This function is shown in Figure 4.5.3-6. Figure 4.5.3-2 shows the function $p(C_1|T_1)$, which is the inverse of $T_1(p_1)$ and which defines the posterior probabilities of class membership in terms of 'time to convergence'. Here, $p(C_1|T_1) = 1/m_T\ (T - T^{\epsilon}_1) + p_{1k\_MIN}$. Note that $p(C_1|T_1)$ is necessarily a one-sided pdf since $T_{jk} \geq 0$ is always true.

In accordance with equation E_4.5.2-2 (page 117), the derivative of $X_k(p_1)$ (Figure 4.5.3-5) divided by the derivative of $T_1(p_1)$ (Figure 4.5.3-6) produces the convergence rate function $\partial X/\partial t$ (Figure 4.5.3-7). There are two instances of $\partial X/\partial t$ in Figure 4.5.3-7, one for each trajectory of the NDS. Since the NDS is an autonomous system (as defined in section 4.2.1), it is more common to express $\partial X/\partial t$ as a function of X. This latter form is shown in Figure 4.5.3-8.

The function $X_k(T_1)$ shown in Figure 4.5.3-4 is the integral of $\partial X/\partial t$ shown in Figure 4.5.3-8 (the integration is described by equation E_4.2.1-2 on page 94). Figure 4.5.3-3 illustrates the relationship $T_{1k}(X_k(0))$, which is the inverse of $X_k(T_1)$.

A glance at the $\partial X/\partial t$ (Figure 4.5.3-8) which results from the choice of $T_1(p_1)$ (Figure 4.5.3-6) immediately reveals three problems:

[viii] Strictly speaking, this is not a true pdf since the integral over R^1 does NOT equal 1.0 as it should [41]. However, this minor detail can be ignored since it does not affect our ability to describe the RTA NN model in any way.

1. $\partial X/\partial t$ is discontinuous at the attractor $\Gamma_{j1} = 0.0$;

2. The NDS condition $\lim_{X \to \Gamma_1} \partial X / \partial t = 0$ is NOT met; and

3. $\partial X/\partial t$ is complex.

The first two problems are serious since they imply that the NDS defined is not achievable in practice (because of the discontinuity) and does not converge globally and asymptotically to the attractor.

The third problem implies that teaching a NN to learn the equivalent convergence rate function $\Delta H(X(n))$ for the discrete-time NDS will be difficult (complexity is discussed in more detail in section 4.7.1).

In the next subsection, it will be shown that the first two problems can be resolved by redefining the function $T_j(p_j)$ so that the resultant convergence rate function $\partial X/\partial t$ meets the requirements of a NDS with a globally and asymptotically stable attractor.

However, the issue of the complexity of $\partial X/\partial t$ cannot be resolved so easily when $T_j(p_j)$ is defined arbitrarily. Since the complexity of $\partial X/\partial t$ is related to the relationship between the derivatives of two functions via equation E_4.5.2-2 (page 117), no simple rules can be derived to define a $T_j(p_j)$ which will produce a $\partial X/\partial t$ with a desired shape or complexity. An easier way to proceed is to define a 'desired' $\partial X/\partial t$ with a known, limited complexity and then determine the required $T_j(p_j)$ which will produce it. This approach is described in section 4.5.4.

4.5.3.1 Reconciling Valid NDS Constraints and Equation E_4.5.2-2

In order to define a NDS which can be achieved in practice (i.e., can be taught to a NN) and has a unique, globally and asymptotically attractor Γ_j, the following constraints must be met (see section 4.2 (page 93) for more details on these):

1. $\partial X/\partial t$ is continuous over the entire input region $\mathbb{R}_{IN} \subset \mathbb{R}^p$ over which input vectors $X_k(0)$ can occur;

2. $\lim_{\vec{X} \to \vec{\Gamma}_1} \partial \vec{X} / \partial t = 0$;

3. $\nabla \cdot \frac{\partial \vec{X}}{\partial t} < 0$ (i.e., the system is dissipative).

The third condition is derived in part from the Lyapunov function conditions and ensures that all points $X_k(0) \in \mathbb{R}_{IN}$ converge to Γ_j. (see section 4.2.1.3). The third condition is easily met in the development of a RTA NN model. That is because, when training is performed, sample trajectories which converge to the attractor are available or, if only initial pairs $\{X_k(0), C_j\}$ are available, suitable trajectories which always converge to the attractor Γ_j are created (this was discussed in section 3.5.1.2, starting on page 81). Hence, the resultant 'desired' $\partial X/\partial t$ will meet the dissipative requirement.

When constraints 1 and 2 are not met, discontinuities may occur at the attractor. This was seen in the example of the previous section. Examine Figures 4.5.3-5 and 4.5.3-6 again. It can be seen that the problem with equation E_4.5.2-2 (page 117) arises because $\partial X/\partial p_1$ increases without bounds as $p \rightarrow p_{1\_MAX}$ whereas $\partial T_1/\partial p_1$ is constant. Hence, $\partial X/\partial t$ also increases without bounds as p approaches p_{1_MAX} (or as X approaches Γ_1, which is equivalent).

Since $p(C_j|X_k(0) = \Gamma_j) = p_{1\_MAX}$ is always true (from equation E_4.4-6, page 107), and given equation E_4.5.2-2, constraint 2 (above) can be reformulated as follows:

$$\lim_{\vec{X}\to\vec{\Gamma}_j} \partial\vec{X}/\partial t = 0 \Leftrightarrow \lim_{p\to p_{j\_MAX}} \frac{\partial\vec{X}/\partial p_j}{\partial T_j/\partial p_j} = 0 \qquad \textbf{(E_4.5.3.1-1)}$$

Since $p(X) = p(C_j|X_k(0))$ is already defined by the training data, the numerator in the above equation is also fixed and cannot be redefined by the user. However, it is possible to define $T_j(p_j)$ so that $\partial T_j/\partial p_j$ can meet the above constraint. But the requirements on $\partial T_j/\partial p_j$ depend in part on the behavior of $X_j(p_j)$ (and $p(C_j|X_k(0))$) near the attractor. If $\partial X/\partial p_j \to \infty$ as $p \to p_{j\_MAX}$ (as is the case in Figure 4.5.3-5 on page 121), then $\partial T_j/\partial p_j$ must approach ∞ at a faster rate to enforce the constraint E_4.5.3.1-1. Conversely, if $\partial X/\partial p_j \to 0$ as $p \to p_{j\_MAX}$, then $\partial T_j/\partial p_j$ approaching a constant value (as shown in Figure 4.5.3-6) would be sufficient.

What is the significance of altering $\partial T_j/\partial p_j$ is such a way ? An example will help answer that question.

Consider again the example of NDS-1 introduced in the previous section and shown in Figures 4.5.3-1 to 4.5.3-8 (page 121). In light of equation E_4.5.3.1-1, it is clear that $T_j(p_j)$ must be modified in such a way that the derivative $\partial T_j/\partial p_j$ converges to infinity faster than the derivative of $X_j(p_j)$ as $p \to p_{j\_MAX}$. Since $\partial X/\partial p_j = \infty$ implies $\partial p_j/\partial X = 0$ (and similarly for $\partial T_j/\partial p_j$), the problem is equivalent to defining a function $p(C_j|T_j)$ which approaches 0.0 faster than $p(C_j|X_k(0))$ approaches it. Since the pdf $p(C_j|X_k(0))$ is an exponential as described by equation E_4.5.3-1, a function $p(C_j|T_j)$ defined as follows can be shown to be suitable for this task:

$$p(C_j|T_j) = p_{j\_MAX} \exp(-T_j^3/\sigma_T^2), \text{ where } \sigma_T^2 = \frac{-(T_j^{\epsilon})^3}{\ln\left(\frac{p_{jk\_MIN}}{p_{j\_MAX}}\right)} \qquad \textbf{(E_4.5.3.1-2)}$$

L'Hôpital's rule [45] can be used to verify that the ratio of this function and equation E_4.5.3-1 approaches infinity as 0.0 is approached. Hence, equation E_4.5.3.1-1 will converge to 0.0 in the limit, as required. With some manipulation, it is easily shown that $T_j(p_j)$ takes this form:

$$T_j(p_j) = \left(-\sigma_T^2 \ln\left(\frac{p}{p_{j\_MAX}} \right) \right)^{\frac{1}{3}} \qquad \text{(E_4.5.3.1-3)}$$

It is equally easy to demonstrate that $X_j(p_j)$ has a similar form as E_4.5.3.1-3 except that a square-root function is involved instead of a cubic root as above. Here again, L'Hôpital's rule can be used to demonstrate that the ratio of the two in equation E_4.5.3.1-1 approaches 0.0 as $p \rightarrow p_{j\_MAX}$. The results of using such a function $T_2(p_2)$ are shown in Figures 4.5.3.1-1 to 4.5.3.1-8. The NDS has been labeled NDS-2 in these figures to distinguish it from the NDS-1 example of section 4.5.3.

Let us now consider function $p(C_2|T_2)$ shown in Figure 4.5.3.1-2 and compare it to $p(C_1|T_1)$ shown in Figure 4.5.3-2 (page 121). The former is still roughly linear for large values of T_{jk} (i.e., small values of p_{jk}) but it is much less so for small values of T_{jk} (i.e., large values of p_{jk}). What does $\partial p_j / \partial T_j \approx 0$ around p_{j_MAX} imply ? It means, for example, that two points $X_k(0)$, and $X_r(0)$ which have $P(C_j|X_k(0)) \approx P(C_j|X_r(0))$ may actually end up having $T_{jk} << T_{jr}$. **Hence, near the attractor Γ_j time to convergence is no longer truly inversely proportional to probability of class membership and the axiom E_3.3-1 of the RTA NN model has been violated**. However, this is necessary to meet constraints 1 and 2 listed at the beginning of the section.

What is the implication of this necessary compromise ? Since T_{jk} is no longer truly representative of class membership, a confidence level based on T_{jk} may not be an accurate representation of $P(C_j|X_k(0))$. But in practice, this is a relatively minor problem, as long as the nonlinearity is confined to a small region around the attractor, far away from the decision boundary B_{IN_j}.

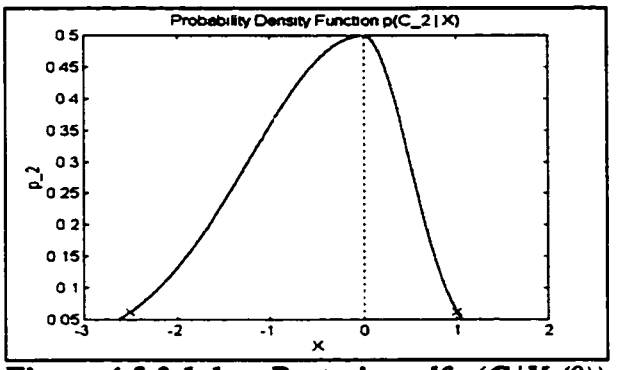


Figure 4.5.3.1-1 : Posterior pdf $p(C_2|X_k(0))$

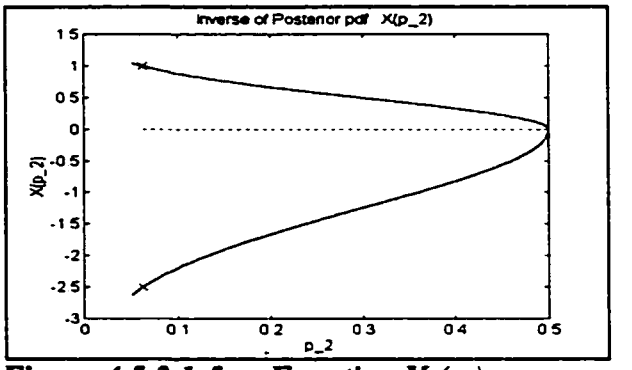


Figure 4.5.3.1-5 : Function $X_k(p_2)$

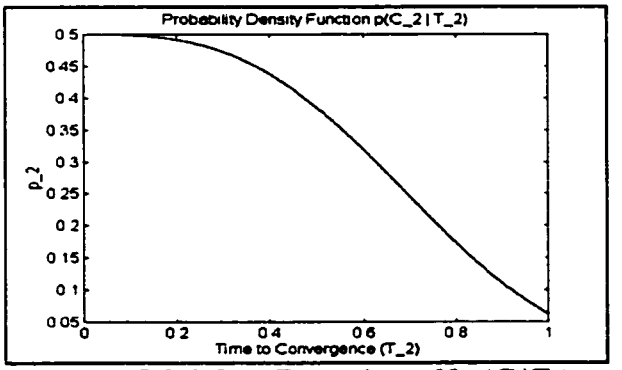


Figure 4.5.3.1-2 : Posterior pdf $p(C_2|T_2)$

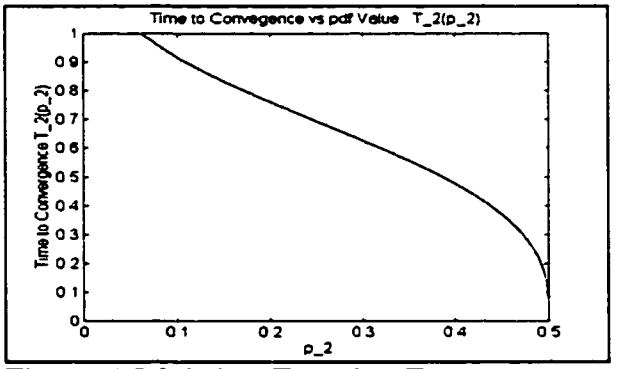


Figure 4.5.3.1-6 : Function $T_2(p_2)$

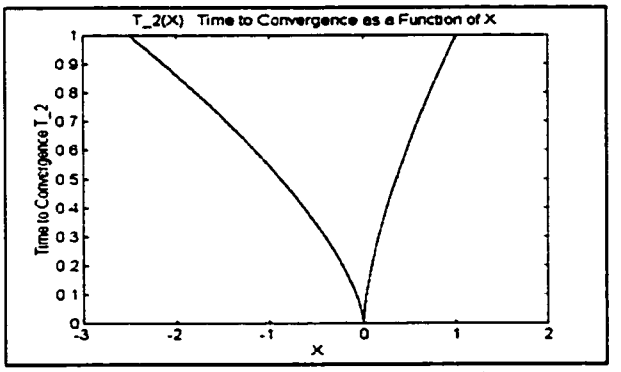


Figure 4.5.3.1-3 : Function $T_{2k}(X_k(0))$

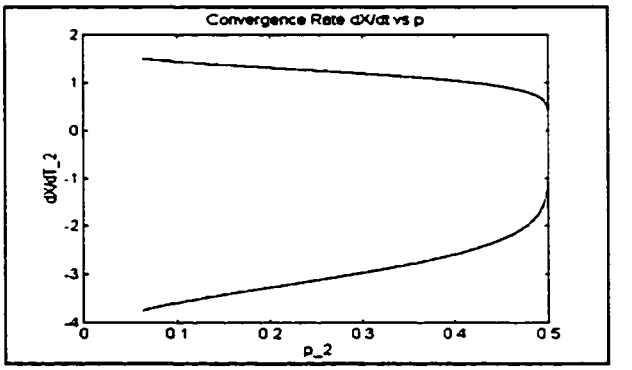


Figure 4.5.3.1-7 : Function $\partial X/\partial t$ versus p

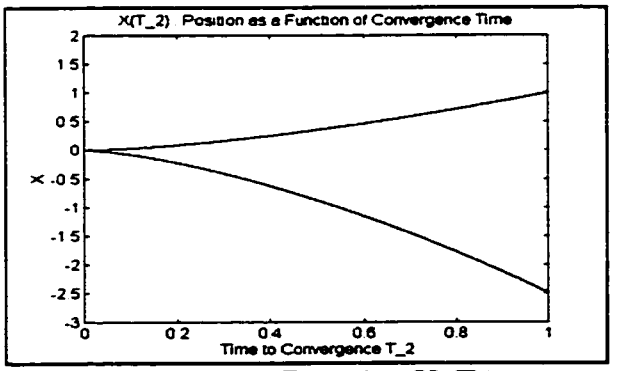


Figure 4.5.3.1-4 : Function $X_k(T_2)$

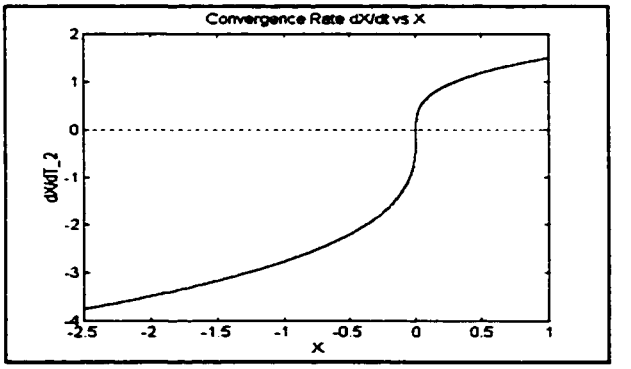


Figure 4.5.3.1-8 : Function $\partial X/\partial t$ versus X

For example, consider two points $X_1(0) = -0.25$ and $X_2(0) = 0.0$ which have $p_1 = 0.49$, $p_2 = 0.5$ and $T_1 = 0.20$ and $T_2 = 0.0$ respectively for NDS-2. Assume that two consecutive races with the RTA NN model are performed. If the T_{2k} of the winners are compared, the results would imply that $P(C_2|X_1(0)) << P(C_2|X_2(0))$ if it was assumed that T_{2k} is inversely proportional to p (and linear) all the way in to the attractor. Although this conclusion would be erroneous, the confidence level computed in each individual race would still be accurate. That is because both X_1 and X_2 are deep within region R_{IN_2}, far away from the boundary point $X = -2.5 \in B_{IN\_2}$. As a result of this, other NDS which participate in the race will necessarily have a $T_{jk} >> T_{2k}$ for both these points. As a result, a confidence level based on the relative T_{jk} of all the NDS within each individual race (or the ratio T_{jk}/T^{ϵ}_{j}) will be an accurate representation of the relative posterior probabilities.

For points farther away from the attractor, $p_j(T_j)$ is almost linear in Figure 4.5.3.1-2, so that a confidence level based on the relative time to convergence of all the NDS within one race will again be an accurate representation of the posterior probabilities.

In practice, it is often more convenient to define an *inner region of convergence* R_{CV_j} where constraint 2 described earlier is enforced on $\partial X/\partial t$ by arbitrarily defining this function as linear and having $\partial X(\Gamma_j)/\partial t \equiv 0$. Then, a suitable T_{jk} (p_j) is defined to meet the constraint of equation E_4.5.3.1-1. The region R_{CV_j} is usually defined so that none of the points on its boundary B_{CV_j} are close to the class-decision boundary B_{IN_j}. In formal terms, this inner region of convergence can be defined as follows:

$$\left.\begin{array}{l} R_{CV\_j} \subset R_{IN\_j} \subseteq R^p \\ \{\vec{X}_k(0)\} \in B_{CV\_j} \\ \{\vec{X}_r(0)\} \in B_{IN\_j} \end{array}\right\} \Rightarrow \begin{cases} d(\vec{X}_k(0), \vec{X}_r(0)) \geq \eta_{CV\_IN\_j}, \forall k, r \\ \eta_{CV\_IN\_j} > 0.0 \end{cases} \qquad \textbf{(E_4.5.3.1-4)}$$

The minimum Euclidean distance $\eta_{CV\_IN\_j}$ between points on the two boundaries B_{CV_j} and B_{IN_j} is made as large as possible to limit the size of the region where axiom E_3.3-1 is violated but small enough to ensure that $\partial X/\partial t$ is not too complex (i.e., it is desirable to

avoid very steep slopes in $\partial X/\partial t$, since these are hard to teach to a NN). An example of such a region is shown in Figure 4.5.3.1-9. This is the $\partial X/\partial t$ of NDS-1 shown in Figure 4.5.3-8 (page 121) but with the convergence rate forced to be linear near the attractor.

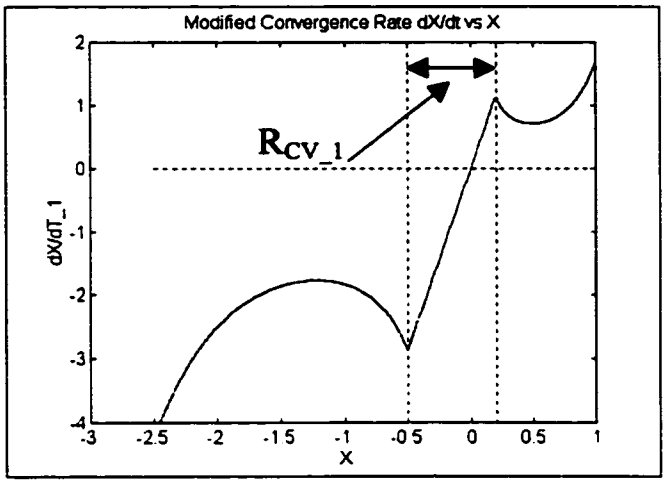


Figure 4.5.3.1-9 : Modified $\partial X/\partial t$ for NDS-1

The advantage of using such a region is that the requirements of a NDS (constraints 1 and 2 described earlier) are met while still maintaining an exactly linear relationship between T_{jk} and p_j beyond the boundary B_{CV_j}. By ensuring the relationship is linear around the critical boundary B_{IN_j}, classification errors are minimized. The use of such regions will be illustrated in chapter 5.

4.5.4 METHOD 2 : Defining $T(p)$ via an Arbitrary $\partial X/\partial t$ Function

As was seen in the previous section, defining an arbitrary $T_j(p_j)$ can result in a $\partial X/\partial t$ which can be a complex function and which may violate the basic constraints of a NDS with a globally and asymptotically stable attractor (i.e., constraints 1 and 2 listed at the beginning of section 4.5.3). Although continuity and convergence to the attractor can be enforced as shown in Figure 4.5.3.1-9 with an inner region of convergence, this does not necessarily reduce the complexity of $\partial X/\partial t$ and may even increase it. As mentioned in section 4.5.1, a complex $\partial X/\partial t$ equates to a difficult training problem for the neural network. It is therefore desirable to minimize complexity.

Another approach to build a suitable NDS model when the posterior pdf $p(C_j|X_k(0))$ is given by the training data is to define a 'desired' $\partial X/\partial t$ and then determine the required relationship $T_j(p_j)$ to implement it. The advantage of this approach is that the complexity of $\partial X/\partial t$ is fixed. This can be a significant benefit when the pdf $p(C_j|X_k(0))$ is such that

$\partial X/\partial t$ would be very complex when 'METHOD 1' of section 4.5.3 is used (e.g., Figure 4.5.3-8 on page 121).

In simple terms, the complexity of a function increases as the number of peaks, valleys, discontinuities, etc, increases and as the slopes of the curves become steeper (see equation E_4.7.1-2 on page 146 for a sample complexity function). Hence, one of the simplest possible convergence rate functions can be shown to be:

$$\frac{\partial \vec{X}_{jk}}{\partial t} = K_{jk} \qquad \textbf{(E_4.5.4-1)}$$

Figures 4.5.4-1 to 4.5.4-8 illustrate an example based on the NDS-1 of section 4.5.3 but where $\partial X/\partial t$ is arbitrarily defined as a constant. This system is referred to as NDS-3 throughout this section. It can be seen from the figures that the convergence rate function is now very simple[ix] relative to that shown in Figure 4.5.3-8 (page 121). However, the relationship between T_j and p_j (or $X_k(0)$) has changed significantly.

By defining an arbitrary $\partial X/\partial t$ as above, equation E_4.5.2-2 (page 117) tells us that the following relationship must be enforced:

$$\frac{\partial T_j}{\partial p_j} = \frac{\partial \vec{X} / \partial p_j}{K_j} = \frac{\partial p^{-1}(C_j \mid \vec{X}) / \partial p_j}{K_j} \qquad \textbf{(E_4.5.4-2)}$$

Hence, the function $T_j(p_j)$ must have this form:

$$T_j(p_j) = \frac{1}{K_j}\left(p^{-1}(C_j \mid \vec{X}) + K_2\right) \qquad \textbf{(E_4.5.4-3)}$$

[ix] To simplify the example, we have neglected to include a 'region of convergence' R_{CV_3} (as defined in section 4.5.3.1). In reality, such a region would be required (e.g., as shown in Figure 4.5.3.1-9) where $\partial X/\partial t$ would have to be at least a linear function, thereby increasing the minimum complexity to a certain extent.

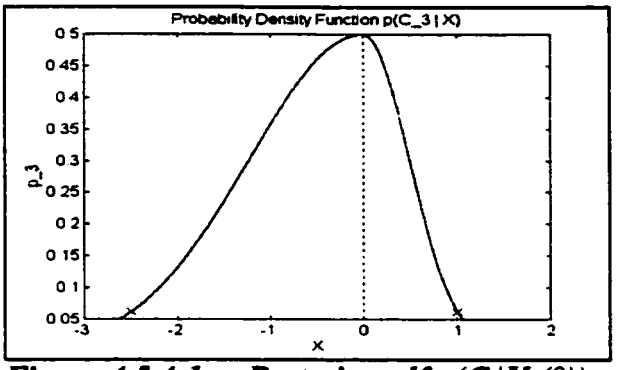


Figure 4.5.4-1 : Posterior pdf $p(C_3|X_k(0))$

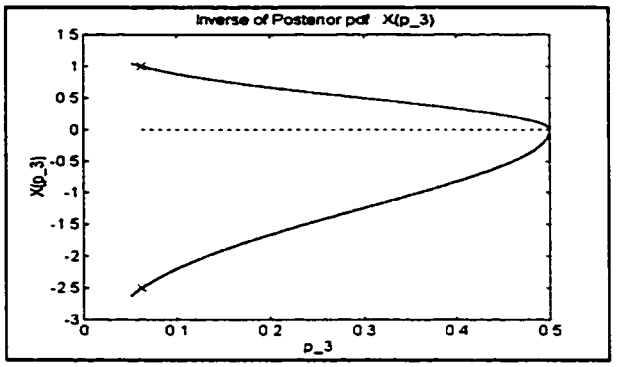


Figure 4.5.4-5 : Function $X_k(p_3)$

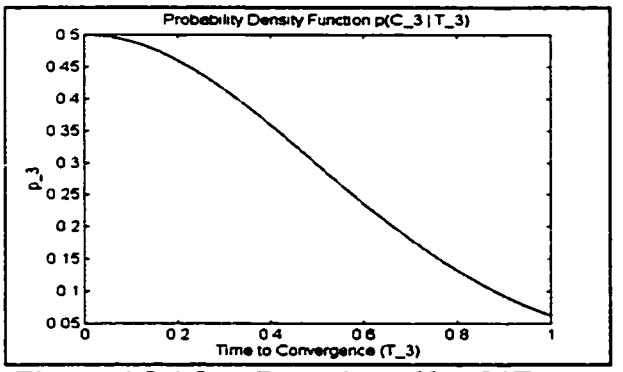


Figure 4.5.4-2 : Posterior pdf $p(C_3|T_3)$

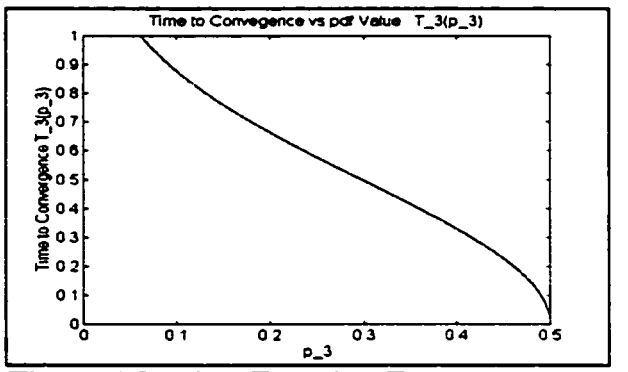


Figure 4.5.4-6 : Function $T_3(p_3)$

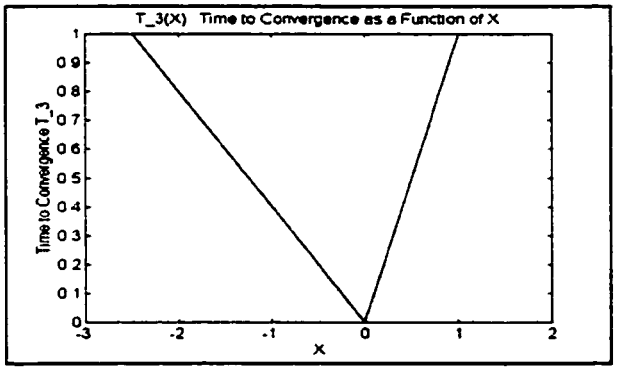


Figure 4.5.4-3 : Function $T_{3k}(X_k(0))$

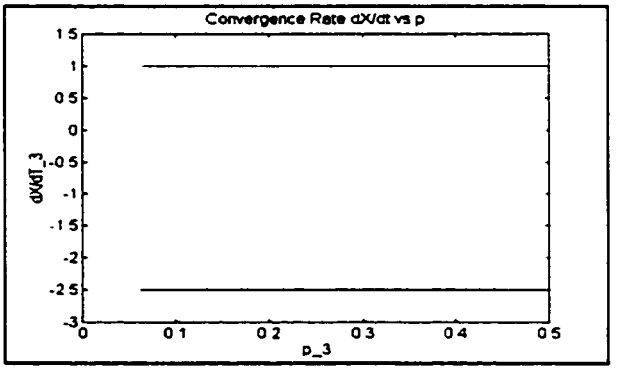


Figure 4.5.4-7 : Function $\partial X/\partial t$ versus p

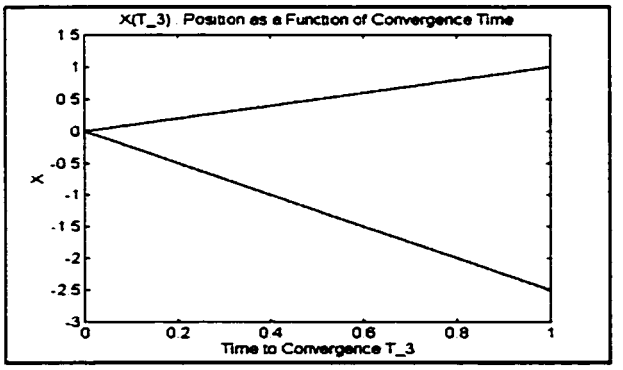


Figure 4.5.4-4 : Function $X_k(T_3)$

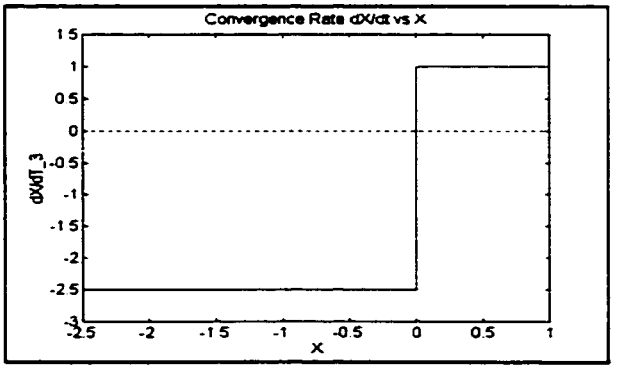


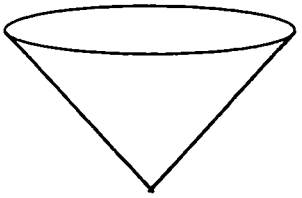
Figure 4.5.4-8 : Function $\partial X/\partial t$ versus X

What is the significance of such a relationship ? Consider again Figure 4.5.2.1-2 on page 118. A small $\partial p/\partial X$ implies that points $X_1 - \delta$ and X_1 have roughly the same probability of class membership even though X_1 is further away from the attractor than $X_1 - \delta$. A small $\partial p/\partial X$ also implies a large $\partial X/\partial p$ so that $\partial T/\partial p$ given by equation E_4.5.4-2 is also large.

But this means that two points with roughly the same probability of class membership will have a very different time to convergence (e.g., $T(X_1 - \delta) << T(X_1)$ even though $P(C_j|X_1(0) - \delta) \approx P(C_j|X_1(0))$). Conversely, a large $\partial p/\partial X$ implies that points $X_1 + \delta$ and X_1 have very different probabilities of class membership despite the fact that they are roughly the same distance to the attractor. Hence, $\partial X/\partial p$ is small and $\partial T/\partial p$ is small so that $T(X_1 + \delta) \approx T(X_1)$ even though $P(C_j|X_1(0) + \delta) << P(C_j|X_1(0))$). **This relationship is exactly the opposite of what is described by axiom E_3.3-1** (page 71).

Note that a similar problem was found to exist for the NDS-2 example of section 4.5.3.1. However, in that case, this relationship was only true deep into the classification region, at values close to p_{j_MAX} (i.e., in the inner convergence region R_{CV_j}). But in the example of NDS-3 discussed here, this relationship is applicable everywhere in the input domain, including the critically important decision boundary B_{IN_3}. Hence, points on either side of that boundary with very different probabilities of class membership could end up having very similar values of T_{jk} , thereby increasing the probability of misclassification errors. Furthermore, the time to convergence can no longer be used as a reliable measure of classification confidence level because it is no longer directly related to the probability of class membership.

By imposing an arbitrary function $\partial X_{jk}/\partial t = K_{jk}$ irrespective of class membership probabilities, $X(T)$ is constrained to be a linear function as shown in Figure 4.5.4-4. But since $X(T)$ is fixed, so is $T(X)$ shown in Figure 4.5.4-3. As a result, $\partial X/\partial t$ is independent of the class membership posterior pdf. The only way to re-establish a dependence between the two is to define $T_j(p_j)$ as shown in equation E_4.5.4-3, in order to enforce the relationship required by equation E_4.5.2-2 (page 117).



In terms of our Uncle Brian example of section 3.3 (page 69), a constant $\partial X/\partial t$ as expressed by equation E_4.5.4-1 implies that our 'bowls' are all shaped as 'right circular cones' as shown here. In such a NDS, all initial points which are the same distance from the attractor will converge to the bottom in exactly the same amount of time, regardless of whether they have the same likelihood of class membership. Here, the time to convergence is inversely proportional to the Euclidean distance to the attractor (and to nothing else).

Although in some cases $P(C_j|X_k(0))$ is a function of the distance from the attractor, this is not necessarily true in general (this was discussed in section 3.3.1, starting on page 72).

However, having T_{jk} as a function of X may be acceptable in some cases. If the $\partial X/\partial t$ derived from a linear $T(p)$ is too complex in certain regions of the input space, it may be preferable to arbitrarily simplify $\partial X/\partial t$ and accept some degradation in the linear relationship of T_{jk} versus p_{jk}. For example, such a compromise may be acceptable deep inside the region of classification (i.e., far away from the decision boundary). This was demonstrated in section 4.5.3.1 with the NDS-2 example, where it was shown that the lack of linearity between T_{jk} versus p_{jk}. near the attractor (i.e., near p_{j_MAX}) had little impact on our ability to perform reliable classifications.

4.5.5 METHOD 3 : Defining $T(p)$ and $\partial X/\partial t$ with a Cost Function

This section discusses briefly the process of combining methods 1 and 2 from sections 4.5.3 and 4.5.4 in order to achieve advantages of both approaches and minimize their respective disadvantages.

As discussed in section 4.5.3, defining an arbitrary function $T(p)$ may result in a complex function $\partial X/\partial t$ which is difficult to teach to a NN. It may also result in $\partial X/\partial t$ being

unsuitable for a NDS with a globally and asymptotically attractor. However, it allows us to compute a reliable classification confidence measure everywhere in the input space.

On the other hand, defining an arbitrary $\partial X/\partial t$ may allow us to limit the complexity of that function and to meet NDS constraints. But it can distort the relationship between time to convergence and probability of class membership to such an extent that classification confidence levels can no longer be assigned reliably.

By combining the two approaches, it is possible to benefit from their respective advantages and limit their disadvantages. An example when such an approach makes sense has already been illustrated in Figure 4.5.3.1-9 (page 129). In that example, a linear $T(p)$ is preserved beyond the inner region of convergence R_{CV_I} so that a reliable classification confidence measure can be assigned near the classification boundary (where it is most important). Inside region R_{CV_I}, an arbitrary linear convergence rate $\partial X/\partial t$ has been enforced so that the requirements of a NDS with a globally stable attractor were met and the complexity of the function $\partial X/\partial t$ was reduced to a certain extent. Although time to convergence near the attractor was no longer an accurate measure of the posterior probability of class membership in that example, this was shown to be a minor problem.

There are potentially many cases when such a combined method can be advantageous. In these instances, the problem of finding the optimal $\partial X/\partial t$ and $T(p)$ can be expressed as a classical nonlinear minimization problem [59], with a cost function having a form similar to this:

$$E(s,d) = L_{PDF\_ERROR}\, d(\hat{p}, p) + L_{CVRG\_CMPX}\, s(\partial \hat{X} / \partial t) \qquad \textbf{(E\_4.5.5-1)}$$

Here, $E(s,d)$ is the cost function to be minimized, $d(\hat{p}, p)$ is a distance measure between the desired posterior pdf $p(C_j|T_j)$ and the achieved pdf (note that these are simply the inverse of $T(p)$). The function $s(\partial X/\partial t)$ assigns a numerical value to the complexity of the convergence rate function (equation E_4.7.1-2 on page 146 is an example of such a

function). The parameters 'L' are weighting parameters, which may be functions of p and X. For example, the weight L_{PDF_ERROR} can be maximum near p_{jk_MIN} (i.e., at the decision boundary) and minimum near p_{j_MAX} (i.e., near the attractor) to reflect the relative importance of having T_{jk} related to p_{jk} linearly in these regions.

In a situation where $p(C_j|X_k(0))$ is known and where estimates of the classification boundaries are known, defining suitable functions $T(p)$ and $\partial X/\partial t$ could proceed as follows:

1. Define a desirable $T(p)$ and determine the function $\partial X/\partial t$ which results from this choice.

2. Calculate the complexity of $\partial X/\partial t$ and determine regions where it is desirable/feasible to reduce it.

3. In these regions, define arbitrary segments of $\partial X/\partial t$ with a desired complexity level and recalculate the resultant $T(p)$.

4. Compute the $d(\hat{p}, p)$ for the new $\partial X/\partial t$ and assess the overall cost.

5. Iterate steps 1 to 4 as required until an acceptable cost measure is achieved.

This is an iterative approach to minimizing the error which is similar in many respects to the training of neural networks with backpropagation training.

However, note that in order to limit the scope of this dissertation, method 3 is not analyzed any further.

4.6 Mathematical Formulation of the RTA NN Model Paradigm

A mathematical model for the RTA NN is developed in section 4.6.1. An example classification problem is described in section 4.6.2 in order to help illustrate how the model operates in practice.

4.6.1 Mathematical Representation of the RTA NN Model

The architecture of the RTA NN is illustrated in Figure 4.6.1-1. Refer again to section 3.4 (page 74) from the previous chapter for a description of the various components of the model.

Let $X_k(0) \in \mathbb{R}_{IN} \subset \mathbb{R}^P$, $X_k(0) \in C_{IN}$, where P is the dimension of the input space. The set $\{X_k(0) \in C_{IN} \mid k = 1,2,...\}$ is a (possibly) infinite set of vectors which must be classified into one of $R + 1$ classes C_j which span the entire input space C_{IN}:

$$\bigcup_{j=1}^{R+1} C_j = C_{IN} \qquad \text{(E_4.6.1-1)}$$

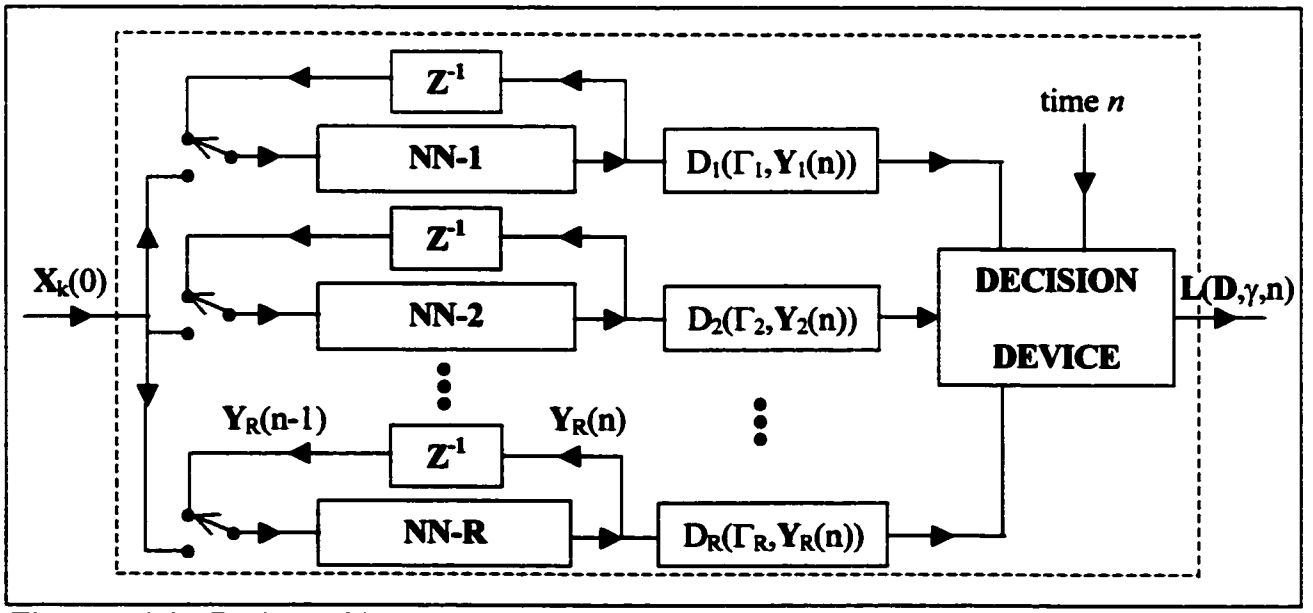


Figure 4.6-1 : Basic Architecture of the Race To the Attractor Neural Network Model

By convention, class C_{R+1} will normally be considered as the "unknown" class, to which vectors $X_k(0)$ are relegated if they do not meet the classification criteria of any of the other R classes.

$X_k(0)$ is the unknown input vector to the supernet S_R consisting of $j = 1$ to R subnets, each one having a unique, globally and asymptotically stable attractor point Γ_j in the region $\mathbb{R}_{IN}$. The attractor Γ_j is the prototype of the class C_j. Each subnet NN-j models a nonlinear dynamical system whose evolution is defined by:

$$\vec{Y}_{jk}(n) = \vec{H}_j\left(\vec{Y}_{jk}(n-1)\right), n \geq 1 \qquad \text{(E_4.6.1-2)}$$

$H_j(\ldots)$ is the iterated mapping function which defines the evolution of the NDS modeled by subnet NN-j. The above equation is also equivalent to this relationship:

$$\vec{Y}_{jk}(n) = \vec{Y}_{jk}(n-1) + \Delta\vec{H}_j\left(\vec{Y}_{jk}(n-1)\right), n \geq 1 \qquad \text{(E_4.6.1-3)}$$

ΔH and H were defined earlier in section 4.2.2, starting on page 98. $Y_{jk}(n)$ is the output of the system at time n, when its initial input at time $n = 0$ was $Y_j(n=0) = X_k(0)$. $Y_{jk}(n)$, $H_j(\ldots)$, Γ_j and $X_k(0)$ have the same dimension P. Note also that <u>all</u> initial values of $X_k(0) \in \mathbb{R}_{IN} \subset \mathbb{R}^P$ will eventually converge to the attractor Γ_j for <u>all</u> subnets since these attractors are globally and asymptotically stable for each NDS-j.

As time n increases, a time $n = N_{jk}$ is eventually reached when:

$$D_j\left(\vec{\Gamma}_j, \vec{Y}_{jk}(n = N_{jk})\right) < \gamma_j, N_{jk} \geq 0 \qquad \text{(E_4.6.1-4)}$$

where D_j is the distance function, defined as follows:

$$D_j(\vec{\Gamma}_j, \vec{Y}_j(n)) \mid \vec{\Gamma}_j, \vec{Y}_j(n) \in R^P, D_j(\cdots): R^P \rightarrow R^1, D_j(\cdots) \geq 0.0 \qquad \text{(E_4.6.1-5)}$$

and where $\gamma_j > 0$ is a user-defined threshold indicating how close $Y_{jk}(n)$ must get to the attractor Γ_j before the approximation $Y_{jk}(n) \approx \Gamma_j = H_j(\Gamma_j)$ is considered valid. The function D_j could be the Euclidean [46] distance (for example) but other forms are permissible.

A trajectory c_{jk} is defined as the sequence of points which are iterated by NDS-j from time $n = 0$ (when $Y_{jk}(0) = X_k(0)$) until time $n = N_{jk}$ (when $Y_{jk}(N_{jk}) \approx \Gamma_j$, as determined through equation E_4.6.1-4):

$$c_{jk} = \{\vec{Y}_{jk}(n)\} \mid n \in [0, N_{jk}], \vec{Y}_{jk}(n) = \vec{H}_j(\vec{Y}_{jk}(n-1)), \vec{Y}_{jk}(0) = \vec{X}_k(0) \qquad \textbf{(E_4.6.1-6)}$$

Hence, c_{jk} is the sequence of points $\{Y_{jk}(0) = X_k(0),\ Y_{jk}(1), \ldots, Y_{jk}(N_{jk}) \approx \Gamma_j\}$. Note that this is the discrete-time equivalent of equation E_4.4-7 (page 107) for a continuous-time NDS.

To implement the RTA NN model, it is also necessary to define a 'race membership function' $M_{RACE}(j)$, which indicates whether subnet j is still in the race or whether it has been disqualified:

$$M_{RACE}(j) = \begin{cases} 0, \textit{subnet } j \textit{ is NOT in the race} \\ 1, \textit{subnet } j \textit{ is in the race} \end{cases}$$

$$N_{jk} > N_j^{\epsilon} \Leftrightarrow M_{RACE}(j) = 0 \qquad \textbf{(E_4.6.1-7)}$$

$$N_{jk} \le N_j^{\epsilon} \Leftrightarrow M_{RACE}(j) = 1$$

The race membership function is used to ensure that the desired classification boundaries N^{ϵ}_j are enforced. In effect, a subnet-j being disqualified from the race is equivalent to saying that NDS-j has classified the input as OUT of class $\mathbb{C}_j$ (i.e., unknown to that specific NDS).

An input vector $X_k(0)$ will be classified as a member of class $\mathbb{C}_j$ in accordance with the following criteria:

$$\vec{X}_k(0) \in C_j \Leftrightarrow \begin{cases} N_{jk} < N_{rk} \forall r \neq j, r \in [1, R] \\ and\ M_{RACE}(j) = M_{RACE}(r) = 1 \end{cases} \qquad \textbf{(E\_4.6.1-8)}$$
$$\vec{X}_k(0) \in C_{R+1} \Leftrightarrow M_{RACE}(j) = 0, \forall j \in [1, R]$$

The entire set of points for which the statement $X_k(0) \in C_j$ is true defines the classification region R_{IN_j}:

$$R_{IN\_j} = \{\vec{X}_k(0), k = 1,2,...\} \mid \vec{X}_k(0) \in C_j \qquad \textbf{(E\_4.6.1-9)}$$
$$R_{IN\_j} \subseteq R_{IN} \subseteq R^P$$

Finally, because N_{jk} is inversely proportional to the posterior probability of class membership $P(C_j|X_k(0))$ (as required by the axiom E_3.3-1 (page 71) and equation E_4.4-3 on page 106), the following statement is always true for one 'race' of the model:

$$N_{jk} > N_{rk} \Leftrightarrow P(C_j \mid \vec{X}_k(0)) < P(C_r \mid \vec{X}_k(0)) \qquad \textbf{(E\_4.6.1-10)}$$

This is the discrete-time equivalent of equation E_4.4-4 (page 106).

4.6.1.1 Mathematical Model for the Decision Device

The decision device in Figure 4.6-1 determines the winner of the race to the attractor (i.e., it classifies the input vector $X_k(0)$ as one of the object classes). The decision is based on the number of iterations to convergence N_{jk} (i.e., the first NN to get within a certain distance γ_j of the attractor Γ_j). If the time to convergence is too long (i.e., $X_k(0)$ has a low probability $P(C_j|X_k(0))$ of being in <u>any</u> of the available R classes), the input is classified as "unknown".

The operation of the decision device $L(\vec{D}, \vec{\gamma}, n)$ shown in Figure 4.6-1 can be modeled as follows:

$$L(\vec{D}, \vec{\gamma}, n) = \vec{A} \times \vec{g}(n) = \sum_{i=1}^{R+1} \alpha_i g_i(n) \qquad \textbf{(E\_4.6.1.1-1)}$$

where $A = [\alpha_1, \alpha_2, ..., \alpha_{R+1}]$ are constant, unique, non-zero values and where $g(n) = [g_1(n), ..., g_{R+1}(n)]^T$ is the vector of orthogonal generating functions required to achieve the mapping approximation as a *Karhunen-Loève* series expansion [41] of the form $\hat{f}(X) = \sum_{i=1}^{R+1} \alpha_i c_i(X) \approx f(X)$. Here, $f(X)$ is the desired mapping function which classifies all inputs $X(0) \in \mathbb{R}_{IN}$ as one of the $R+1$ classes. Note that in the function $L(D, \gamma, n)$, D is the vector of distance functions $\{D_1(...), ... D_R(...)\}$ and γ is the vector of threshold values $\{\gamma_1, ..., \gamma_R\}$.

Independence (and orthogonality) of the generating functions learned by the various NN will be achieved by ensuring that only one subnet wins the race for any input vector or that none of the subnets win the race when the input is unknown. This can be achieved by enforcing the following rules:

$$g_j(n) = \begin{cases} 1 \Leftrightarrow D_j(\Gamma_j, Y_j(n)) \le \gamma_j, M_{RACE}(j) = 1, \text{and} \\ \qquad D_r(\Gamma_r, Y_r(n)) > \gamma_r, M_{RACE}(r) = 1, \forall r \ne j \\ 0, \text{otherwise} \end{cases}, \forall r, j \in [1, R]$$

$$g_{R+1}(n) = \begin{cases} 1 \Leftrightarrow M_{RACE}(j) = 0, \forall j \in [1, R] \\ 0, \text{otherwise} \end{cases} \qquad \textbf{(E_4.6.1.1-2)}$$

Initially, at time $n = 0$, $\vec{g}(n) = \vec{0}$. As time progresses, one (and only one) of the generating functions will assume a value > 0, thereby indicating that the race is over. This can occur as early as $n = 1$, when $X_k(0) = \Gamma_j$ (it is assumed that at least one iteration is required to propagate the input of the NDS to the output). Note that the maximum amount of time required to classify an input will always correspond to an 'unknown' classification since this requires that all subnets classify the input as unknown (i.e., all subnets become disqualified as $N_{jk} > N^{\epsilon}_j$, $\forall j \in [1,R]$). Hence, the maximum classification time n_{MAX} is given by:

$$n_{MAX} = \max\{N^{\epsilon}_1, ..., N^{\epsilon}_R\} + 1 \qquad \textbf{(E_4.6.1.1-3)}$$

Once the race is over, the input $X_k(0)$ is classified and the subnets can be reset for the next input vector to classify.

Finally, note that the decision device can also be used to assign a *confidence level*, based on the number of iterations to convergence. For example, comparing the ratio N_{jk}/N^{ϵ}_{j} for all the subnets $j = 1$ to R will indicate if the initial point $X_k(0)$, was close to one or more boundaries. In such a case, the ratio will be close to 1.0 for two or more NN. If this occurs, a low confidence level can be assigned to the classification. Conversely, if the input is close to an attractor, the ratio will be close to 0.0 (since N_{jk} is close to 0) for that NN and will likely be high for the other NN (e.g., greater than 1.0) so that a high confidence level can be assigned to the classification decision. Many such functions could be defined but in order to limit the scope of this dissertation, this subject will not be pursued any further.

4.6.2 One-dimensional Example of a RTA NN Classification Solution

A one-dimensional example of a RTA NN model is shown in Figure 4.6.2-1. Here, there are three basic classes C_1 (with $\Gamma_1 = 2$),C_2 (with $\Gamma_2 = 4$), C_3 (with $\Gamma_3 = 6$) and the 'unknown' class C_4. The vertical lines in the top of the figure show the desired class boundaries and/or rejection thresholds. The corresponding values of the pdf define the minimum posterior probability for membership in the class on the different trajectories. Note that there is one boundary between classes C_1 and C_2 and rejection thresholds (as defined in section 4.3.2) between classes C_2 and C_3. As a result, inputs $X_k(0)$ will be classified as follows:

$$\begin{aligned} &a < X_k(0) < b \Leftrightarrow X_k(0) \in C_1 \\ &b \le X_k(0) < c \Leftrightarrow X_k(0) \in C_2 \\ &d < X_k(0) < e \Leftrightarrow X_k(0) \in C_3 \\ &X_k(0) \le a, X_k(0) \ge e, c \le X_k(0) \le d \Leftrightarrow X_k(0) \in C_4 \end{aligned} \qquad \textbf{(E_4.6.2-1)}$$

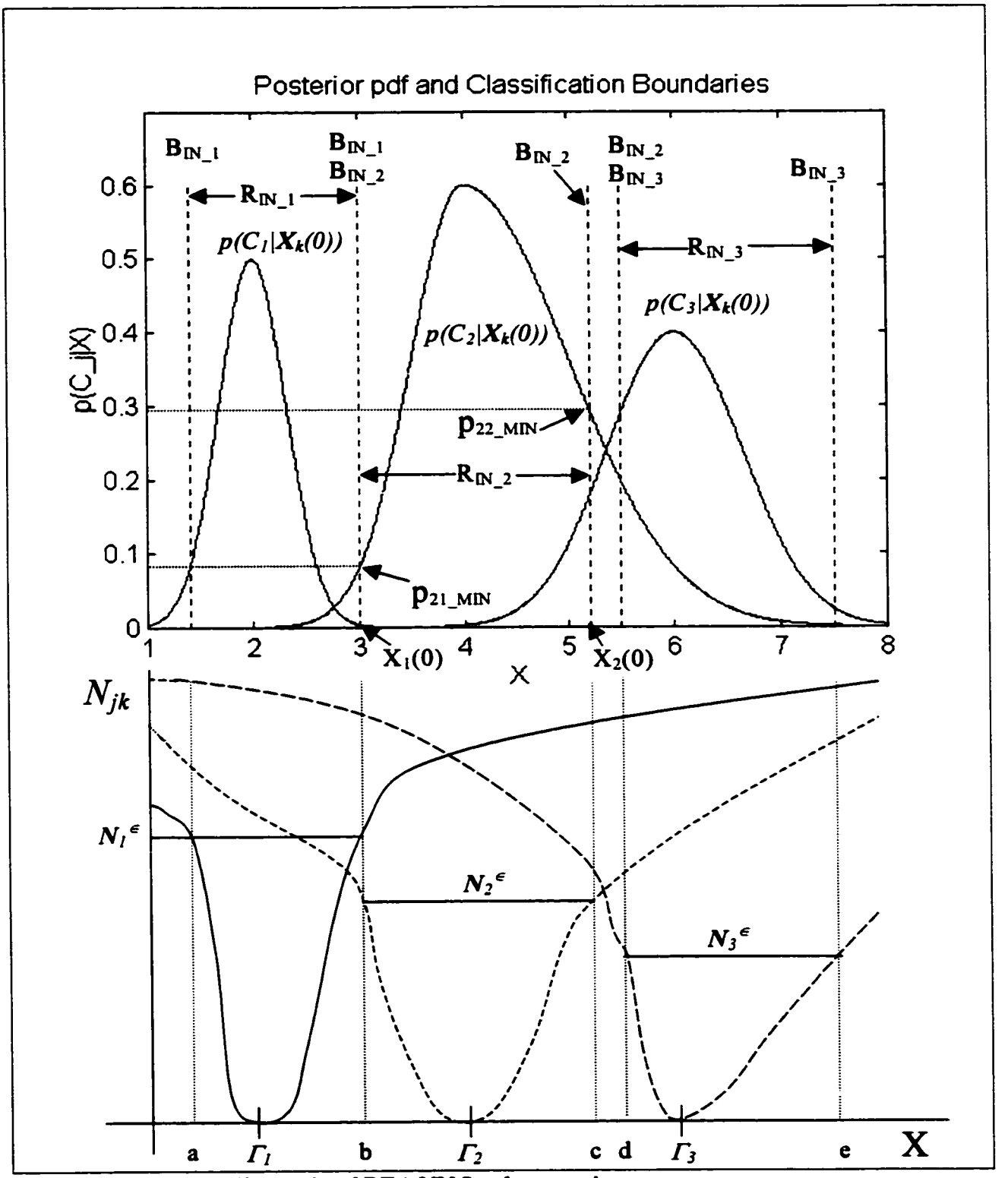


Figure 4.6.2-1 : 1D Example of RTA NN Implementation

The bottom part of Figure 4.6.2-1 shows the maps of iterations convergence N_{jk} versus initial value $X_k(0)$ for the three NDS to be implemented with three independent NN. These plots are similar to the example shown earlier in Figure 3.2-5 (page 67). Each NDS must learn a convergence rate function $H_j(X_k(n))$ which produces the equivalent N_{jk} mapping shown in the bottom of 4.6.2-1. For example, recall that the $H_j(X_k(n))$ shown in Figure 3.2-2 (page 66) is the required mapping function to produce the N_{jk} plot of Figure 3.2-5.

For class C_1 the classification boundaries shown in Figure 4.6.2-1 are achieved by adjusting the iterated mapping function $H_1(X_k(n))$ such that the number of iterations to convergence at the boundary points is equal to the defined maximum value N_1^{ϵ} for the given pdf value of p_{jk_MIN}. During a race, when N_{1k} exceeds that value, NDS-1 is disqualified from the race by the decision device (through the class membership function $M_{RACE(j)}$). In essence, this means that NDS-1 has identified the input as being "OUT" of class C_1. This approach has the equivalent effect of 'truncating' the classification boundary at the $X_k(0)$ value corresponding to that N_{jk}.

A similar approach is used for the other two NDS so that the desired classification boundaries are achieved for all classes. For example, note that even though some points $X_k(0) < b$ have $N_{2k} < N_{1k}$, NDS-1 will still win the race because NDS-2 will be disqualified from the race after N_2^{ϵ} iterations (i.e., it will effectively identify such inputs as OUT of class C_2 because their convergence time exceed the N_2^{ϵ} limit).

Note also that the mappings of N_{jk} versus $X_k(0)$ for <u>each</u> NDS extend over the <u>entire</u> input space $\mathbb{R}_{IN}$ over which $X_k(0)$ may occur. This is necessary to ensure that only the 'correct' NDS identifies an input as a member of its class.

Figure 4.6.2-2 illustrates a suitable output $L(D,\gamma,n)$ from the decision device for the example of Figure 4.6.2-1. Here, the vector A of equation E_4.6.1.1-1 (page 139) has this form : $A = [1,2,3,4]$. Note that the decision device output is shown for time $N^{\epsilon}_1 + 1$

because it is guaranteed to have achieved a classification decision by that time, as dictated by equation E_4.6.1.1-3.

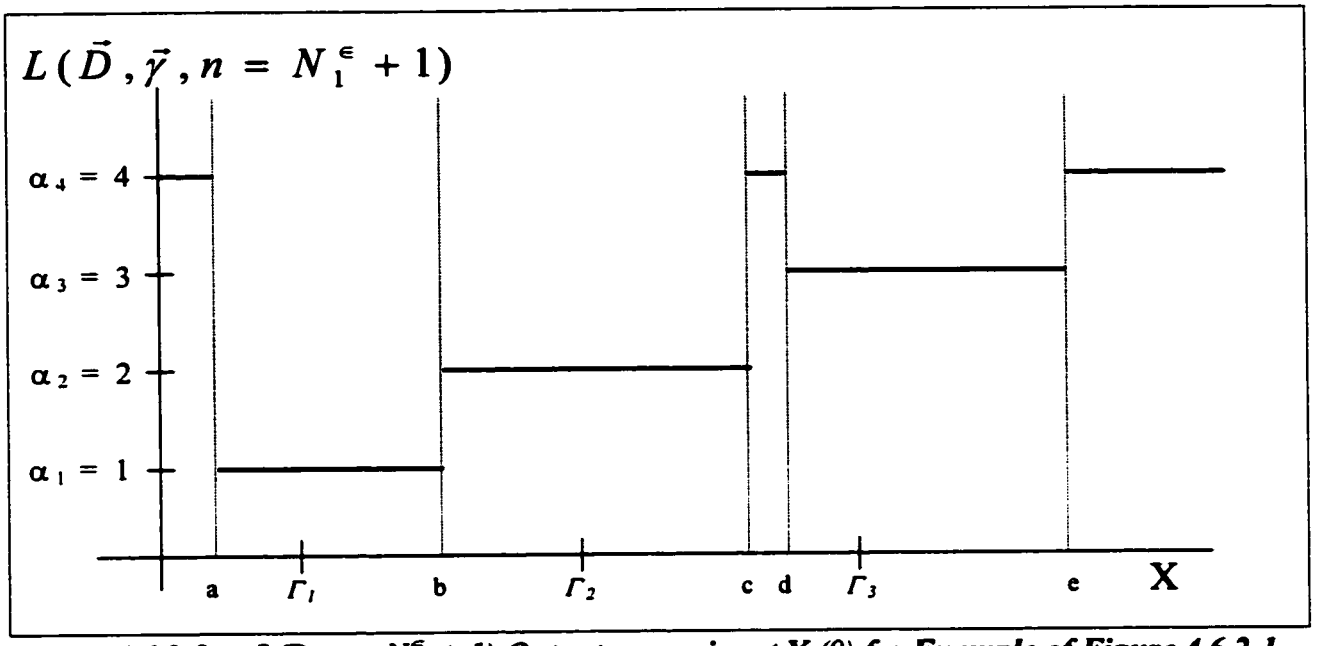


Figure 4.6.2-2 : L(D,γ,n =$N^{\epsilon}{}_1$ + 1) Output versus input $X_k(0)$ for Example of Figure 4.6.2-1

4.7 Training Aspects of the RTA NN Model

In this section, the training of neural networks is examined again (this was first discussed in section 3.5 of chapter 3, starting on page 75). The primary focus in this section is on the 'curse of dimensionality' [4,26] and how it limits the ability of a NN to learn a desired mapping. This problem is analyzed in detail in section 4.7.1. In section 4.7.2, mechanisms available to the RTA NN model to reduce the effect of this curse are reviewed briefly.

In section 4.7.3, the choice of architecture and training algorithm for the neural subnets of the RTA NN model is discussed briefly.

4.7.1 The Curse of Dimensionality and the Learning Task Complexity

In this section, we discuss an important theoretical concept which explains how the complexity of a training task is related to the amount of training data used and to the complexity of the mapping function being taught. This information is relevant because a measure of complexity provides us with an indication of the required size of the neural net and of the required amount of time to teach it properly. Furthermore, knowing what causes complexity in the first place will sometimes allow us to formulate a problem in such a way that complexity is reduced.

When teaching a neural network to approximate a function by using a set of training vectors, it has been demonstrated that the number of training vectors V required to achieve a given degree of accuracy increases exponentially as the ratio of the input dimension P to the degree of smoothness s of the mapping function increases [44]:

$$V = Le^{\frac{P}{s}} \qquad \text{(E_4.7.1-1)}$$

Here, s decreases towards 0.0 as the complexity of the function increases and L is a positive constant. This relationship is often referred to as the 'curse of dimensionality' [26] because many NN training problems require a large input dimension, which results in an exponential increase in the number of training vectors required to teach the NN adequately.

A neural network with M free parameters, has a limited learning capacity, which can be defined by the Vapnik-Chervonenkis (VC) dimension [26]. If the size of the training set increases, the size of the neural net must also increase in order to increase the number of free parameters. This necessarily increases the complexity of the learning task because more free parameters must now be computed and more training time is required to compute them. Hence, an increase in V ultimately results in an exponential increase in the complexity of the learning task. As a result of this, an increase in P or an increase in the complexity of the mapping function (i.e., decrease in s) will ultimately result in an

increase in the NN size and in the amount of time required to train it[x].

The increase in V as the complexity of $\partial X/\partial t$ increases can also be understood from the point of view of the Nyquist sampling rate theorem discussed in section 4.2.2. Recall that the required sampling interval Δt given by equation E_4.2.2-4 (page 99) must decrease as the frequency spectrum of the mapping function increases. As the complexity of $\partial X/\partial t$ increases, it has more peaks and valleys with steeper slopes. These effectively increase the amount of signal energy at higher frequencies, so that samples must be taken at smaller time intervals. More samples on any given trajectory means more training points V, leading to an exponential increase in the complexity of the mapping function.

By minimizing the learning task complexity, it becomes possible to achieve a high mapping accuracy in the minimum amount of time and with a NN of minimal size. This is important to us in view of the fact that the RTA NN model for this research has been implemented in software, on a serial-instruction computer. Because this approach results in much longer training times than a hardware (parallel) implementation, it is imperative that complexity of the mapping function $\partial X/\partial t$ be minimized as much as possible.

The complexity function s itself can take many forms. An example is shown below for a one-dimensional function $f(x)$:

$$s_f = s(f(x)) = \frac{1}{\int_{x=0}^{G} |f''(x)| dx} \qquad \text{(E_4.7.1-2)}$$

Here, $[0, G]$ is the interval over which the function is defined. The sum of the absolute value of second derivatives ensures that complexity will increase (i.e., s decreases) as the number of peaks or valleys increases and as their slopes become steeper. Note that a function such as $\partial X/\partial t = K$ has $s = \infty$ or minimum complexity. Many other smoothness

[x] Note that in this dissertation, the input dimension will normally be fixed whereas the mapping function $\partial X/\partial t$ complexity can vary. Thus, it is more relevant for us to think of the 'curse of dimensionality' as a 'curse of complexity' instead.

functions *s(...)* can be defined [26], without the need to constrain *f(x)* to be twice differentiable.

4.7.2 RTA NN Model Mechanisms Available to Reduce Learning Task Complexity

In this section, the mechanisms available to the RTA NN model to reduce the learning task complexity are reviewed briefly. These mechanisms are as follows:

1. **Use an iterated mapping function instead of a static one**: By defining the mapping function in such a way that the input is gradually transformed into the class prototype over several steps instead of in one single step, the complexity of the mapping function is significantly reduced. This was shown in the example of section 2.4.1.1 of chapter 2 (starting on page 41). In the RTA NN, this mechanism is implemented by the subnets which are taught the iterated mapping functions $H(X(n))$.

2. **Have each subnet specialize in recognizing only one class**: In other words, subnet '*j*' only classifies inputs $X_k(0)$ as one of two possible categories: "$X_k(0)$ is a member of class C_j" or "$X_k(0)$ is NOT a member of class C_j". Note that this approach is really nothing more than the mathematical concept of *orthogonalization*. This was introduced in section 4.6.1.1 where the *Karhunen-Loève* series expansion [41] formulation of the RTA NN model was discussed.

   For example, in Figure 4.6.2-1 (page 142), the NDS-2 specializes in mapping all inputs in the range *[a, e]* as IN or OUT of the class C_2 via the number of iterations to convergence. The mapping function $\Delta H(X(n))$ which generates the map of N_{jk} versus input $X_k(0)$ is much simpler than if one network was taught to learn all three attractors shown on that figure.

3. **Simplify the convergence rate function**: By simplifying $\partial X/\partial t$ (or $H(X(n))$), the complexity of the learning problem is also reduced. There are two ways to do this.

One way is to redefine the attractor Γ_j as a region from a single point. An example was shown in section 2.4.1.2 of chapter 2 (starting on page 42) where this approach produced a simpler function.

Another approach was discussed in sections 4.5.4 and 4.5.5. In these sections, it was shown how to enforce smoothness constraints on $\partial X/\partial t$ and still achieve acceptable classification performance.

4. **Re-use a previously-learned mapping function to learn a new one**: This was discussed in section 2.4.1.3 of chapter 2 (page 42) with a simple example. In the RTA NN model, this can be implemented by using a trained NN for NDS-j with attractor Γ_j and reusing it (i.e., making a copy of it) to learn a new NDS-r with attractor Γ_r. This approach was introduced in section 3.2.1 (page 67).

In chapter 5, it will be shown that the ability of the RTA NN to implement these mechanisms allows it to overcome the curse of complexity to a significant extent and achieve reliable classification for difficult mapping problems.

4.7.3 Architecture of the RTA NN Subnets and their Training

So far in this chapter, there has been no discussion of the architecture of the neural subnets of the RTA NN model (shown in Figure 4.6-1 on page 136) or of the algorithms to be used for their training.

The reason for not discussing this subject before is that **the choice of neural net architecture and the training algorithm used is independent of the RTA NN model paradigm**. What is meant by this is that the RTA NN model does NOT preclude any neural net architecture or training algorithm from being used for implementation. Any neural architecture which can learn an iterated mapping function $\Delta H(X(n))$ can be used for the subnets, and any algorithm which can teach those nets to learn the required mapping functions can be used. The RTA NN paradigm is concerned mainly with

defining a process to assign 'time to convergence' with posterior probabilities of class membership and with implementing mechanisms which greatly simplify the complexity of the learning task. How this simplified learning task is actually implemented with currently available neural models is largely an unrelated problem.

Hence, many of the net structures described in section 2.5 of chapter 2 could be used for this task, such as MLP, RBF or TDNN feedforward nets taught with backpropagation learning.

In chapter 5, a specific implemetation example will be discussed in detail. The discussion of the neural net architecture and training algorithm selected is relegated to that chapter since these subjects are not directly related to the RTA NN paradigm theory discussed in the current chapter.

4.8 Summary and Conclusions

In this chapter, mathematical models were developed and analyzed for the various aspects of the RTA NN paradigm.

In section 4.2, characteristics of nonlinear dynamical systems (NDS) were analyzed. The conditions necessary to ensure that the NDS has an attractor which is globally and asymptotically stable were defined. The relationship between a continuous-time NDS and a discrete-time NDS was also clarified.

In section 4.3, the basic principles of statistical pattern recognition problems and their optimal solution in the Bayesian sense were developed. This section showed how training samples, class membership probabilities and classification boundaries were inter-related in order to achieve optimal classification performance. In section 4.4, it was shown how the concepts of statistical Bayesian learning and NDS were related to each other through the RTA NN model paradigm. This section described how the fundamental axiom of the RTA NN model (i.e., equation E_3.3-1 on page 71) determines the manner in which the

'time to convergence' is related to the posterior class membership probabilities for the attractor and for points on the classification boundary. That section also described why points on a common trajectory of a NDS must have a posterior probability density function (pdf) which is monotonically decreasing from the attractor outward. However, it was also shown that this constraint on the pdf could be removed in practice by redefining the attractor in a higher dimension (section 4.4.2).

In section 4.5, The relationship between 'time to convergence' and posterior probabilities of class membership was analyzed in detail. Equation E_4.5.2-2 (page 117) was shown to capture the required relationship between posterior class membership probabilities and NDS convergence rate function in order to achieve optimal classification performance with the RTA NN.

Following this, a mathematical model of the RTA NN supernet was described in section 4.6. This model captures the rules and constraints which regulate the operation of the RTA NN.

Finally, section 4.7 discussed some aspects of training for the subnets of the RTA NN supernet. Section 4.7.1 discussed the curse of dimensionality and its effect on the complexity of the learning task for a neural network. Section 4.7.2 reviewed mechanisms available to the RTA NN in order to reduce this learning task complexity. Section 4.7.3 concluded the discussion on training with a few words about the choice of NN architecture for the subnets of the RTA NN and the choice of the training algorithm. It was argued that the user is free to choose any architecture which can be taught an iterated mapping function. The user is also free to choose any training algorithm which can teach the net this function.

With this information in hand, we are now ready to describe how the RTA NN performs in a real classification problem. This is the subject of chapter 5.

Chapter 5 :

Performance of the RTA NN Model for a Sample Classification Task

5.1 Introduction

This chapter describes the ability of the RTA NN model to learn a practical classification task in $\mathbb{R}^2$. The main objective of this chapter is to demonstrate that the RTA NN model achieves the key characteristics of learning in the brain (described in section 2.3 of chapter 2 (page 37)) and that it does so through the mechanisms described in section 2.4 of that chapter (page 39). Another objective is to identify disadvantages and limitations of the RTA NN model and propose possible ways to overcome these.

Let us summarize briefly the key characteristics of learning in the brain:

1. the ability to learn complex classification boundaries using simple devices (i.e., neurons);

2. the ability to learn new classes without forgetting the ones already in memory (i.e., little or no memory loss);

3. the ability to learn a new class with a level of effort related mostly to the complexity of that class (e.g., as a child learns letter 'Z' with roughly the same level of effort as letter 'A' despite the fact that 'A', 'B',....'Y' are already in memory).

4. the ability to compensate for damage caused by the death of neurons and continue to perform reliable classifications.

It was argued in chapter 2 that the following mechanisms will allow a neural net (NN) to achieve the above characteristics to a certain extent:

a. implement the classification mapping as an iterated function instead of using a static function (i.e., reduce the complexity of the learning task by implementing the mapping gradually over time instead of in one single step).

b. implement the classification using independent modules which each specialize in classifying inputs as IN or OUT of one class (i.e., orthogonalize the mapping functions to reduce the overall complexity and reduce memory loss).

c. re-use existing, trained subnets to learn new classes. This simplifies the overall complexity of the learning task.

d. redefine the attractor from a point to a region. This simplifies the complexity of the learning task and allows the NN to repair damage. It also allows previously taught NN to be re-used to learn new classes.

In section 5.2, the types of neural nets (NN) used in the RTA NN *supernet* is discussed as well as the training algorithm selected. Section 5.3 describes in detail the two-dimensional (2D) classification problem which was used to test the performance of the RTA NN model. That section also describes how mechanism 'b' achieves a reduction in the complexity of the learning task. In section 5.4, results of the training sessions are described and analyzed.

Section 5.5 describes how to redefine attractor Γ_j from a point to a 2D region in order to simplify the learning task complexity. It will also be shown in that section that this approach allows each subnet to achieve the desired classification mapping to a high degree of accuracy. In section 5.6, the mechanism of re-using an existing subnet to learn a new class is demonstrated. Section 5.7 demonstrates how the RTA NN model can be made to learn new classes without forgetting old ones and with a level of difficulty

related mostly to the new classification boundary itself. Section 5.8 demonstrates the ability of the RTA NN model to repair neural damage relatively easily and restore its classification accuracy to a large extent.

Section 5.9 discusses the disadvantages and limitations of the RTA NN model. Where possible, ideas are introduced which could help overcome these. Finally, section 5.10 summarizes the information presented in this chapter.

5.2 Neural Net Types and Training Algorithm Used for the RTA NN Model

Section 5.2.1 describes the types of neural nets used for the subnets of the RTA NN and provides a rationale for these choices. Section 5.2.2 discusses the algorithm used to train these nets. Section 5.2.3 describes briefly the software used to implement and train the subnets of the RTA NN model.

5.2.1 Neural Net Types Used and Rationale for the Choices Made

The basic architecture for the subnets of the RTA NN model is the feedforward architecture described earlier in section 2.5.1.3.1 of chapter 2 (page 53). This structure may have more than one layer (e.g., Multi-Layer Perceptron (MLP)) but there are no feedback connections between neurons (although self-feedback connections may exist). The types of neurons used were as follows:

1. the McCulloch-Pitts model [26], with a set of weights $\boldsymbol{W}$ connecting to the input vector $\boldsymbol{X}$ and an activation function $\varphi(v(x))$ which can be linear or nonlinear (i.e., sigmoid function or *tanh* function).

2. simple Radial Basis Function (RBF) neurons which have a Gaussian activation function.

3. locally recurrent RBF neurons. These are neurons which have a feedback connection to their own input (i.e., as opposed to a feedback connection to other neurons).

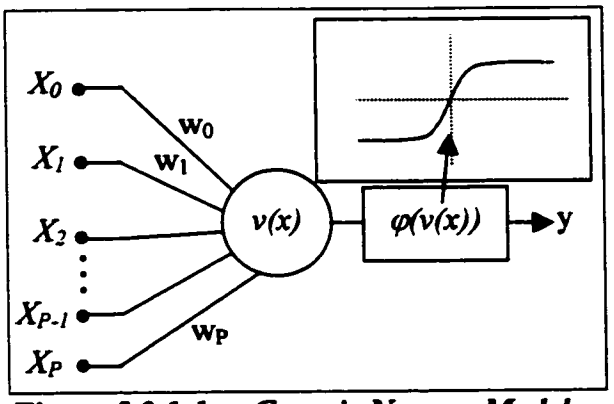


Figure 5.2.1-1 : Generic Neuron Model

Figure 5.2.1-1 illustrates a general structure which is applicable to all these neuron types. For the McCulloch-Pitts neuron, the activation potential function $v(x)$ takes this form:

$$v(x) = w_0 x_0 + \sum_{j=1}^{P} w_j x_j \qquad \textbf{(E_5.2.1-1)}$$

where $\boldsymbol{X} = [x_1, \ldots, x_P]$ is the P dimensional input vector and where $x_0 = -1$ is a bias term whose value does not change with time. The sigmoid and *tanh* activation functions have the form shown below:

$$\varphi(v(x)) = \begin{cases} \dfrac{K}{1+\exp(-a\,v(x))} \Rightarrow \textit{sigmoid function} \\ K \tanh\left(\dfrac{a\,v(x)}{2}\right) \Rightarrow \tanh \textit{ function} \end{cases} \qquad \textbf{(E_5.2.1-2)}$$

The main difference between the two is that the sigmoid has an output value in the range $[0,1]$ whereas the *tanh* has an output in the range $[-1,1]$ (neglecting the scaling factor K). A sketch of the latter is shown in the inner box of Figure 5.2.1-1.

For the simple RBF neuron, the activation function $\varphi(v(x))$ takes this form in $\mathbb{R}^1$:

$$y(n) = \phi(v(n)) = \exp\left(-\sum_{i=1}^{i=P} \frac{(w_i x_i(n) - \mu_i)^2}{\sigma_i^2}\right) \qquad \textbf{(E_5.2.1-3)}$$

Two types of locally recurrent neurons were used : *linear-recurrent RBF* (LR-RBF) and *center-recurrent* RBF (CR-RBF) [62]. These neurons have $P+1$ inputs, and an output $y(n)$ defined as follows:

$$y(n)=\phi(v(n))=\begin{cases}\exp\left(-\sum_{i=1}^{i=P}\left(\frac{(w_i x_i(n)-\mu_i)^2}{\sigma_i^2}\right)-y(n-1)\right)\Rightarrow LR-RBF\\ \exp\left(-\sum_{i=1}^{i=P}\left(\frac{(w_i x_i(n)-\mu_i)^2}{\sigma_i^2}\right)-\frac{(y(n-1)-\mu_0)^2}{\sigma_0^2}\right)\Rightarrow CR-RBF\end{cases} \quad \textbf{(E_5.2.1-4)}$$

The feedforward net architecture was selected because it is relatively easy to teach by using the backpropagation algorithm [26]. It also learns relatively quickly with that approach. Fully recurrent nets are harder to teach and may exhibit instability problems as well. Time Delay Neural Networks (TDNN) are essentially a feedforward MLP net with a finite-impulse-response (FIR) filter at each synapse. They were not selected in part because implementation and training is harder (because of the FIR). Furthermore, through the "unfolding in time" [26] algorithm, they can be shown to be equivalent to a static feedforward MLP. As a result, there is no real advantage to use TDNN instead of a static MLP net.

Both the Hopfield net and Boltzmann machine can be taught to associate an input with a given attractor but the trajectory c_{jk} and time to convergence N_{jk} cannot be adjusted through training. Since c_{jk} and N_{jk} are essential elements of the RTA NN model paradigm, these net types are therefore unsuitable for use with our model.

The neuron types listed earlier were selected because their relative learning abilities complement each other well for the type of iterated mapping functions $H(X(n))$ which the nets must learn. A MLP net with McCulloch-Pitts neurons usually exhibits a good ability to extrapolate or interpolate in regions where little or no training data is provided. This is due mainly to the fact that McCulloch-Pitts neurons perform *global approximations* of the mapping function being learned [4,19,26]. This ability is of interest to us because it allows the net to learn a mapping function adequately with training data which is more

sparsely distributed in the input space $\mathbb{R}_{IN}$ than would be possible otherwise. Hence, a smaller data set can be used, resulting in shorter training times.

By contrast, RBF neurons are much better at performing *local approximations* [4,19,26]. This ability is useful to learn the narrow peaks and sharp slopes of $H(X(n))$. Locally recurrent neurons are particularly well suited for learning iterated mapping functions $H(X(n))$ since their local feedback allows them to better capture the dynamics of such functions than 'static' neurons without any feedback [19,62]. However, they are harder to teach because of potential stability problems. Appendix A (starting on page 227) provides a more detailed discussion of the properties of LR-RBF and CR-RBF neurons and how they compare to simple RBF.

Note that by combining McCulloch-Pitts neurons and RBF neurons (with or without feedback) in the same MLP net, it is possible to capture the advantages of both types to a certain extent. This was done in most of the NN trained.

5.2.2 Training Algorithm for the Neural Nets

The training algorithm used for the feedforward nets was the backpropagation algorithm with momentum term [4,26]. Since this algorithm is well explained in the classical literature, there is no need to develop it in this chapter. However, the reader can find a short description of that algorithm in Appendix A.

The backpropagation algorithm was selected primarily because of the simplicity of its implementation in software but also because it has been shown to be an effective approach to teach NN arbitrary mapping functions [4,26,37,43].

There are many variations based on the backpropagation algorithm : the *conjugate-gradient* algorithm, Newton methods, quasi-Newton methods, the Levenberg-Marquardt algorithm, Kalman filters, etc, just to name a few [4,26,33]. Although some of these can achieve a significant increase in the training speed and an improved ability to avoid local

minima of the error function, the required computational complexity is much higher. Any of these approaches could have improved training performance for the subnets of the RTA NN model. However, it was decided NOT to implement any of these algorithms because the improvement which could be gained was NOT essential to achieve the objectives of this chapter. Recall from section 5.1 that we seek only to demonstrate the ability of the RTA NN model to achieve the four characteristics of learning in the brain by using the four mechanisms described in that section. As will be seen in sections 5.4 onward, these objectives can be achieved with simple backpropagation training.

5.2.3 NNET Software Program Description

All the simulations performed in this research work were done with a custom-coded software package called NNET. This program was written in Object Oriented Programming (OOP) methodology, using C++.

The program NNET has close to 47000 lines of codes. Neurons and 'distance' functions $D_j(\Gamma_j, Y_j(n))$ as shown in Figure 4.6-1 (page 136) are implemented as stand-alone 'objects' which can be connected anywhere in the net. This flexibility allows MLP with several layers to be constructed, where the inputs to a layer are NOT restricted to be only from the previous layer. Also, various neuron types can be used in the same net. A sample neural net is shown in Figure 5.2.3-1, with different neurons and other objects available in NNET.

Each individual neuron has its own training parameters to maximize flexibility. Also, NNET can iterate the output of the NN back to the input in order to plot the achieved trajectories. Neurons can also be 'killed' in order to assess how their death affects the rest of the network. Training can be performed in *batch mode* or *pattern mode* [26] and can be terminated at a specific epoch, at a specific error level or if the error decreases by less than a specified amount in a user-defined number of epochs. The software has extensive logging capabilities for all objects of the net so that a detailed analysis of the learning process can be performed. Finally, NNET allows a large number of nets to be trained in

sequence through the use of batch files so that training, logging and testing of neural nets is fully automated and does not require user-intervention.

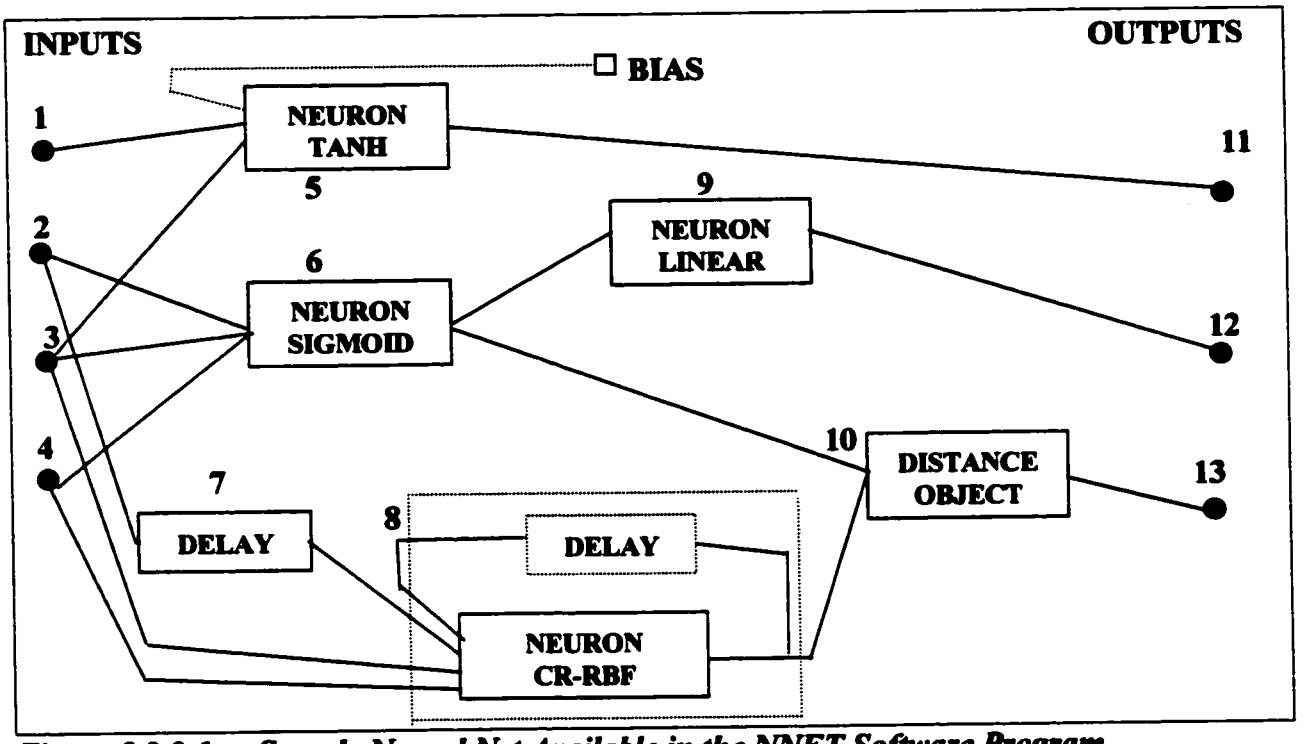


Figure 5.2.3-1 : Sample Neural Net Available in the NNET Software Program

A detailed software description document exists for NNET. But it has not been included in this dissertation since it is not directly relevant to the thesis itself.

5.3 Description of the Classification Problem and Effect of Orthogonalizing *H(X(n))*

In this section, the 2D training problem used to test the capabilities of the RTA NN Model is described (section 5.3.1). In addition, the reduction in the learning task complexity achieved by the orthogonalization of the mapping functions (i.e., mechanism 'b' of section 5.1) is demonstrated in section 5.3.2.

5.3.1 Description of the 2D Classification Problem

Recall from section 3.5.2 (page 86) that the 'maximum knowledge memorization learning' approach is used to train the subnets of the RTA NN model. Essentially, this means that everything is known about the classification problem (i.e., optimal class boundaries, trajectories, convergence rate, attractor, etc) so that each subnet 'j' only needs to be taught to memorize the function $H_j(X(n))$ through the training trajectories.

For the 2D problem selected, the desired optimal classification boundaries B_{IN_j} are shown in Figure 5.3.1-1. There are five classes C_A to C_E with the corresponding attractor Γ_j shown as an 'o' inside each region. The center of each classification region R_{IN_j} is shown as an 'X'. Note that for R_{IN_B}, the attractor is offset from the center. Note also that the boundary of region 'C' is a *hypocycloid* [53] with different scaling to the left and right of the attractor. Region 'D' is based on a *cardoid* [53]. Regions 'A' and 'B' are simple rectangular and circular boundaries respectively but region 'E' is a composite of the other four boundaries. These classification boundaries were selected because they are highly nonlinear and therefore constitute a difficult learning problem for any type of neural network, particularly those discussed in chapter 2. As discussed in section 4.6 (page 136) and in accordance with equation E_4.4-8 (page 108), all points on or inside B_{IN_j} must be classified as members of class C_j by subnet 'j' while points outside that boundary must be classified as non-members of class C_j.

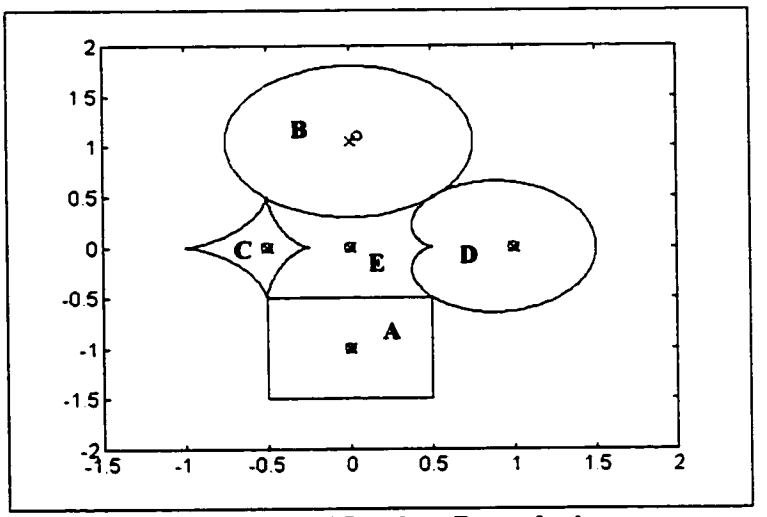


Figure 5.3.1-1 : Classification Boundaries

It is important to appreciate that even though the boundaries do not overlap, this does NOT imply that the posterior pdf of the classes have no overlap. As explained earlier in

chapter 4, these B_{IN_j} represent only the optimal rejection thresholds to minimize classification errors. The class pdf themselves usually overlap (e.g., see Figure 4.3.2-1 on page 105).

Figures 5.3.1-2 to 5.3.1-13 illustrate various characteristics of the NDS-A for class C_A. Figure 5.3.1-2 shows the map of N_{jk} versus input vector $X(0) = [x(0), y(0)]$. Figure 5.3.1-3 shows several contour lines of N_{jk} versus $X(0)$. Note that the rounded corners of some of these contour lines is only an artifact of the MATLAB utility which generates contours via interpolation. The boundary region B_{IN_A} is shown as a dotted line in that figure, with a 'X' symbol at the corners. Note that this boundary corresponds to $N^{\epsilon}_{A} = 20$, which is the maximum number of iterations permitted for an input $X(0)$ to be declared a member of class C_A. This number was selected large enough to ensure that the magnitude of the iterated mapping function $\Delta H_A(X(n))$ was not too large since excessively fast convergence rates may result in instability or oscillations near the attractor. This effect is somewhat similar to that seen in linear second order control systems where a fast rise time (to a step input response) will often result in large overshoots (i.e., 'ringing") and/or long settling times [2]. Conversely, N^{ϵ}_{A} was made as short as possible to ensure that points far away from B_{IN_A} (but still in the global input region R_{IN}) did NOT have an excessively high N_{jk}. The latter situation would result in training trajectories starting at these outer points having an excessively large number of points. This, in turn, would increase the required training time for the NN, as explained earlier in section 4.7.1 (page 145).

The other dotted line at $N_{jk} = 10$ (Figure 5.3.1-3) is the boundary B_{CV_A} of the *inner region of convergence* R_{CV_A} (marked with a 'o' at the corners). This was described in section 4.5.3.1 with an example (see Figure 4.5.3.1-9 on page 129). This is the region where $\Delta H_A(X(n))$ is forced to converge towards 0.0 as $X(n)$ approaches Γ_A in order to ensure that the NDS-A has a global and asymptotically stable attractor at that point.

Figures 5.3.1-4 and Figures 5.3.1-5 illustrate 3D views of the iterated mapping function $\Delta H_A(X(n)) = \Delta H_A(x(n),y(n)) = [\Delta H_{A\_x}(x(n),y(n)),\ \Delta H_{A\_y}(x(n),y(n))]$. Figures 5.3.1-6 and

5.3.1-7 show side-views of $\Delta H_{A\_x}(x(n),y(n))$ and $\Delta H_{A\_y}(x(n),y(n))$ respectively. Note again that within region R_{CV_A}, the magnitude of $\Delta H_A(X(n))$ is forced to converge to 0.0 linearly. Note also that the convergence rate outside the classification region R_{IN_A} is small relative to its magnitude inside that region. In effect, this implements a situation analogous to the Uncle Brian example shown in Figure 3.3-1 (page 69) where the slope of the bowl is fairly steep close to the attractor (i.e., fast convergence) but becomes shallower further out (i.e., slow convergence).

Figures 5.3.1-8 to 5.3.1-11 illustrate the 32 training trajectories used to teach the NN to learn the required $\Delta H(X(n))$. It can be noted that the N_{jk} increases rapidly outside B_{IN_A}, which is consistent with the slower convergence rate in that region. In reality, the training trajectories should extend over the region $\mathbb{R}_{IN}$ over which inputs to the supernet are expected to occur. Here, $\mathbb{R}_{IN} = \{x,y \mid x \in [-1.5,2],\ y \in [-2,2]\}$ as shown in Figure 5.3.1-1. However, training trajectories have been initially defined to be centered over the attractor of each region R_{IN_j} in order to ensure that the best possible match to $\Delta H(X(n))$ could be achieved in each such region.

Finally, figures 5.3.1-12 and 5.3.1-13 illustrate the scaled posterior pdf of class membership for NDS-A[xi]. Because the function $T_j(p_j)$ was made linear, the pdf is essentially an inverted and scaled version of the map of N_{jk} versus $X(0)$. The values of the pdf below p_{Ak_MIN} have been truncated to make it easier to see the boundary below it.

Figures 5.3.1-14 to 5.3.1-25 illustrate the characteristics of NDS-B. Note that because the attractor $\Gamma_B = (0.05,\ 1.1)$ does NOT correspond to the center of the region B_{IN_B} (at (0.0,1.05)), the contour lines N_{jk} are not concentric and the magnitude of $\Delta H_B(X(n))$ is different on either side of the attractor (e.g., Figures 5.3.1-18 and 5.3.1-19). This assymmetry is also plainly visible in the side views of the trajectories (Figures 5.3.1-21 and 5.3.1-22). The jagged edges in the pdf shown in Figure 5.3.1-24 are caused by the limited accuracy of the MATLAB interpolations between the points available.

[xi] Note that the area under the curve does NOT add up to 1.0, as required of a true pdf [41]. Scaling and translation were used along the z-plane to make the function easier to see.

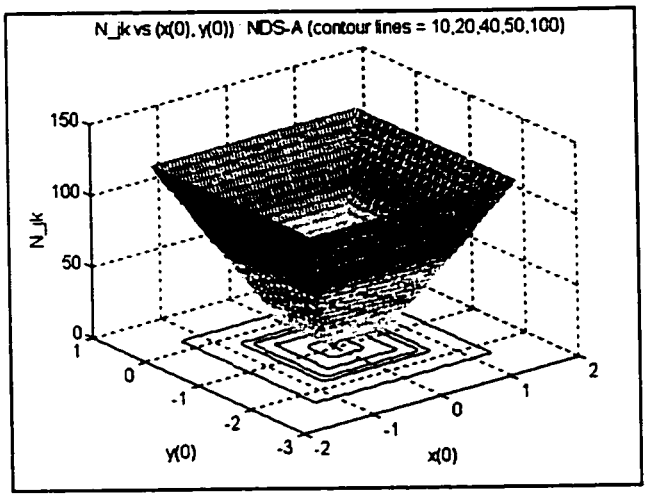


Figure 5.3.1-2 : N_{jk} versus X(0) for NDS-A

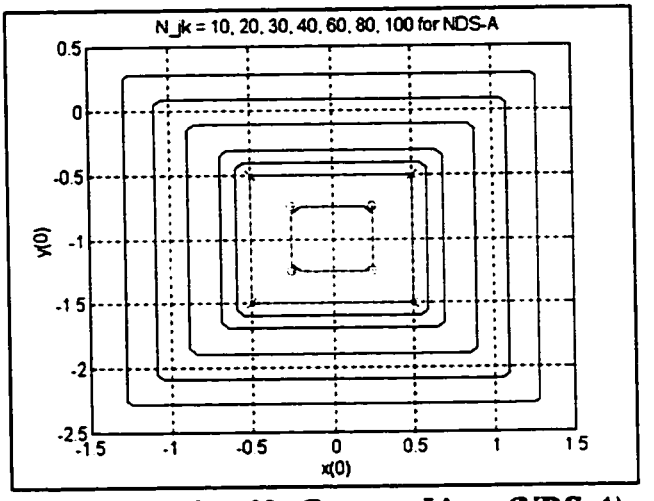


Figure 5.3.1-3 : N_{jk} Contour Lines (NDS-A)

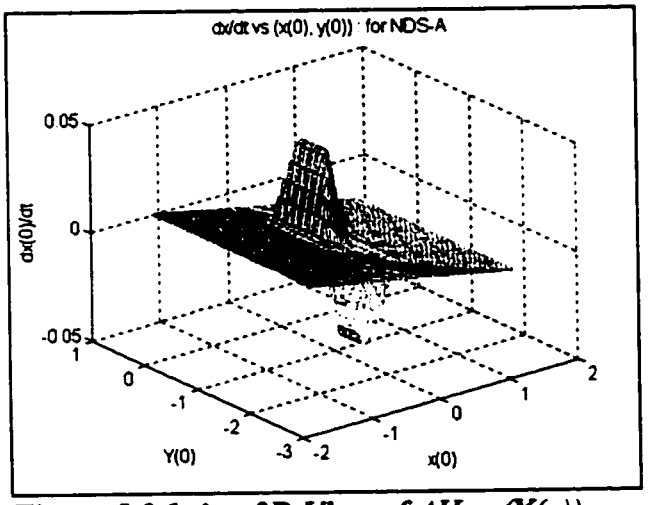


Figure 5.3.1-4 : 3D View of $\Delta H_{A\_x}(X(n))$

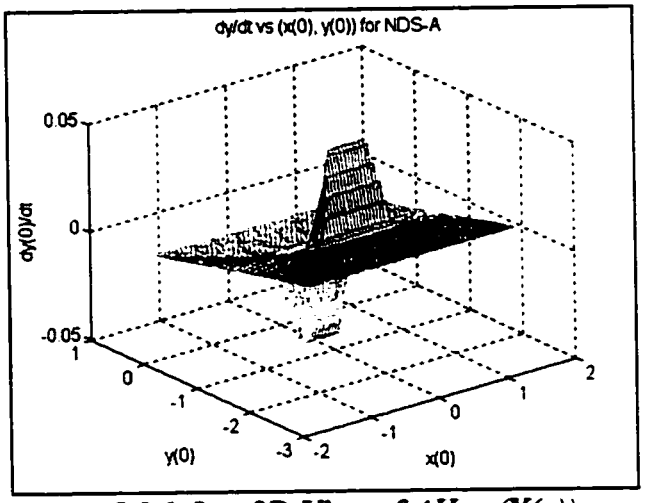


Figure 5.3.1-5 : 3D View of $\Delta H_{A\_y}(X(n))$

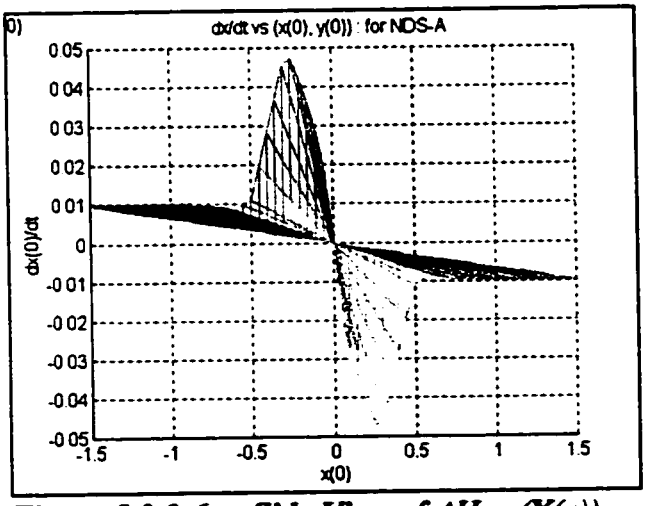


Figure 5.3.1-6 : Side View of $\Delta H_{A\_x}(X(n))$

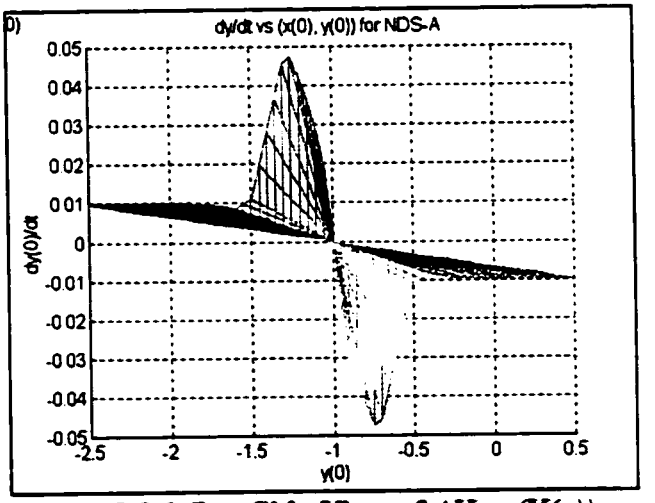


Figure 5.3.1-7 : Side View of $\Delta H_{A\_y}(X(n))$

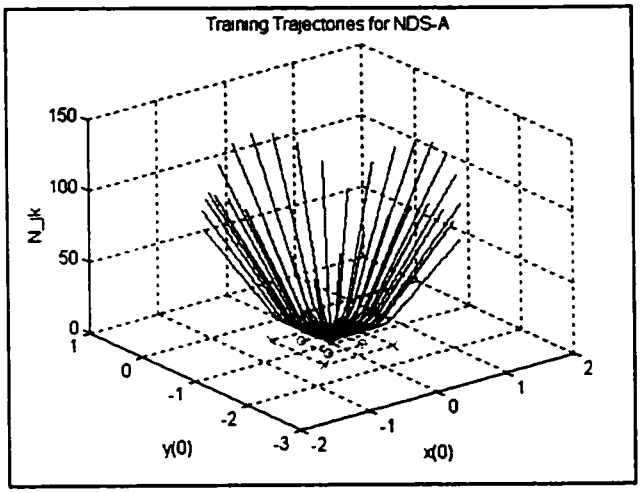


Figure 5.3.1-8 : 3D View of c_A for NDS-A

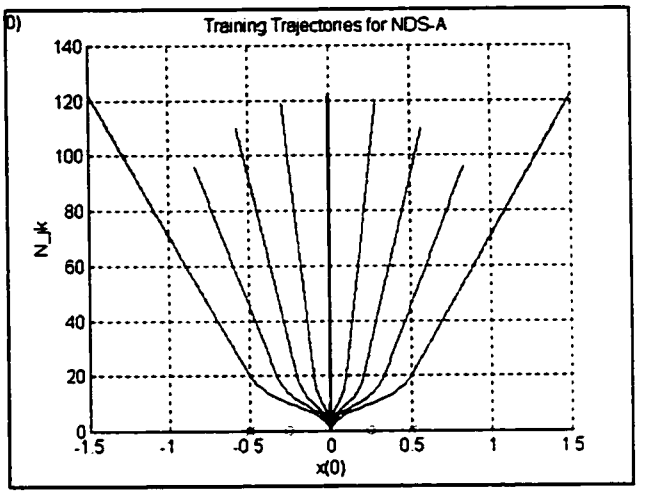


Figure 5.3.1-9 : x-Side View of c_A (NDS-A)

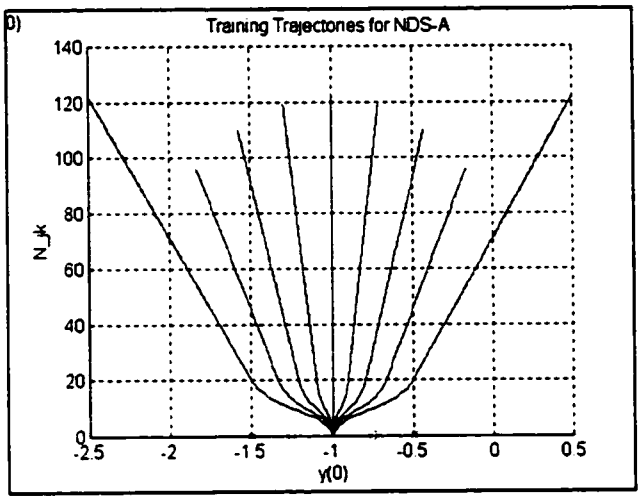


Figure 5.3.1-10 : y-Side View of c_A (NDS-A)

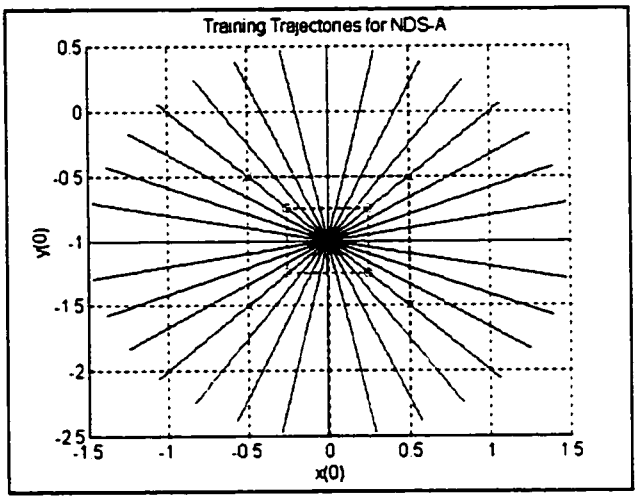


Figure 5.3.1-11 : Top View of c_A (NDS-A)

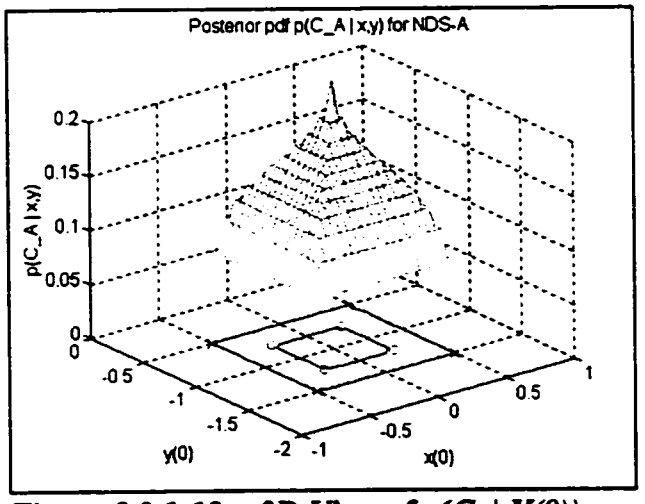


Figure 5.3.1-12 : 3D View of $p(C_A \mid X(0))$

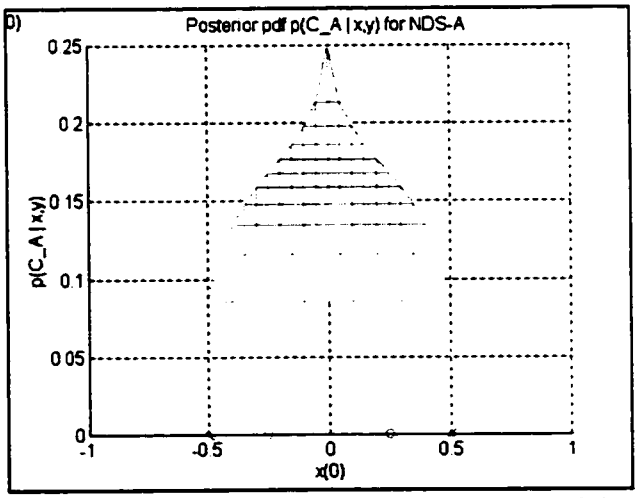


Figure 5.3.1-13 : x-Side View of $p(C_A \mid X(0))$

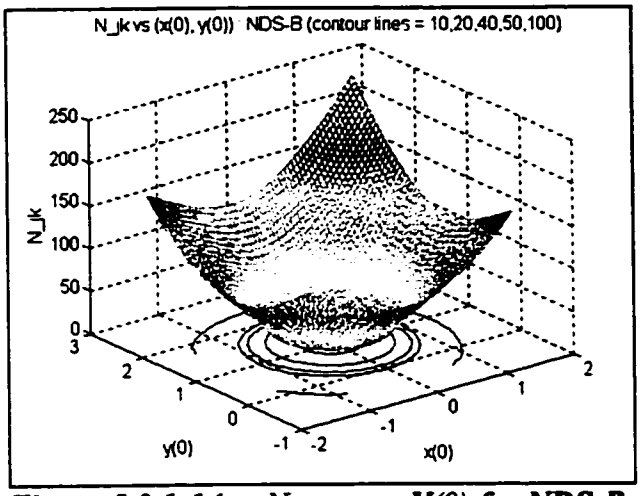


Figure 5.3.1-14 : N_{jk} versus X(0) for NDS-B

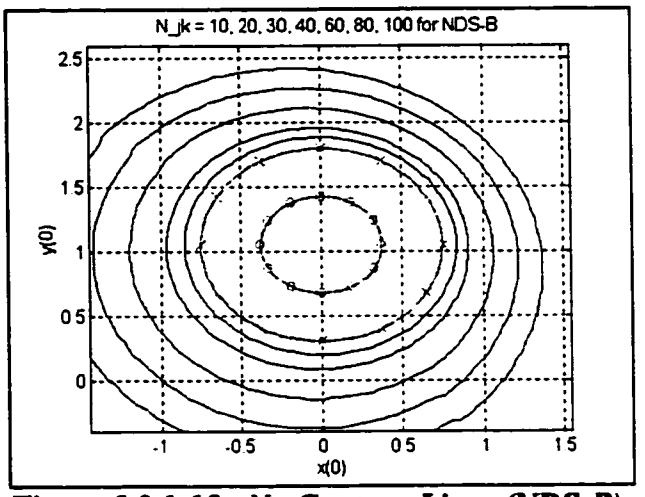


Figure 5.3.1-15 : N_{jk} Contour Lines (NDS-B)

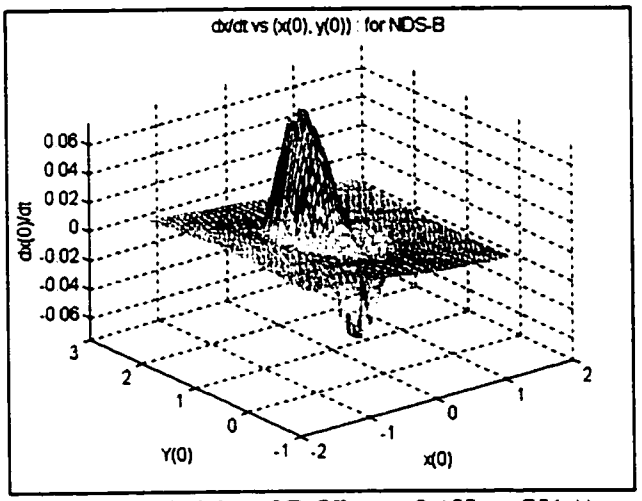


Figure 5.3.1-16 : 3D View of $\Delta H_{B\_x}(X(n))$

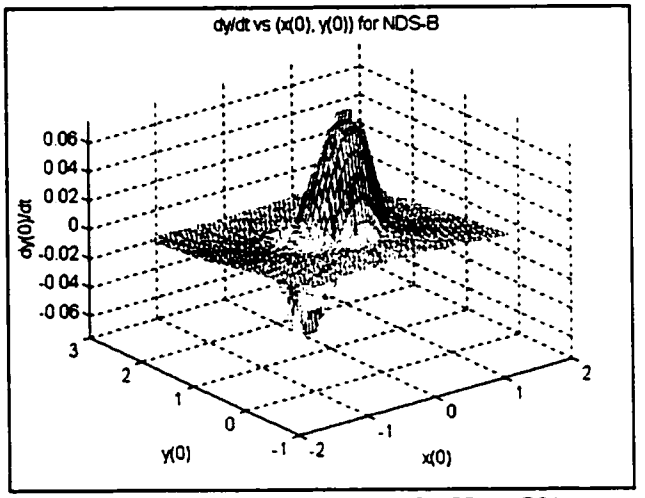


Figure 5.3.1-17 : 3D View of $\Delta H_{B\_y}(X(n))$

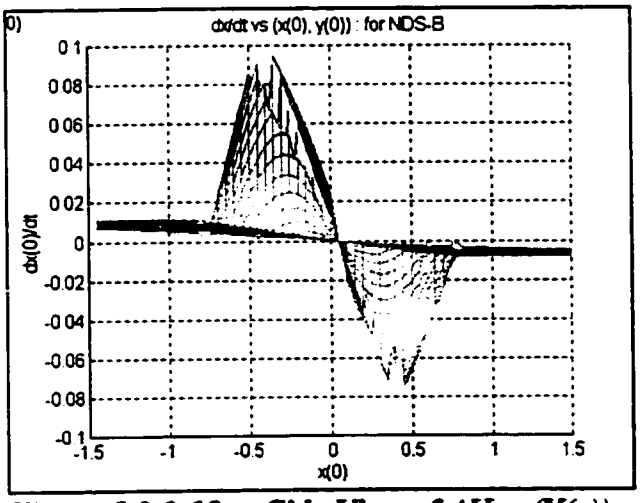


Figure 5.3.1-18 : Side View of $\Delta H_{B\_x}(X(n))$

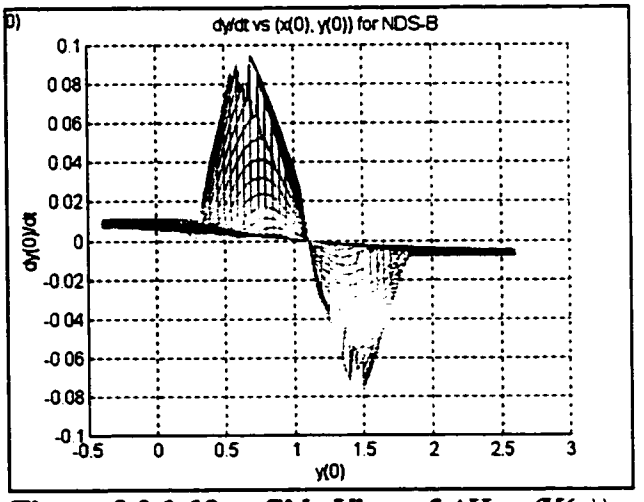


Figure 5.3.1-19 : Side View of $\Delta H_{B\_y}(X(n))$

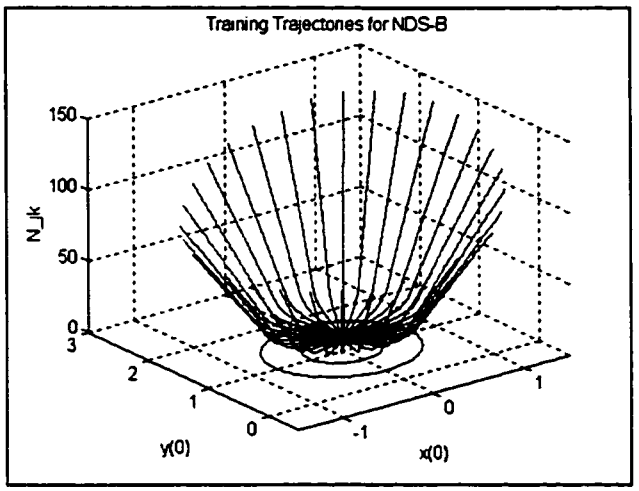


Figure 5.3.1-20 : 3D View of c_B for NDS-B

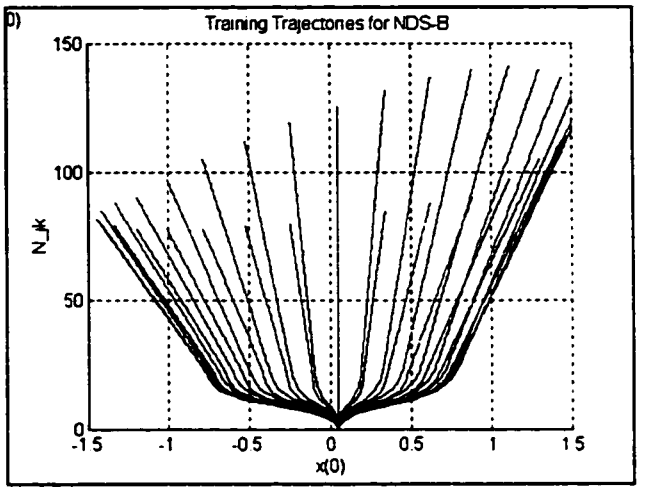


Figure 5.3.1-21 : x-Side View of c_B (NDS-B)

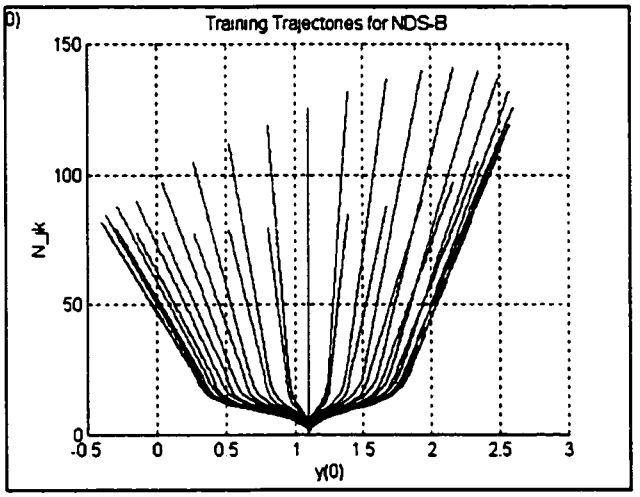


Figure 5.3.1-22 : y-Side View of c_B (NDS-B)

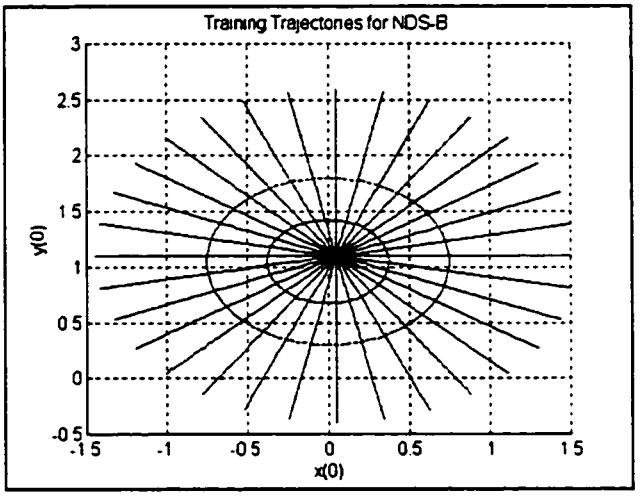


Figure 5.3.1-23 : Top View of c_B (NDS-B)

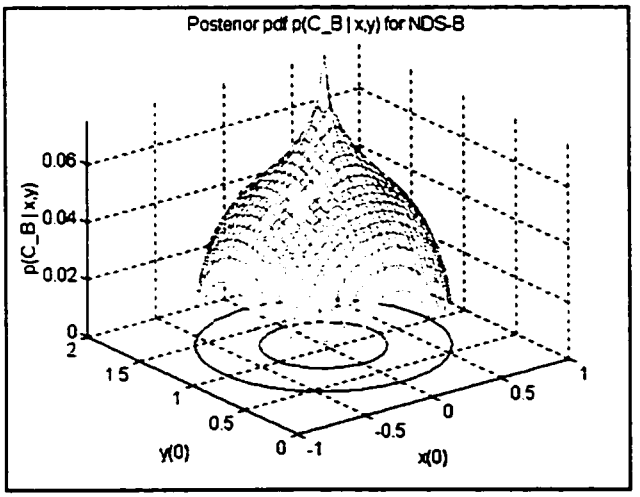


Figure 5.3.1-24 : 3D View of $p(C_B \mid X(0))$

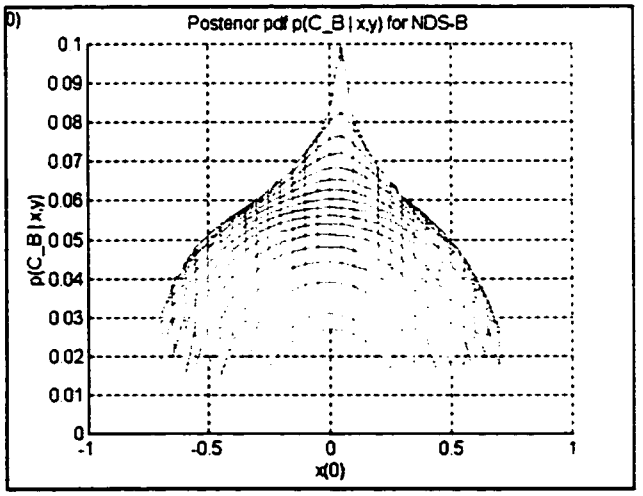


Figure 5.3.1-25 : x-Side View of $p(C_B \mid X(0))$

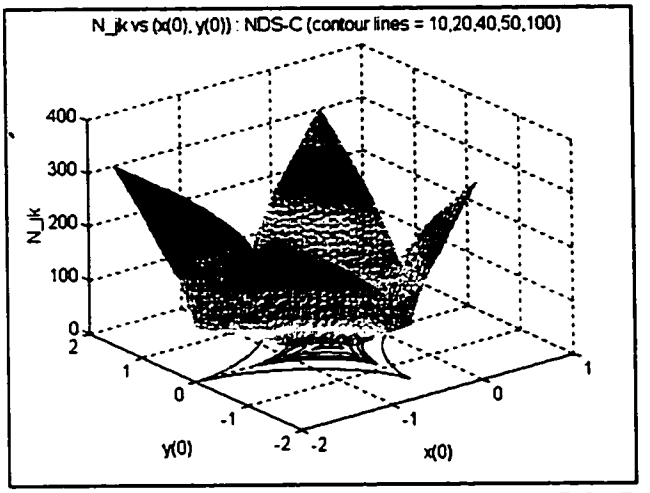


Figure 5.3.1-26 : N_{jk} versus X(0) for NDS-C

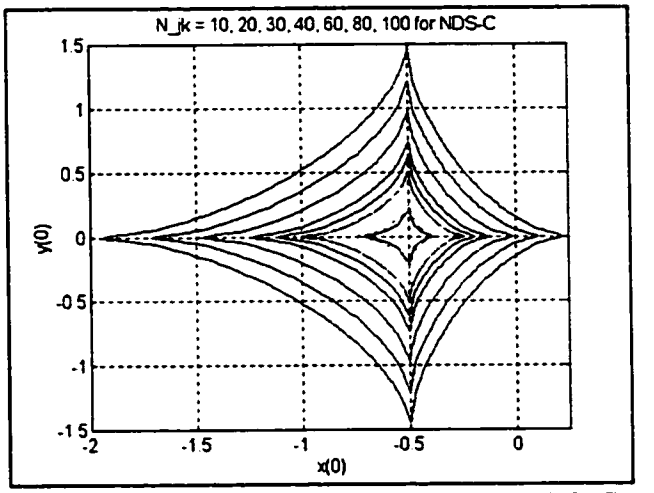


Figure 5.3.1-27 : N_{jk} Contour Lines (NDS-C)

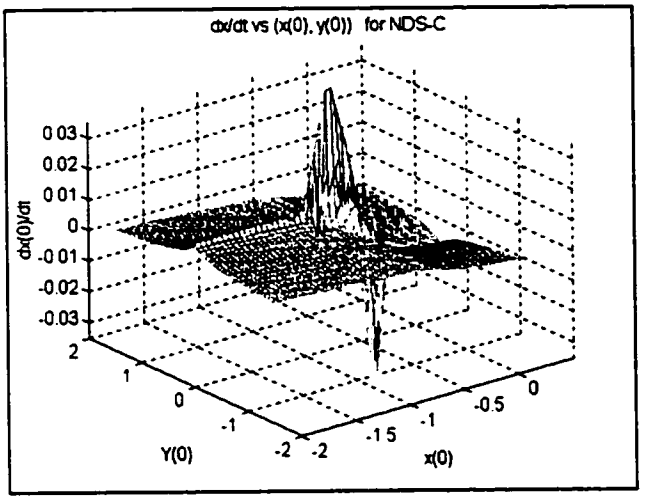


Figure 5.3.1-28 : 3D View of $\Delta H_{C\_x}(X(n))$

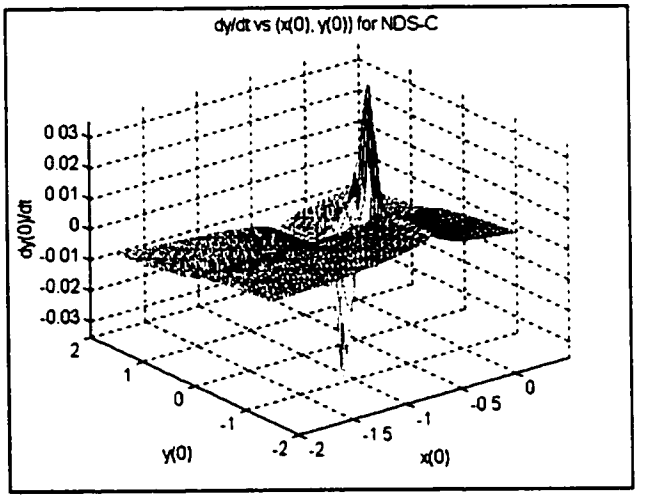


Figure 5.3.1-29 : 3D View of $\Delta H_{C\_y}(X(n))$

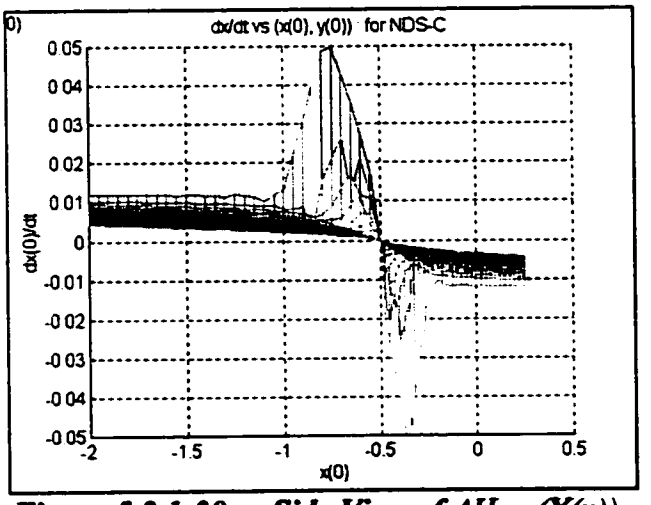


Figure 5.3.1-30 : Side View of $\Delta H_{C\_x}(X(n))$

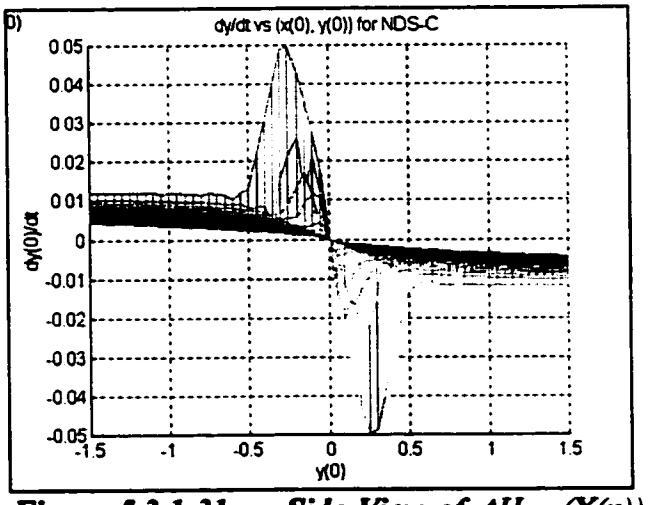


Figure 5.3.1-31 : Side View of $\Delta H_{C\_y}(X(n))$

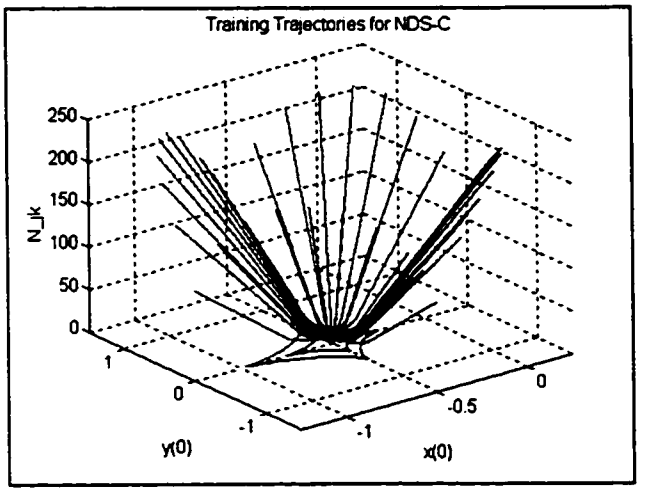


Figure 5.3.1-32 : 3D View of c_c for NDS-C

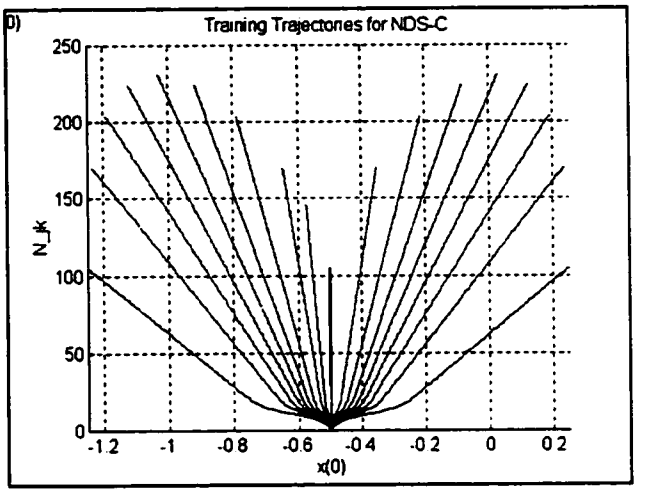


Figure 5.3.1-33 : x-Side View of c_c (NDS-C)

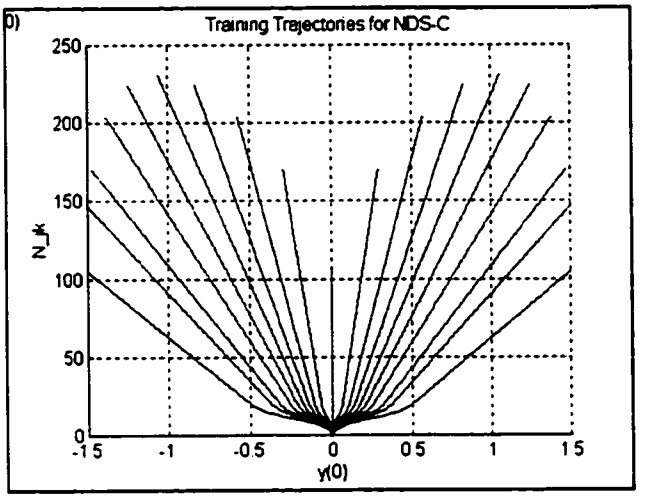


Figure 5.3.1-34 : y-Side View of c_c (NDS-C)

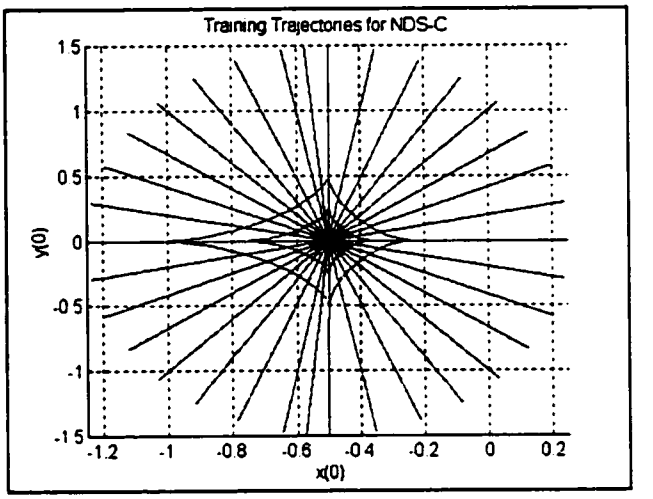


Figure 5.3.1-35 : Top View of c_c (NDS-C)

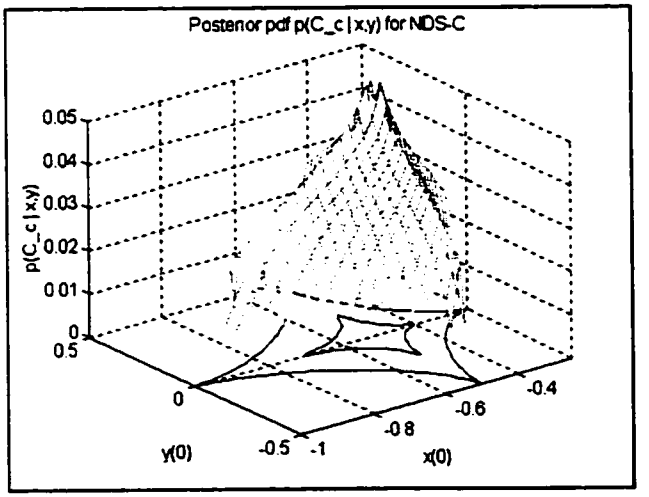


Figure 5.3.1-36 : 3D View of $p(C_C \mid X(0))$

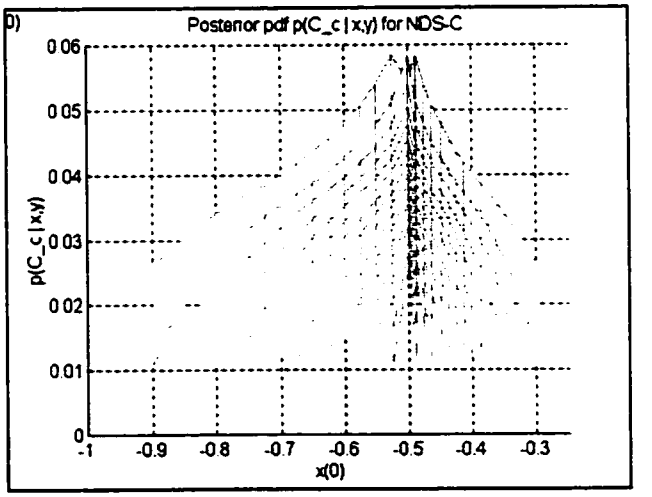


Figure 5.3.1-37 : x-Side View of $p(C_C \mid X(0))$

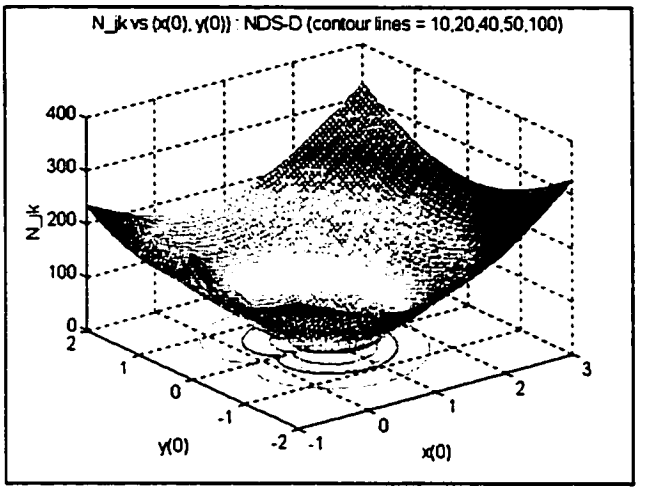


Figure 5.3.1-38 : N_{jk} versus X(0) for NDS-D

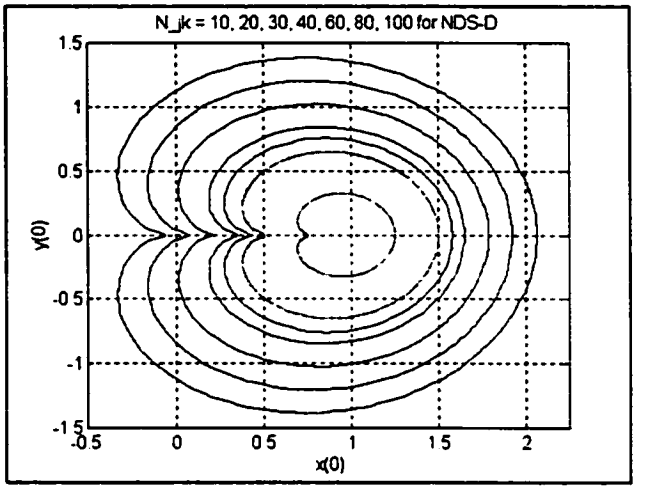


Figure 5.3.1-39 : N_{jk} Contour Lines (NDS-D)

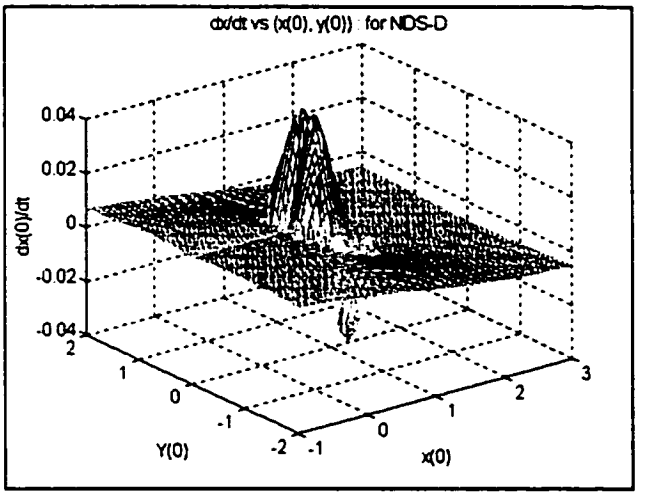


Figure 5.3.1-40 : 3D View of $\Delta H_{D\_x}(X(n))$

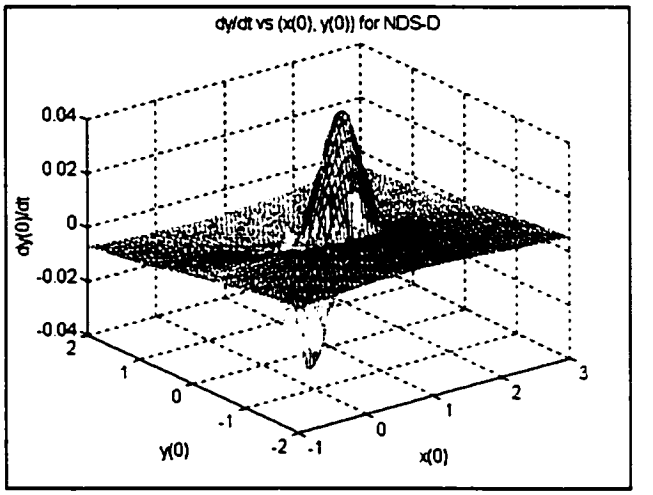


Figure 5.3.1-41 : 3D View of $\Delta H_{D\_y}(X(n))$

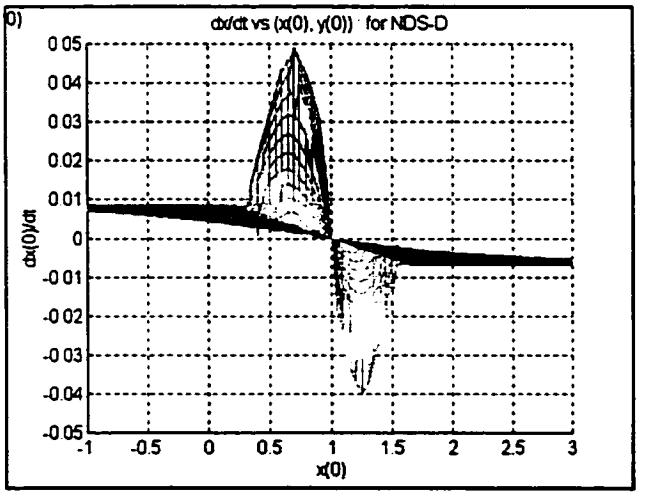


Figure 5.3.1-42 : Side View of $\Delta H_{D\_x}(X(n))$

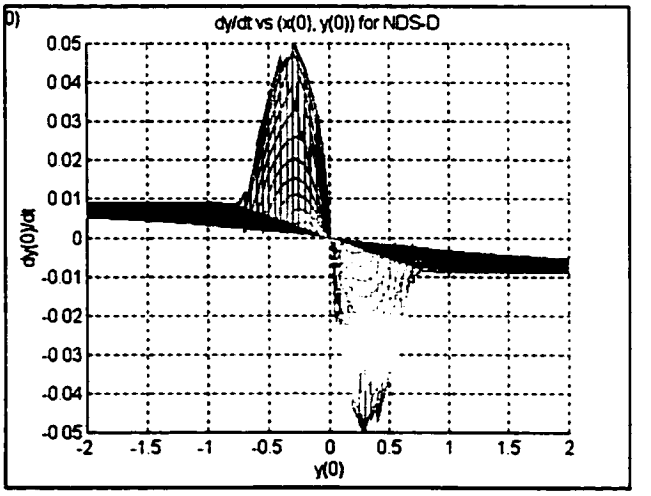


Figure 5.3.1-43 : Side View of $\Delta H_{D\_y}(X(n))$

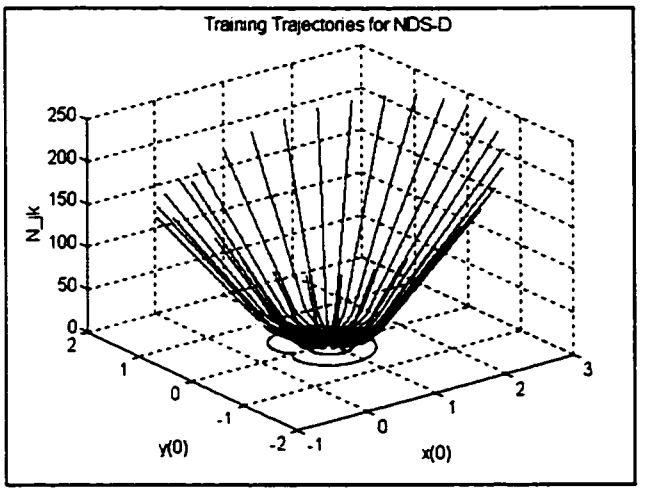


Figure 5.3.1-44 : 3D View of c_D for NDS-D

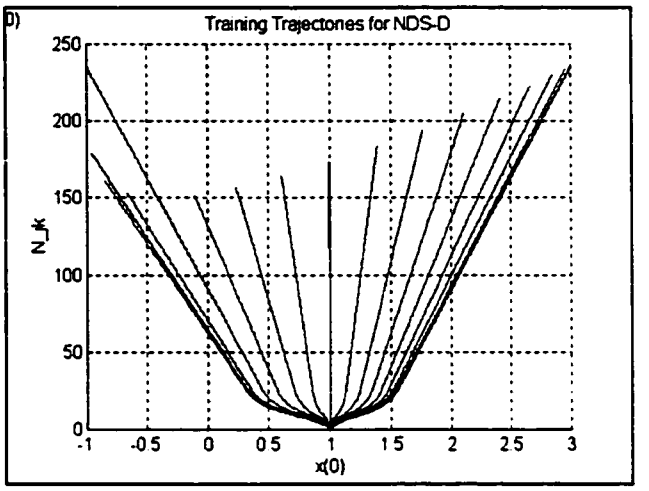


Figure 5.3.1-45 : x-Side View of c_D (NDS-D)

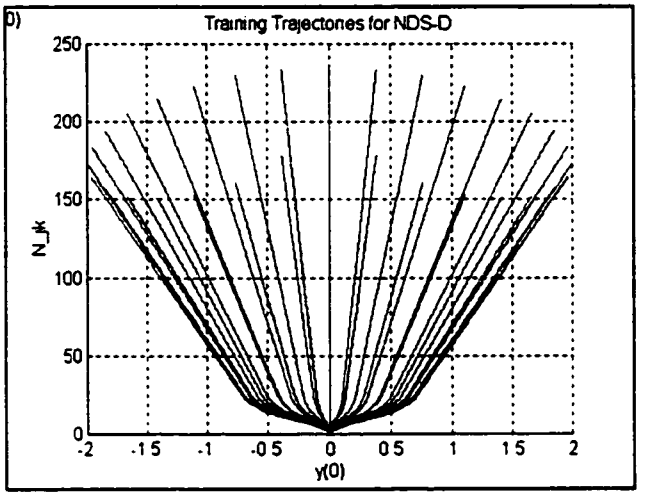


Figure 5.3.1-46 : y-Side View of c_D (NDS-D)

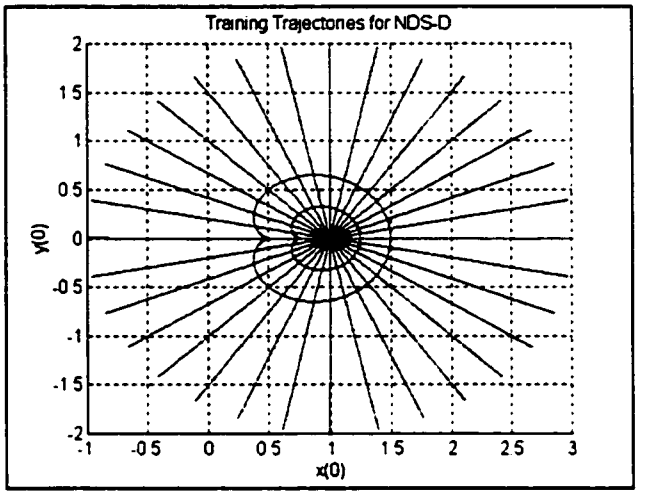


Figure 5.3.1-47 : Top View of c_D (NDS-D)

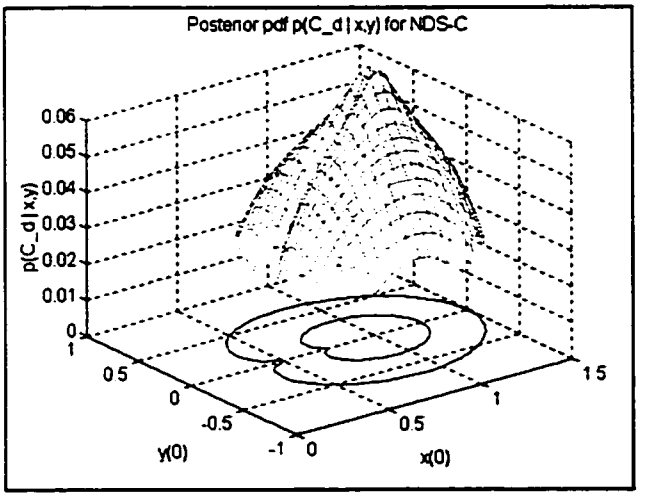


Figure 5.3.1-48 : 3D View of $p(C_D \mid X(0))$

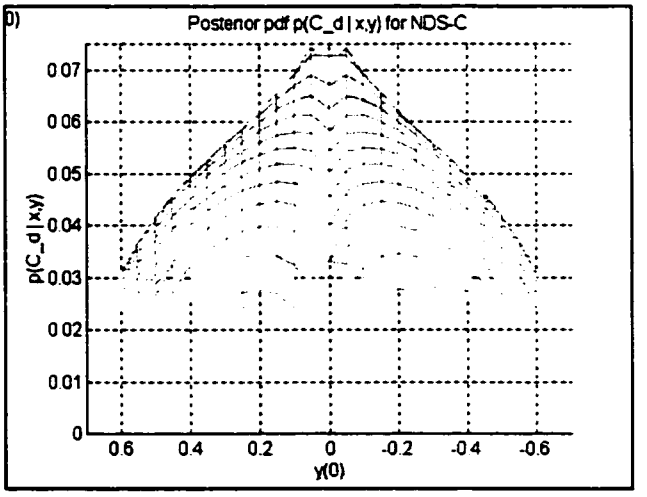


Figure 5.3.1-49 : x-Side View of $p(C_D \mid X(0))$

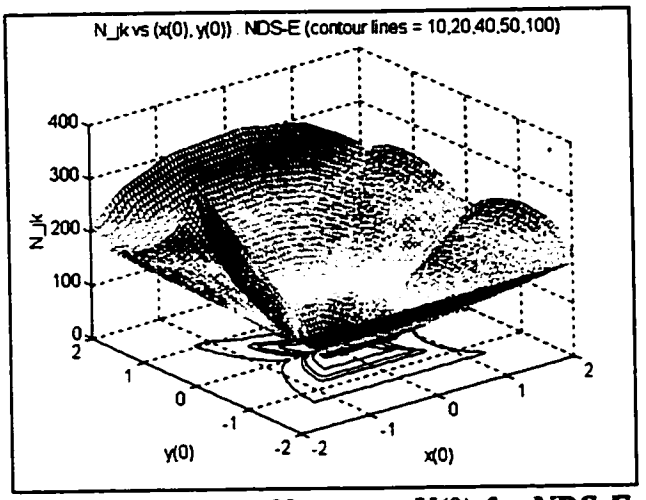


Figure 5.3.1-50 : N_{jk} versus X(0) for NDS-E

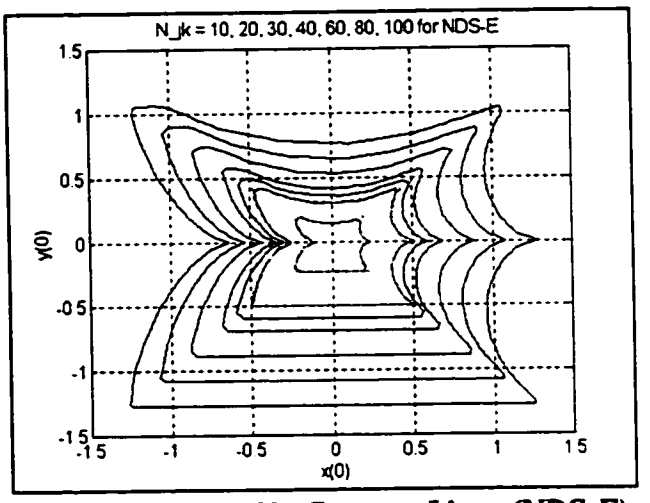


Figure 5.3.1-51 : N_{jk} Contour Lines (NDS-E)

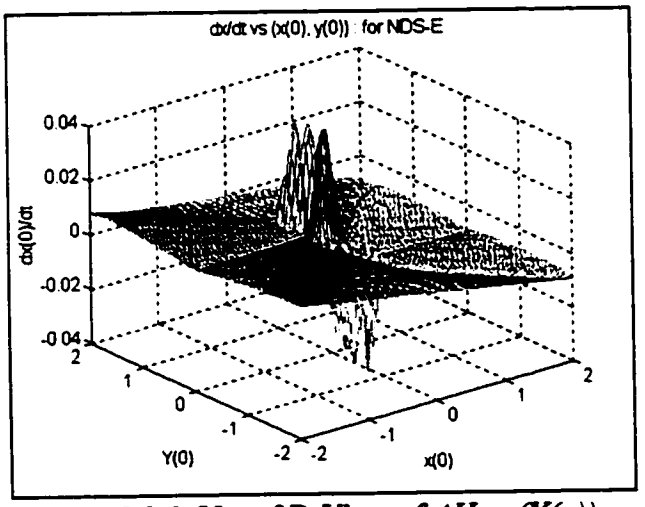


Figure 5.3.1-52 : 3D View of $\Delta H_{E\_x}(X(n))$

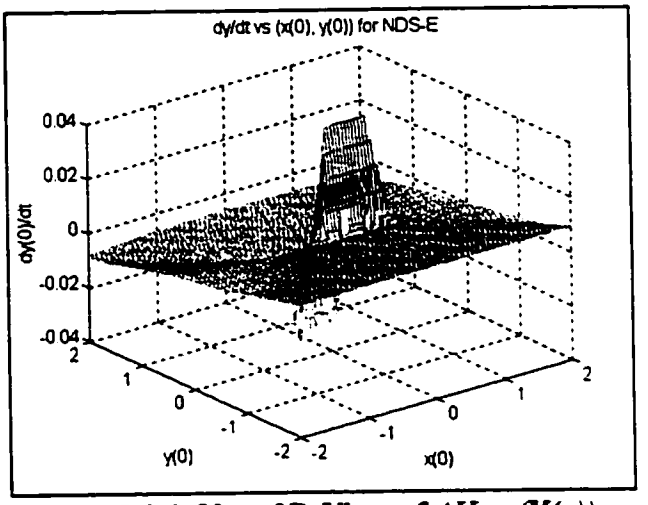


Figure 5.3.1-53 : 3D View of $\Delta H_{E\_y}(X(n))$

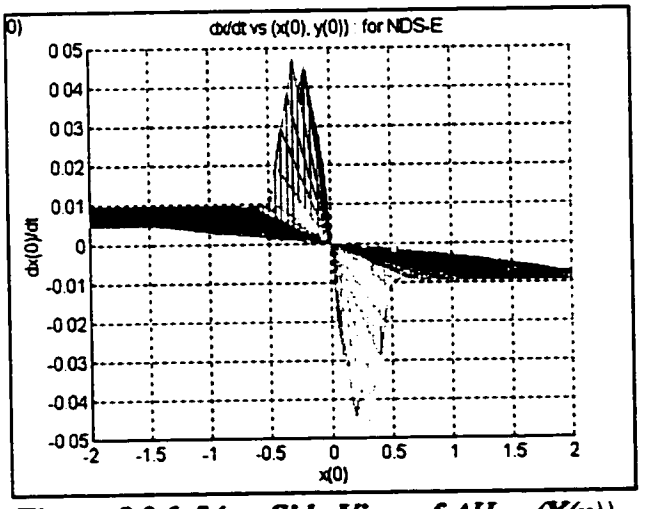


Figure 5.3.1-54 : Side View of $\Delta H_{E\_x}(X(n))$

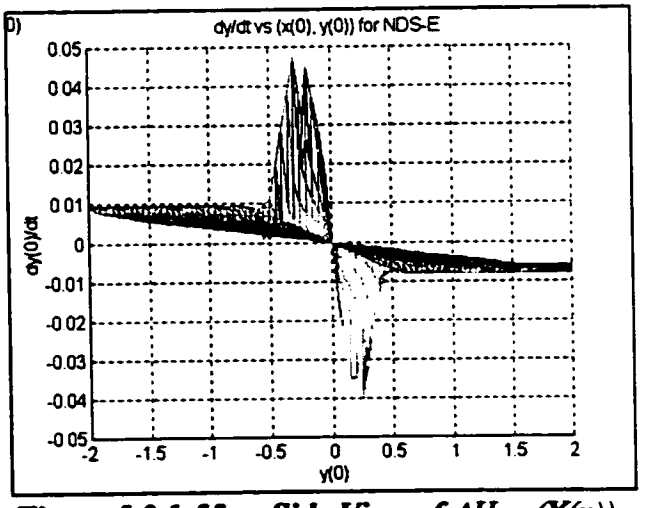


Figure 5.3.1-55 : Side View of $\Delta H_{E\_y}(X(n))$

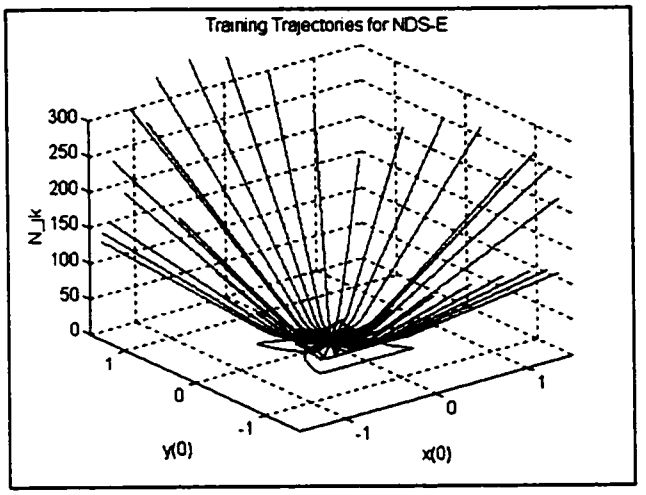


Figure 5.3.1-56 : 3D View of c_E for NDS-E

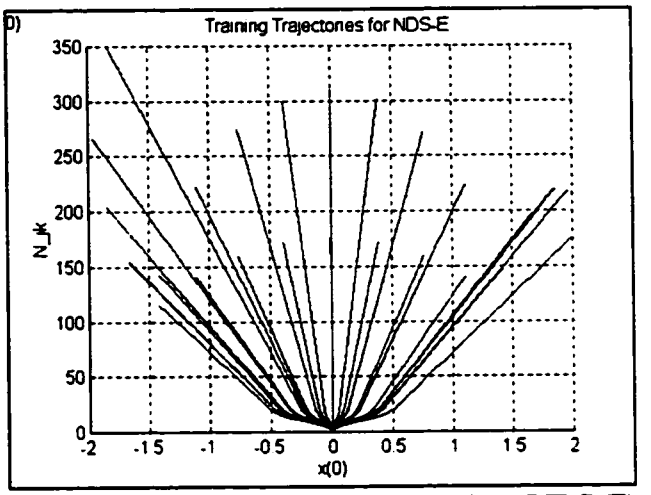


Figure 5.3.1-57 : x-Side View of c_E (NDS-E)

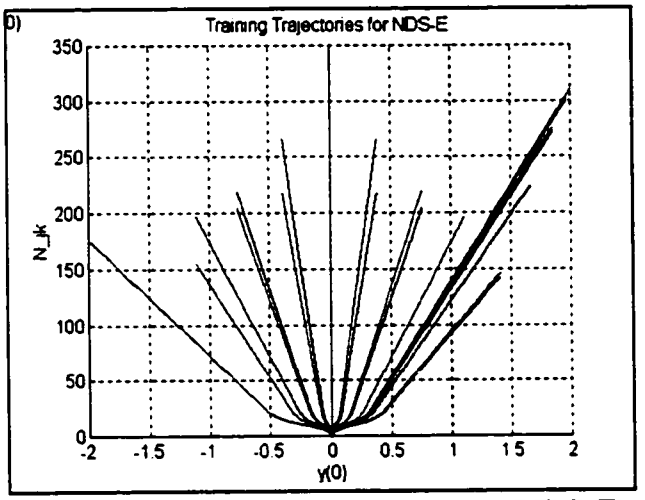


Figure 5.3.1-58 : y-Side View of c_E (NDS-E)

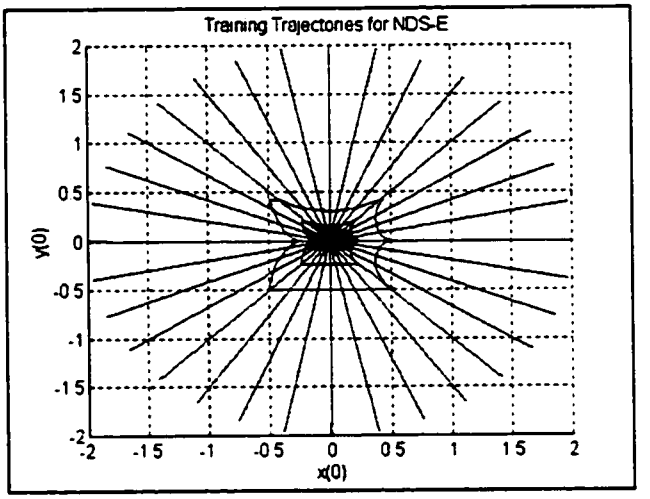


Figure 5.3.1-59 : Top View of c_E (NDS-E)

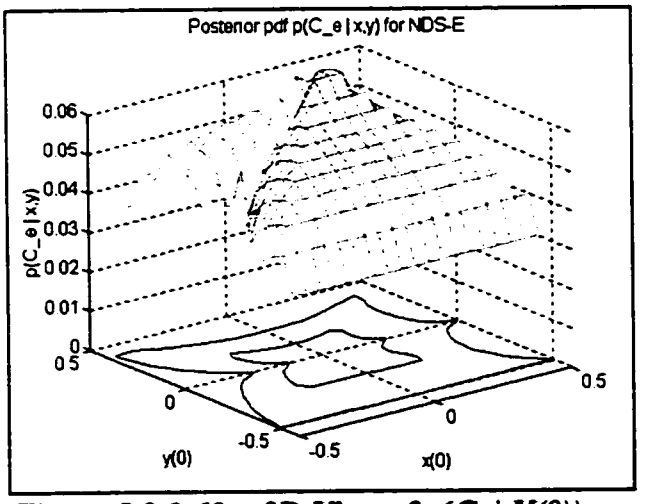


Figure 5.3.1-60 : 3D View of $p(C_E \mid X(0))$

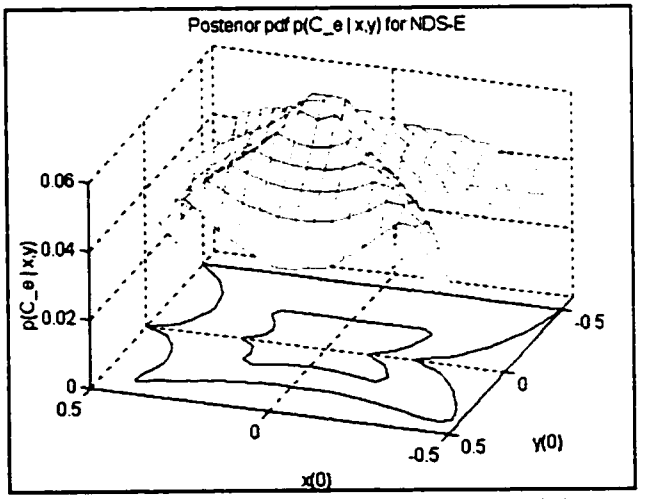


Figure 5.3.1-61 : 3D View of $p(C_E \mid X(0))$

Figures 5.3.1-26 to 5.3.1-37 illustrate the characteristics of NDS-C. The jagged edges in the pdf of Figure 5.3.1-36 are again due to MATLAB interpolation errors. There is in fact a peak to the pdf at the attractor. Figures 5.3.1-38 to 5.3.1-49 illustrate the characteristics of NDS-D. Figures 5.3.1-50 to 5.3.1-61 illustrate the characteristics of NDS-E.

For all of these NDS, the classification boundary B_{IN_j} corresponds to $N^{\epsilon}_j = 20$ while the boundary B_{CV_j} corresponds to $N_{jk} = 10$. For the five NDS, the training data files consist of 3351 to 6454 training vectors, where each vector has four coordinates : *[{x(n), y(n)}, {x(n+1), y(n+1)}]* which represent the current input and the desired output. The NN is therefore required to learn the iterated mapping function $\boldsymbol{H(X(n))}$ over the input region.

5.3.2 Reduction in Learning Task Complexity By Orthogonalizing $H(X(n))$

In this section, we illustrate how the learning task complexity has been reduced by orthogonalizing the iterated maps $\boldsymbol{H(X(n))}$ of each subnet.

It was indicated before that each subnet-*j* is made to specialize in only one task : determine if an input $X_k(0)$ is IN class $\mathcal{C}_j$ or OUT of it. It was shown in section 4.6.1.1 (page 139) that this is essentially equivalent to orthogonalizing the mapping functions of the subnets.

This specialization process was described in sections 2.4.2 and 2.4.3 of chapter 2 (page 43) and in section 4.7.2 (page 147) as a mechanism through which the complexity of the learning task is reduced. This can be shown to be true by considering a *global approximation approach* instead : consider the case where one neural network is required to learn all five NDS with their respective attractors and trajectories. In such a case, the global mapping function $\Delta H_y(X(n))$ which that NN must learn for the 2D example introduced in the previous section would has the appearance shown in Figure 5.3.2-1.

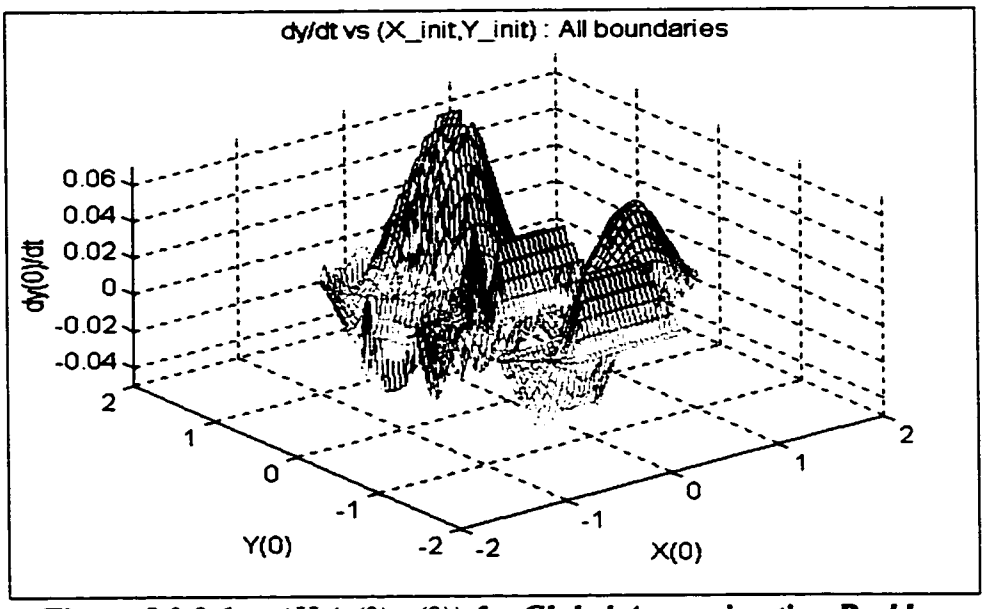


Figure 5.3.2-1 : $\Delta H_y(x(0),y(0))$ for Global Approximation Problem

The function $\Delta H_x(X(n))$ is equally complex. Compare Figure 5.3.2-1 to Figure 5.3.1-5 (page 162), which is $\Delta H_{A\_y}(X(n))$ for the NDS-A specializing in class $\mathcal{C}_A$. The complexity of $\Delta H_y(X(n))$ above is clearly much higher than that of $\Delta H_{A\_y}(X(n))$ (e.g., in terms of the complexity function s given by equation E_4.7.1-2 on page 146). This is also true for the other $\Delta H_{j\_y}(X(n))$ shown in Figures 5.3.1-17, 5.3.1-29, 5.3.1-41 and 5.3.1-53.

As explained in section 4.7.1 (page 145) and through equation E_4.7.1-1, the additional complexity in the global approximation approach would result in the need for a much larger NN and would also result in a significantly longer training time requirement. Hence, it should be apparent from these figures that the mechanism of 'specialization' (or orthogonalization) used in the RTA NN model does indeed reduce the learning task complexity significantly.

5.4 Results of Training the Subnets with the 2D Classification Problem Data

In this section, the results of the training sessions are described and analyzed briefly. Section 5.4.1 describes the scope of training performed and the different net structures utilized. Section 5.4.2 describes the results of training these various nets on the data for

regions A to E. In section 5.4.3 various techniques used to try to improve the mapping ability of these nets are discussed. Section 5.4.4 provides an overall analysis and conclusions based on these results.

5.4.1 Scope of Training Sessions and Neural Net Structures Utilized

The training sessions for each data set involved the two basic net structures shown in Figure 5.4.1-1 We refer to these as *coupled* and *uncoupled* structures.

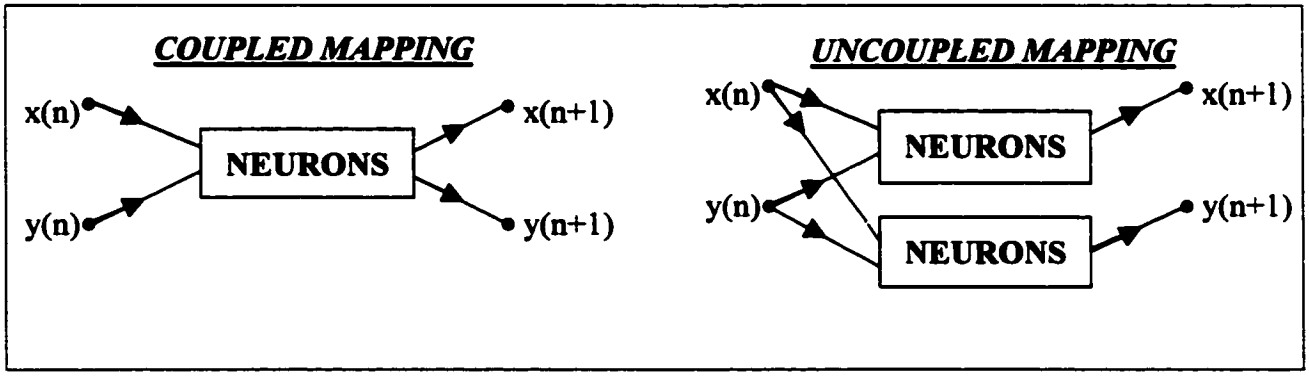


Figure 5.4.1-1 : Coupled and Uncoupled Structures for the RTA NN Subnets

In the 'coupled' structure, the two mapping functions $\Delta H_{j\_x}(x(n),y(n))$ and $\Delta H_{j\_y}(x(n),y(n))$ (e.g., as shown in Figures 5.3.1-4, 5.3.1-5 on page 162) are learned by the same network of neurons. Hence, each neuron is involved in both mapping functions. However, in this 2D classification problem, ΔH_{j_y} happens to be independent of ΔH_{j_x}. As a result, these functions can be taught separately to the NN. This has been done in the 'uncoupled' structure shown above, where a group of neurons is involved in only one of the two mapping functions.

In Figure 5.4.1-1, the box marked 'neurons' may consist of one or more layers of neurons which can be RBF, LR-RBF, CR-RBF, McCulloch-Pitts neurons, or combinations of these. The RBF neuron types are either spread out uniformly over the input space or spread out with a higher density in R_{IN_j}. The McCulloch-Pitts neurons start off with randomized weights in accordance with heuristic rules proposed by reference [26]. The net sizes vary between 70 and 450 neurons.

For each training session, at least 10 instances of a given net implementation are trained on the data for one NDS, where each instance may have different initial weights or RBF parameters. Most implementations involved combinations of RBF neuron types and McCulloch-Pitts neurons in a ratio of roughly 6 to 1. Different training sessions involved different learning rate or momentum parameters, and different termination conditions. Both coupled and uncoupled net structures were tested on each data set.

Overall, several hundred nets were trained over a period of several months.

5.4.2 Results of the Training Sessions

In this section, the results of the training sessions are described briefly.

Generally, the trained neural nets achieved a poor match of the desired $\Delta H(X(n))$ mapping functions. Figures 5.4.2-1 to 5.4.2-12 illustrate the mapping achieved by a trained neural net labeled as D2A23 for the data of NDS-A, versus the 'desired' mappings. Odd-numbered figures are the 'achieved' results while even-numbered figures are the desired results. Figures 5.4.2-13 to 5.4.2-16 show some of the best results achieved for all nets trained on NDS-B data. Similarly, Figures 5.4.2-17 to 5.4.2-20 show results for NDS-C data, Figures 5.4.2-21 to 5.4.2-24 are for NDS-D data and Figures 5.4.2-25 to 5.4.2-28 are for NDS-E data. The poor match between the achieved contour lines $\underline{N}^{\epsilon}_{j} = 20$ and the desired boundary $B_{IN\,j}$ for each NDS implies that many classification errors will be made by these NN.

The best neural nets involved simple RBF, LR-RBF or CR-RBF neurons combined with about a dozen McCulloch-Pitts neurons (for a total of about 70 to 100 neurons in one hidden layer). The coupled and uncoupled structures seemed to perform at about the same level of ability. Generally, the CR-RBF and LR-RBF produced better approximations of the desired functions than simple RBF but the former were harder to teach because of potential stability problems. This subject is discussed in more detail in Appendix A (starting on page 227).

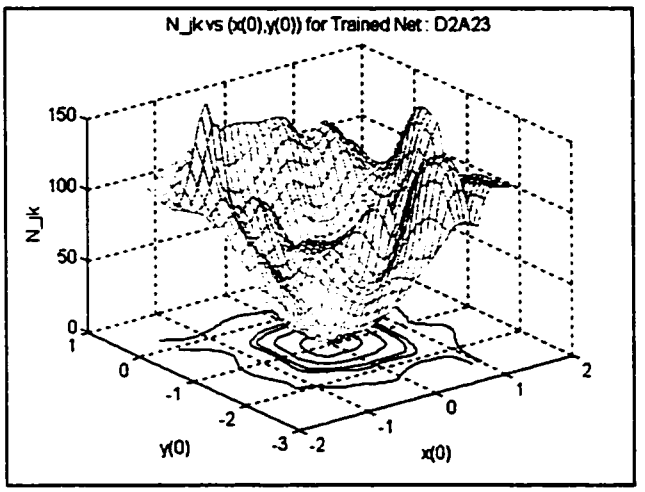


Figure 5.4.2-1 : Achieved N_{jk} Map (NDS-A)

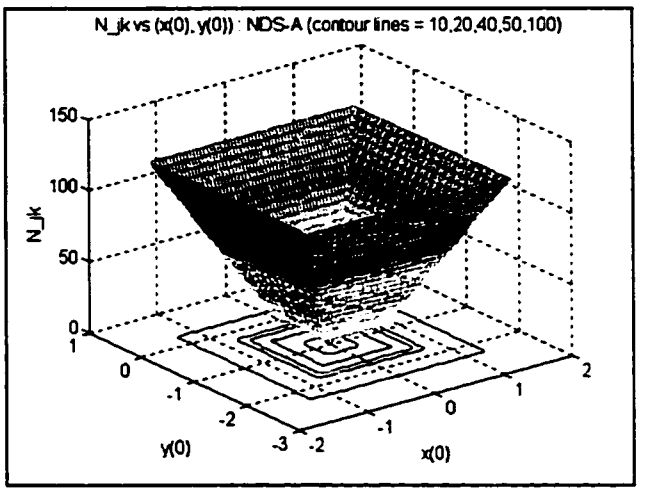


Figure 5.4.2-2 : Desired N_{jk} Map (NDS-A)

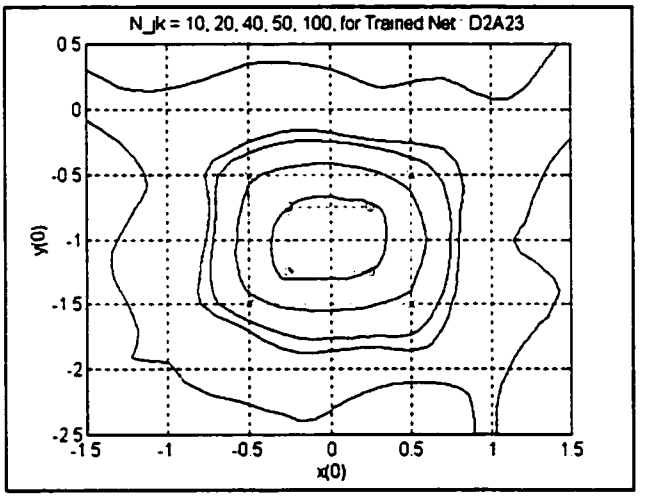


Figure 5.4.2-3 : Achieved Contour Lines N_{Ak}

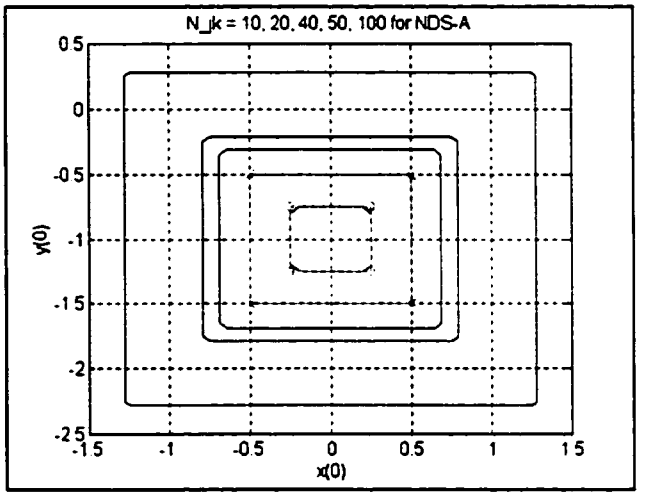


Figure 5.4.2-4 : Desired Contour Lines N_{Ak}

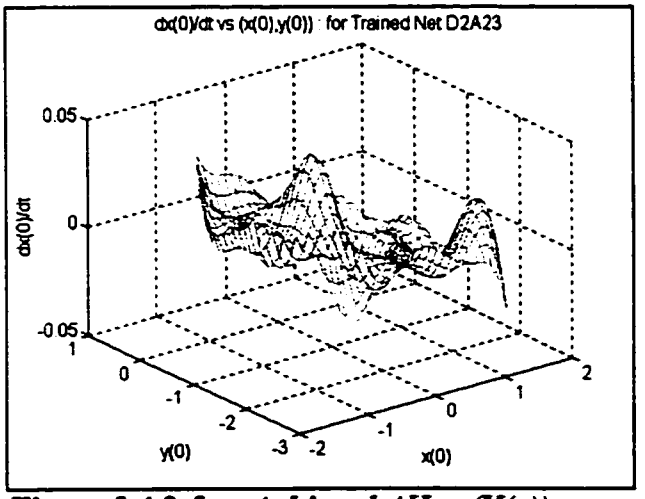


Figure 5.4.2-5 : Achieved $\Delta H_{A\_x}(X(n))$

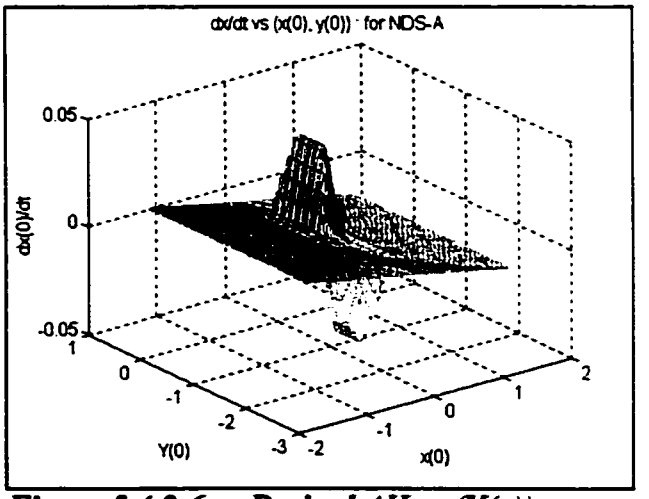


Figure 5.4.2-6 : Desired $\Delta H_{A\_x}(X(n))$

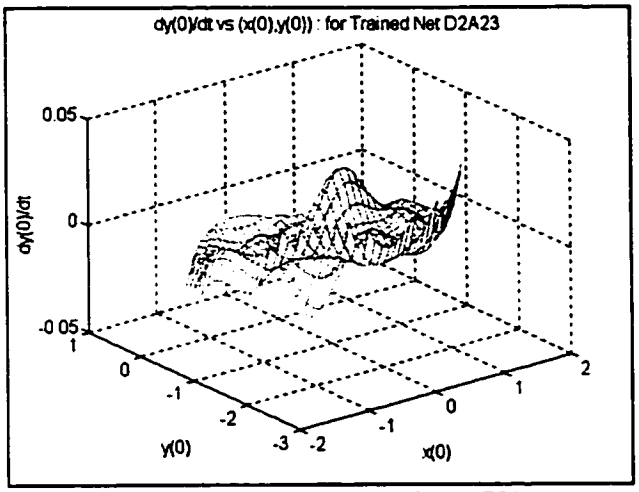


Figure 5.4.2-7 : Achieved $\Delta H_{A\_y}(X(n))$

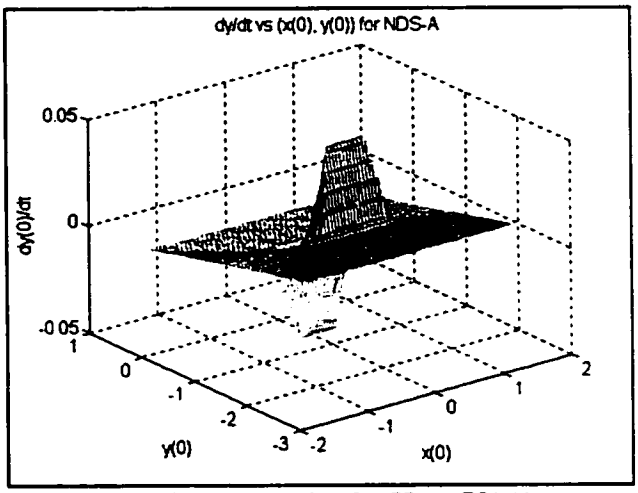


Figure 5.4.2-8 : Desired $\Delta H_{A\_y}(X(n))$

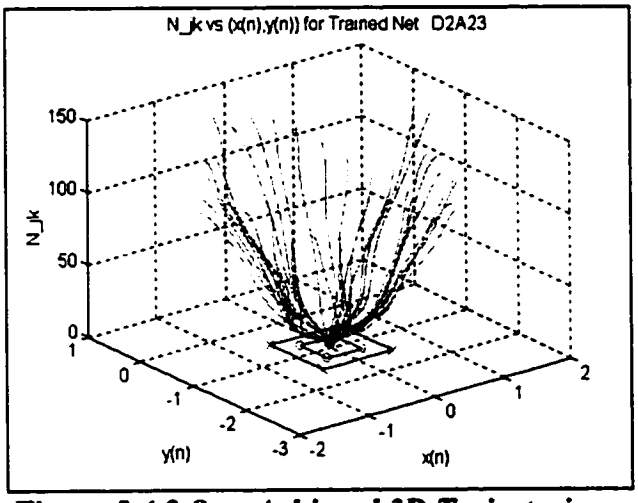


Figure 5.4.2-9 : Achieved 3D Trajectories c_A

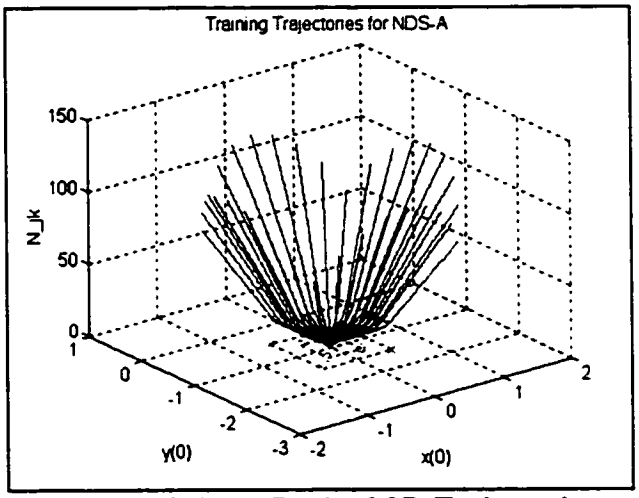


Figure 5.4.2-10 : Desired 3D Trajectories c_A

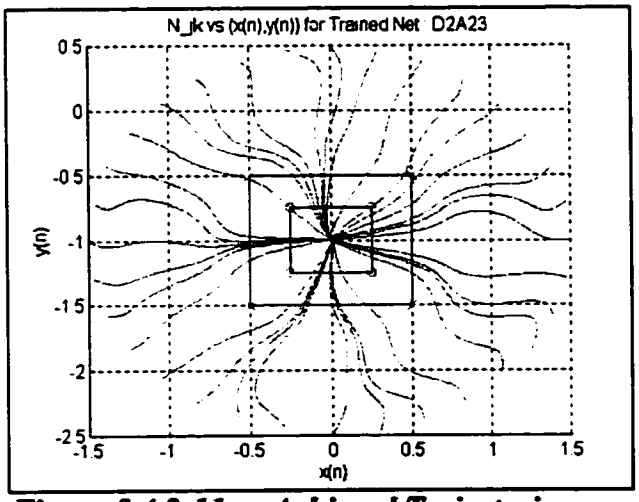


Figure 5.4.2-11 : Achieved Trajectories c_A

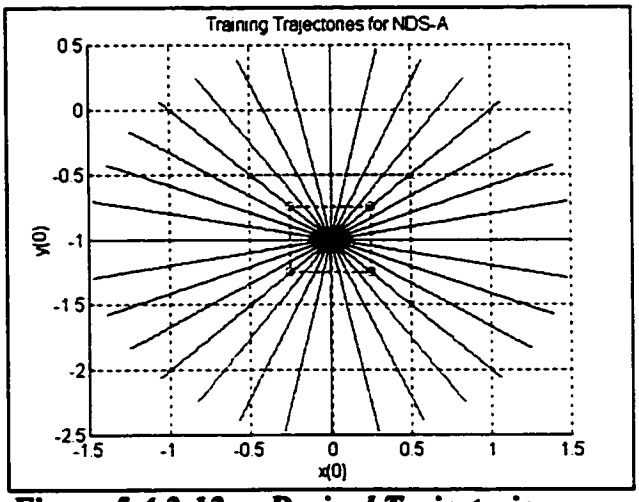


Figure 5.4.2-12 : Desired Trajectories c_A

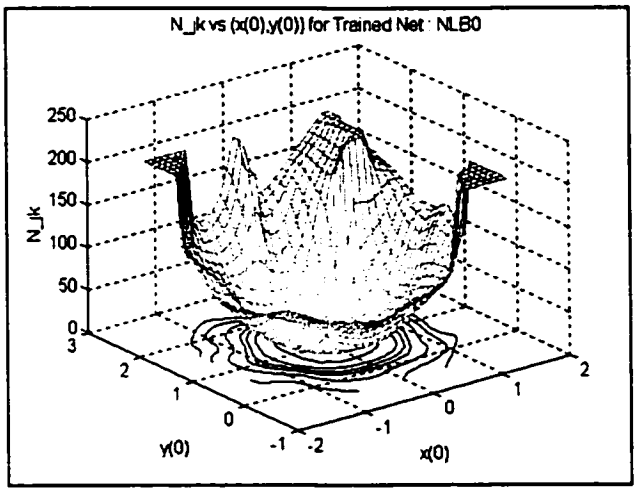


Figure 5.4.2-13 : Achieved N_{jk} Map (NDS-B)

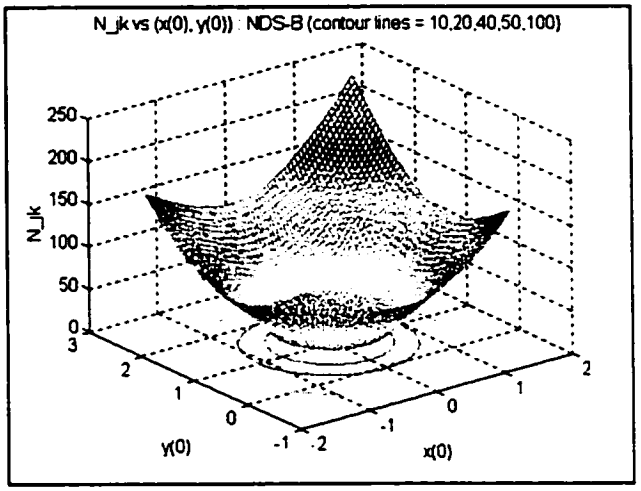


Figure 5.4.2-14 : Desired N_{jk} Map (NDS-B)

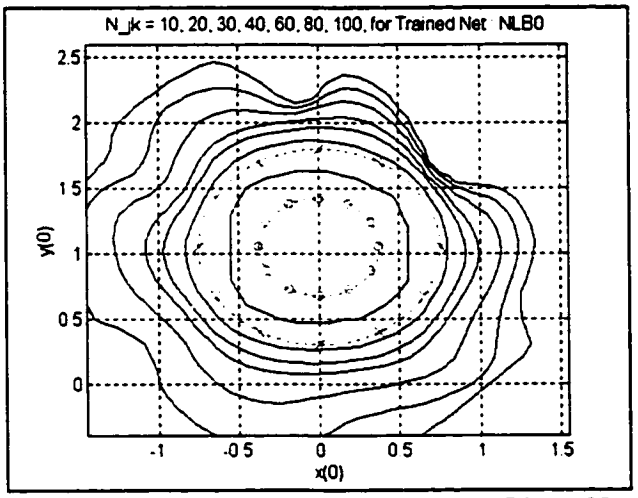


Figure 5.4.2-15 : Achieved Contour Lines N_{Bk}

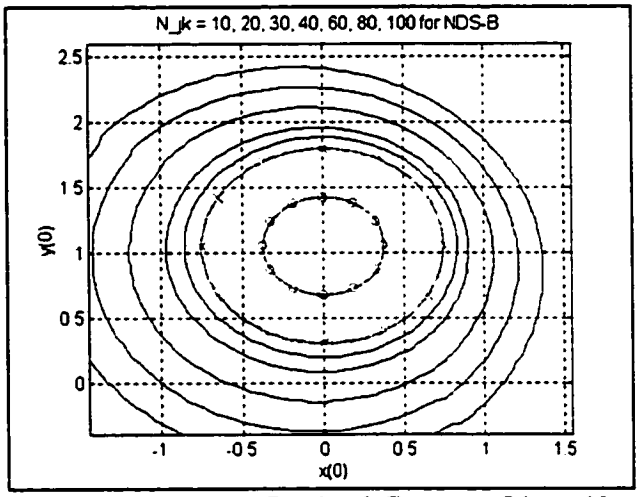


Figure 5.4.2-16 : Desired Contour Lines N_{Bk}

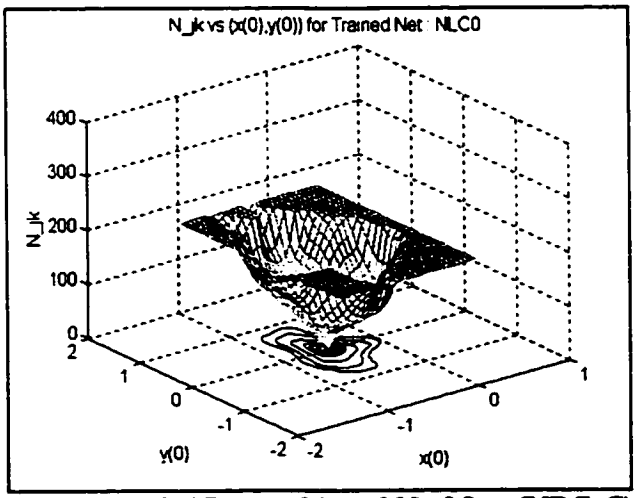


Figure 5.4.2-17 : Achieved N_{jk} Map (NDS-C)

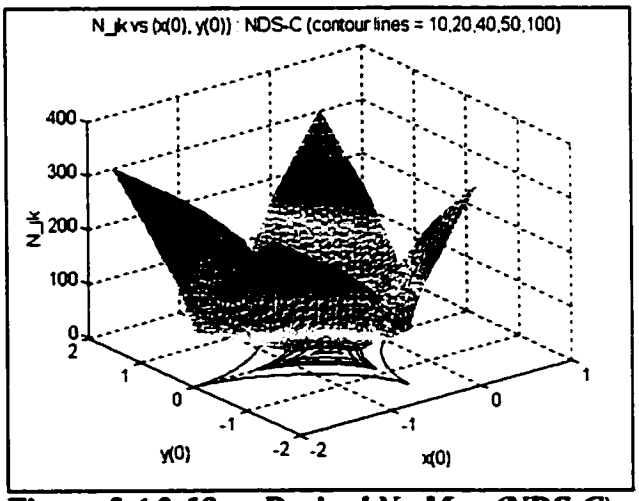


Figure 5.4.2-18 : Desired N_{jk} Map (NDS-C)

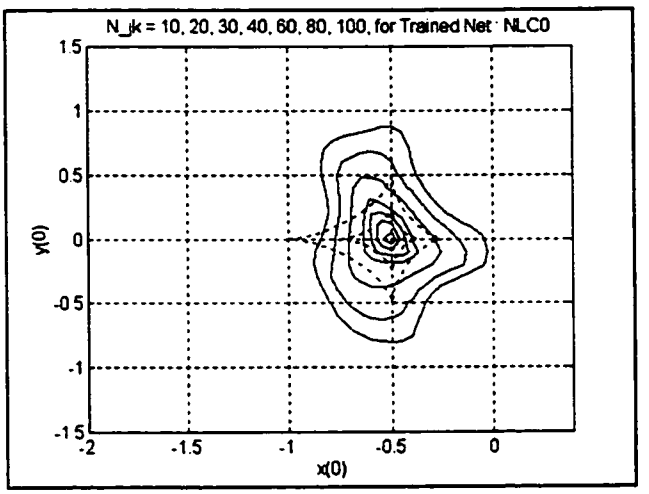


Figure 5.4.2-19 : Achieved Contour Lines N_{Ck}

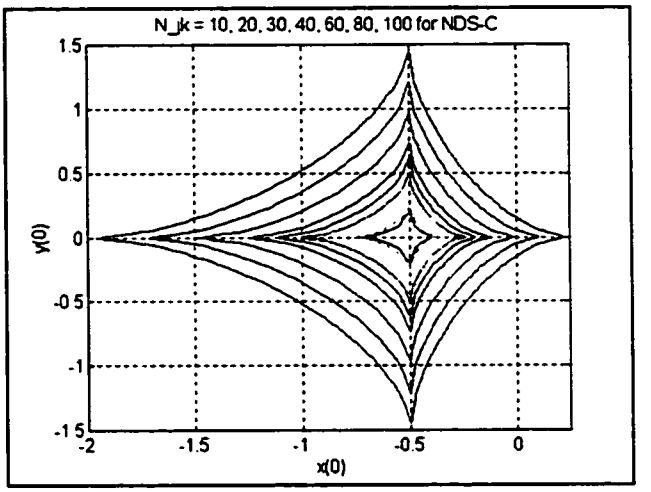


Figure 5.4.2-20 : Desired Contour Lines N_{Ck}

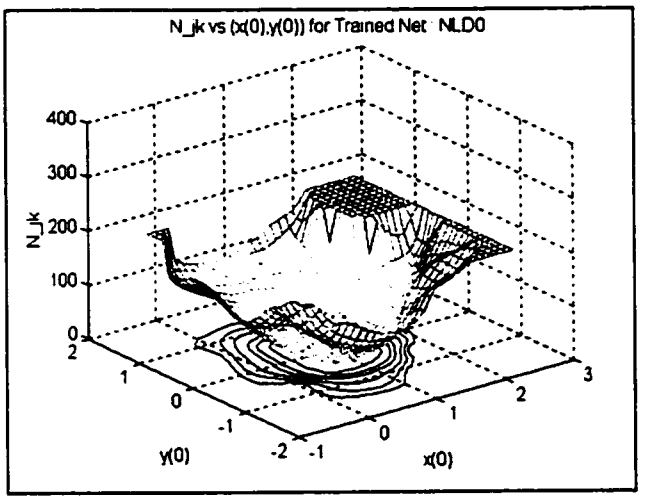


Figure 5.4.2-21 : Achieved N_{jk} Map (NDS-D)

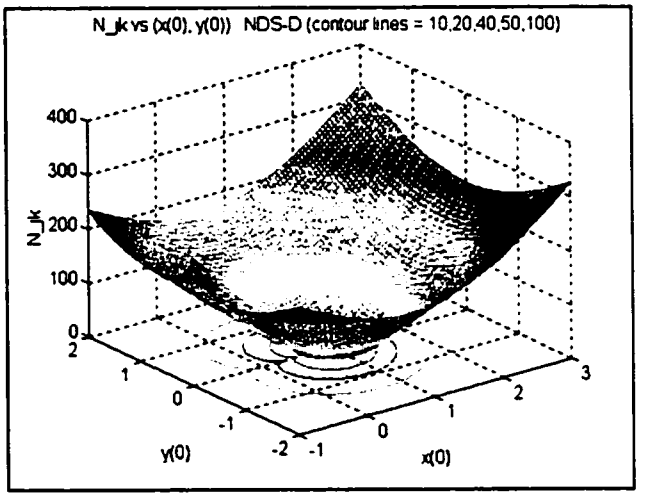


Figure 5.4.2-22 : Desired N_{ik} Map (NDS-D)

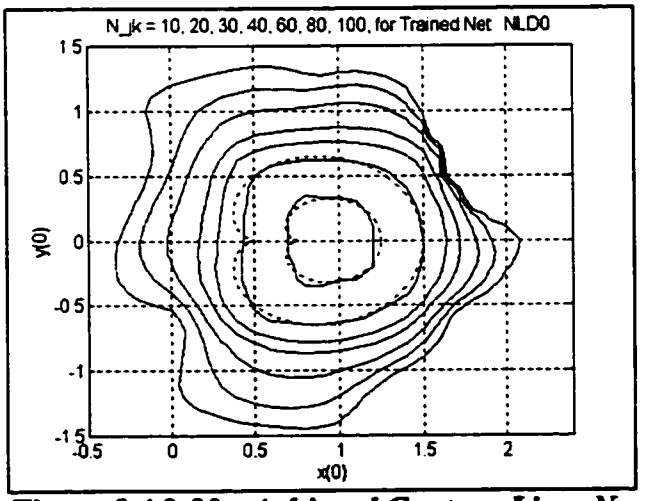


Figure 5.4.2-23 : Achieved Contour Lines N_{Dk}

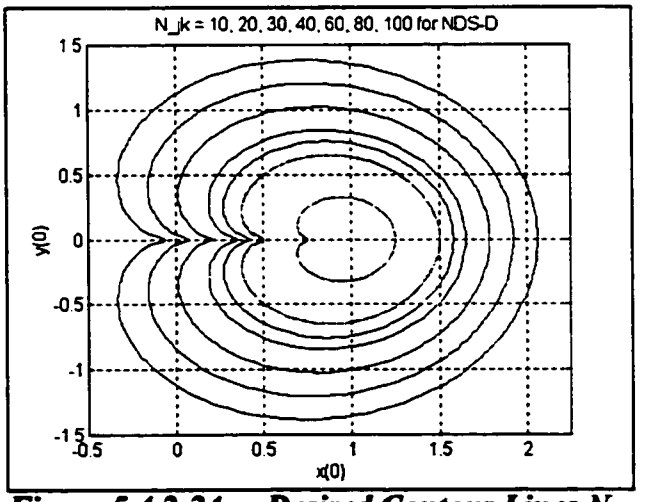


Figure 5.4.2-24 : Desired Contour Lines N_{Dk}

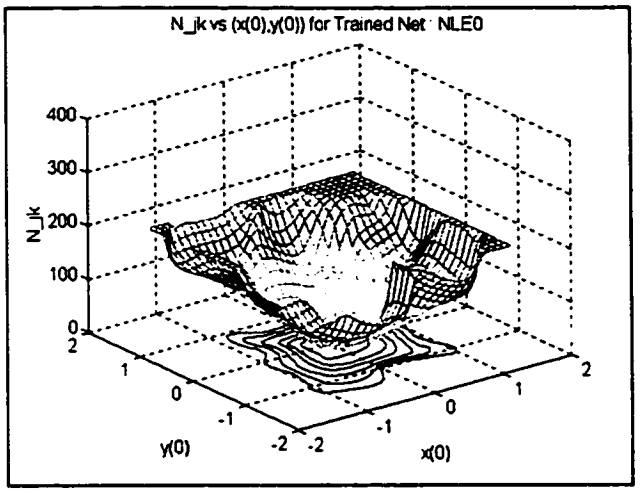


Figure 5.4.2-25 : Achieved N_{jk} Map (NDS-E)

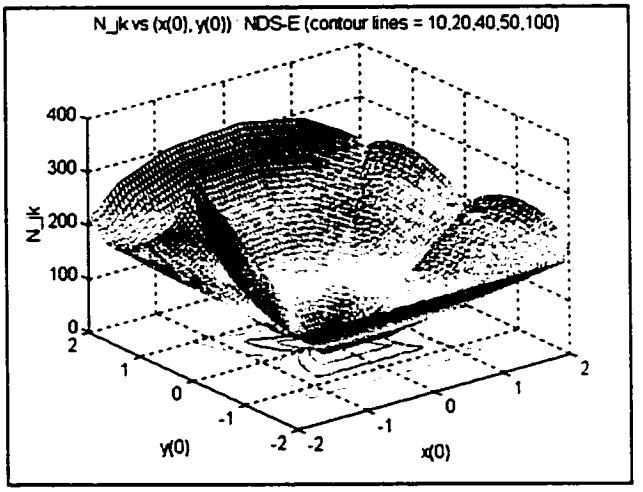


Figure 5.4.2-26 : Desired N_{jk} Map (NDS-E)

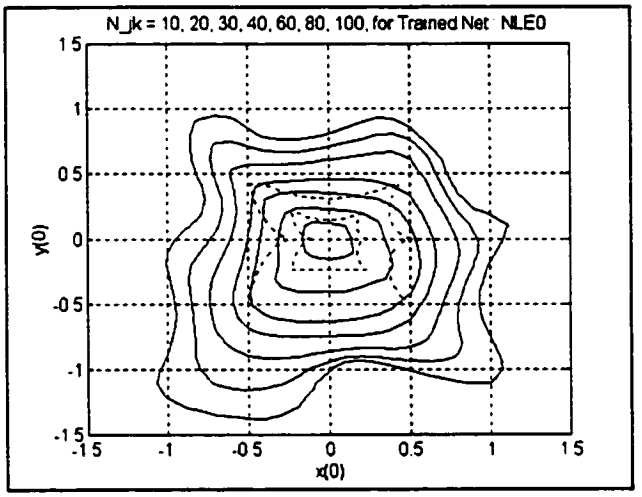


Figure 5.4.2-27 : Achieved Contour Lines N_{Ek}

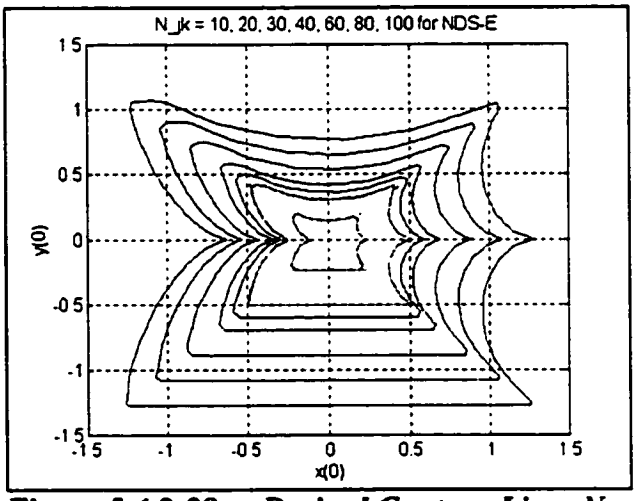


Figure 5.4.2-28 : Desired Contour Lines N_{Ek}

Feedforward nets made up of only McCulloch-Pitts neurons were NOT able to learn any of the required mappings, regardless of the number of neurons or layers involved. This is probably due to the fact that these MLP perform only global approximations and are unable to capture the local features of the desired iterated mapping functions $\Delta H(X(n))$. Reference [19] discusses this problem at length.

5.4.3 Techniques Used to Improve the Mapping Ability of NN

In an attempt to improve the accuracy of the mappings achieved by the neural nets, the following changes to the training sessions were implemented:

1. The size of the neural nets were increased from 70-100 up to a maximum of 450 neurons; and/or

2. The training data files were doubled in size, with a higher point density near the classification boundary B_{IN_j}. This was combined with a modified error function for backpropagation training, which applied 10 times more weight to errors around the boundary B_{IN_j} than to errors elsewhere.

The first change was designed to increase the number of free parameters to allow greater flexibility to learn the required mapping function $\Delta H(X(n))$. The second change was designed to improve the match at the classification boundary since this is a where most classification errors will occur.

Training sessions involved nets with various types of neurons and with the coupled and uncoupled structures described in section 5.4.1. Results were sometimes *slightly* better around the boundary B_{IN_j} but often at the expense of increased mapping error outside the region R_{IN_j}. Figure 5.4.3-1 illustrates the overall best match achieved at a boundary, using a net with 450 simple RBF neurons in an uncoupled structure. A comparison of that figure with Figures 5.4.2-3 and 5.4.2-4 (page 176) indicates that the better match at the boundary has been achieved at the expense of less accurate contour lines elsewhere.

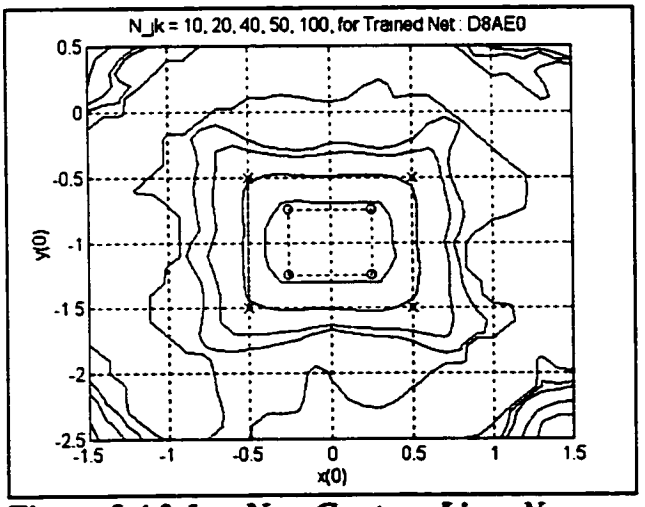


Figure 5.4.3-1 : New Contour Lines N_{Ak}

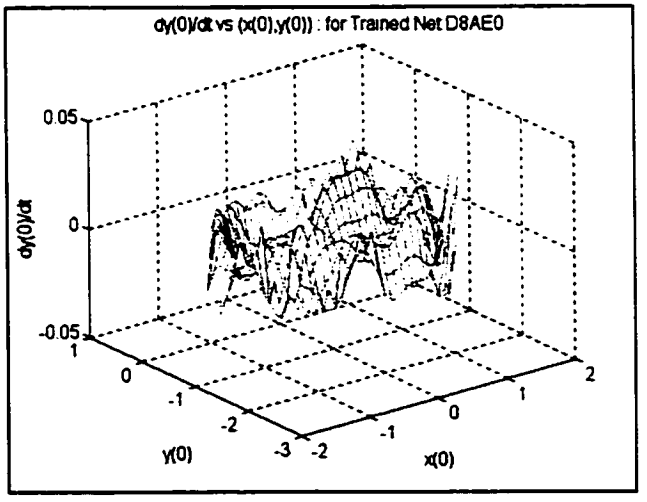


Figure 5.4.3-2 : New Achieved $\Delta H_{A\_y}(X(n))$

Figure 5.4.3-2 shows the achieved mapping function $\Delta H_{A\_y}(x(0),y(0))$. A comparison with Figures 5.4.2-7 and 5.4.2-8 (page 177) also indicates that there is a slightly better match of the desired function in some regions but not everywhere.

For more complex boundaries such as region C and E, none of the trained nets ever came close to achieving an acceptable match.

5.4.4 Overall Conclusions Based on the Results of the Training Sessions

Results of section 5.4.2 indicate that the NN were not able to match the classification boundaries well. This was particularly true of the more complex boundaries (e.g., NDS-C or E). Attempts to improve the performance by increasing the size of the neural net and/or by putting more training emphasis at the classification boundary have produced a very modest improvement in the case of NDS-A but were not sufficient to achieve an acceptable match to the boundaries of more complex regions.

The fact that a significantly larger NN did not result in a significant improvement in the mapping ability of the NN suggests that these NN may have reached the limit of their learning ability. Perhaps other net structures and a more sophisticated algorithm could improve the mapping accuracy to a certain degree. However, it is NOT considered likely that any other net structure or algorithm would allow us to achieve a very accurate mapping of the desired $\Delta H(X(n))$ for regions A, C, D or E.

To understand our rationale for this conclusion, examine again some of the desired functions $\Delta H_{j\_x}(X(n))$ and $\Delta H_{j\_y}(X(n))$ (shown in Figures 5.3.1-4 to 7 (page 162), Figures 5.3.1-16 to 19, Figures 5.3.1-28 to 31, Figures 5.3.1-40 to 43 and Figures 5.3.1-52 to 55). Each one of these mapping functions has some narrow peaks, steep slopes and (occasionally) corners with abrupt changes in slope. These characteristics make each one of these functions fairly complex. Although the complexity of each such function has been reduced significantly as a result of the orthogonalization process utilized in the RTA

NN model (e.g., compare them to Figure 5.3.2-1 on page 173), these are still too complex for neural networks of realistic size and complexity to learn to a high degree of precision.

The fundamental problem is one of excessive mapping function complexity; it is NOT an inherent limitation of the net structure used for training. Hence, it is unlikely that other net architectures described in section 2.5 of chapter 2 would perform significantly better. These nets are also susceptible to the curse of dimensionality (which is really a curse of complexity in our case). As a result, these nets will probably not perform much better.

5.5 Simplification of the Learning Task Complexity by Redefining the Attractor

Since the complexity of the desired mapping functions $\Delta H(X(n))$ is still too high for these NN to learn them adequately, it is essential to somehow reduce the complexity of these functions. Fortunately, this can be accomplished through one of the mechanisms introduced in chapter 2 : redefining the attractor Γ_j from a point to a region.

Section 5.5.1 describes how to define the *attractor region* Γ^R_j in such a way that the subnets of the RTA NN model can be made to match the desired classification boundary to a high degree of accuracy. Section 5.5.2 illustrates how this approach achieves the promised reduction in learning task complexity and results in the NN matching the classification boundary B_{IN_j} to a high degree of accuracy.

5.5.1 Redefining the Attractor as a Region for the Subnets of the RTA NN

Consider again Figure 5.4.2-3 (page 176) which shows the achieved contour lines $\underline{N}_{Ak}$ of NN D2A23 trained on NDS-A data (note that the underline signifies that these are the 'achieved' results as opposed to the 'desired' ones). The contour line for $\underline{N}^{\epsilon}_{A} = 20$ is such that points on the desired boundary B_{IN_A} are either inside that contour line (meaning they will converge to the attractor too quickly) or outside that contour line (meaning they will converge to the attractor too slowly). It is therefore necessary to re-adjust the achieved boundary of $\underline{N}^{\epsilon}_{A} = 20$ until it matches the desired one. This process is described below.

Let $\underline{N(B_{IN\_j})_{MIN}}$ represent the minimum number of iterations required by any of the points $X_k(0)$ on the desired boundary B_{IN_j} to reach the point attractor Γ_j. In mathematical terms, we can define this parameter as follows:

$$\underline{N}(B_{IN\_j})_{MIN} = \min\{\underline{N}_{jk}\}\forall \bar{X}_k(0) \in B_{IN\_j} \qquad \text{(E\_5.5.1-1)}$$

Let us define a number of iterations $N_{j\_\Gamma\_REGION} \leq \underline{N(B_{IN\_j})_{MIN}}$ such that <u>all</u> points on the boundary B_{IN_j} will be iterated by exactly $N_{j\_\Gamma\_REGION}$ iterations and the final point on the trajectory $X_k(n = N_{j\_\Gamma\_REGION})$ will become a point on the boundary of the attractor region Γ^R_j. If we do this for all points on the boundary B_{IN_j}, the corresponding attractor region Γ^R_j is defined by all the end points $X_k(n = N_{j\_\Gamma\_REGION})$. Mathematically, this can be expressed as follows:

$$\Gamma^R_j = \{\bar{X}_k(n = N_{j\_\Gamma\_REGION})\}\forall \bar{X}_k(0) \in B_{IN\_j} \qquad \text{(E\_5.5.1-2)}$$

In practice, a finite number of points on the boundary B_{IN_j} are iterated and line segments connecting the adjacent end points $X_k(n = N_{j\_\Gamma\_REGION})$ approximate the attractor region. In addition, $N_{j\_\Gamma\_REGION}$ is usually made smaller than $\underline{N(B_{IN\_j})_{MIN}}$ by a few iterations to ensure that none of the end points $X_k(n = N_{j\_\Gamma\_REGION})$ actually reach the point attractor Γ_j. This ensures that the point attractor is fully contained within the region attractor Γ^R_j (i.e., Γ_j does NOT lie on the boundary of Γ^R_j). This is done only to simplify the software implementation of the distance object in the RTA NN model (i.e., function box $D_j(\Gamma^R_j, Y_j(n))$ shown in Figure 4.6-1 on page 136).

With the attractor redefined as a region, the distance object $D_j(\Gamma^R_j, Y_j(n))$ now measures the <u>minimum</u> distance between the current NN iterate $Y_j(n)$ and the boundary of the region attractor Γ^R_j. But with the region boundary of Γ^R_j defined as per equation E_5.5.1-2, it follows that <u>all</u> points on the desired classification boundary B_{IN_j} will reach the boundary of Γ^R_j in <u>exactly</u> $N_{j\_\Gamma\_REGION}$ iterations. In order to ensure that <u>all</u> points on B_{IN_j} finish the race in exactly N^{ϵ}_j iterations, it is then only necessary to add a delay of $N^{\epsilon}_j - N_{j\_\Gamma\_REGION}$ before NN-j is declared the winner. This delay begins after

$D_j(\Gamma^R_j, Y_j(n) = 0.0$ for the first time. This delay could be added directly after the $D_j(\Gamma^R_j, Y_j(n))$ function box of Figure 4.6-1, or could be added in the decision device itself.

An example will help illustrate how this process works in practice. Consider again the neural net D2A23 whose achieved performance on NDS-A data is shown side-by-side with the desired performance in Figures 5.4.2-1 to 5.4.2-12 (starting on page 176). First, an array of 150 points $X_k(0) = (x(0),y(0))$ on the rectangular boundary B_{IN_A} is created. Each one of these points is iterated $N_{A\_\Gamma\_REGION} = 15$ iterations. This value ensures that none of these trajectories actually reach the point attractor Γ_A. The trajectories of all these points are shown in Figure 5.5.1-1. The line segments joining the adjacent end points of all these trajectories approximates the attractor region Γ^R_A. This region attractor is shown in more detail in Figure 5.5.1-2

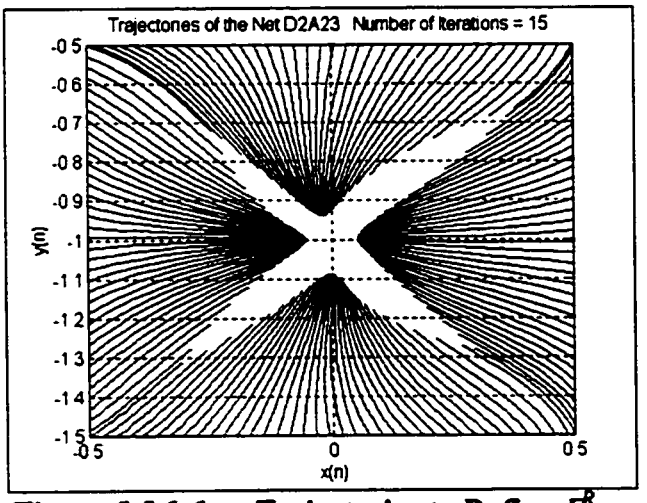


Figure 5.5.1-1 : Trajectories to Define Γ^R_A

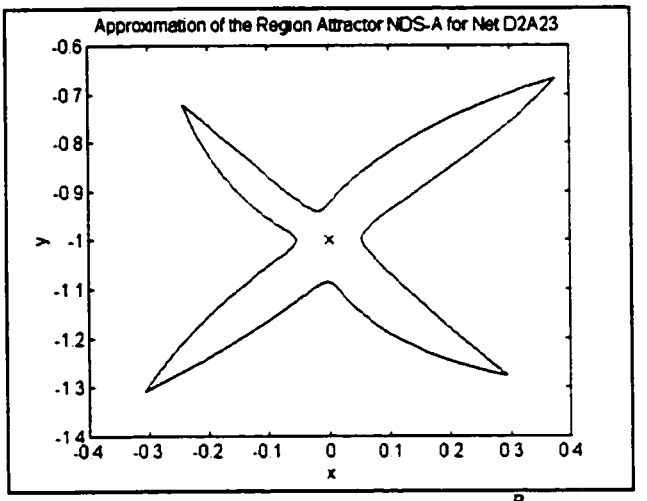


Figure 5.5.1-2 : Region Attractor Γ^R_A

By programming the distance object $D_A(\Gamma^R_A Y_A(n))$ with the region shown in Figure 5.5.1-2, any point from the boundary B_{IN_A} will reach the attractor Γ^R_A in exactly 15 iterations. By adding an extra delay of $N_{A\_DELAY} = 5$ iterations (e.g., at the output of the distance object of Figure 4.6-1 on page 136), every point on B_{IN_A} now reaches the attractor in exactly $N^{\epsilon}_j = 20$ iterations, as desired.

The mapping performance of the net D2A23 with the region attractor (and delay) implemented is shown in Figures 5.5.1-3 and 5.5.1-4. For convenience, the performance of D2A23 without the region attractor (from Figures 5.4.2-1 and 5.4.2-3) are shown in Figures 5.5.1-5 and 5.5.1-6. The desired mappings are shown again in Figures 5.5.1-7 and 5.5.1-8. It can be seen that, with the region attractor, the achieved contour line $\underline{N}^{\epsilon}_{A} = 20$ now matches the desired boundary B_{IN_A} (dashed line) to a high degree of accuracy. Hence, errors at the classification boundary will now be minimized. Note that the achieved contour line $\underline{N}^{\epsilon}_{A}$ can be made to match B_{IN_A} exactly by defining the region attractor more accurately (i.e., using more initial points $X_k(0)$ on the boundary B_{IN_A} so that the line segments used to approximate Γ^{R}_{A} are shorter and more numerous).

Figure 5.5.1-9 illustrates another example for NDS-E where the attractor was redefined from a point to a region. The top of that figure shows the mapping performance when a point attractor is used. Clearly, this NN will perform many classification errors at the boundary B_{IN_E} (outer dashed line) because its $\underline{N}^{\epsilon}_{E} = 20$ contour line is a poor match relative to that boundary. By contrast, when a region attractor Γ^{R}_{E} is defined in the same manner described above, the match to the boundary becomes much better (bottom of Figure 5.5.1-9). Figure 5.5.1-10 shows the approximate shape of the region attractor Γ^{R}_{E} required to achieve the improved mapping.

In both cases, note that the introduction of a region attractor has NOT changed the N_{jk} contour lines further away from B_{IN_j} by a significant amount. Most of the impact of the region attractor has been close to and inside the boundary B_{IN_j} itself.

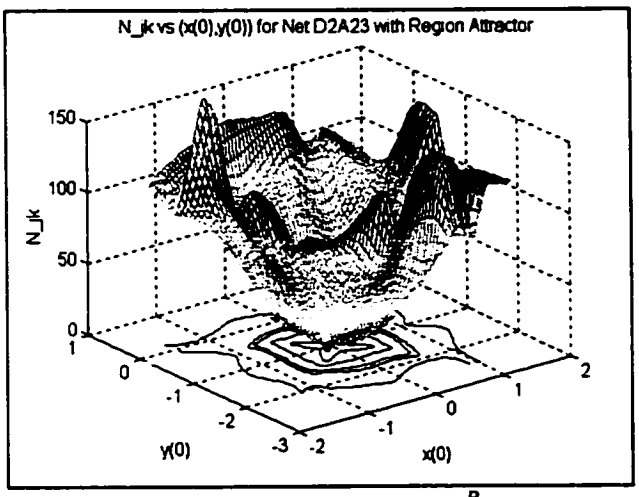


Figure 5.5.1-3 : N_{Ak} Map with Γ^R_A

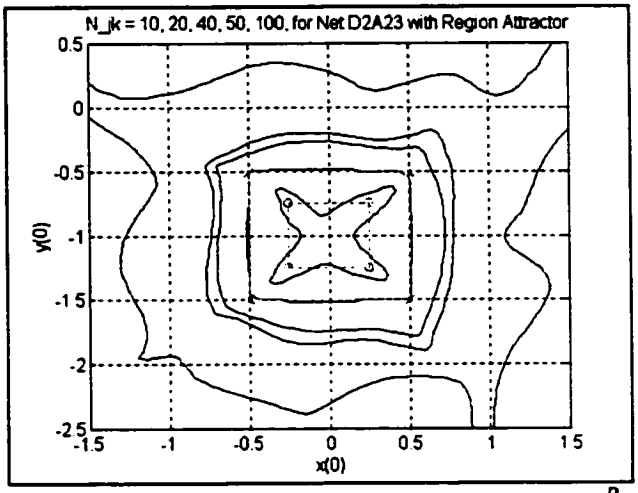


Figure 5.5.1-4 : Contour Lines N_{Ak} with Γ^R_A

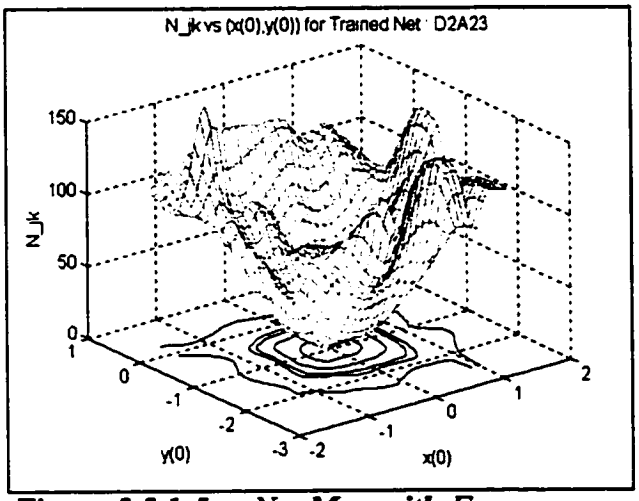


Figure 5.5.1-5 : N_{Ak} Map with Γ_A

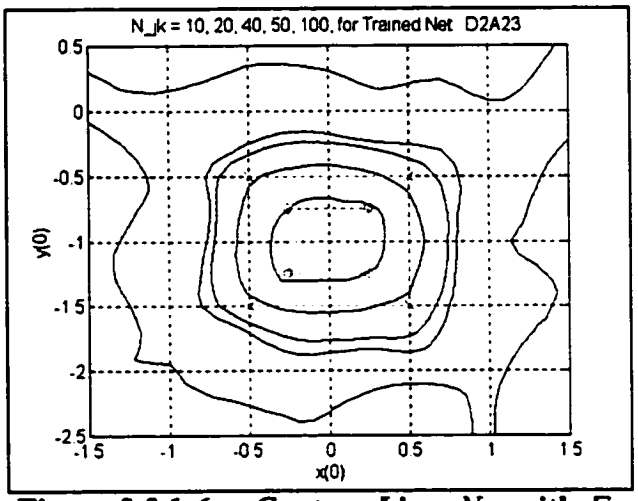


Figure 5.5.1-6 : Contour Lines N_{Ak} with Γ_A

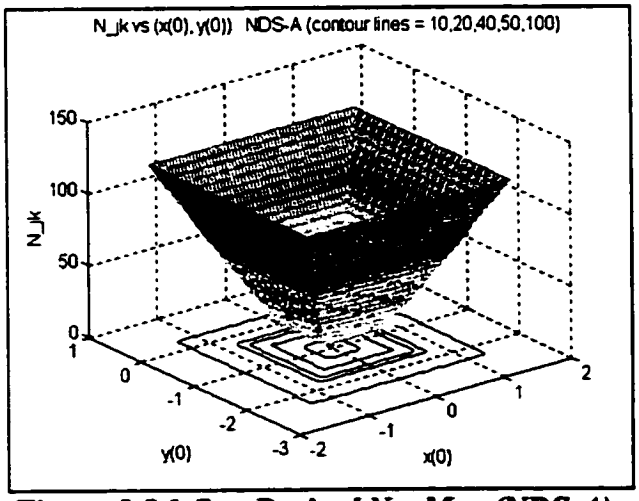


Figure 5.5.1-7 : Desired N_{Ak} Map (NDS-A)

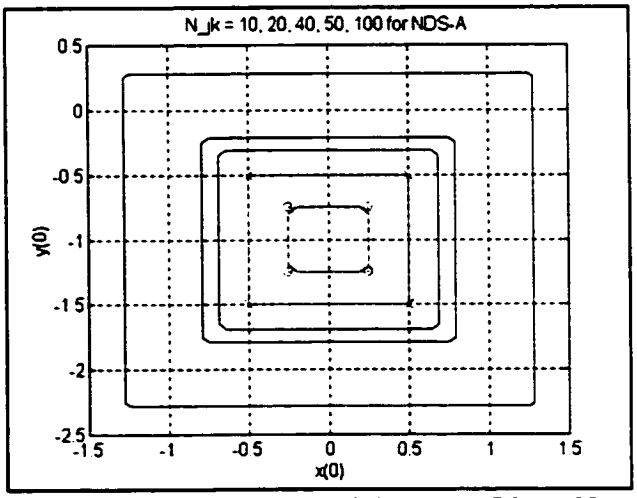


Figure 5.5.1-8 : Desired Contour Lines N_{Ak}

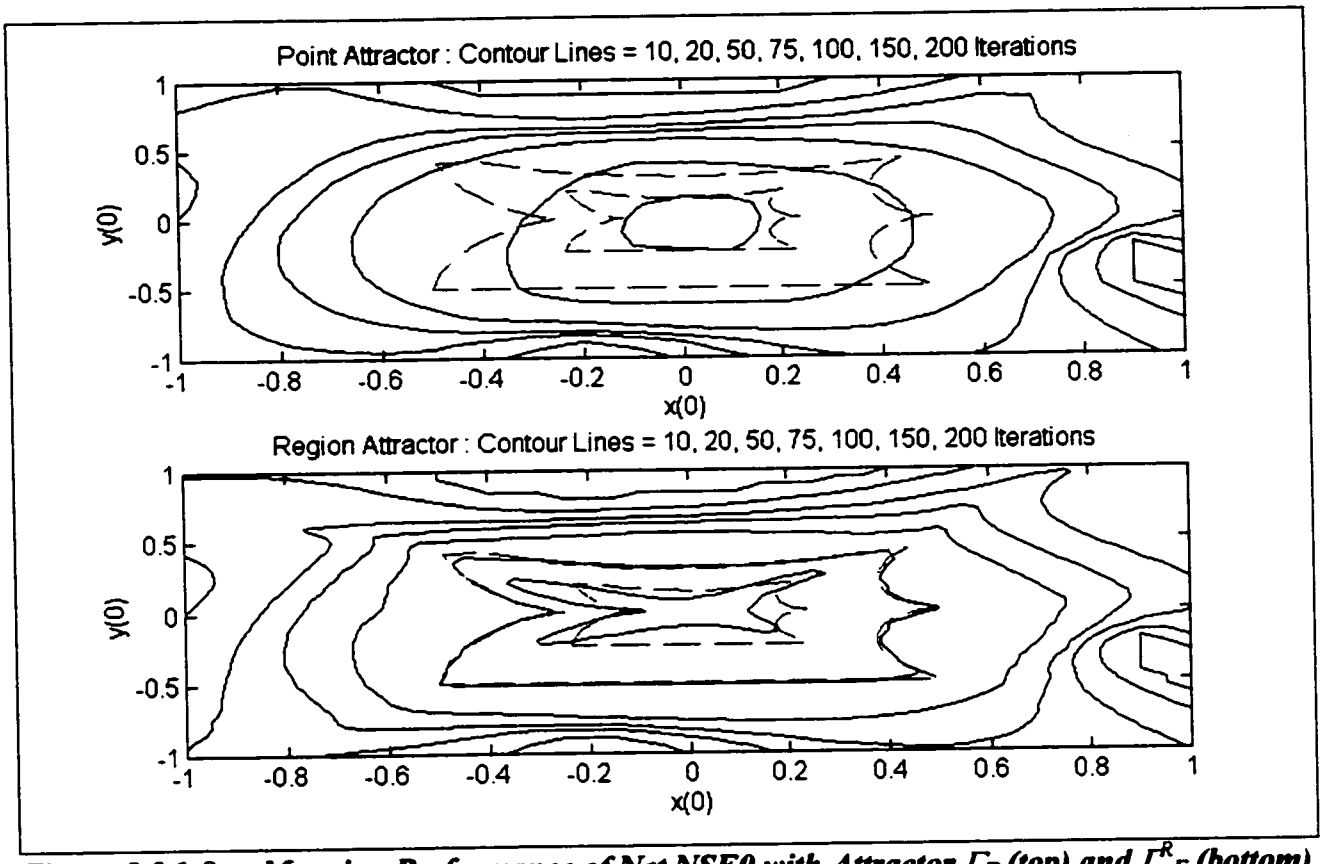


Figure 5.5.1-9 : Mapping Performance of Net NSE0 with Attractor Γ_E (top) and Γ^R_E (bottom)

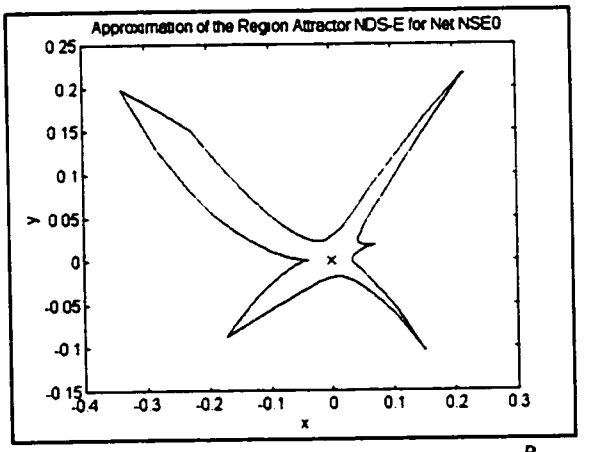


Figure 5.5.1-10 : Attractor Region Γ^R_E

5.5.2 Reduction in the Learning Task Complexity

In order to understand how the redefinition of the point attractor has reduced the complexity of the learning task for the neural network, it helps to examine the required

$\Delta H(X(n))$ before and after the new region attractor is implemented. Consider Figures 5.5.2-1 and 5.5.2-2 shown on the following pages.

In both figures, the top part illustrates how the complexity of the attractor has been increased from a point to a region through the process described in section 5.5.1. The next two plots on the left-hand side show the mapping functions $\Delta H_{j\_x}(X(n))$ and $\Delta H_{j\_y}(X(n))$ required to ensure that the contour line $N^{\epsilon}_{j} = 20$ matches exactly the boundary region B_{IN_j} when the attractor is a point. The two plots to the right of these illustrate the mapping functions $\Delta H_{j\_x}(X(n))$ and $\Delta H_{j\_y}(X(n))$ required to ensure that the contour line $N^{\epsilon}_{j} = 20$ matches exactly the boundary region B_{IN_j} when the attractor is a region. The reduction in complexity is evident in that all the sharp edges and corners of the functions on the left have been replaced by smooth maps with relatively shallow slopes, wide hills and no corners.

Hence, complexity has been transferred from the iterated mapping function $\Delta H(X(n))$ to the attractor itself. Note that this attractor complexity is now implemented outside the neural network, in the distance object of the RTA NN model. No additional changes or training to the NN itself was required to achieve the desired mapping. Finally, note that the classification problem complexity has only been transferred; NOT eliminated.

This process can be summarized in a few words as follows:

Redefining the attractor from a simple point to a more complex region allows us to TRANSFER some of the complexity of the classification problem from the mapping function $\Delta H(X(n))$ to the region attractor.

Since the region attractor is implemented outside the neural net, the complexity of the classification problem for the neural net is reduced.

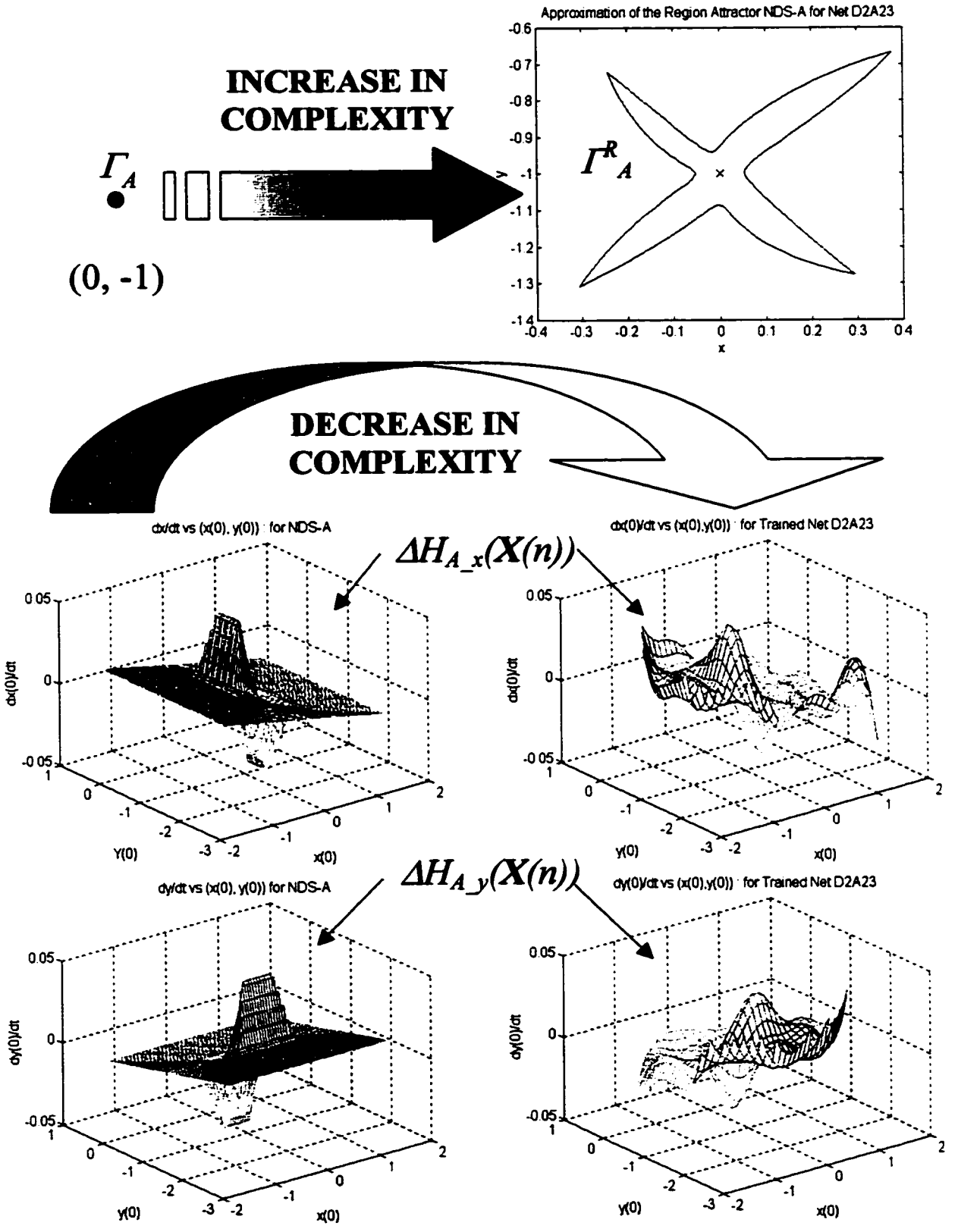


Figure 5.5.2-1 : Transfer of Complexity from $\Delta H_A(X(n))$ to Γ^R_A

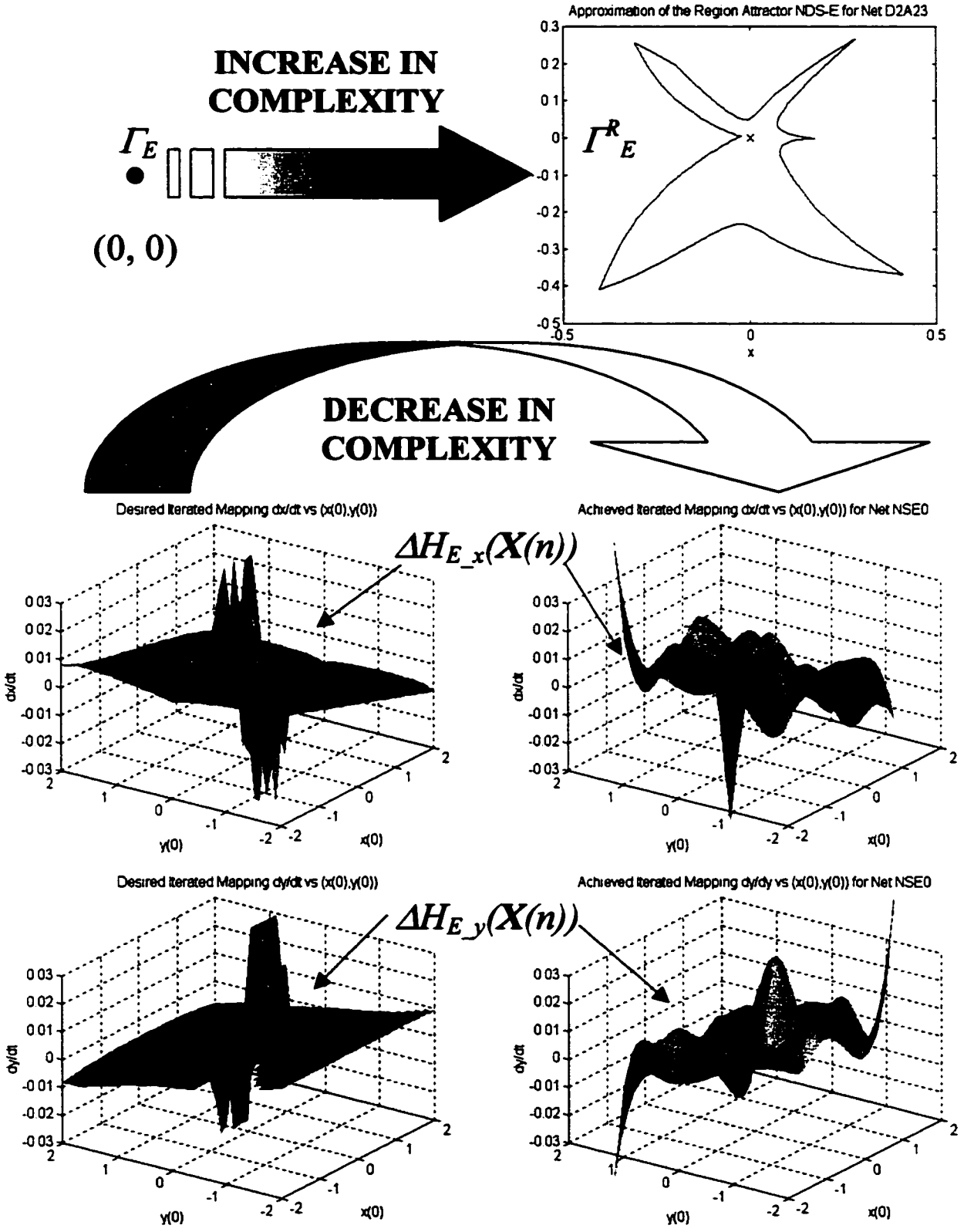


Figure 5.5.2-2 : Transfer of Complexity from $\Delta H_E(X(n))$ to Γ^R_E

In terms of 'memorization learning' (see section 3.5.2, page 86), the process of training a neural subnet to match the desired classification boundary now involves these steps:

1. Train the neural net (NN) to learn $\Delta H_{j\_x}(X(n))$ and $\Delta H_{j\_y}(X(n))$ for the point attractor to the best of the neural net's ability.

2. Redefine the attractor as explained in section 5.5.1 so that the $\Delta H_{j\_x}(X(n))$ and $\Delta H_{j\_y}(X(n))$ achieved by the NN during training in step 1 will be adequate to implement the desired decision boundary accurately.

For the incremental learning approach described in section 3.5.1 (page 78), one can envision that the process of Bayesian inference learning also involves gradual changes of the attractor into a region as new data arrives, in order to compensate for the limited learning ability of simple neurons.

5.6 Re-using a Trained Subnet to Learn a New Class

This section describes how the mechanism of re-using a previously trained NN can be implemented with the RTA NN model in order to reduce the overall complexity of the learning task.

Essentially, this is accomplished by a generalization of the mechanism described in section 5.5. In the previous section, it was shown how a NN trained on NDS-A data (for example) could be made to achieve the desired classification mapping boundary B_{IN_A} by redefining the point attractor Γ_A into a region attractor Γ^R_A with the help of trajectories whose initial points were on the desired boundary B_{IN_A}. (i.e., in accordance with equation E_5.5.1-2 on page 184).

But this process is NOT limited to the use of initial points on the boundary B_{IN_A} itself. By using a set of initial points on B_{IN_B}, B_{IN_C}, B_{IN_D}, and B_{IN_E}, the NN trained on the NDS-A data can be made to match its $N^{\epsilon}_A = 20$ contour line to the shape of any one of

the other classification boundaries. Furthermore, the fact that the point attractors Γ_A to Γ_E have different coordinates in R^2 is not a significant obstacle since the use of appropriate translation offsets x_{offset_in} and y_{offset_in} will allow any NN trained to any attractor Γ_j to converge instead to another attractor Γ_r. Let us illustrate these concepts with an example.

Consider again NN D2A23 which was initially trained to map NDS-A functions (see Figures 5.4.2-1 (page 176) to 5.4.2-12) and whose performance was improved in section 5.5.1 by redefining the point attractor Γ_A as a region attractor Γ^R_A (see Figures 5.5.1-1 (page 185) to 5.5.1-8). The same process discussed in section 5.5.1 has been repeated with a set of 150 points $X_k(0)$ on boundaries B_{IN_B}, B_{IN_C}, B_{IN_D}, and B_{IN_E}. A suitable offset $(x_{offset\_in}, y_{offset\_in})$ was applied at the input of the NN D2A23 to ensure that the point attractor Γ_j was translated to the point attractor Γ_A which NN D2A23 utilizes.

Recall that this use of offsets has already been discussed in section 3.2.1 (page 67) and is illustrated in Figure 5.6-1. Based on equation E_3.2.1-1, the required offsets to achieve the translation for the 2D problem have this form:

$$\begin{aligned}(x_{offset\_in}, y_{offset\_in}) &= (\Gamma_{A\_x} - \Gamma_{j\_x}, \Gamma_{A\_y} - \Gamma_{j\_y}) \\ (x_{offset\_out}, y_{offset\_out}) &= (-x_{offset\_in}, -y_{offset\_in})\end{aligned} \qquad \textbf{(E\_5.6-1)}$$

For example, if NN-D2A23 is to be used to map the NDS with Γ_B = (0.05, 1.05), the input offsets are (-0.05, -2.05). If the input at time n is $Y_B(n)$ = (0.05, 1.05), the 'real' input to the NN after translation is $Y_A(n)$ = (0.0,-1.0) which is the attractor Γ_A. Hence, the output of the NN will be $Y_A(n+1) = \Gamma_A$ = (0.0,-1.0) but after the output translation, this becomes $Y_B(n)$ = (0.05, 1.05) = Γ_B. Hence, to an observer outside the dotted line shown in Figure 5.6-1, this NN implements NDS-B with point attractor Γ_B but the NN inside the box actually implements NDS-A with point attractor Γ_A . By ensuring that the region attractor Γ^R_A has the shape required to match B_{IN_B}, the NN implements the correct classification boundary for NDS-B.

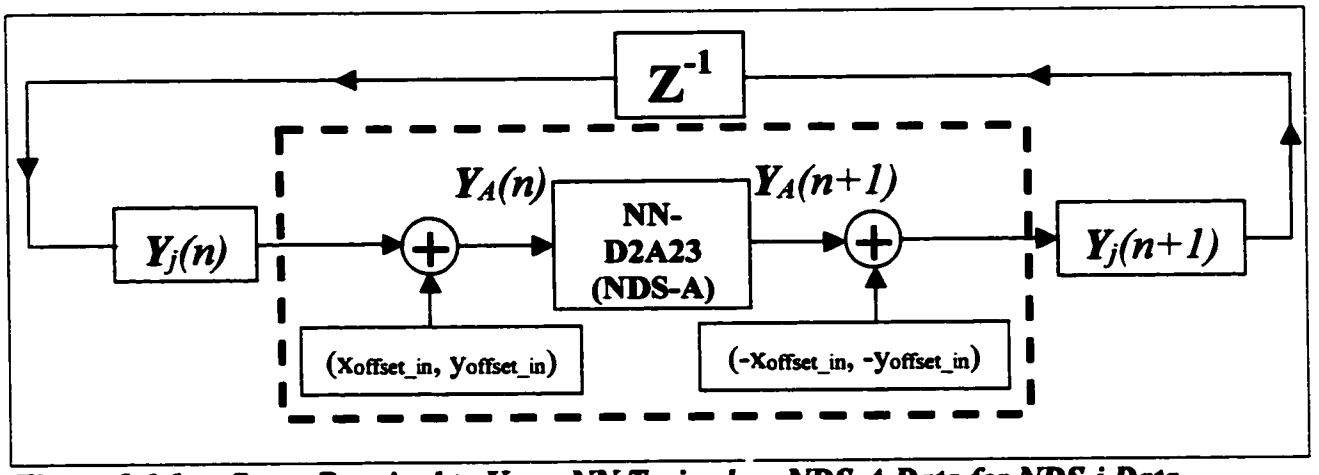


Figure 5.6-1 : Setup Required to Use a NN Trained on NDS-A Data for NDS-j Data

Figures 5.6-2 to 5.6-9 illustrate the trajectories used to define the region attractors and the approximate region attractor shapes required to implement NDS-B to NDS-E with the NN trained on the data of NDS-A.

In order to implement the entire RTA NN using only NN D2A23, it is then simply necessary make four extra copies of NN D2A23 and to insert them in the RTA NN with the appropriate offsets to achieve the classification boundaries B_{IN_A} to B_{IN_E}. Each distance object implements the required region attractors Γ^R_A (Figure 5.5.1-2 on page 185) and Γ^R_B to Γ^R_E (Figures 5.6-3, 5.6-5, 5.6-7 and 5.6-9). The final RTA NN supernet required to implement the 2D classification problem takes the form shown in Figure 5.6-10. Each box containing the neural net is actually a structure as shown in Figure 5.6-1 with the appropriate offsets as defined by equation E_5.6-1.

Note that the number of iterations required before the attractor is reached ($N_{j\_\Gamma\_REGION}$) is not the same for all regions : it is 15, 25, 10, 15 and 10 for regions A to E respectively. As indicated in section 5.5.1, it is necessary to add a delay $N_{j\_DELAY} = N^{\epsilon}_j - N_{j\_\Gamma\_REGION}$ in the decision device (or at the output of each distance object) to ensure that each NN wins the race in the required N^{ϵ}_j whenever an input $X_k(0)$ is on or inside boundary B_{IN_j}. Note also that for subnet-B, the N^{ϵ}_B has to be increased from 20 to 30 iterations because the trajectories of Figure 5.6-2 require 25 iterations to reach the region attractor.

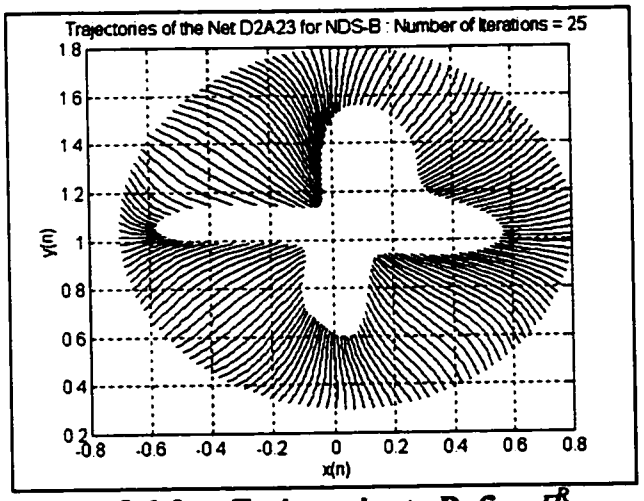


Figure 5.6-2 : Trajectories to Define Γ^R_B

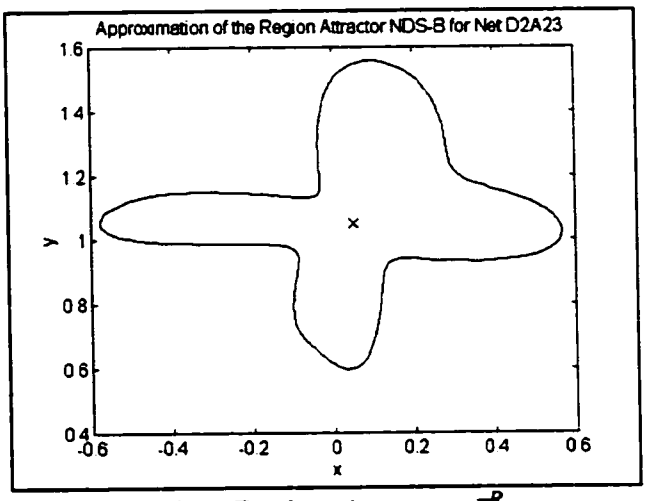


Figure 5.6-3 : Region Attractor Γ^R_B

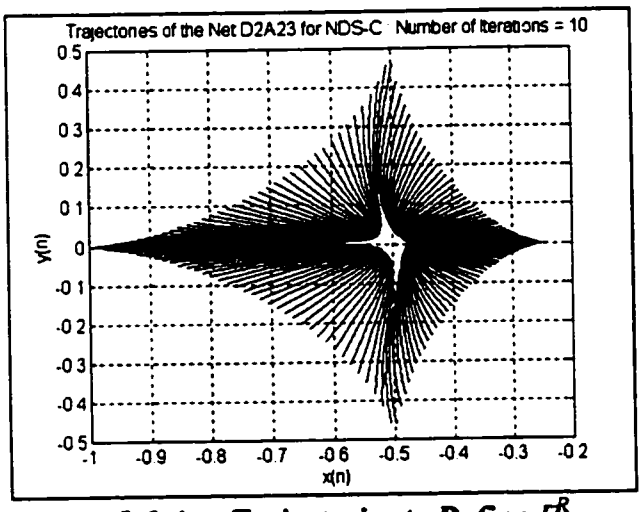


Figure 5.6-4 : Trajectories to Define Γ^R_C

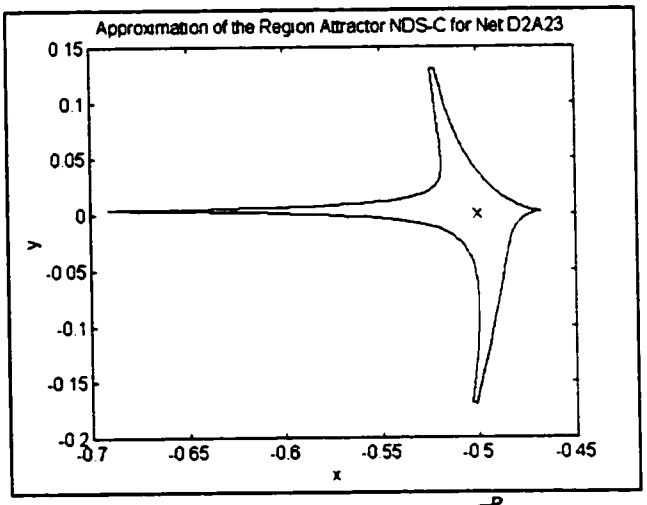


Figure 5.6-5 : Region Attractor Γ^R_C

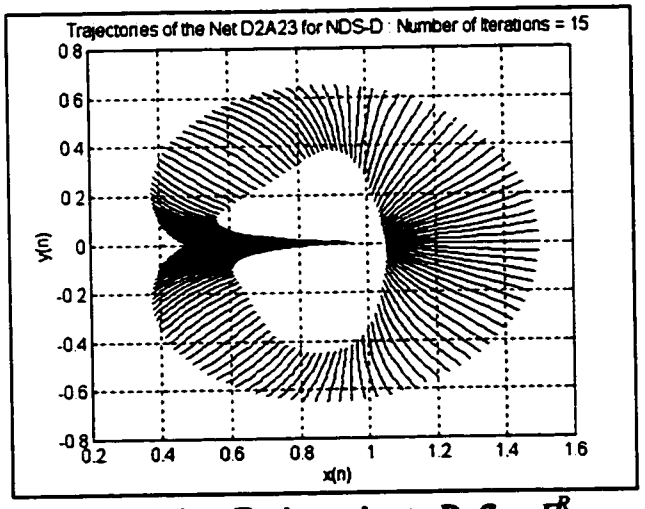


Figure 5.6-6 : Trajectories to Define Γ^R_D

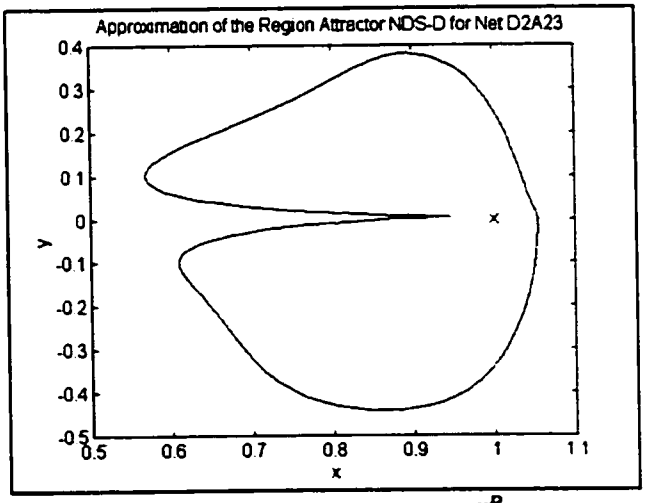


Figure 5.6-7 : Region Attractor Γ^R_D

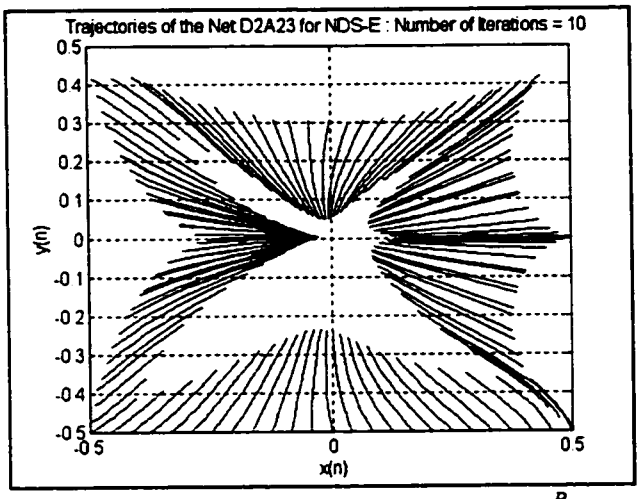


Figure 5.6-8 : Trajectories to Define Γ^R_E

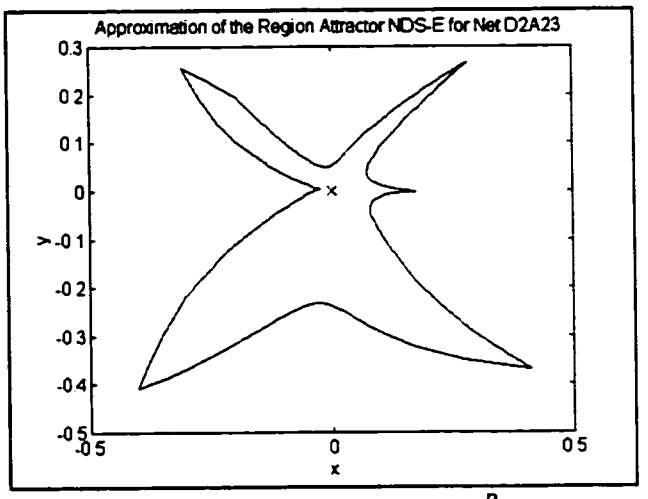


Figure 5.6-9 : Region Attractor Γ^R_E

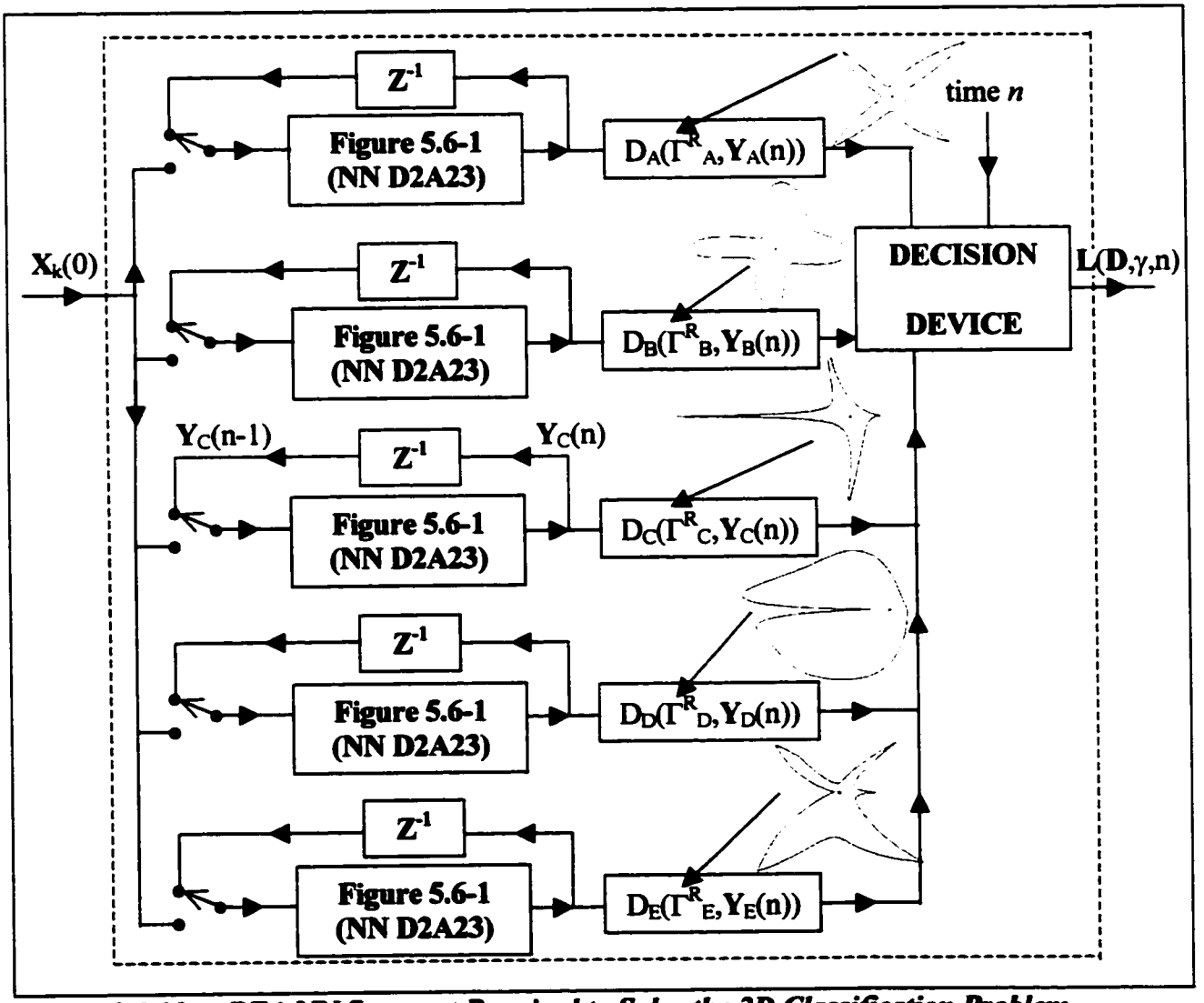


Figure 5.6-10 : RTA NN Supernet Required to Solve the 2D Classification Problem

However, the fact that subnet-B has a different N^{ϵ}_j than the other four NDS (with $N^{\epsilon}_j = 20$) has no detrimental effect on the ability of the RTA NN to classify inputs correctly. That is because each NN is disqualified from the race after it reaches its N^{ϵ}_j. As a result of this, subnets A, C, D, and E will NOT invade the classification region of subnet-B (the disqualification process was discussed in section 4.6.1 (page 136) and is described by equation E_4.6.1-7 on page 138).

Figures 5.6-11 to 5.6-15 illustrate the achieved N_{jk} contour lines with the RTA NN model of Figure 5.6-10. It can be seen that the critical region around the boundaries B_{IN_J} are matched to a fairly high degree of accuracy in all cases. Hence, the RTA NN model will achieve reliable classifications in the regions shown.

Let us examine the how much reduction in the learning task complexity has been achieved with this mechanism. Recall from section 4.7.1 (page 145) that the learning task complexity increases as the number of training vectors required V increases, with V given by:

$$V = L e^{\frac{P}{s}} \qquad \text{(E\_5.6-2)}$$

where L is a positive constant, P is the input dimension and s is a measure of the complexity of the iterated mapping function $H(X(n))$ such that s decreases as the function complexity increases. As explained in section 4.7.1, as V increases, the required size of the NN increases, which results in an increase in the training time. Hence, as V increases exponentially, so does the learning task complexity.

For the sake of argument, assume that the combined complexity of the achieved mapping functions $\Delta H_{A\_x}(X(n))$ and $\Delta H_{A\_y}(X(n))$ shown in Figure 5.5.2-1 (page 190) is $s = 0.245$ (including the use of the region attractor). With $L = 1$, equation E_5.6-2 results in $V = 3510$, (with $P = 2$) which is roughly equal to the number of training pairs actually used to train NN D2A23.

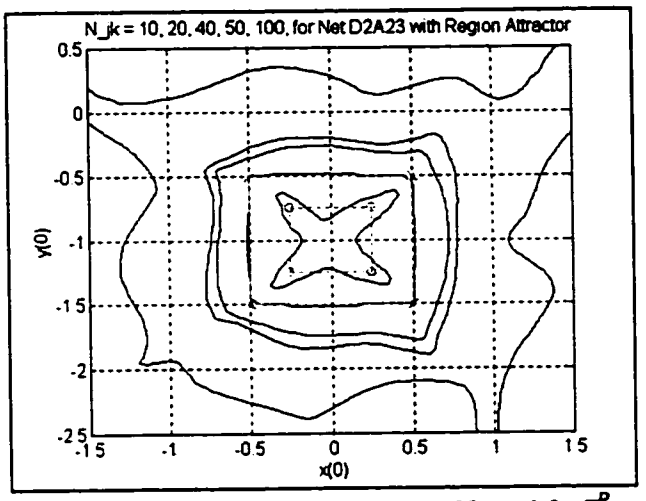


Figure 5.6-11 : Contour Lines N_{Ak} with Γ^R_A

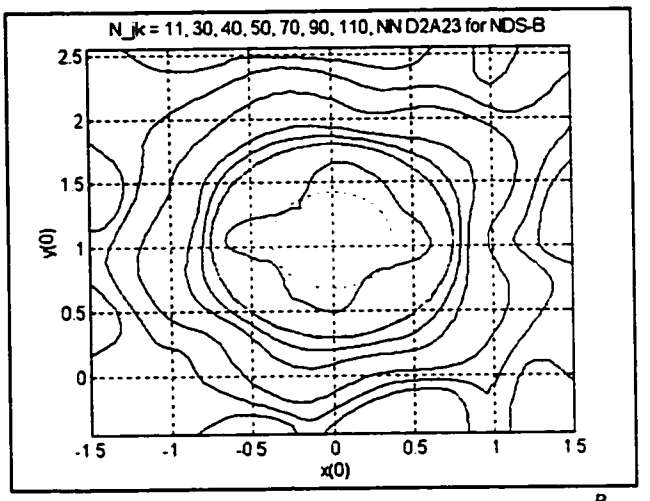


Figure 5.6-12 : Contour Lines N_{Bk} with Γ^R_B

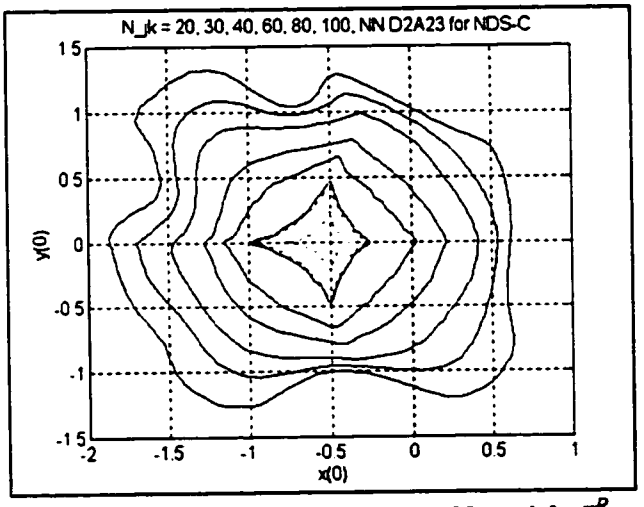


Figure 5.6-13 : Contour Lines N_{Ck} with Γ^R_C

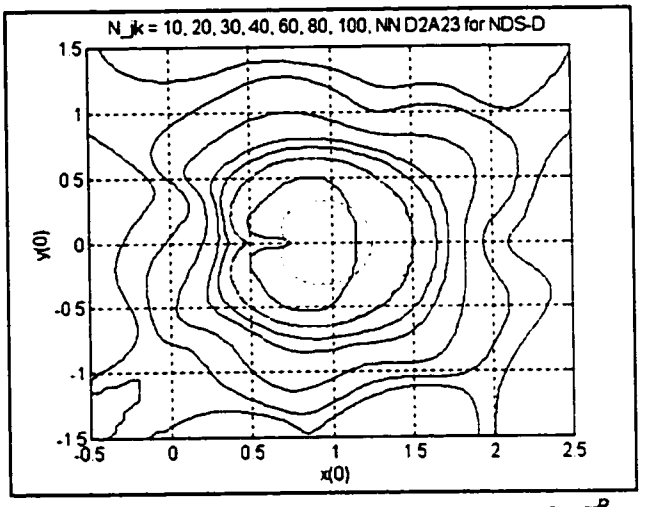


Figure 5.6-14 : Contour Lines N_{Dk} with Γ^R_D

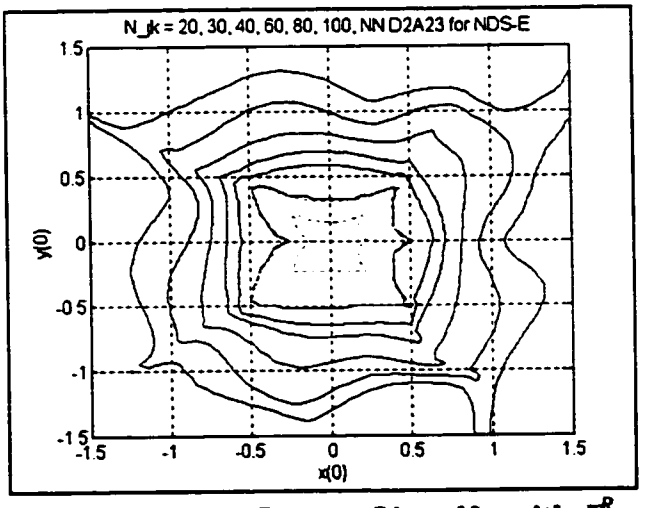


Figure 5.6-15 : Contour Lines N_{Ek} with Γ^R_E

From Figure 5.5.2-2 (page 191), it can be appreciated that the achieved functions $\Delta H_{E\_x}(X(n))$ and $\Delta H_{E\_y}(X(n))$ are not significantly more complex than those of Figure 5.5.2-1. Hence, as a rough approximation, it seems reasonable to assume that the complexity of the other four mapping functions is also around $s = 0.245$. Therefore, if it was necessary to train five separate NN to learch each separate set of functions $\Delta H_{j\_x}(X(n))$, $\Delta H_{j\_y}(X(n))$, the overall complexity of the training task would be related to 5 X 3510 = 17550 training points, which is 5 times more complex than learning one mapping function.

But by re-using one trained NN to map all five NDS, the total learning task complexity for the NN is fixed and has a complexity equivalent to 3510 training points. Hence, a problem whose complexity increases linearly with the number of classes (when one NN is trained for each NDS) has been reduced to a problem of almost constant complexity by the use of this mechanism. Note that, strictly speaking, the task of defining the attractor region has a non-zero complexity but this is a trivial task relative to the 5-20 hours required to teach a NN to learn the iterated mapping function.

The overall reduction in complexity relative to a global approximation problem is even more significant. Consider again Figure 5.3.2-1 (page 173) which shows the iterated mapping function required if one NN had to learn all five classes as one global function. If $s = 0.245$ is the complexity of $H(X(n))$ for one class, the complexity for 5 classes can be argued to be in the order of $s = 0.245 \div 5 = 0.049$. Using this value in equation E_5.6-2 produces a value $V = 5.32 \text{ X } 10^{17}$ which is several orders of magnitude more complex than the RTA NN model paradigm with the mechanism of re-use ($V = 3510$) or even without that mechanism used ($V = 17550$).

These results illustrate clearly how mechanisms 'b', 'c' and 'd' listed in section 5.1 (starting on page 151) allow the RTA NN model to achieve significant reductions in the complexity of a learning task.

5.7 Adding a New Class to a Fully Trained RTA NN Supernet

This section illustrates how the RTA NN model is able to learn a new class without memory loss for the previously learned classes. In addition, it will be shown that the complexity of the learning task is limited by the complexity of the new class boundary itself and is NOT affected by the number of classes already in memory (i.e., learning characteristic 3 as defined in section 5.1, page 151).

Consider the new class C_F as shown in Figure 5.7-1, with point attractor $\Gamma_F = (1, -1.4)$ and with the optimal classification boundary B_{IN_F} as shown in that figure. In order to add a new class to the RTA NN model, it only necessary to train a new subnet NN-F to learn the appropriate trajectories for $X_k(0) \in R_{IN} \subset R^2$, where $R_{IN} = \{x \in [-1.5,2], y \in [-2,2]\}$. Here again, the training seeks to ensure that all points on the boundary B_{IN_F} or inside it will converge to the point attractor in $N_{Fk} \leq N^{\epsilon}_F = 20$ and will be classified as IN class C_F whereas points outside that boundary (with $N_{Fk} > 20$) will be classified as OUT of that class.

Note that because the other 5 NN already classify points in R_{IN_F} as OUTSIDE their respective classes, <u>there is no need to retrain the other subnets</u>. Hence, there is no memory loss and the complexity of the learning task is determined exclusively by the complexity of the new mapping function $\Delta H_F(X(n))$, whose complexity is related mainly to the complexity of the boundary region B_{IN_F}.

But what if the optimal classification boundary of the new class <u>overlaps</u> one of the existing boundaries ? This situation is shown in Figure 5.7-2, where the new optimal classification boundaries are such that B'_{IN_A} and B'_{IN_D} need to be redefined as shown. In this case, subnets NN-A and NN-D will also have to be re-trained in order to match the new desired (and optimal) classification boundaries. However, NN-B, NN-C and NN-E need not be retrained so that the learning task complexity and potential memory loss is minimized (i.e., it is confined to NN-A and NN-E).

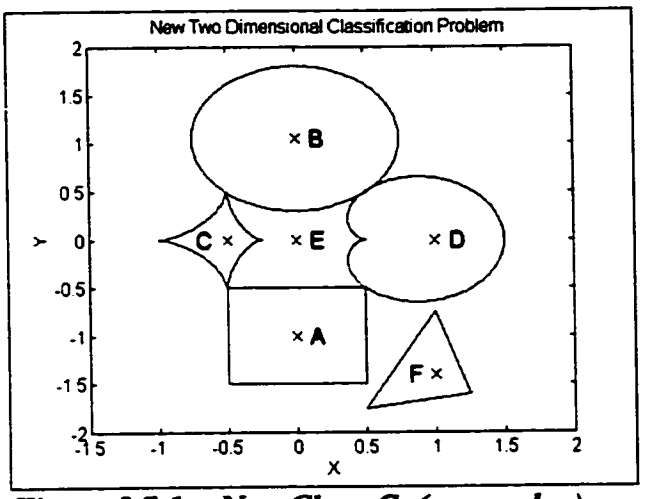


Figure 5.7-1 : New Class C_F (no overlap)

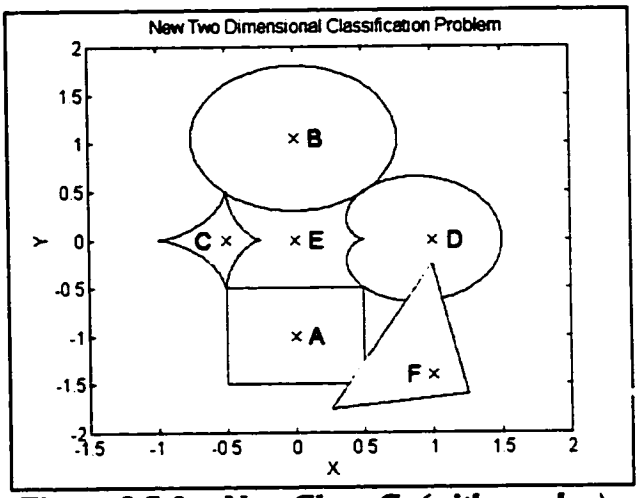


Figure 5.7-2 : New Class C_F (with overlap)

Furthermore, the mechanisms described in sections 5.5 and 5.6 can be used to greatly simplify the addition of a new class to the RTA NN. For example, the subnet-A can be re-used to learn the new class C_F as shown in Figure 5.7-1 or 5.7-2 by redefining a suitable attractor region for subnet-A and applying suitable x and y offsets (as shown in Figure 5.6-1, page 194). For the new regions B'_{IN_A} and B'_{IN_D} shown in Figure 5.7-2, the attractor regions I^R_A and I^R_D can be redefined as explained in section 5.5.1 to match the new boundaries.

Note that with the use of these two mechanisms, there is no need to train a new neural net or even retrain existing ones. As a result of this, it is possible to add an arbitrary number of new classes with various shapes to the RTA NN and achieve accurate classifications for all classes with a minimum amount of effort. This is somewhat similar to an adult who, having learned the alphabet 'A' to 'Z' can learn the Greek alphabet 'α' to 'ω' with relatively little effort.

5.8 Repairing Neural Damage in the RTA NN Supernet

This section describes how neural damage affects the performance of the RTA NN model and how mechanism 'd' discussed in section 5.1 allows the RTA NN model to repair such damage to a large extent.

Section 5.8.1 provides a brief description of the results of tests where one or more neurons in a trained subnet were killed. This section also introduces the important concept of the *utilization factor* (U_F) which provides a measure of the degree of contribution which individual neurons bring to the overall mapping achieved by a net.

In section 5.8.2, the process of repairing neural damage is explained and illustrated with several examples.

5.8.1 Definition of the Utilization Factor and Results of Neural Damage Tests

For both the MLP or RBF neurons, 'death' implies that the neuron's output is always 0.0 regardless of the input. This is equivalent to removing it from the summed output mapping.

One must be very careful when assessing the impact of the death of a neuron on the NN. The reason why caution is necessary is that neurons in a fully trained neural net may not contribute equally to the mapping task. Past experience has shown that some neurons in a net can contribute nothing to the mapping while a few others make most of the contribution [19]. In such a net, killing neurons which are already useless when alive has little significance.

Hence, before assessing the impact of killing a neuron, it is essential to determine how useful it is to the NN when alive. To that end, the concept of the *utilization factor* (U_F) is now introduced. The utilization factor is a measure of the usefulness of a neuron in the network. It is derived by comparing the Mean Square Error (MSE) between the desired and achieved trajectories when a neuron is dead to the MSE when that neuron is alive.

Let E_{AV_jk} be the mean square error (MSE) between the 'desired' trajectory c_{jk} and the trajectory achieved by the trained NN-j when all its neurons are alive (c_{jk}). E_{AV_jk} is given by:

$$E_{AV\_jk} = d\left(c_{jk}, \underline{c}_{jk}\right)^2 = \frac{1}{N_{jk}} \sum_{n=1}^{N_{jk}} \left(x_{jk}(n) - \underline{x}_{jk}(n)\right)^2 + \left(y_{jk}(n) - \underline{y}_{jk}(n)\right)^2 \qquad \text{(E\_5.8.1-1)}$$

where $[x_{jk}(n), y_{jk}(n)]$ is the n-th iterate of the desired trajectory c_{jk} and where $[\underline{x}_{jk}(n), \underline{y}_{jk}(n)]$ is the n-th iterate of the achieved trajectory $\underline{c}_{jk}$, with all the neurons alive.

The overall MSE over the M desired trajectories of the training data set is identified as E_{AV_j} and is given by:

$$E_{AV\_j} = \frac{1}{M} \sum_{k=1}^{M} E_{AV\_jk} \qquad \text{(E\_5.8.1-2)}$$

Hence, E_{AV_j} is the MSE averaged over the entire training data set. Let us define $E_{AV\_j}(r \notin NN)$ as the MSE of equation E_5.8.1-2 with neuron r in the NN-j 'killed'. The utilization factor for neuron r of NN-j is identified as $U_{F\_j}(r)$ and is given by:

$$U_{F\_j}(r) = \frac{E_{AV\_j}(r \notin NN)}{E_{AV\_j}}, \quad E_{AV\_j} \neq 0 \qquad \text{(E\_5.8.1-3)}$$

Hence, $U_{F\_j}(r)$ is a measure of how much the mapping error increases (or decreases) when neuron r is removed from NN-j. The larger $U_{F\_j}(r)$ is, the more important (i.e., utilized) is neuron r in the NN since its removal causes a large increase in the MSE. Conversely, a $U_{F\_j}(r) \leq 1.0$ implies that neuron r is not useful (or is even detrimental) to the mapping since its removal leaves the MSE unchanged (or makes it lower) than before its death.

5.8.1.1 Summary of Neural Damage Test Results

For these tests, four basic net types were used:

a. **SRBF_68A.NET** with 68 simple RBF and 10 MLP in a coupled structured (i.e., one group of neurons generates both $H_{j\_x}(X(n))$ and $H_{j\_y}(X(n))$, as shown in

Figure 5.4.1-1 on page 174). The net was trained on NDS-A data and has a region attractor I^R_A.

b. **SRBF_74A.NET** with 74 simple RBF and 16 MLP split into two groups, as an uncoupled structure (i.e., separate groups of neurons generate $H_{j\_x}(X(n))$ and $H_{j\_y}(X(n))$ separately, as shown in Figure 5.4.1-1). The net was trained on NDS-A data and has a region attractor I^R_A (different than the one in 'a' above).

c. **LRBF_74A.NET** similar to SRBF_74A.NET but linear-recurrent RBF (LR-RBF) neurons are used.

d. **CRBF_74A.NET** similar to SRBF_74A.NET but center-recurrent RBF (CR-RBF) neurons are used.

For each net, every neuron was killed in sequence and its U_f was determined. A few results for the four nets tested are summarized in the table below:

Table T_5.8.1.1-1 : Average Utilization Factors U_f for the Neural Nets Tested

RBF average U_f: $E_R[U_f]$	393.7	161.6	34.2	17.6
McCulloch-Pitts average U_f: $E_{Mc}[U_f]$	910.8	503.0	225.4	321.0
Overall Average : $E[U_f]$	463.7	230.4	69.7	73.3

In this table $E[U_f]$ is the average U_f for all neurons of the type indicated. The following main observations can be made:

1. McCulloch-Pitts neurons have a U_f which is consistently higher than RBF neurons. This may be because they perform global approximations versus local ones for RBF. Hence, a McCulloch-Pitts neuron is more likely to be involved in mapping a large number trajectories versus only a few for a RBF neuron.

2. The uncoupled structure has an average U_f which is roughly half that of the coupled structure. This seems sensible since a dead neuron in the coupled structure will affect both $H_{j\_x}(X(n))$ and $H_{j\_y}(X(n))$ whereas it would affect only one of these in an uncoupled structure.

3. Both LR-RBF and CR-RBF neurons have a lower U_f than simple RBF neurons. This may indicate that neurons with local feedback have a better ability to compensate for the death of their neighbors than those without such feedback.

Note that a detailed discussion and analysis of the above results will not be conducted, since this is not necessary to demonstrate that the RTA NN can repair neural damage.

The effect of killing a RBF neuron in the SRBF_74A net can be appreciated by looking at Figures 5.8.1.1-1 to 5.8.1.1-12. Odd-numbered figures show the net performance before the neuron is killed while even numbered figures show the performance after the neuron is killed. The neuron itself is a simple RBF involved in the mapping of $H_{A\_y}(X(n))$ and has a $U_f = 22.3$. Its center coordinates $(\mu_x, \mu_y) = (1, -1.15)$ are shown as a circle 'o' in Figures 5.8.1.4, 5.8.1.1-10 and 5.8.1.1-12.

From Figure 5.8.1.1-4, it can be seen that the contour line for $N^{\epsilon}_{A} = 20$ no longer matches the desired boundary B_{IN_A} everywhere. As a result of this, the damaged NN will perform many classification errors in the region where the two no longer match.

Figures 5.8.1.1-6 and 5.8.1.1-8 illustrate how the iterated mapping function $\Delta H_{A\_y}(X(n))$ has changed in the damaged net relative to the undamaged one (Figures 5.8.1.1-5 and 5.8.1.1-7). Note that $\Delta H_{A\_x}(X(n))$ has not been shown in these figures because the dead neuron was not involved in the X coordinate mapping. As a result, that mapping is the same for both the damaged and undamaged NN.

The effect of the dead neuron on the trajectories is shown in Figures 5.8.1.1-10 and 5.8.1.1-12 relative to the undamaged net (5.8.1.1-9 and 5.8.1.1-11).

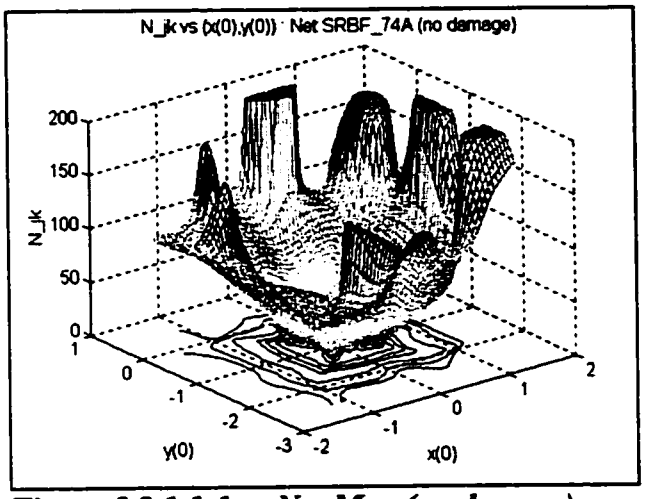


Figure 5.8.1.1-1 : N_{Ak} Map (no damage)

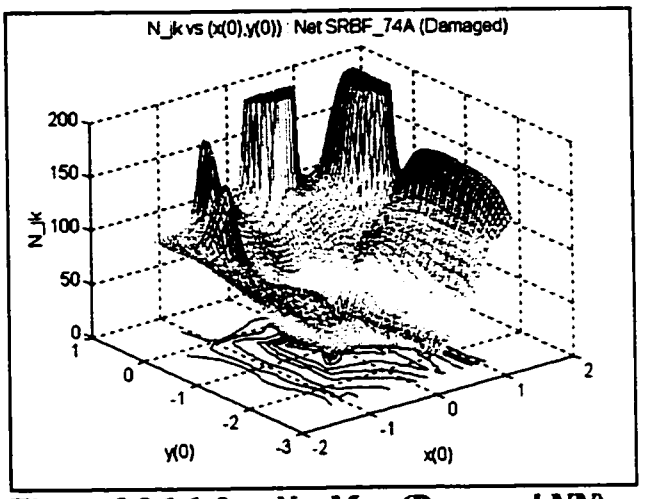


Figure 5.8.1.1-2 : N_{Ak} Map (Damaged NN)

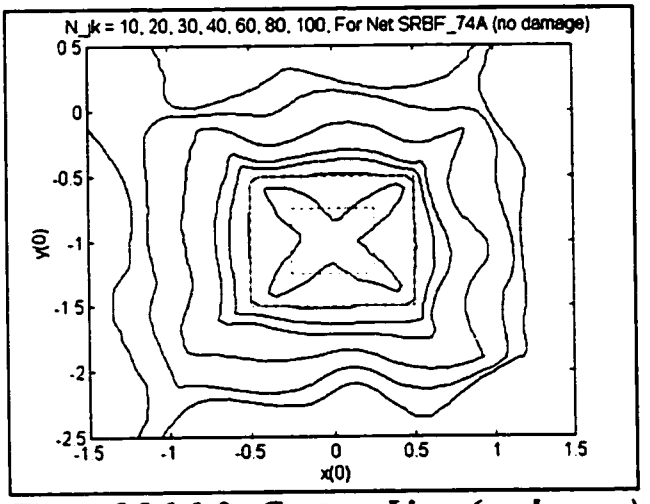


Figure 5.8.1.1-3 : Contour Lines (no damage)

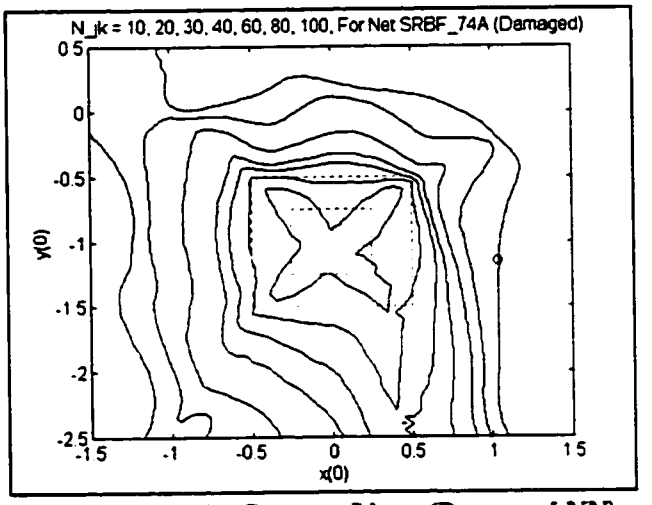


Figure 5.8.1.1-4 : Contour Lines (Damaged NN)

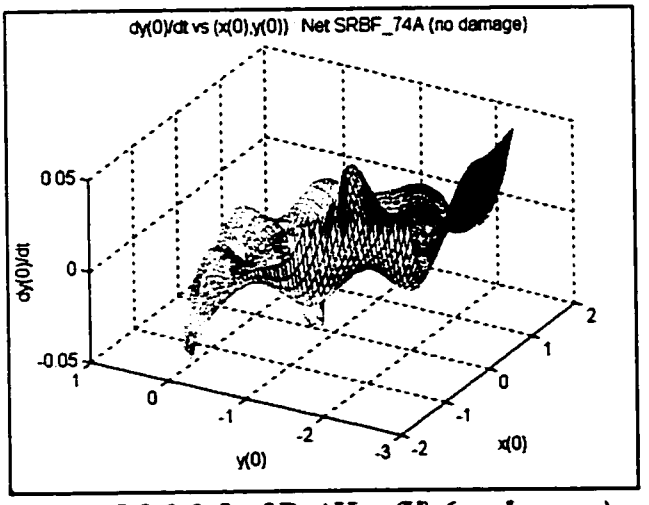


Figure 5.8.1.1-5 : 3D $\Delta H_{A\_y}(X)$ (no damage)

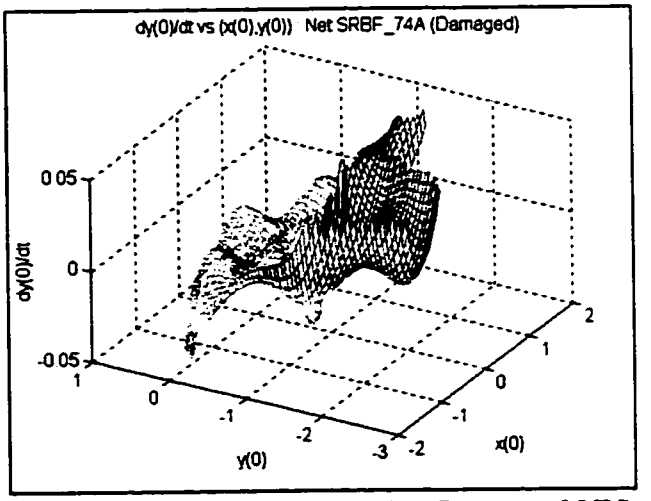


Figure 5.8.1.1-6 : 3D $\Delta H_{A\_y}(X)$ (Damaged NN)

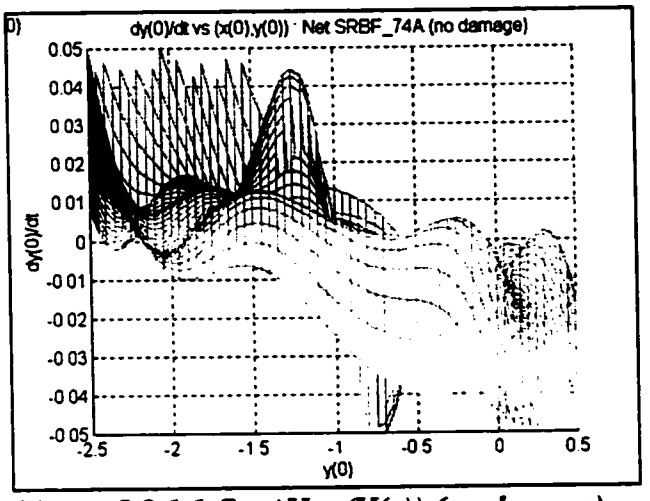


Figure 5.8.1.1-7 : $\Delta H_{A\_y}(X(n))$ (no damage)

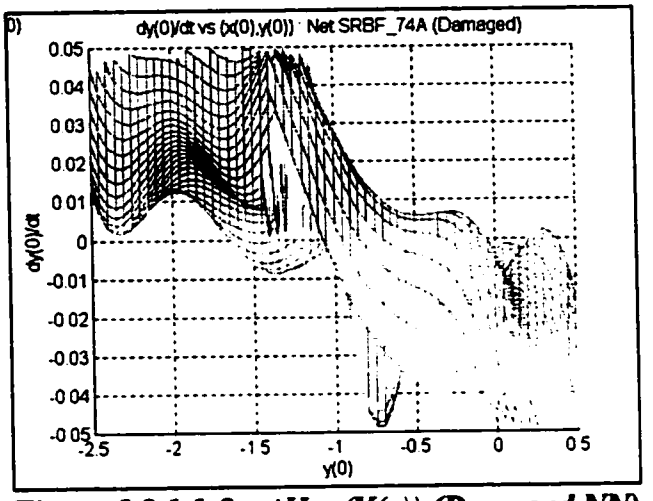


Figure 5.8.1.1-8 : $\Delta H_{A\_y}(X(n))$ (Damaged NN)

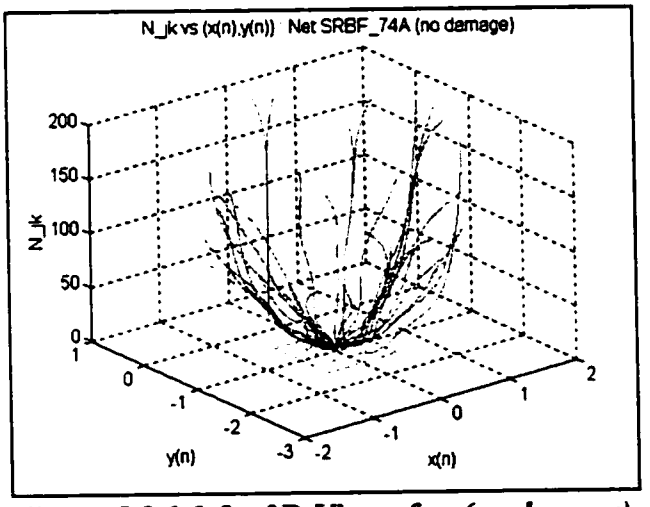


Figure 5.8.1.1-9 : 3D View of c_A (no damage)

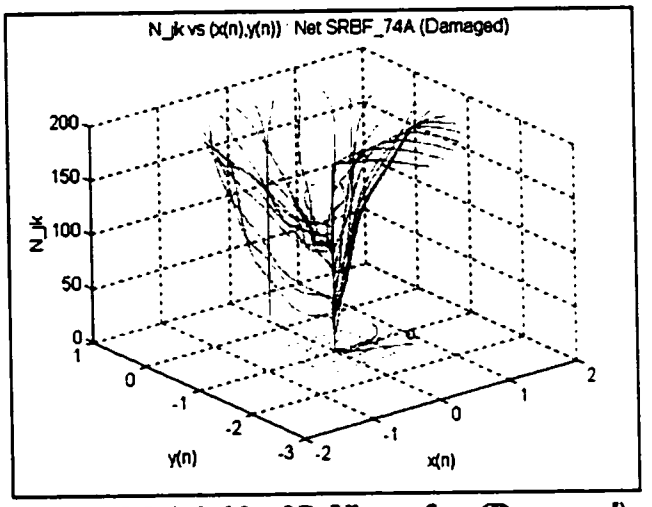


Figure 5.8.1.1-10 : 3D View of c_A (Damaged)

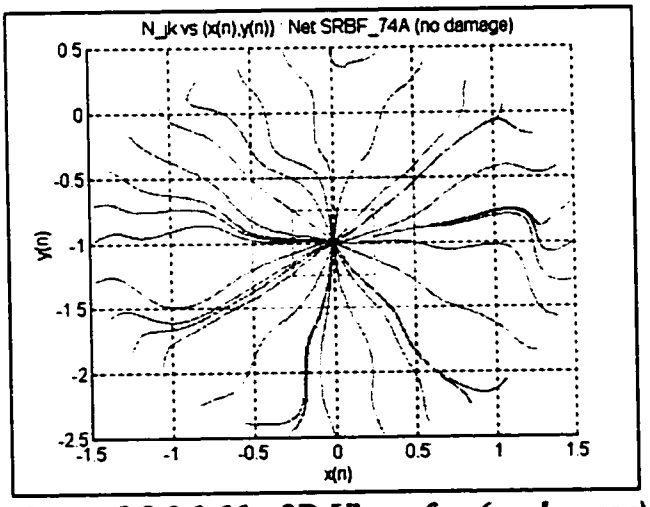


Figure 5.8.1.1-11 : 2D View of c_A (no damage)

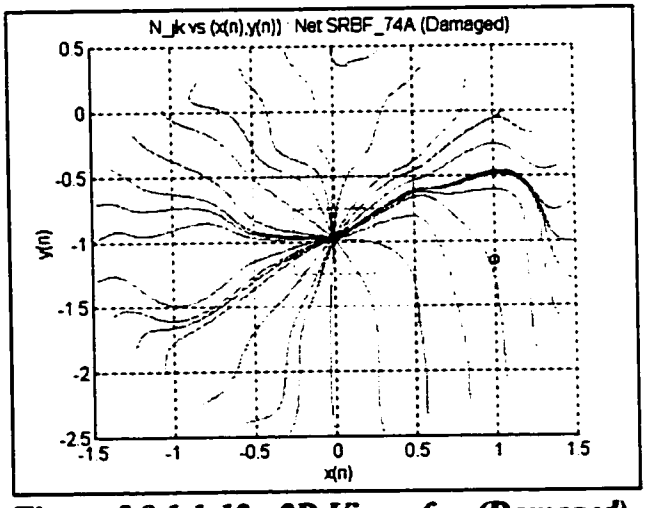


Figure 5.8.1.1-12 : 2D View of c_A (Damaged)

It is significant to note from Figure 5.8.1.1-12 that the damage has been more-or-less confined to the region where the dead neuron resided when alive. This is typically the case for RBF neurons, which perform local approximations only. For McCulloch-Pitts neurons, the damage is more generalized, which is consistent with their global approximation characteristic.

It is also relevant to note from Table T_5.8.1.1-1 that most neurons in the NN generally have a very high utilization factor. Hence, the death of any such neuron will have a significant impact on the NN performance. This is believed to be unlike the brain where the death of a few neurons rarely seems to have much of an effect [14,26]. This difference is believed to be due to an attribute of the brain which we have NOT attempted to include in the RTA NN model : **redundancy**.

In the brain, it is believed that thousands or millions of neurons are used to accomplish a mapping where a much smaller number would probably suffice [14,26]. Therefore, each neuron probably has a low U_f value (i.e., just slightly above 1.0) so that the death of a few will have little impact.

Note that such redundancy could be added in the RTA NN model. The mapping produced by each individual neuron in the hidden layer after initial training could be taught to another neural net with 100 hidden neurons (for example). It would then only be necessary to replace each hidden neuron of the original net with the newly-trained net with 100 neurons. By repeating this for every hidden neuron, the redundancy of the subnets of the RTA NN model could be increased in a straightforward, systematic way.

5.8.2 How to Repair Neural Damage and Restore Classification Performance

In this section, the process of repairing neural damage is explained and illustrated with several examples. The ability to restore classification performance through such repairs is also demonstrated.

Consider again Figures 5.8.1.1-5 and 5.8.1.1-6 from the previous section. In the former case, the $\Delta H_{A\_y}(X(n))$ represents the iterated mapping function required to ensure that $N^{\epsilon}_{A} = 20$ matches exactly the classification boundary B_{IN_A} when a suitable region attractor Γ^{R}_{A} has been defined as described in section 5.5.1. The $\Delta H_{A\_y}(X(n))$ shown in Figure 5.8.1.1-6 for the damaged NN shows a different iterated mapping which is no longer 'matched' to Γ^{R}_{A} . This results in a mismatch between the contour line and the classification boundary. But this situation is really no different than the problem we had initially when the NN was taught the trajectories with the point attractor Γ_A. In that case, the achieved $\Delta H_A(X(n))$ was different from the desired one and the N^{ϵ}_{A} contour line did not match the desired boundary either (e.g., see Figures 5.4.2-3 to 5.4.2-8, starting on page 176). It was then necessary to redefine the point attractor into a region attractor as explained in section 5.5.1 to correct for this mismatch. The process of repairing neural damage follows a similar approach.

The simplest way to proceed is to discard the old attractor region Γ^{R}_{A} from the distance object and repeat the steps of section 5.5.1, as if the damaged NN now represented the best possible mapping after an initial training session.

These steps were followed for the damaged NN of the previous section. The top of Figure 5.8.2-1 illustrates the 'original' region attractor Γ^{R}_{A} which the NN required to achieve the contour line mapping shown in Figure 5.8.1.1-3. The bottom of that figure shows the new Γ^{R}_{A} which is now required to ensure that the damaged NN will match the classification boundary B_{IN_A} with N^{ϵ}_{A} iterations. For convenience, Figures 5.8.2-2 to 5.8.2-4 show the N_{jk} contour lines of the original NN, the damaged NN and the repaired NN respectively. Although the contour lines of the repaired NN are different from those of the original net in several areas, the repaired net does match the classification boundary fairly well. Hence, its ability to classify inputs reliably has been restored to a significant degree. The distorted contour lines outside B_{IN_A} are not a major concern since the NN will be eliminated from the race for inputs in these regions when N^{ϵ}_{A} iterations have elapsed.

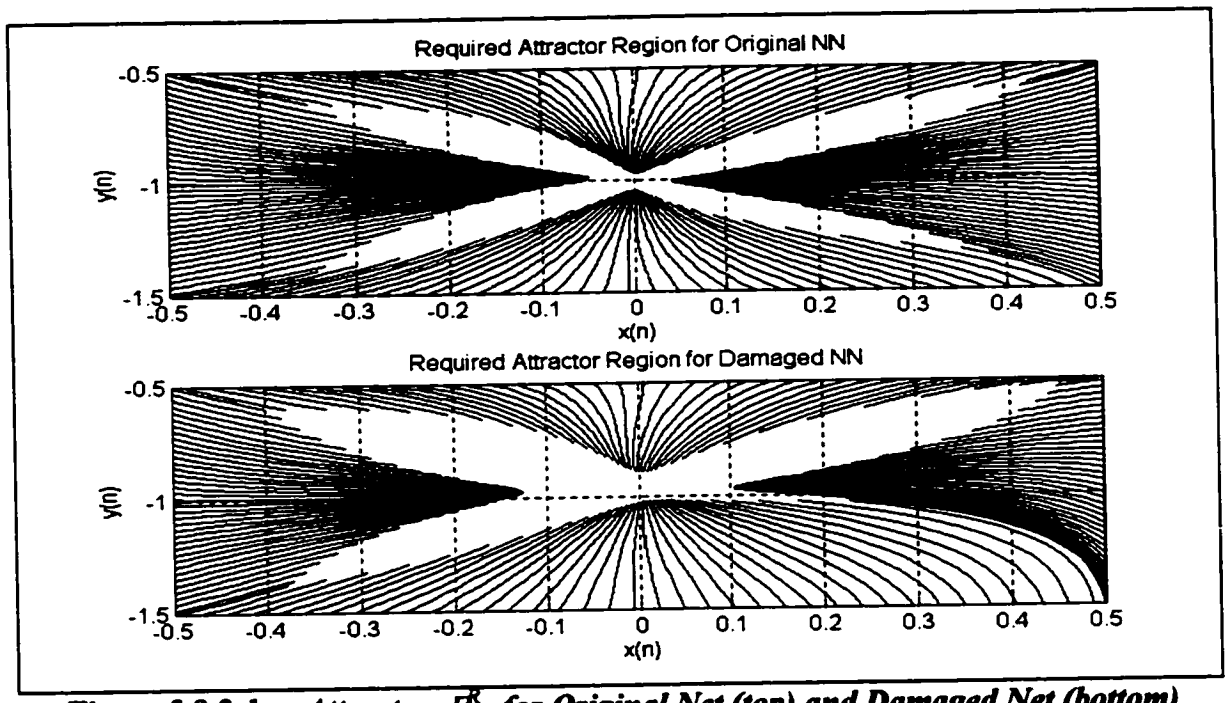


Figure 5.8.2-1 : Attractor Γ^{R}_{A} for Original Net (top) and Damaged Net (bottom)

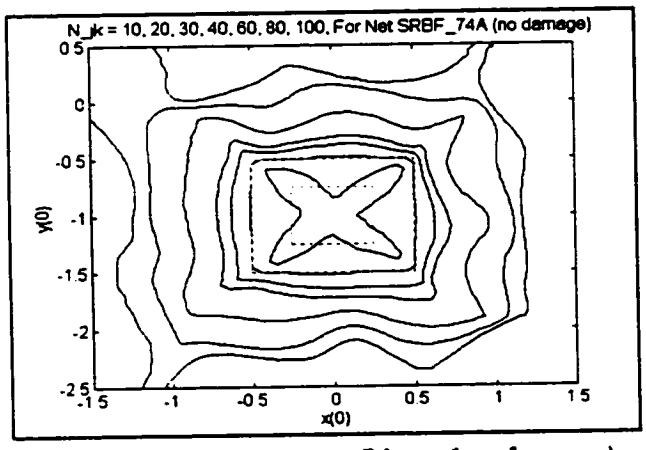


Figure 5.8.2-2 : Contour Lines (no damage)

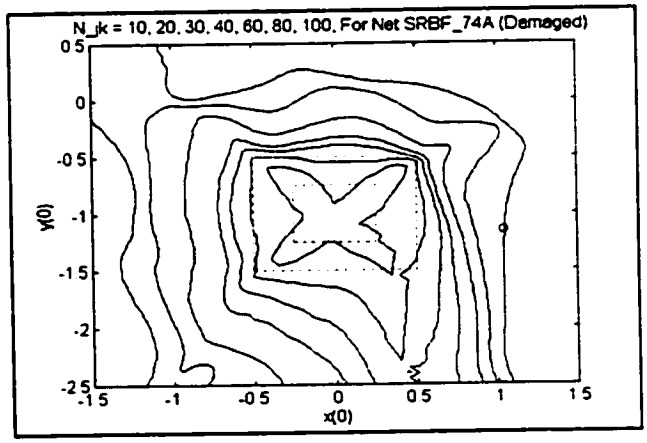


Figure 5.8.2-3 : Contour Lines (Damaged)

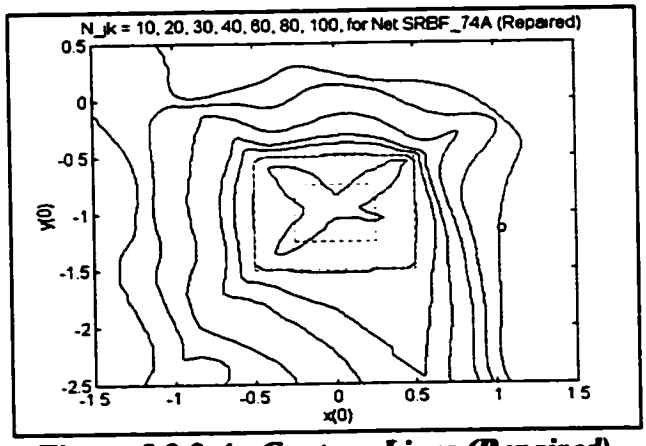


Figure 5.8.2-4 : Contour Lines (Repaired)

This repair technique has been tried with all the nets listed in section 5.8.1.1 and with different neurons killed. In many cases, the repaired NN regained most of its ability to classify without errors. One other example (which we label NN-2) is illustrated in Figures 5.8.2-5, 5.8.2-7 and 5.8.2-9. These figures show the mapping performance of the original NN, damaged NN and repaired NN respectively. This example involved a dead RBF neuron of U_f = 238.6 with (μ_x, μ_y) = (-0.7,1.1) for NN SRBF_74A.

Figures 5.8.2-6, 5.8.2-8 and 5.8.2-10 show another example (i.e., NN-3) for the coupled net structure SRBF_68A with a dead RBF neuron of U_f = 89 and (μ_x, μ_y) = (-1.3,-0.2). The circle 'o' shown in some of these figures identifies the location of the center (μ_x, μ_y) of the RBF neuron in the input space. Figures 5.8.2-11, 5.8.2-13 and 5.8.2-15 show an example based on the LRBF_74A structure (NN-4) with a dead LR-RBF neuron of U_f = 59.4 and (μ_x, μ_y) = (-0.99, 0.12). Figures 5.8.2-12, 5.8.2-14 and 5.8.2-16 show an example based on the CRBF_74A structure (NN-5) with a dead CR-RBF neuron of U_f = 2.22 and (μ_x, μ_y) = (-0.01, 0.03).

In some cases, the damage is too extensive to be repaired by this approach. Figures 5.8.2-17 and 5.8.2-18 illustrate two examples based on SRBF_74A where this is the case (see also Figure 5.8.1.1-9 (page 207) for the trajectories of the undamaged net). In one instance, the attractor has been split into two main attractors when a RBF neuron involved in the $H_{A\_x}(X(n))$ mapping was killed (Figure 5.8.2-17). That neuron had a U_f = 80.2 and (μ_x, μ_y) = (-1.02,-0.54) (shown as a circle 'o' in Figure 5.8.2-17). In another case, the attractor was shifted outside the classification boundary B_{IN_A} when a McCulloch-Pitts neuron involved in the $H_{A\_x}(X(n))$ mapping (with U_f = 76.9) was killed (Figure 5.8.2-18). Hence, there is a limit to how much damage (and which type of damage) can be fixed with this mechanism. However, note that the 'real' brain also has limits to its ability to repair damage. It was argued in section 5.8.1.1 that each neuron in the brain probably has a U_f just slightly above 1.0 as a result of the massive redundancy present. Hence, the trauma in the brain which would be equivalent to a U_f of 20-250 (as tested in the RTA NN) probably corresponds to the death of hundreds of thousands of neurons (i.e., severe damage).

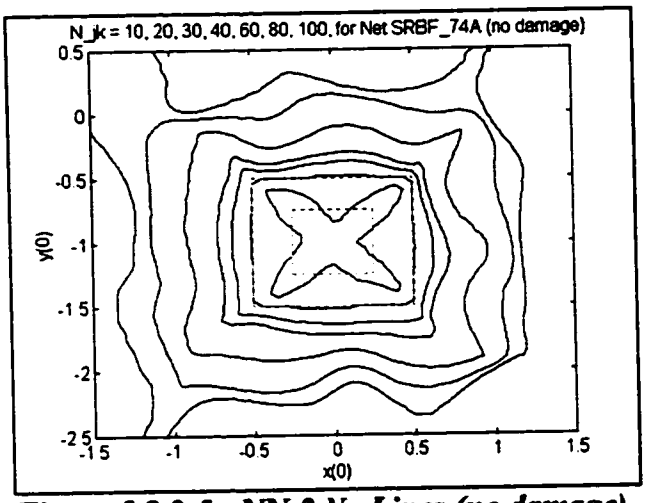


Figure 5.8.2-5 : NN-2 N_{jk} Lines (no damage)

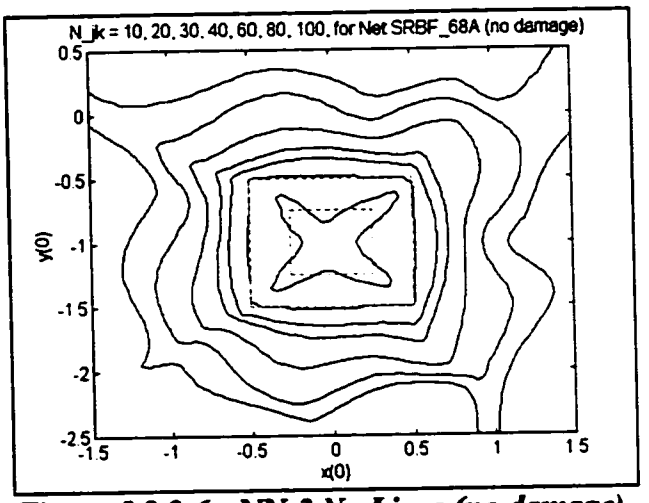


Figure 5.8.2-6 : NN-3 N_{jk} Lines (no damage)

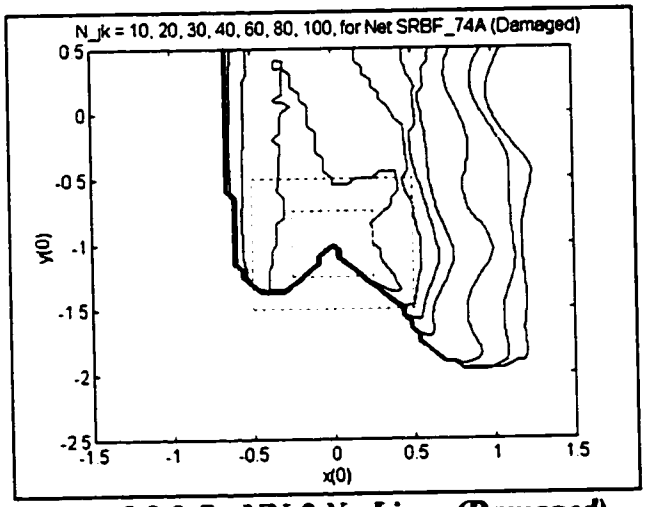


Figure 5.8.2-7 : NN-2 N_{jk} Lines (Damaged)

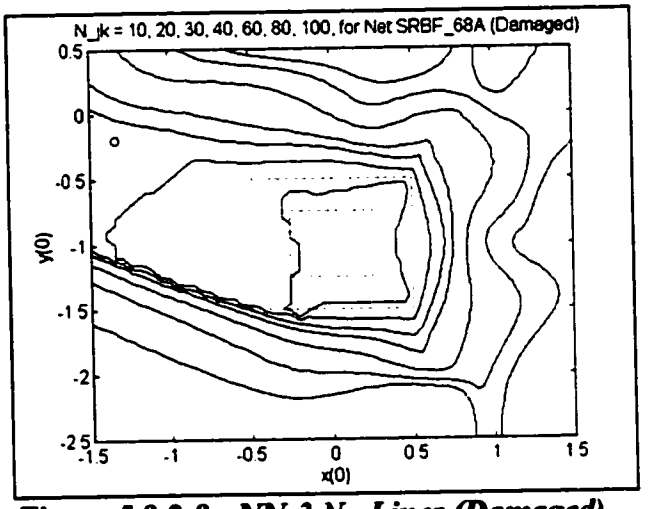


Figure 5.8.2-8 : NN-3 N_{jk} Lines (Damaged)

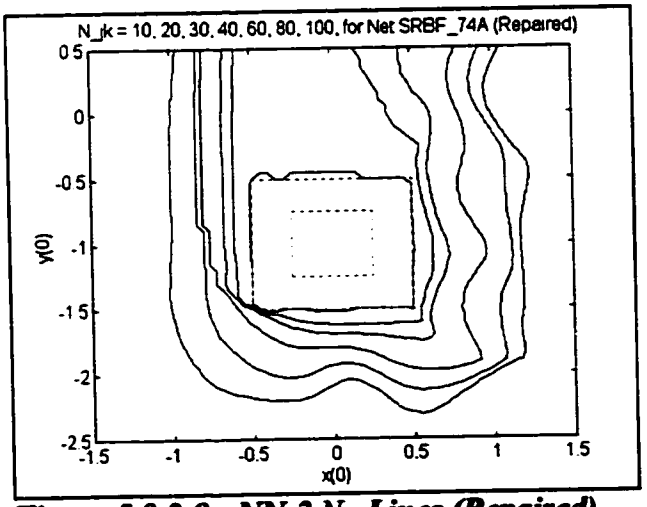


Figure 5.8.2-9 : NN-2 N_{jk} Lines (Repaired)

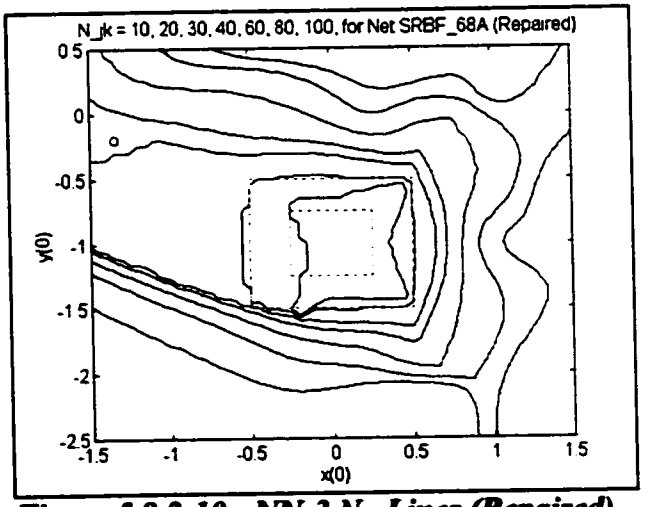


Figure 5.8.2-10 : NN-3 N_{jk} Lines (Repaired)

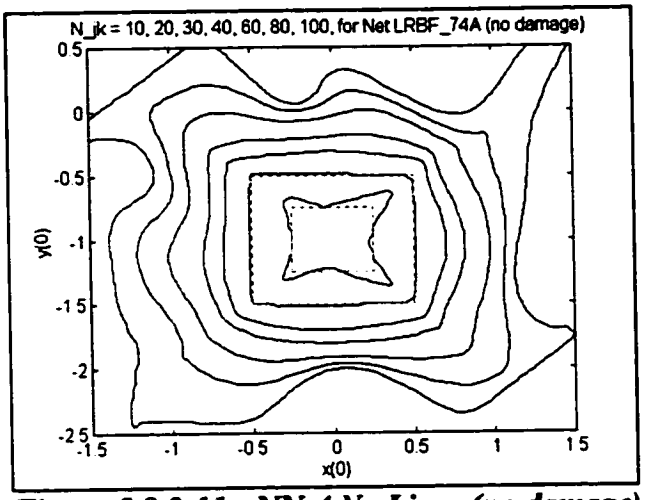


Figure 5.8.2-11 : NN-4 N_{jk} Lines (no damage)

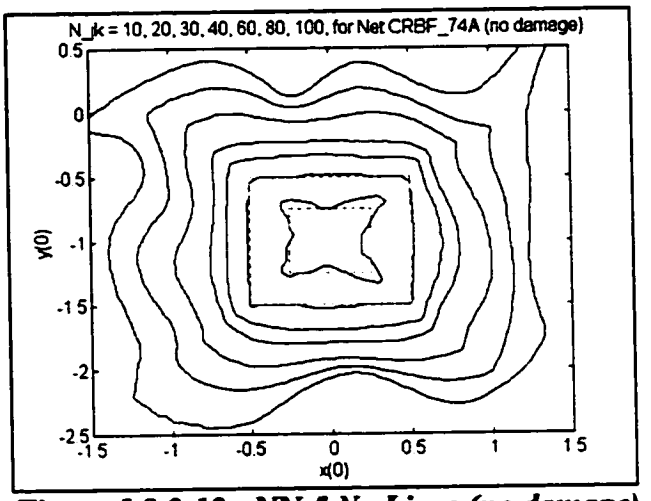


Figure 5.8.2-12 : NN-5 N_{jk} Lines (no damage)

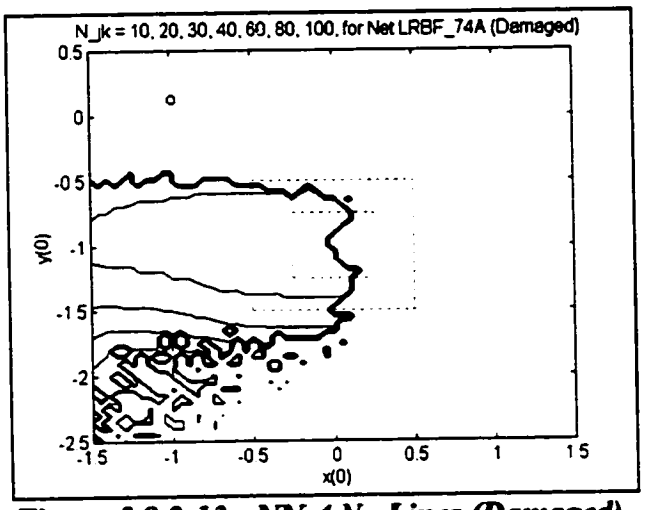


Figure 5.8.2-13 : NN-4 N_{jk} Lines (Damaged)

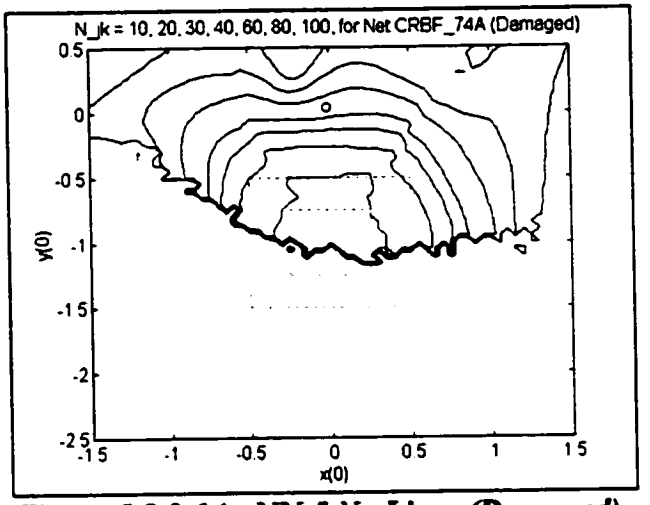


Figure 5.8.2-14 : NN-5 N_{jk} Lines (Damaged)

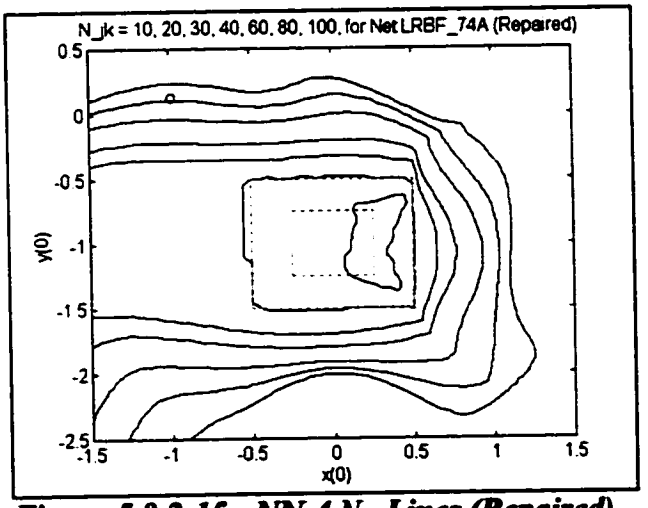


Figure 5.8.2-15 : NN-4 N_{jk} Lines (Repaired)

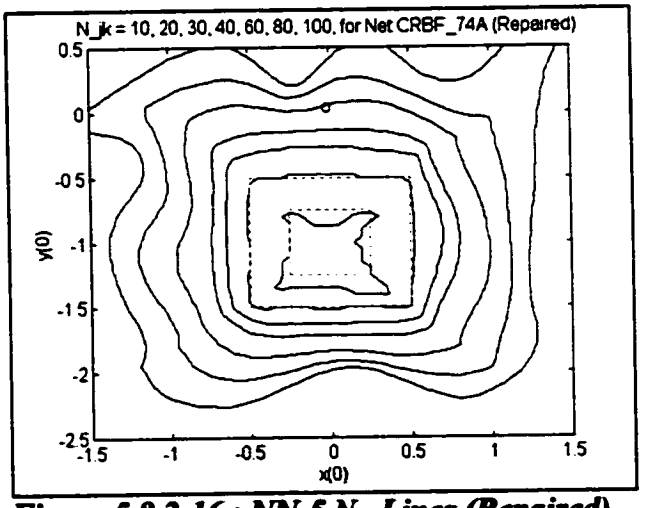


Figure 5.8.2-16 : NN-5 N_{jk} Lines (Repaired)

Nevertheless, the examples described in this section demonstrate that the technique of redefining the attractor provides the RTA NN model with an effective mechanism through which a moderate amount of neural damage can be repaired relatively easily.

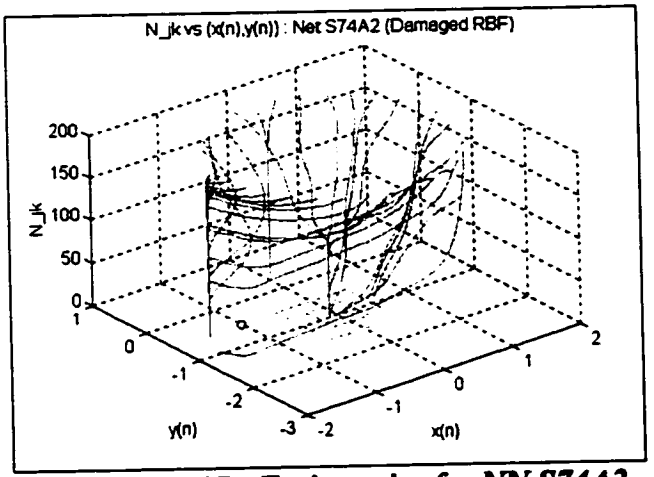


Figure 5.8.2-17 : Trajectories for NN S74A2

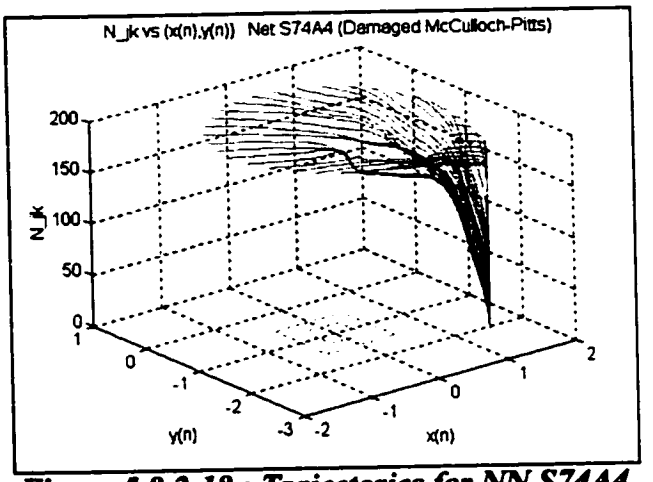


Figure 5.8.2-18 : Trajectories for NN S74A4

5.9 Disadvantages and Limitations of the RTA NN Model Paradigm

So far in this dissertation, only the capabilities and advantages of the RTA NN model paradigm have been discussed at length. But in the interest of objectivity, it is also essential to discuss disadvantages and limitations as well. These are discussed in this section.

Each major disadvantage or limitation of the RTA NN model is covered in a separate subsection. In some cases, ideas are proposed through which these limitations or disadvantages could possibly be overcome.

5.9.1 Lack of Coupling Between NDS Subnets of the RTA NN Model

In the RTA NN model, the '*R*' NDS implemented with '*R*' neural networks are *independent* of each other. In simple terms, this means that there is no interference between nets, so that each one 'runs' in the race as if it was alone. This situation is

analogous to a horse race where each horse stays in its track and is unaffected by the other horses.

By contrast, a roller derby where skaters actively block and hit each other during the race would be analogous to a situation where the various NDS are NOT independent but *coupled* [15,56]. In such a case, there would be significant interaction (and 'real' competition) between the various NDS.

The presence of such competition in neural nets is not new. For example, the Kohonen net (section 2.5.1.3.2, on page 54) makes extensive use of feedback connections to ensure that neurons compete against each other before a winner emerges. In section 3.6.3 (page 89) it was argued that the Freeman neurobiological model also utilizes some form of competition to the extent that various nerve cell assemblies all compete to lure the NDS towards a specific attractor.

Coupled NDS often exhibit behavior which cannot be replicated accurately by uncoupled NDS [15,40,56]. Hence, it is possible that the RTA NN model is limited in the type of classification tasks it can implement as a result of its usage of independent NDS. However, the significance of that limitation is not clear at this time.

5.9.2 NDS Limited to Systems where $X(n+1) = H(X(n))$

In chapters 2 to 5, the analysis and use of discrete-time NDS was limited to systems where the output at time '$n+1$' depends only on the state of the system at time 'n' (these are sometimes referred to as 'first-order' systems in the literature [15,56]). This was done partly because the mathematical analysis for systems where the 'memory' goes further back in time (e.g., $X(n+1) = H(X(n), X(n-1),...X(n-Q))$) is much more difficult [2,15]. Another reason for this omission is that systems which depend only on the state at the previous increment were sufficient to explain the RTA NN paradigm.

But higher order NDS can generate trajectories which first-order NDS are incapable of approximating adequately [2,56]. Hence, limiting the RTA NN model to first-order systems would reduce the model's ability to implement arbitrary classification problems. It is therefore desirable to allow the use of higher-order NDS in the RTA NN model.

5.9.3 Limited Development of the Incremental Learning Concept

The concept of incremental learning was discussed in section 3.5.1 (page 78). In that section, the manner in which the RTA NN model could learn trajectories, attractors and convergence rates incrementally was discussed in general terms.

However, a detailed algorithm to actually implement such incremental learning does not exist at this time (recall that training in chapter 5 was done using 'memorization learning', as described in section 3.5.2 (page 86)). Much work remains to be done in order to define an algorithm which is guaranteed to converge to an optimal solution.

5.9.4 Classification Regions must be Connected Regions in the Input Space

One limitation of the RTA NN model as described in this thesis is that the classification regions $R_{IN\,j}$ are restricted to be *connected regions* [3] in the input space.

In simple terms, a region $R_{IN\,j}$ is connected if, for any two points in the region, a trajectory joining them exists such that the trajectory never exits the region $R_{IN\,j}$. For example, the classification regions shown in Figure 3.3.1-2 (page 74) and Figure 5.3.1-1 (page 159) are all connected but the one shown in Figure 5.9.4-1 is not. In the latter figure, the classification region $R_{IN\,j}$ is disconnected so that points in the region to the right cannot reach the attractor in the region to the left while staying inside $R_{IN\,j}$.

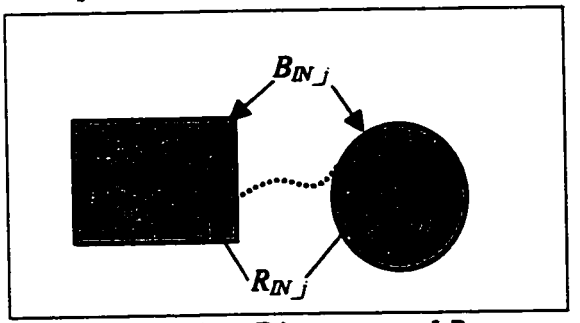


Figure 5.9.4-1 : Disconnected $R_{IN\,j}$

This limitation of the RTA NN model exists because the convergence rate function $\partial X/\partial t$ of a NDS is constrained to be continuous in practice (refer again to section 4.4.1 (page 110) for a discussion of this subject). As a result of this, a trajectory starting in one of the disconnected regions making up R_{IN_j} cannot 'jump' into the other region containing the attractor without touching the space in between.

For the RTA NN model, it is essential that the trajectories with initial points $X_k(0) \in R_{IN\_j}$ remain in that region in order to ensure that the condition $N_{jk} \leq N^{\epsilon}_j$ is always met (i.e., from equation E_4.4-8, on page 108). This requirement can be explained with the help of Figure 5.9.4-1. In that figure, a trajectory starting at point $X_k(0)$ will leave the region R_{IN_j} to the right before reaching the attractor in the region to the left, as a result of $\partial X/\partial t$ being continuous. But points on that trajectory <u>outside</u> B_{IN_j} must have a time to convergence $N_{jk} > N^{\epsilon}_j$ in order be classified as OUTSIDE class C_j by the NDS (i.a.w. equation E_4.4-8). However, because point $X_k(0)$ is further away from the attractor on that same trajectory, it must have a N_{jk} which is larger than those points, as explained in section 4.4.1. But this contradicts the requirement that $N_{jk} \leq N^{\epsilon}_j$ in order for $X_k(0)$ to be classified correctly as 'inside' class C_j. This dilemma cannot be resolved by the NDS.

However, there is a way to allow the RTA NN model to handle such disconnected regions. This would involve defining 'teams' of neural nets where there are as many nets as there are disconnected regions. Then, each NN implements a NDS with a connected region having its own attractor. Whenever one NN of the team wins, the input is classified as a member of the class which the team represents as a whole. In such a race, it is largely irrelevant which member of the team actually wins. Figure 5.9.4-2 illustrates this concept for the example of the disconnected regions of Figure 5.9.4-1.

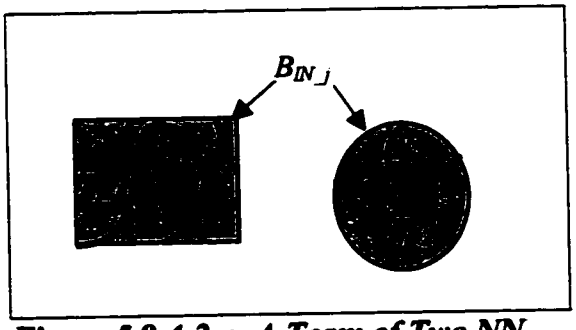


Figure 5.9.4-2 : A Team of Two NN

The significance of the NDS attractor as a class prototype must necessarily be changed in such a system since the class is now defined by several attractors. These attractors can be

viewed as a set of *intermediate prototypes*. One can then interpret the classification solution as consisting of a NDS which has the property that all points $X_k(0)$ which are member of the class C_j have a trajectory which intersects one intermediate attractor from that set while points outside that class intersect none of these attractors. Figure 5.9.4-3 illustrates an example of a hypothetical three-dimensional NDS which has this property (the intermediate attractors are those shown in Figure 5.9.4-2). Note that in that example, there exists a global prototype attractor in a different plane than the intermediate attractors. In such an example, it is not necessary to allow the trajectories to reach the global attractor since it is known that they are members of the class as soon as they reach the intermediate attractors. In a sense, one can view these intermediate attractors as a disconnected *region attractor* I^R_j in the same way that such attractors were utilized in section 5.5.1 (page 183).

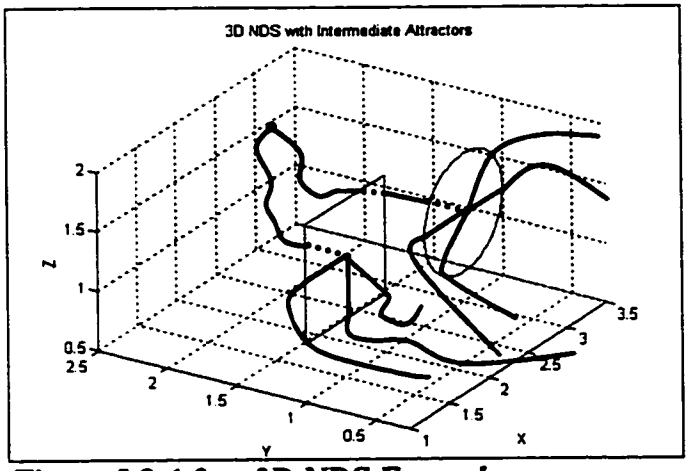


Figure 5.9.4-3 : 3D NDS Example

5.9.5 Each Subnet Must be Trained Over the Entire Input Space

In order to ensure that each neural net in the RTA NN model classifies all inputs correctly, the training trajectories must span the entire input space R_{IN} over which such inputs are expected to occur. For the example of Figure 5.3.1-1 (page 159), this means that each neural net must be taught trajectories which extend over the entire range:
$\{(x,y) \mid x \in [-1.5,2], y \in [-2,2]\}$.

But if the input space R_{IN} is doubled in size (for example), the number of training points for each NN will increase significantly, resulting in an exponential increase in the complexity of the training task as a result of the curse of dimensionality (as discussed in section 4.7.1, starting on page 145). As explained in chapters 2 to 5 this curse is less of a

problem for the RTA NN than for conventional classification paradigms. However, the RTA NN model advantage will become meaningless in practice when the input space region R_{IN} is very large.

One possible way of reducing the impact of a very large input space might be to partition R_{IN} into several subspaces and then use a separate RTA NN supernet for each subspace.

5.9.6 Limited Ability to Repair Neural Damage

It was shown in chapter 5 that the RTA NN model is capable of repairing the damage caused by the death of neurons in some cases (see section 5.8.2, starting on page 208). However, this repair ability did not exist when the neural damage was such that the NDS point attractor Γ_j became split into several attractors or when the attractor was shifted outside the classification region. Examples of these situations were shown in Figures 5.8.2-17 and 5.8.2-18 (page 214).

In some of these cases, it might be possible to repair the damage with the use of suitable offsets (e.g., as shown in Figure 5.6-1 on page 194) or by using a disconnected region attractor Γ^R_j (i.e., one region segment around each attractor of the damaged NDS).

5.10 Summary and Conclusions

This chapter discussed the ability of a RTA NN supernet to achieve a difficult classification problem in practice.

The types of neural nets used and the training algorithm used were described in section 5.2. Section 5.3 described the two-dimensional classification problem which the supernet was required to learn. That section also demonstrated how the learning task complexity was reduced by the process of orthogonalizing the iterated mapping functions learned by the subnets.

The results of training the subnets on the desired classification maps was discussed in section 5.4. It was shown that, in general, the subnets were unable to replicate the desired mapping functions adequately because the complexity of these functions was still too high. However, it was shown in section 5.5 that some of that complexity could be transferred to the distance object of the RTA NN model through the mechanism of redefining the attractor from a point to a region. By using this mechanism, the subnets were able to achieve the desired classification boundaries to a high level of accuracy. As a result, the RTA NN was able to achieve reliable classifications with a performance close to that of the optimal Bayesian classifier (the latter was discussed in chapter 4).

Section 5.6 demonstrated how a subnet trained on the NDS data of one class could be re-used to achieve the desired mapping for another class. It was shown that, through the use of this mechanism, the overall complexity of a learning task could be reduced significantly. As a result, the 'curse of dimensionality' [26] (or 'curse of complexity' in our case) was mitigated to a considerable extent.

In section 5.7, it was shown how the various mechanisms available to the RTA NN model could enable it to learn a new class with little or no memory loss. It was also shown that these mechanisms result in a learning task complexity which was limited to the complexity of the new classification boundary itself (i.e., no exponential growth in complexity as new classes are added).

Section 5.8 demonstrated how neural damage in the subnets of the RTA NN model could be repaired to a significant extent through the mechanism of redefining the attractor. Finally, section 5.9 described some of the main disadvantages and limitations of the RTA NN model. Where possible, ideas were proposed which could mitigate or eliminate these.

Recall that the main objective of this chapter was to demonstrate that the RTA NN model achieves the four key characteristics of learning in the brain described in section 5.1 (page 151) and that it does so through the mechanisms listed in that section. The successful simulation results presented in this chapter allow us to claim that this objective

has been met. A secondary objective of this chapter was to identify the main disadvantages and limitations of the RTA NN model. This has been achieved through the discussion in section 5.9.

Chapter 6 :

Conclusions and Future Work

6.1 Introduction

In this chapter, we examine again the fundamental tenet of the thesis. In light of the analysis and results presented in chapters 2 to 5, we assess to what extent the thesis has been proven true. We then look forward to the future and identify areas where valuable research could be conducted in order to improve on the RTA NN model.

Section 6.2 provides a brief summary of the information presented in this dissertation. Section 6.3 revisits the statement of the thesis and provides relevant conclusions. Section 6.4 discusses possible areas of future research.

6.2 Summary of Thesis

In chapter 2, a model was introduced which sought to explain how learning and recognition of patterns occurs in the 'real' brain (i.e., the Freeman model [23]). In this model, unique class prototypes are learned as unique chaotic attractors by an area of the brain which is essentially a large NDS with the capacity to store many such attractors. Recall is triggered by Nerve Cell Assemblies (NCA) which respond to input (e.g., visual, odorant, etc) by exciting this NDS in such a way that the NDS converges to the chaotic attractor corresponding to the class to which the input belongs.

Following this discussion, chapter 2 described four key characteristics which make the brain such a good classifier (section 2.3 on page 37). It was then shown how these characteristics could be achieved with a software/mathematical model by using four

mechanisms (these are summarized in section 2.4.5, starting on page 48). This discussion was followed by a review of conventional classification systems which utilize neural networks. These were based on the paradigms of one-step associative memory or neurodynamical systems. It was argued that none of these solutions achieve the key characteristics of learning in the brain to a significant extent.

In chapter 3, the RTA NN model paradigm was introduced. It was argued that the fundamental axiom of that paradigm is as follows: *an unknown input vector is iterated by a NDS in such a way that the time required for the input to converge to the NDS attractor is inversely proportional to the probability that the input is a member of the class whose prototype is the NDS attractor* (i.e., axiom E_3.3-1 on page 71). As long as this axiom is enforced, classification is achieved by racing several independent NDS (one for each class) and simply noting which NDS wins the race. The RTA NN paradigm was then compared to the conventional paradigms introduced in chapter 2. It was argued that the use of 'time to convergence' in the RTA NN model constitutes a novel approach to solve classification problems. It was also argued that the RTA NN model achieves the four key learning characteristics to a higher degree than those traditional paradigms by using the four mechanisms described in chapter 2.

In chapter 4, a mathematical analysis of the RTA NN model paradigm was performed. The relationship between Bayesian classification theory, NDS theory and the fundamental axiom of the RTA NN model (equation E_3.3-1, page 71) was clarified. It was shown how these various aspects of the classification problem lead to the fundamental relationship captured by equation E_4.5.2-2 (page 117). This equation defines how the convergence rate of a NDS must be related to the class membership probability in order to enforce that fundamental axiom of the RTA NN (thereby allowing optimal classification performance to be achieved). The definition of this convergence rate in practice was also discussed at length in that chapter.

In chapter 5, the performance of the RTA NN model on a sample classification problem was described. The ability of the RTA NN model to achieve the four key learning

characteristics introduced in chapter 2 was demonstrated. It was also shown how the four mechanisms described in chapter 2 were implemented in practice in order to achieve these characteristics. This chapter also discussed the main disadvantages and limitations of the RTA NN model.

6.3 Conclusions

Time is an essential and pervasive element of the computational process in the brain. The fundamental tenet of this thesis is that it is possible to model the pattern recognition process in the brain with a novel paradigm where time is the key element used to determine the classification answer.

The RTA NN model sought to capture the importance of time in the computational process and also mimic the time-dependent dynamical evolution of states which the brain utilizes. The results presented in this thesis indicate that reliable classification of patterns can indeed be achieved with the RTA NN model.

The RTA NN model was also shown to possess several key advantages which traditional paradigms based on neural networks fail to achieve to a similar extent. One of these is the ability to learn complex classification boundaries using only simple neurons. Another one is the ability to learn incrementally (i.e., without memory loss). A third advantage is the ability to learn a new pattern class with a level of effort related mostly to the complexity of the class boundary itself (and NOT related to how many classes are already in memory). Finally, the RTA NN model exhibits robustness through its ability to repair neural damage and restore its classification performance almost entirely. As was mentioned in chapter 2, these advantage are more commonly associated with the human brain than with conventional classifier models. As a result, the RTA NN paradigm not only captures the importance of time in the computational process of the brain; it also achieves advantages of the brain (as a classifier) to a higher degree than these other models.

It was stated in chapter 2 that no one really knows how the brain works. At the end of this thesis, this statement remains true. Nevertheless, the classification paradigm presented here constitutes a small step forward in the quest for a better understanding of the classification process in the brain and in the quest to build a better classifier to solve practical engineering problems.

6.4 Ideas for Future Research Work

In section 5.9 of chapter 5, several limitations and disadvantages of the RTA NN model were introduced. In this final section, we propose some areas where future research could allow many of these to be eliminated:

1. The use of independent NDS (i.e., uncoupled NDS) may limit the type of trajectories which can be implemented by the RTA NN. Further research in the use of coupled NDS is therefore desirable.

2. Another limitation of the model described in this thesis is due to the fact that only NDS systems with memory limited to one time step in the past were considered. Here again, the range of achievable trajectories is limited as a result of this. Hence, more research in the use of NDS with memory extending several increments in the past is warranted.

3. Further research is also necessary in order to define a detailed incremental learning algorithm which can be used in practice.

4. Another area of concern is that the current RTA NN model is limited to the use of connected classification regions. Since disconnected regions can occur in real classification problems, it is desirable to research various approaches to overcome this limitation.

5. The use of multiple RTA NN supernets to cover a large input space is also an area of interest for further research.

6. Research to define additional approaches to repair neural damage would be advantageous since the mechanism described in this thesis is limited in some respects.

On a more ambitious scale, we must highlight the need to develop a better model which truly captures the manner in which the brain seems to operate (e.g., the Freeman model described in chapter 2). The prize of success in such an endeavor is enormous when one considers the amazing ability of the brain to learn complex tasks and to successfully solve difficult classification problems.

Appendix A :

Performance and Stability Issues for Linear Recurrent Radial Basis Function (LR-RBF) Neurons and Center-Recurrent RBF (CR-RBF)

A.1 Introduction

In this appendix, two types of locally recurrent RBF neurons are described and analyzed : the linear recurrent RBF (LR-RBF) and the center-recurrent RBF (CR-RBF). For each type, the error gradient equations required for parameter update with backpropagation training are developed in section A.2 and compared to those of the simple RBF (S-RBF) neuron (i.e., a static RBF neuron without feedback). Then, the stability of these recurrent RBF neurons is investigated in section A.3 and conditions under which they exhibit periodic and/or chaotic behavior are examined.

Section A.4 discusses how the update of net parameters during backpropagation training can change the stability properties of these neurons. The critical importance of these stability characteristics to the success or failure of the training session is illustrated with an example. Based on these results, suggestions are proposed to help increase the probability of success of the training session. Section A.5 summarizes the arguments presented in this appendix.

A.2 Architecture and Training Equations for LR-RBF and CR-RBF Neurons

The architecture for both the LR-RBF and the CR-RBF is illustrated in Figure A.2-1. Both neuron types involve the feedback of the neuron's previous output into a new coordinate, separate from those of the independent input vector $X(n)$. The mathematical descriptions of the LR-RBF and CR-RBF are provided in sections A.2.1 and A.2.2.

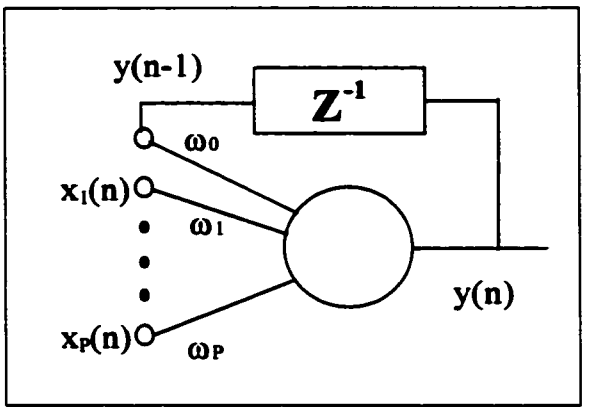


Figure A.2-1 : LR-RBF, CR-RBF Neuron

A.2.1 Equations for the LR-RBF Neuron

The output of the LR-RBF neuron 'i' is $y_i(n) = e^{v_i(n)}$ where $v_i(n)$ is given by:

$$v_i(n) = -\left[\omega_{i,0} y_i(n-1) + \sum_{j=1}^{P} \frac{(\omega_{i,j} x_j(n) - \mu_{i,j})^2}{\sigma_{i,j}^2}\right] \qquad \text{(E_A.2.1-1)}$$

Since $y_i(n-1) \in [0, 1] \ \forall\, n$, the response of the neuron at time n to the input $X(n)$ is effectively attenuated when the neuron was active at the previous iteration (i.e., $y_i(n-1) > 0$). Similarly, the sensitivity of the neuron is maximum at time n when it was previously inactive (i.e., $y_i(n-1) = 0$). In that way, the feedback term can be viewed as a mechanism to adjust the sensitivity of the neuron, based on its response (or lack thereof) to the previous input.

During gradient-based training, the parameters ω, σ and μ of equation E_A.2.1-1 can all be updated. However, the weights ω are usually fixed at a value of 1.0 because updating them is equivalent to updating the centers μ. This can easily be seen by comparing $(\alpha x - \mu)^2$ and $\alpha^2(x - \mu/\alpha)^2$, which are equivalent. Hence, any weight having a value different than 1.0 can be replaced with a value of 1.0 with a proper scaling for the center μ. A similar argument is applicable for any change required to ω as a result of training. Hence, to simplify calculations, all weight values are normally set at 1.0 and do <u>not</u> change during backpropagation training.

For a neuron i, the update of a parameter $p_{i,j}$ during backpropagation training involves the error gradient at the output of that neuron:

$$\frac{\partial E_i(n)}{\partial p_{i,j}} = \frac{\partial E_i(n)}{\partial y_i(n)} \frac{\partial y_i(n)}{\partial v_i(n)} \frac{\partial v_i(n)}{\partial p_{i,j}} \qquad \textbf{(E\_A.2.1-2)}$$

where $E_i(n)$ is the difference between the 'desired' output and the actual neuron output at time n. When the neuron is hidden, the error $E_i(n)$ is a weighted sum of the error at the output layer, propagated back to that neuron [26]. For a RBF neuron, $\partial y_i(n)/\partial v_i(n) = y_i(n)$ since $\partial e^u/u = e^u$. The third term of equation E_A.2.1-2 is our primary concern since it captures the memory of the recurrent neuron. For the center parameters μ of the neuron,

$$\frac{\partial v_i(n)}{\partial \mu_{i,l}} = \frac{\partial y_i(n-1)}{\partial \mu_{i,l}} - \sum_{j=1}^{P} \frac{\partial}{\partial \mu_{i,l}} \left(\frac{(x_j(n) - \mu_{i,j})^2}{\sigma_{i,j}^2} \right) \qquad \textbf{(E\_A.2.1-3)}$$

$$\text{where } \frac{\partial y_i(n-1)}{\partial \mu_{i,l}} = \frac{\partial y_i(n-1)}{\partial v_i(n-1)} \frac{\partial v_i(n-1)}{\partial \mu_{i,l}} = y_i(n-1) \frac{\partial v_i(n-1)}{\partial \mu_{i,l}} \qquad \textbf{(E\_A.2.1-4)}$$

The second term of equation E_A.2.1-3 can be simplified as follows:

$$\frac{\partial}{\partial \mu_{i,l}} \left(\frac{(x_j(n) - \mu_{i,j})^2}{\sigma_{i,j}^2} \right) = \begin{cases} -\dfrac{2}{\sigma_{i,l}^2}(x_l(n) - \mu_{i,l}), & j = l \\ 0, & j \neq l \end{cases} \qquad \textbf{(E\_A.2.1-5)}$$

As a result, $\partial v_i(n)/\partial \mu_{i,l}$ becomes:

$$\frac{\partial v_i(n)}{\partial \mu_{i,l}} = \frac{2}{\sigma_{i,l}^2}(x_l(n) - \mu_{i,l}) - y_i(n-1) \frac{\partial v_i(n-1)}{\partial \mu_{i,l}} \qquad \textbf{(E\_A.2.1-6)}$$

and the error gradient at neuron 'i' becomes:

$$\frac{\partial E_i(n)}{\partial \mu_{i,l}} = \frac{\partial E_i(n)}{\partial y_i(n)} y_i(n) \left[\frac{2}{\sigma_{i,l}^2} (x_l(n) - \mu_{i,l}) - y_i(n-1) \frac{\partial v_i(n-1)}{\partial \mu_{i,l}} \right] \qquad \text{(E_A.2.1-7)}$$

This gradient is then used to update the parameter $\mu_{i,l}$ at the end of a training epoch r, in accordance with:

$$\mu_{i,l}(r+1) = \mu_{i,l}(r) + \Delta\mu_{i,l}(r) \qquad \text{(E_A.2.1-8)}$$

$$\text{where } \Delta\mu_{i,l}(r) = -\eta \frac{\partial \overline{E}_i(r)}{\partial \mu_{i,l}} + \alpha \Delta\mu_{i,l}(r-1) \qquad \text{(E_A.2.1-9)}$$

In the latter equation, η is the training update rate and α is a *momentum* parameter used to improve learning [26]. In batch mode training, M training vectors are processed through the net and the error gradient is an averaged value:

$$\frac{\partial \overline{E}_i(r)}{\partial \mu_{i,l}} = \frac{1}{M} \sum_{j=1}^{M} \frac{\partial E_i(j)}{\partial \mu_{i,l}} \qquad \text{(E_A.2.1-10)}$$

whereas, in pattern mode training, the gradient involves only one training pair (i.e., one iteration) [26]. In a similar manner, the update rule for the variance parameter σ can be derived from:

$$\frac{\partial v_i(n)}{\partial \sigma_{i,l}^2} = -\frac{\partial y_i(n-1)}{\partial \sigma_{i,l}^2} - \sum_{j=1}^{P} \frac{\partial}{\partial \sigma_{i,l}^2} \left(\frac{(x_j(n) - \mu_{i,j})^2}{\sigma_{i,j}^2} \right) \qquad \text{(E_A.2.1-11)}$$

which simplifies to:

$$\frac{\partial v_i(n)}{\partial \sigma_{i,l}^2} = \frac{(x_l(n) - \mu_{i,l})^2}{(\sigma_{i,l}^2)^2} - y_i(n-1) \frac{\partial v_i(n-1)}{\partial \sigma_{i,l}^2} \qquad \text{(E_A.2.1-12)}$$

By contrast, a simple RBF neuron (without feedback) has these update equations [26]:

$$\frac{\partial v_i(n)}{\partial \sigma_{i,l}^2} = \frac{(x_l(n) - \mu_{i,l})^2}{(\sigma_{i,l}^2)^2}$$
$$\frac{\partial v_i(n)}{\partial \mu_{i,l}} = \frac{2}{\sigma_{i,l}^2}(x_l(n) - \mu_{i,l}) \qquad \textbf{(E\_A.2.1-13)}$$

Comparing these to equations E_A.2.1-6 and E_A.2.1-12, the memory inherent in the linear recurrent neuron is evident.

A.2.2 Equations for the CR-RBF Neuron

For the CR-RBF neuron, the output of neuron 'i' is $y_i(n) = e^{v_i(n)}$, where $v_i(n)$ is:

$$v_i(n) = -\left[\frac{(\omega_{i,0} y_i(n-1) - \mu_{i,0})^2}{\sigma_{i,0}^2} + \sum_{j=1}^{P} \frac{(\omega_{i,j} x_j(n) - \mu_{i,j})^2}{\sigma_{i,j}^2}\right] \qquad \textbf{(E\_A.2.2-1)}$$

To simplify the equations, the weights are normally kept fixed at a value of 1.0 at all times. Comparing equation E_A.2.2-1 to E_A.2.1-1, it can be argued that the CR-RBF is a generalization of the LR-RBF, with the former having the ability to adjust its sensitivity at time n to achieve a maximum when $y(n-1) = \mu_0$.

The development of the gradient equations for the CR-RBF is similar to that of the LR-RBF. It can easily be shown that the gradient for parameters μ takes this form:

$$\frac{\partial v_i(n)}{\partial \mu_{i,l}} = \begin{cases} -\frac{2}{\sigma_{i,0}^2}(y_l(n-1) - \mu_{i,0}) \times \left(y_i(n-1)\frac{\partial v_i(n-1)}{\partial \mu_{i,0}} - 1\right), & l = 0 \\ \frac{2}{\sigma_{i,l}^2}(x_l(n) - \mu_{i,l}) - \frac{2}{\sigma_{i,0}^2}(y_i(n-1) - \mu_{i,0}) \times y_i(n-1)\frac{\partial v_i(n-1)}{\partial \mu_{i,l}}, & l > 0 \end{cases} \qquad \textbf{(E\_A.2.2-2)}$$

while the equivalent equations for parameters σ take this form:

$$\frac{\partial v_i(n)}{\partial \sigma_{i,l}^2} = \begin{cases} \dfrac{(y_i(n-1)-\mu_{i,0})^2}{(\sigma_{i,0}^2)^2} - \dfrac{2}{\sigma_{i,0}^2}\left(y_i(n-1)-\mu_{i,0}\right) \times y_i(n-1)\dfrac{\partial v_i(n-1)}{\partial \sigma_{i,0}^2}, & l=0 \\ \dfrac{(x_i(n)-\mu_{i,l})^2}{(\sigma_{i,l}^2)^2} - \dfrac{2}{\sigma_{i,0}^2}\left(y_i(n-1)-\mu_{i,0}\right) \times y_i(n-1)\dfrac{\partial v_i(n-1)}{\partial \sigma_{i,l}^2}, & l>0 \end{cases} \qquad \textbf{(E\_A.2.2-3)}$$

Comparing equations E_A.2.2-2, E_A.2.2-3 with E_A.2.1-6 and E_A.2.1-12, it can be seen that the extra term $(y_l(n-1)-\mu_{i,0})$ in the CR-RBF equations scales the recursive term involving $v_i(n-1)$ in a such a way that the neuron behaves like a simple RBF when $y_i(n-1) = \mu_{i,0}$ (i.e., as per equations E_A.2.1-13). Otherwise, the gradient may be increased or decreased, depending on the signs of the individual terms in the equation.

A.3 Stability of LR-RBF and CR-RBF Neurons

Consider again equation E_A.2.1-1 for a LR-RBF neuron. Let us determine its stability around its singular points. Equation E_A.2.1-1 can be simplified by defining K_X as follows:

$$K_X = \sum_{j=1}^{P} \frac{(x_j(n)-\mu_j)^2}{\sigma_j^2} \qquad \textbf{(E\_A.3-1)}$$

where the independent variables $x_j(n)$ are assumed to be constant over several iterations at the attractor(s) of the neuron. Hence, the LR-RBF output can be expressed as:

$$y(n) = e^{-y(n-1)}e^{-K_x} \qquad \textbf{(E\_A.3-2)}$$

The local stability of the NDS represented by equation E_A.3-2 can be determined by linearization around the attractor(s) [56]. Here, $y(n) = F(y(n-1), K_x)$ so that a first-order Taylor series expansion results in $\Delta y(n) \approx A\Delta y(n-1)$ with

$$A = \left.\frac{\partial F}{\partial y}\right|_{y_o} \qquad \textbf{(E\_A.3-3)}$$

where y_o is an attractor point of the NDS. For a discrete-time NDS, local stability is guaranteed if all the poles of the linearized system are inside the unit circle in the Z-transform plane (i.e., the eigenvalues of matrix 'A' have a magnitude smaller than 1.0 [56]).

At the attractor(s) $y(n) \approx y(n-1)$ so that $\frac{\partial F}{\partial y} = -e^{-y}e^{-K_X}$ evaluated at y_o becomes:

$$y_o = e^{-y_o}e^{-K_X} \text{ or } \ln(y_o) + y_o + K_X = 0 \quad \textbf{(E_A.3-4)}$$

where $0 \le y_o \le 1.0$ and $K_X \ge 0$ is always true. When the input vector $X = \mu$, $K_X = 0.0$ so that y_o reaches a maximum value ≈ 0.56714. As K_X increases, y_o decreases towards 0.0. This relationship is illustrated in Figure A.3-1 for a few values of K_X. Evaluating equation E_A.3-3 at the attractors produces poles in the range $-0.56714 \le y_o \le 0.0$. Since their magnitude is always smaller than 1.0, the LR-RBF is always stable over its entire parameter space. Figure A.3-2 illustrates some iterated maps of the LR-RBF.

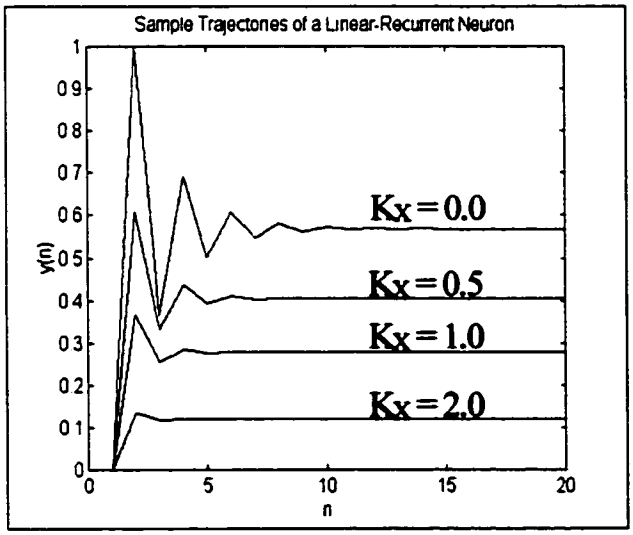


Figure A.3-1 : LR-RBF Neuron Trajectories

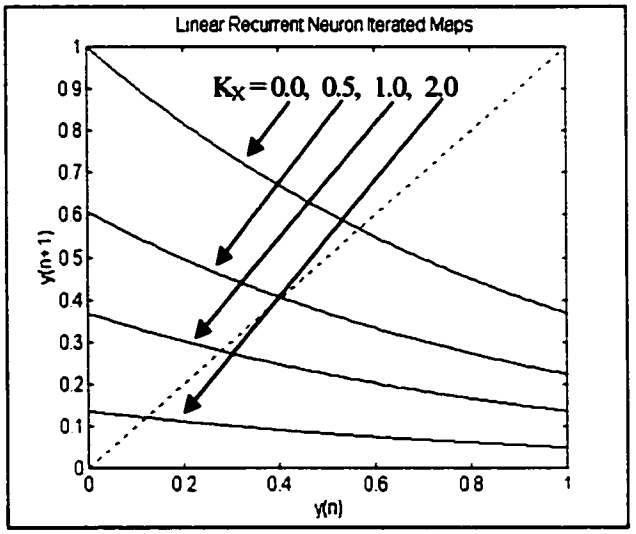


Figure A.3-2 : LR-RBF Iterated Maps

The Lyapunov exponent around an attractor point can be shown to be natural logarithm of the absolute value of equation E_A.3-3 [56]. It is known that a Lyapunov exponent smaller than 1.0 indicates a contraction mapping (i.e., a stable attractor) and conversely

for a value greater than 1.0. Hence, for the LR-RBF, the Lyapunov exponent is always smaller than 1.0 so that stability is always present, regardless of the neuron's parameters.

A.3.1 Stability of a CR-RBF Neuron

Here again, K_X is defined as shown in equation E_A.3-1 and is assumed constant over several iterations at the attractors of the CR-RBF neuron. The output of the neuron NDS can then be expressed as follows:

$$y(n) = e^{\frac{-(y(n-1)-\mu_0)^2}{\sigma_0^2}} e^{-K_X} \qquad \text{(E_A.3.1-1)}$$

where $y(n) = F(y(n-1), K_X)$ and linearization as per equation E_A.3-3 can be performed near the attractors to assess the stability properties of this system. Performing the derivation yields these results for the matrix A of eigenvalues:

$$\left.\frac{\partial F}{\partial y}\right|_{y_o} = \frac{-2y_o}{\sigma_0^2}(y_0 - \mu_o) \qquad \text{(E_A.3.1-2)}$$

where y_o are the attractor points of the NDS and stability requires that $-1 < \frac{-2y_o}{\sigma_0^2}(y_0 - \mu_o) < 1$. Note that, for the CR-RBF, the eigenvalue may have a magnitude greater than 1.0 so that the NDS can be unstable and even chaotic. By substituting E_A.3.1-1 into E_A.3.1-2 and simplifying, this equation is obtained:

$$\ln(y_o) + \frac{(y_o - \mu_o)^2}{\sigma_0^2} + K_X = 0.0 \qquad \text{(E_A.3.1-3)}$$

Depending on the values of μ_o σ_o and K_X, this equation may have 1, 2 or 3 solutions which may be stable, unstable and even lead to a chaotic regime. Figure A.3.1-1 illustrates a family of curves for equation E_A.3.1-3, with several values of K_X. The points which intersect with the horizontal dotted line at 0.0 are solutions of equation E_A.3.1-3 (i.e., attractor points). As K_X increases from 0.0, the number of solutions falls

from 3 to 2 and then to a unique solution near $y_0 \approx 0.0$. This is not surprising since K_X acts as an attenuation factor applied to the neuron's output $y(n)$ (see equation E_A.3.1-1).

Figure A.3.1-2 illustrates the iterated maps for the CR-RBF with the parameters used to generate Figure A.3.1-1. Each intersect point with the line $y(n) = y(n+1)$ represents an attractor point y_o for the NDS. It can be shown that when the slope of the curve at the intersect point has a magnitude greater than 1.0, the attractor is unstable (conversely, it is stable if the slope magnitude is less than 1.0) [56].

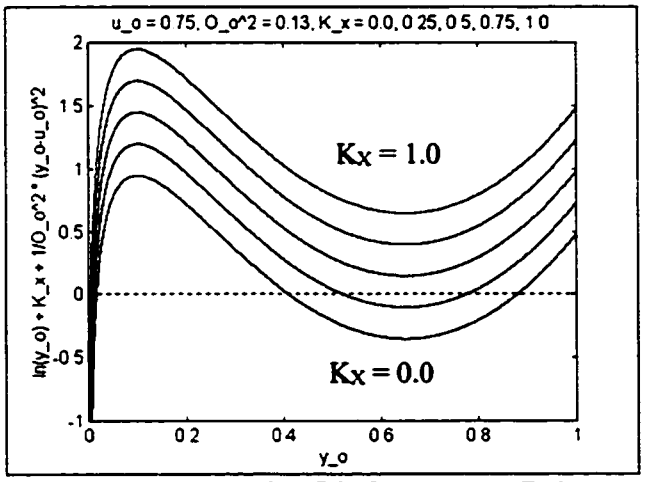


Figure A.3.1-1 : CR-RBF Attractor Points

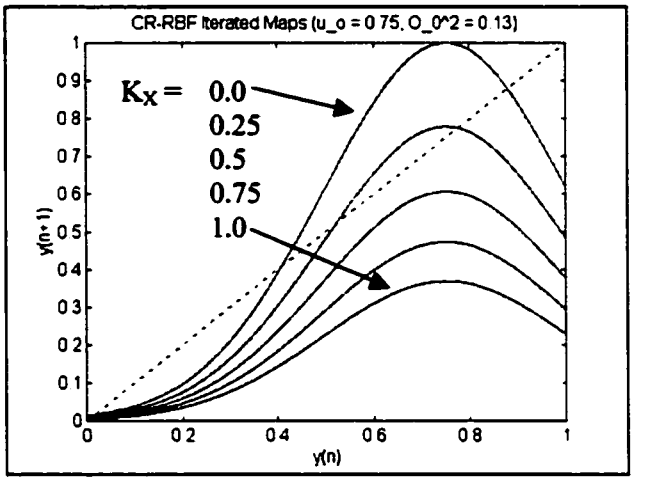


Figure A.3.1-2 : CR-RBF Iterated Maps

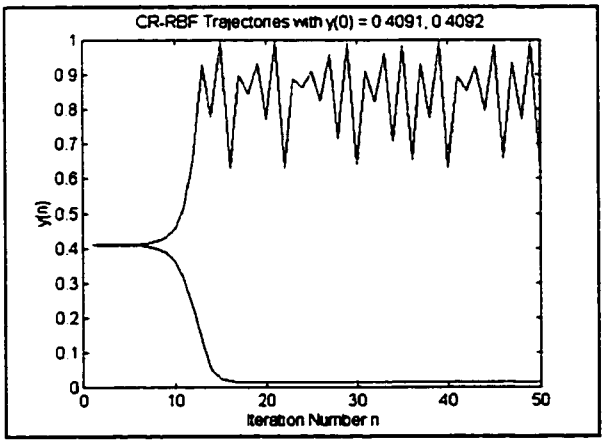


Figure A3.1-3 : CR-RBF Trajectories

Two trajectories for the CR-RBF of Figure A.3.1-1 with $K_X = 0.0$, $\mu_o = 0.75$ and $\sigma^2_0 = 0.13$ are shown in Figure A.3.1-3. Depending on the initial value $y(0)$, the CR-RBF either converges to a stable attractor or falls into a chaotic orbit around an unstable attractor.

In order to appreciate the rich complexity of the CR-RBF neuron as a NDS, it is necessary to examine its *bifurcation diagram* [40,56]. Such a diagram is generated by iterating the CR-RBF with different initial conditions for a large number of iterates (e.g., $I = 200$) and then plotting the values of the next R iterates (e.g., R = 300). If the CR-RBF has reached

a stable attractor after I iterations, the next R iterates will produce a single point in the bifurcation map for the given set of CR-RBF parameters. If the CR-RBF has settled into a periodic orbit of period K, the next R iterates will produce K fixed points on the bifurcation map. For a chaotic orbit (i.e., an orbit with an infinite period), the next R iterates will produce R points on the bifurcation map.

By systematically modifying the CR-RBF neuron parameters set and plotting these R iterated points for each set, the progression of the NDS from stability to chaos is mapped as a function of the CR-RBF parameters. Two bifurcation maps for the CR-RBF are shown in Figures A.3.1-4 and A.3.1-5, for values of $K_X = 0.0$ and 0.5 and as a function of parameters μ_o and σ^2_o. It can be seen that a decreasing value of σ_0^2 will result in period doubling over a large range of values for μ_0 and will lead to chaotic orbits when μ_0 is roughly in the range 0.1 to 0.7.

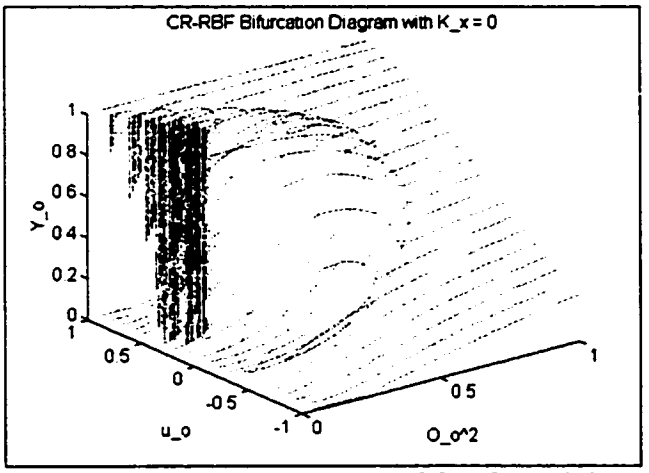


Figure A.3.1-4 : Bifurcation Map $K_X = 0.0$

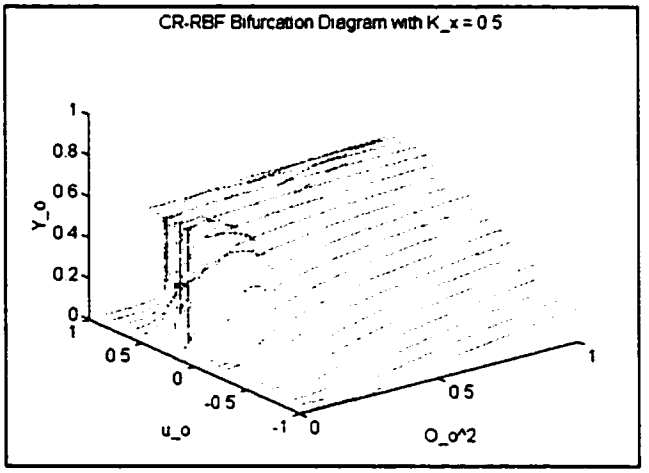


Figure A.3.1-5 : Bifurcation Map $K_X = 0.5$

It can be noted from Figure A3.1-5 that, when the independent input vector is <u>not</u> coincident with vector μ, (i.e., $K_X > 0$), the CR-RBF neuron becomes stable over a wider range of parameter values. Again, this is partly because K_X acts as a simple damping factor on the NDS, as evidenced by equation E_A.3.1-1.

When the CR-RBF neurons are trained with simple backpropagation, the parameter vectors μ and σ will migrate towards values which could make the CR-RBF neuron

stable or unstable. This possibility and its consequences are discussed briefly in the next section.

A.4 Prediction Ability of S-RBF, LR-RBF and CR-RBF Neural Nets

Results so far indicate that CR-RBF neurons have the ability to produce stable orbits of period 1 (and multiples thereof) as well as chaotic orbits. By contrast, the LR-RBF and S-RBF neurons are limited to period 1 stable orbits. The wider range of regimes which the CR-RBF neurons can achieve suggests that they may be better suited for predicting the output of some complex NDS than the other two types. This was seen to be the case in an example where a NN with CR-RBF neurons produced a better prediction of the Lorenz attractor [40] (a chaotic NDS) than NN with LR-RBF or S-RBF neurons [19].

Pitfalls and problems unique to the training of CR-RBF neurons are discussed in section A.4.1. The superior prediction ability of the CR-RBF is illustrated with a simple example in section A.4.2.

A.4.1 Problems in the Training of CR-RBF Neurons

When NN are trained with the backpropagation equations described in section A.2, the neuron parameter vectors σ and μ migrate towards values which are dictated exclusively by the 'desired' response and the gradient it produces for each neuron, during each training epoch.

For LR-RBF neurons, these parameter changes cannot make the individual neurons unstable since they are inherently stable over the entire parameters space. For the S-RBF neurons, the fact that they are static (i.e., no feedback) also ensures that they are stable over the entire parameters space. However, for CR-RBF neurons, this migration of σ and μ parameters may result in the CR-RBF transiting in and out of stable and/or chaotic orbits (refer again to Figures A.3.1-4, A.3.1-5). Some of these instantaneous stability characteristics may or may not be compatible with the stability characteristics of the NDS

being taught to the net. As a result, learning may be hindered (and in some cases improved) by these changes in stability characteristics of CR-RBF neurons.

Furthermore, the value of the input vector $X(n)$ may not be constant throughout a training epoch, so that K_X changes with time. This may cause the bifurcation map of the CR-RBF to change on an iteration-by-iteration basis. The progression of the values for σ and μ for each CR-RBF throughout the training is therefore a complex function of the training data, initial CR-RBF parameter values, learning rates and the effect of the instantaneous CR-RBF trajectories on the error gradient.

However, a complete understanding of these interactions is not always essential to achieve the successful training of a CR-RBF neural net. In some cases, the user can select initial values of μ and σ to match the periodicity properties of the NDS being taught. For example, if the NDS being taught has a stable attractor of period 1, it makes sense to set all hidden CR-RBF neurons with parameters μ and σ which produce a single, stable attractor. By using small learning rates and avoiding excessively long training time, the probability that these parameters will migrate in a region of instability is significantly reduced.

A.4.2 Training Example with S-RBF, LR-RBF and CR-RBF NN

Consider the NDS whose trajectories are illustrated in Figure A.4.2-1. This NDS has the property that all initial points on the boundary of rectangles centered on the attractor (at (0,0)) converge to that attractor in the same number of iterations N_{jk}. For instance, all initial points on the three rectangles shown with dashed lines in Figure A.4.2-1 converge to the attractor in 10, 20 and 25 iterations respectively. In that way, points fed to the NDS at time $n = 0$ can be classified as members (or non-members) of a class (or region) based on the number of iterations they require to reach the attractor. Figure A.4.2-2 illustrates how variations in the convergence rate are used increase N_{jk} for initial points outside the square boundary at $X,Y = \pm 0.5$.

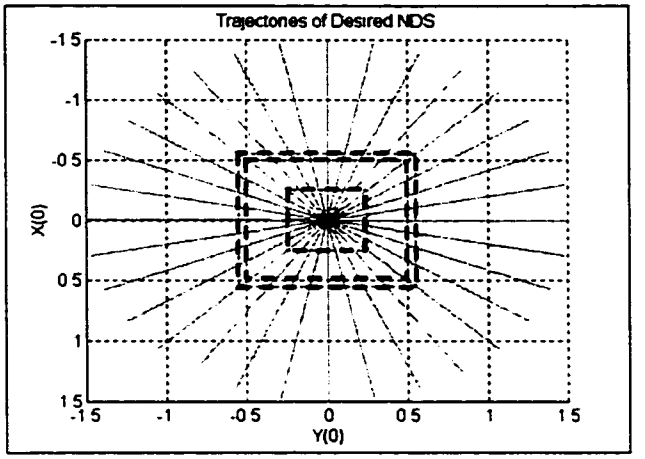


Figure A.4.2-1 : Training Trajectories

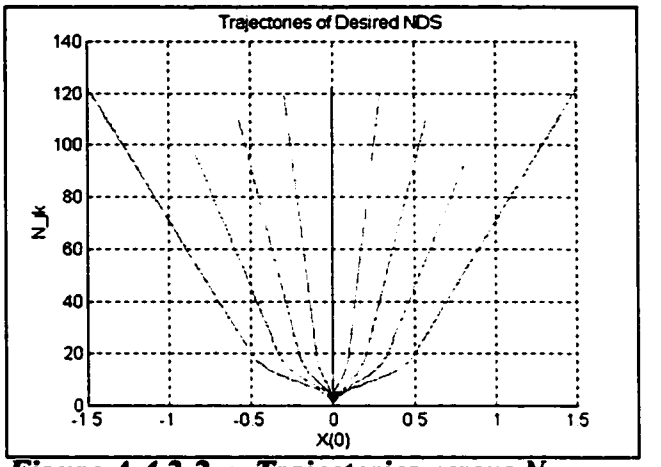


Figure A.4.2-2 : Trajectories versus N_{jk}

Four NN with the same architecture and connections between layers were trained to learn the trajectories of this NDS. One net had S-RBF neurons (i.e., NET-1), one had LR-RBF (i.e., NET-2) and the other two had CR-RBF neurons (i.e., NET-3 and 4). The only difference between the two CR-RBF NN was that one had initial values of σ and μ which were selected to produce period 2 orbits and/or chaotic orbits (i.e., NET-3) while the other one had values designed to produce period 1 stable orbits (i.e., NET-4). All the nets had two inputs and two outputs, with the latter being generated from linear, weighted sums of several hidden neurons.

Parameters μ and σ of the nets were trained using simple backpropagation in batch mode with a momentum term (i.e., equations E_A.2.1-8 and E_A.2.1-9). Training was performed until 10000 epochs were reached or until the average epoch mean square error E_{AV} ceased to decrease by 10% or more over 500 epochs. The 32 training trajectories shown in Figure A.4.2-1 add up to 3428 training pairs, where *[x(n),y(n)]* are the input at iteration *n* and *[x(n+1),y(n+1)]* are the desired output against which the net output is compared.

All of these nets completed training in roughly 7000-10000 epochs and achieved a final E_{AV} in the range of 4.25e-5 to 9.93e-5. The trajectories achieved by NET 1 to 4 after training are shown in Figures A.4.2-3 to A.4.2-6 respectively. It can be seen that NET-2 (LR-RBF) has a better overall mapping than NET-1 (S-RBF) and NET-3 (CR-RBF).

However, NET-4 (CR-RBF) has the best overall mapping and, in particular, is seen to be vastly superior to NET-3.

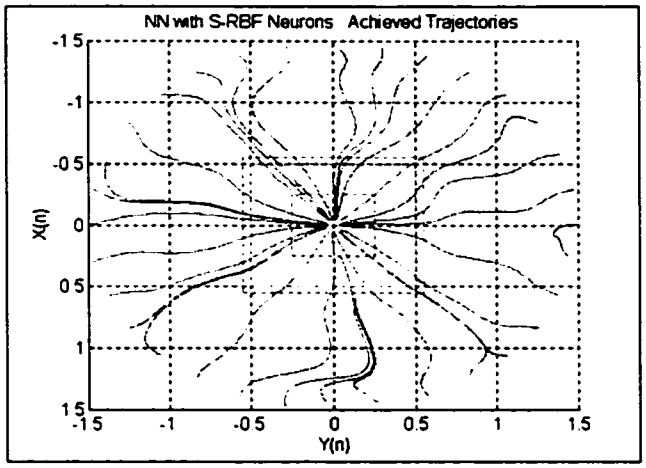


Figure A.4.2-3 : NET-1 (S-RBF) Results

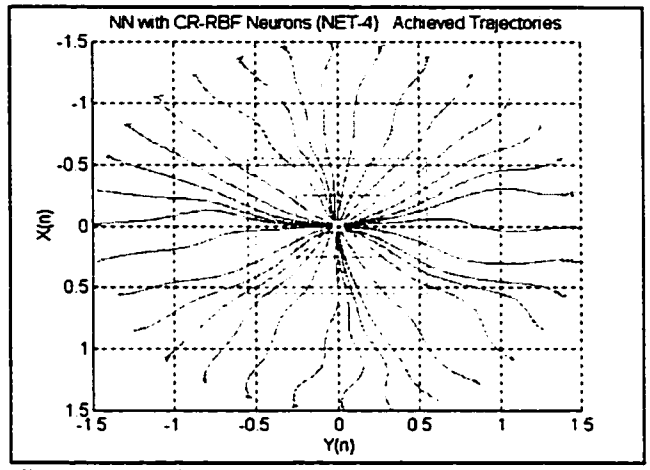


Figure A.4.2-4 : NET-2 (LR-RBF) Results

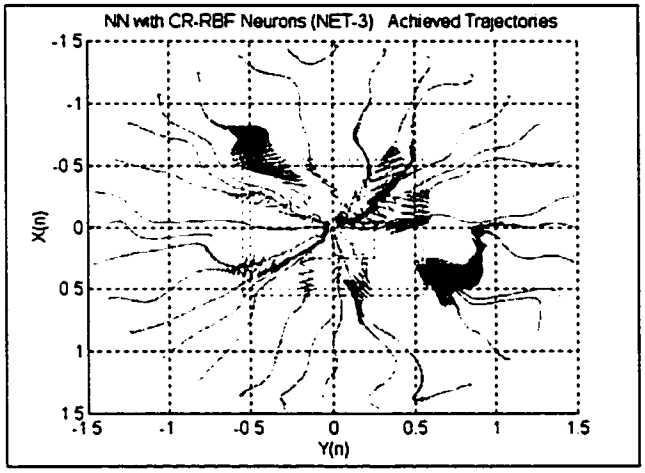


Figure A.4.2-5 : NET-3 (CR-RBF) Results

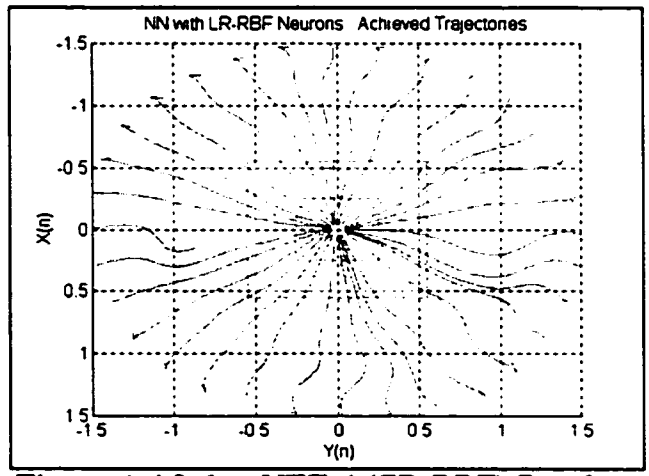


Figure A.4.2-6 : NET-4 (CR-RBF) Results

For NET-3, the initial values of σ and μ resulted in periodic oscillations which were still present at the end of training and which clearly hindered the net's ability to learn the desired NDS. However, the NET-4 initial parameters σ and μ were well within the CR-RBF region with stable orbits of period 1 and remained in that region throughout the training. For the desired NDS illustrated in Figures A.4.2-1 and A.4.2-2, all trajectories are asymptotically stable with period 1 orbits. Clearly, a NN with CR-RBF neurons having initial parameters σ and μ resulting in stable period 1 orbits was the most suitable to learn such a NDS. By contrast, the net with initial values of σ and μ resulting in

unstable orbits never learned the required trajectories properly. Furthermore, a large number of CR-RBF neurons still had unstable orbits by the end of the training, which caused the learned trajectories to exhibit instabilities (Figure A.4.2-5). This example illustrates the importance of selecting initial parameters carefully when using CR-RBF neurons.

A.5 Conclusions and Summary

In this appendix, the backpropagation training equations for linear recurrent and center recurrent RBF neurons were developed and compared to those of simple RBF neurons. It was shown that the presence of local feedback at the RBF neuron introduces memory in the training equations. This feedback term effectively controls the sensitivity of the neuron at time n to an independent input vector $X(n)$, based on the neuron's output magnitude at time $n-1$ (i.e., $y(n-1)$). For a LR-RBF neuron, the sensitivity is maximum when $y(n-1)=0$ and decreases as $y(n-1)$ increases. The CR-RBF neuron was shown to be a generalization of the LR-RBF, with the ability to tune the sensitivity at time n as a function of the Euclidean distance between $y(n-1)$ and μ_0 (with scaling provided by σ_0^2).

The stability characteristics of LR-RBF and CR-RBF neurons were examined. It was shown that the LR-RBF has stable attractors of period one over its entire space of parameters. By contrast, the CR-RBF neuron was shown to have more complex stability characteristics, with the ability to assume stable orbits of period 1 (and multiples) as well as chaotic orbits.

Training examples discussed in this paper and in reference [19] suggest that NN with CR-RBF neurons may have a better learning ability for certain types of NDS than those with LR-RBF and S-RBF neurons. This additional ability may be related to the wider range of stability characteristics which CR-RBF neurons can exhibit relative to LR-RBF or S-RBF.

However, the results described in this appendix indicated that the successful training of CR-RBF neurons can be highly dependent on the initial parameter values selected. Results suggest that the user should select these to ensure that the initial stability characteristics of the CR-RBF are compatible with those of the NDS being taught.

List of References

[1] Yuri V. Andreyev, Y. L. Belsky, S. Dmitriev, D. A. Kuminov, "Information Processing Using Dynamical Chaos : Neural Networks Implementation", IEEE Transactions on Neural Networks, Vol. 7, No. 2, March 1996, pp. 290-298.

[2] John D'Azzo, Constantine H. Houpis, "Linear Control System Analysis and Design", second edition, McGraw-Hill Book Company, 1981, ISBN-0-07-016183-6.

[3] Michael Barnsley, "Fractals Everywhere", Academic Press Inc., 1988.

[4] Christopher Bishop, "Neural Networks for Pattern Recognition", Clarendon Press, 1995, ISBN 0-19-853864-2.

[5] D.S. Borrett, T.H. Yeap, H.C. Kwan, "Neural Networks and Parkinson's Disease", Le Journal Canadien des Sciences Neurologiques, Vol. 20, pp. 107-113, Feb. 1993.

[6] Martin Casdagli, "Nonlinear Prediction of Chaotic Time Series", Physica D, Vol. 35, pp. 335-356, 1989.

[7] M. Ceccarelli, J.T. Hounsou, "RBF Networks vs. Multilayer Perceptrons for Sequence Recognition", from "Neural Networks Theory, Technology and Applications", P.K. Simpson ed., IEEE Technology Update Series, 1996, ISBN-078032-564-8.

[8] Guanrong Chen & Xiaoning Dong, "From Chaos to Order", World Scientific Publishing, 1998, ISBN 9810-225-695.

[9] Luowan Chen, K. Aihara, "Chaotic Simulated Annealing by a Neural Network Model with Transient Chaos", Pergamon Press, 1995, 0893-6080/95.

[10] S. Chen, S.A. Billings, "Modeling and Analysis of Non-Linear Time Series", International Journal of Control, 1989, Vol. 50, No. 6, pp. 2151-2171.

[11] S. Chen, S.A. Billings, C.F.N. Cowan, P.M. Grant, " Practical Identification of NARMAX models using radial basis functions", International Journal of Control, 1990, Vol. 52, No. 6, pp. 1327-1350.

[12] E.S. Chng, B. Mulgrew, "Gradient Radial Basis Function Networks for Nonlinear and Nonstationary Time Series Prediction", IEEE Transactions on Neural Networks, Vol-7, No. 1, Jan 1996, pp. 190-194.

[13] M.Y. Choi, B.A. Huberman, "Dynamic Behavior of Nonlinear Networks", Physical Review A, Vol. 28, No. 2, August 1983.

[14] P.S. Churchland, T.J. Seznowski, "The Computational Brain", MIT Press, 1991, ISBN 0-262-3188-4.

[15] Peter A. Cook, "Nonlinear Dynamical Systems", Prentice-Hall International, 1986, ISBN 0-13-623216-7.

[16] Francis Crick, "The Recent Excitement About Neural Networks", Nature, Vol. 337, No. 12, January 1989.

[17] A.S. Dmitriev, A.I. Panas, S.O. Starkov, "Storing and Recognizing Information based on Stable Cycles of One-Dimensional Maps", Physics Letters A, Vol. 155, No. 8-9, 27 May 1991, pp. 494-499.

[18] E. Domany et al, "Models of Neural Networks III", Springer-Verlag, 1995, ISBN 0-387-94368-4.

[19] G. Ferland, T.H. Yeap, "Prediction of Nonlinear Dynamical System Output with multi-layer Perceptron and Radial Basis Function Neural Networks", Proceedings of the International Joint Conference on Neural Networks 1999 (IJCNN-99), ISBN 0-7803-5532-6.

[20] Guy Ferland, Tet Yeap, "The Race to the Attractor Model for Classifying Objects", IEEE Instrumentation & Measurements Magazine, Vol. 3, No. 4, Dec 2000.

[21] Guy Ferland, Tet Yeap, "A New Paradigm for the Object Recognition Process : 'The Race to the Attractor Neural Network Model'", presented at the International Workshop on Virtual and Intelligent Measurement Systems, Annapolis, MD, 29-30 April 2000. See also http://www.ims.unico.it/conferences/conferenze/2000vims.

[22] Guy Ferland, Tet Yeap, "A New Paradigm for Classification Tasks : the 'Race to the Attractor' Neural Network Model", Proceedings of the International Joint Conference on Neural Networks 2001 (IJCNN-01), ISBN 0-7803-7046-5.

[23] Walter J. Freeman "The Physiology of Perception", Scientific American, February 1991.

[24] Michael R. Guevara, Leon Glass, M.C. Mackey, A. Shrier, "Chaos in Neurobiology", IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-13, No. 5, Sept-Oct 1993, pp. 790-798.

[25] Albert R. Haig, Evian Gordon, J.J. Wright, R.A. Meares, H. Bahramali, "Synchronous Cortical Gamma-Band Activity in Task-Relevant Cognition", Computational Science, Vol. 11, No 4, March 2000, pp. 669-675 (or see http://www.mhri.edu.au/bdl/papers).

[26] Simon Haykin, 'Neural Networks: A Comprehensive Foundation', second edition. MacMillan, 1999, ISBN 0-13-273350-1.

[27] Robert Hecht-Nielsen, "Neurocomputing : picking the human brain", IEEE Spectrum, March 1988.

[28] N.S. Jayant, P. Noll, "Digital Coding of Waveforms", Prentice-Hall, 1984, ISBN 0-13-211913-7-01.

[29] Erwin Kreyszig, "Advanced Engineering Mathematics", 6-th edition, John-Wiley & Sons, 1988, ISBN 0-471-85824-2.

[30] Jeffrey R.M. Kunz, Asher J. Finkel, "Family Medical Guide", Random House New York, 1987, ISBN 0-394-55582-1.

[31] Hon C. Kwan, Tet H. Yeap, D. Barrett, B.C. Jiang, "Network Relaxation as Biological Computation", Behavioral and Brain sciences, Vol. 14 No. 2, pp. 354-356.

[32] Hon C. Kwan, Tet H. Yeap, Bai C. Jiang, Donald Borrett, " Neural Network and Control of Simple Limb Movements", Behavioral and Brain sciences, Vol. 15, No. 4, 1992.

[33] Timothy Masters, "Neural, Novel & Hybrid Algorithms for Time Series Prediction", John Wiley & Sons, 1995, ISBN 0-471-13041-9.

[34] T. Miyoshi, H. Ichihashi, S. Okamoto, T. Hayakawa, "Learning Chaotic Dynamics in Recurrent RBF Network", IEEE technical Series, 0-7803-2768-3/95.

[35] D.J. Munro, O.K. Ersoy, M.R. Bell, J.S. Sadowsky, "Neural Network Learning of Low-Probability Events", IEEE Transactions on Aerospace and Electronic Systems, Vol. 32, No. 3, July 1996.

[36] Kumpati Narendra, Kannan Parthasarathy, "Identification and Control of Dynamical Systems Using Neural Networks", IEEE Transactions on Neural Networks, Vol. 1 No. 1, March 1990.

[37] Kumpati S. Narendra, Kannan Parthasarathy, "Gradient Methods for the Optimization of Dynamical Systems Containing Neural Networks", IEEE Transactions on Neural Networks, Vol. 2, No. 2, March 1991, pp. 252-262.

[38] A.V., Oppenheim, R.W. Schafer, "Discrete-Time Signal Processing", Prentice-Hall, 1989, ISBN 0-13-216292-X.

[39] Edward Ott, Celso Grebogi, J.A. Yorke, "Controlling Chaos", Physical Review Letters, Vol. 64, No. 11, pp. 1196-1199, March 1990.

[40] Edward Ott, "Chaos in Dynamical Systems", Cambridge University Press, 1993, ISBN 0-521-43799-7.

[41] Athanasios Papoulis, "Probability, Random Variables and Stochastic Processes" 2nd ed., McGraw-Hill, 1984, ISBN 0-07-048468-6.

[42] J. Park, I.W. Sandberg, "Nonlinear Approximations Using Elliptic Basis Function Networks", from "Neural Networks Theory, Technology and Applications", P.K. Simpson ed., IEEE Technology Update Series, 1996, ISBN-078032-564-8

[43] Barak A. Pearlmutter, "Gradient Calculations for Dynamic Recurrent Neural Networks : A Survey", IEEE Transactions on Neural Networks, Vol. 6, No. 5, September 1995.

[44] T. Poggio, F. Girosi, "Networks for Approximation and Learning", Proceedings of the IEEE, Vol. 78, No. 9, Sept 1990, pp. 1481-1496.

[45] Murray H. Protter, Charles B. Morrey, "Modern Mathematical Analysis", Addison-Wesley Publishing Company, Don Mills Ontario, 1964, ISBN 0-201-05995-9.

[46] Murray H. Protter, Charles B. Morrey, "College Calculus with Analytic Geometry", 3rd edition, Addison-Wesley Publishing Company, Don Mills Ontario, 1977, ISBN 0-201-06030-2.

[47] M. Rosenblum, L.S. Davis, "An Improved Basis Function Network for Visual Autonomous Road Following", IEEE Transactions on Neural Networks, Vol. 7, No. 5, Sept-1996, pp. 1111-1120.

[48] R.M. Sanner, J.J.E Slotine, "Gaussian Networks for Direct Adaptive Control", IEEE Transactions on Neural Networks, Vol-3, No. 6, Nov 1992, pp. 837-863.

[49] Haber Schaim, Cross, Dodge, Walter : "Physics", fourth edition, D.C. Heath and Company, 1976, ISBN 0-669-97451-X.

[50] Troy Shinbrot, Edward Ott, Celso Grebogi, J.A. Yorke, "Using Chaos to Direct Trajectories to Targets", Physical Review Letters, Vol. 65, No. 26, December 1990.

[51] Stanley M. Shinners, "Modern Control System Theory and Design", John Wiley & Sons Interscience, 1992, ISBN 0-471-55008-6.

[52] Christine A. Skarda, Walter J. Freemam, "How brains make chaos in order to make sense of the world", Behavioral and Brain Sciences, 1987, 10-2, pp. 162-195.

[53] Murray R. Spiegel, "Mathematical Handbook", Schaum's Outline Series, McGraw-Hill Book Company, 1968.

[54] A. C. Tsoi, A.D. Back, " Locally Recurrent Globally Feedforward Networks : A Critical Review of Architectures", IEEE Transactions on Neural Networks, Vol. 5, No. 2, March 1994, pp. 229-239.

[55] Claudio Turchetti, Massimo Conti, Paolo Crippa, Simone Orcioni, "On the Approximation of Stochastic Processes by Approximate Identity Neural Networks", IEEE Transactions on Neural Networks, Vol. 9, No. 6, November 1998.

[56] Ferdinand Verhulst, "Nonlinear Differential Equations and Dynamical Systems", second edition, Springer-Verlag, 1991, ISBN 3-540-60394-2.

[57] Lipo Wang, John Ross, "Chaos, Multiplicity, Crisis,, and Synchronicity in Higher-Order Neural Networks", Physical Review A, Vol. 44, No. 4, August 1991.

[58] Lipo Wang, "Oscillatory and Chaotic Dynamics in Neural Networks Under Varying Operating Conditions", IEEE Transactions on Neural Networks, Vol. 7, No. 6, November 1996.

[59] Wayne L. Winston, "Introduction to Mathematical Programming : Applications and Algorithms", 2nd Edition, Duxbury Press, Wadsworth Inc, 1995, ISBN 0-534-23046-6.

[60] J.J. Wright, D.T.J. Liley, "Dynamics of the Brain at Global and Microscopic Scales : Neural Networks and the EEG", Behavioral and Brain Sciences, 1996, No 19, pp. 285-320 (or see http://www.mhri.edu.au/bdl/papers).

[61] J.J. Wright, et al, "Toward an Integrated Continuum Model of Cerebral Dynamics : The Cerebral Rhythms, Synchronous Oscillation and Cortical Stability", on the web at http://www.mhri.edu.au/bdl/papers.

[62] X. Ye, N.K. Loh, "Dynamic System Identification Using Recurrent Radial Basis Function Network", in "Neural Networks Theory, Technology and Applications", Patrick Simpson editor, IEEE technology update series, 1996, ISBN 0-7803-2564-8.

[63] T.H. Yeap, N.U. Ahmed, "Feedback Control of Chaotic Systems", Dynamica and Control, Vol. 4, pp.- 97-114, April 1994.