

A COMPARATIVE STUDY OF SELECTED NEURAL NETWORK MODELS

by

Adi Indrayanto

A Thesis

presented to the University of Manitoba

in partial fulfillment of the

requirements of the degree of

Master of Science

in the

Department of Electrical and Computer Engineering

Winnipeg, Manitoba

March, 1992



National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file *Votre référence*

Our file *Notre référence*

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-77948-9

Canada

A COMPARATIVE STUDY OF SELECTED NEURAL NETWORK MODELS

BY

ADI INDRAYANTO

A Thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

© 1992

Permission has been granted to the LIBRARY OF THE UNIVERSITY OF MANITOBA to lend or sell copies of this thesis, to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film, and UNIVERSITY MICROFILMS to publish an abstract of this thesis.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's permission.

ABSTRACT

Recent progress in artificial neural network models has led to a need for developing unified benchmarks and a unified methodology of benchmarking in order to facilitate better understanding and applicability of these models for particular applications. This thesis places emphasis on the study of the characteristic features of four selected models, namely, the Bidirectional Associative Memory (BAM), Backpropagation (BP), Counterpropagation (CPN), and Adaptive Resonance Theory 1 (ART-1) neural network models. The study includes the identification and comparison of the characteristic features of the selected models, the development of a neural network software simulator, and some experimental study. The software is designed using the object-oriented design methodology, and is implemented on the Macintosh computer. Since there are two distinct classes of neural network models, the experimental study must employ two benchmark problems, namely, the associative memory and the pattern classification problems. The experiment confirms some of the characteristic features of the selected models and reveals other characteristic features that have not been previously identified. For example, the results of the associative memory experiment show that the selection of the stored patterns in the BAM network seems inconsistent with a selection based on the closest Hamming distance. Results confirm that the three-layer BP network produces fewer spurious patterns than two-layer BP network, and CPN performs similarly to a look-up table whose entries are separated by a radius of association. For the given binary classification problem, results show that the two-layer and three-layer BP with a number of hidden neurons above 6 classify 90% of the total test samples correctly, while the CPN network shows a 95% classification. The experimental results of ART-1 confirm that the learning of the network is stable only at the subset pattern, and this characteristic becomes significant for higher vigilance values. The thesis presents results from numerous experiments.

ACKNOWLEDGEMENTS

The completion of this thesis is possibly only with the support, encouragement, and active involvement of many people. First and foremost, I would like to express my sincere thanks to Dr. W. Kinsner for his patience and helpful support throughout the thesis process, and also for the thesis topic. I would also like to thank my fellow colleagues, Ken Ferens, Armein Langi, Budi Rahardjo, Chris Love, Larry Wall, and Warren Grieder for their careful reading of the manuscript, discussion, support, and friendship. A special thanks goes to the Inter University Centre (IUC) on Microelectronics, ITB, Indonesia, and World University Service of Canada (WUSC) for their financial support of this work. Last but not least, I would like to thank my father, Dr. M. Mardjono, my mother, S. Mardjono, and my best friend, Eri Kamardi for their support and encouragement through the many hours of work on this thesis.

TABLE OF CONTENTS

	Page
ABSTRACT	ii
ACKNOWLEDGEMENTS	iii
LIST OF FIGURES	vii
LIST OF TABLES	ix
LIST OF ABBREVIATIONS AND ACRONYMS	xi
I INTRODUCTION	1
Objective	1
Motivation	1
Organization of Thesis	4
II BACKGROUND ON ARTIFICIAL NEURAL NETWORKS	5
The Biological Model	5
Artificial Neural Networks (ANNs)	7
The Early Foundations	8
McCulloch-Pitts Neurons	9
Hebb's Learning Law	10
The Perceptron	10
New Computing Paradigm	12
The Networks	12
Feedforward Network	13
Feedback Network	13
Learning Methods	14
Supervised Learning	14
Reinforcement Learning	14
Unsupervised Learning	15
Summary	15
III ESSENTIAL FEATURES OF SELECTED ANN MODELS	16
Bidirectional Associative Memory (BAM)	16
Network Topology	17
BAM Weight Modification Procedure	19
Problems in BAM Model	22
Backpropagation (BP)	24
Network Topology	25

BP Weight Modification Procedure	29
Problems in BP Model	33
Counterpropagation (CPN)	36
Network Topology	37
CPN Weight Modification Procedure	39
Problems in CPN Model	42
Adaptive Resonance Theory 1 (ART-1)	42
Network Topology	43
ART-1 Weight Modification Procedure	46
Problems in ART-1 Model	49
Summary	49
IV SOFTWARE IMPLEMENTATION	51
Specifications	51
Design Methodology	52
Architecture	54
User Interface	54
Neural Network Model	61
Tools	74
Verification	76
Software Verification	76
Neural Network Model Verification	77
Summary	77
V ASSOCIATIVE MEMORY EXPERIMENT	79
Pattern Sets	80
Measurement Technique	82
Autoassociative Experimentation	84
Two-association	85
Network Configurations	85
Storing	86
Recalling	89
Three-association	100
Network Configurations	102
Storing	102
Recalling	103
Four-association	109
Network Configurations	109
Storing	110
Recalling	112
Heteroassociative Experimentation	117
Two-association	117
Network Configurations	117
Storing	119

Recalling	120
Three-association	125
Network Configurations	125
Storing	126
Recalling	127
Discussion	129
Summary	137
VI CLASSIFICATION EXPERIMENT	139
Pattern Sets	140
Measurement Technique	141
BP Experimentation	143
Network Configurations	143
Training	144
Testing	147
CPN Experimentation	152
Network Configurations	152
Training	152
Testing	153
ART-1 Experimentation	155
Network Configurations	155
Learning	156
Discussion	173
Summary	178
VII CONCLUSIONS AND RECOMMENDATIONS	179
REFERENCES	186
APPENDIX A: TRAINING PATTERNS	A1
Patterns Used in Associative Memory Experiment	A1
Patterns Used in the Classification Experiment	A12
APPENDIX B: EXPERIMENTAL RESULTS OF ART-1	B1
Predicted Categories for vigilance 0.5	B1
Predicted Categories for vigilance 0.8	B2
Predicted Categories for vigilance 1.0	B6
APPENDIX C: NEURAL NETWORK SIMULATOR'S WINDOWS	C1

LIST OF FIGURES

Figure	Page
2.1 A typical neuron in the human nervous system	6
2.2 A model of an artificial neuron	8
3.1 Topology of a BAM network	18
3.2 Topology of a single hidden layer BP network	26
3.3 Effect of a bias term θ to the sigmoid function	27
3.4 Steepest descent method on the error surface with respect to w_1 and w_2	36
3.5 Topology of a Forward-only CPN	38
3.6 Topology of an ART-1 network	45
4.1 Block diagrams of the main modules	53
4.2 Software class hierarchy	56
4.3 Interactions among objects in the main modules	58
4.4 The <i>main event loop</i> of the program	60
4.5 The <i>ProcessEvent</i> method of the <i>CSwitchboard</i> class	61
4.6 Process of creating and initializing a <i>CNeuralNetsModel</i> object	63
4.7 The weights initialization and the start learning events of the <i>CNeuralNetsModel</i> object	65
4.8 Interaction of many objects during a learning session	66
4.9 A multitask event during an idle time	68
4.10 Objects of subclasses of the <i>CChore</i> class	69
4.11 Stop, resume, and finish learning events of the <i>CNeuralNetsModel</i> object	70
4.12 Interaction of many objects during the testing process	71
4.13 Process of creating the <i>CProbe</i> object and establishing the communication path between the <i>CProbe</i> object and the <i>CDataInterface</i> object	72
4.14 Process of creating the <i>CDisplay</i> object and establishing the communication path between the <i>CDisplay</i> object and the <i>CDataInterface</i> object	73
4.15 Process of creating and initializing the <i>CPatternEditor</i> object	75
5.1 7x5 binary pixel projection of E & G and I & J	81
5.2 Projections of A, B, A_{noisy} , $A \cap B$, $A_{\mu-B}$, and A^c	84
5.3 Total error versus number of epochs of a two-layer BP on S_{AB} set	89
6.1 Error rate versus number of hidden neurons	148
6.2 Error rate versus stopping error criterion	149
6.3a Error rate versus number of bits flipped (noise)	149
6.3b Error rate versus number of bits flipped (noise)	150
6.3c Error rate versus number of bits flipped (noise)	150
6.3d Error rate versus number of bits flipped (noise)	151
6.3e Error rate versus number of bits flipped (noise)	151

6.4	Error rate versus number of training cycles (epochs)	153
6.5a	Error rate versus number of bits flipped (noise), for the network trained using learning rate 0.1 with random order presentation	154
6.5b	Error rate versus number of bits flipped (noise), for the network trained using learning rate 0.1 with random order presentation	154

LIST OF TABLES

Table	Page
5.1 Training sets for the associative memory experimentation	82
5.2 Two-association training sets for the autoassociative experiment	86
5.3a Learning rates, momentum terms, total errors and training cycles of the two-layer BPs	88
5.3b Learning rates, momentum terms, total errors and training cycles of the three-layer BPs	88
5.4a Test results of training set S_{AB}	92
5.4b Test results of training set S_{AH}	95
5.4c Test results of training set S_{AI}	96
5.4d Test results of training set S_{BG}	97
5.4e Test results of training set S_{EG}	98
5.4f Test results of training set S_{HI}	99
5.4g Test results of training set S_{IJ}	100
5.5 Three-association training sets for the autoassociative experiment	101
5.6a Learning rates, momentum terms, total errors and training cycles of two-layer BP	102
5.6b Learning rates, momentum terms, total errors and training cycles of three-layer BP	103
5.7 Verification results of the three-association experiment	103
5.8a Test results of training set S_{ABI}	105
5.8b Test results of training set S_{AFH}	107
5.8c Test results of training set S_{EFG}	107
5.8d Test results of training set S_{BHI}	108
5.9 Four-association training sets	110
5.10a Learning rates, momentum terms, total errors and training cycles of two-layer BP	111
5.10b Learning rates, momentum terms, total errors and training cycles of three-layer BP with two hidden neurons	111
5.10c Learning rates, momentum terms, total errors and training cycles of three-layer BP with three hidden neurons	111
5.11 The verification test results of BAM	112
5.12a Test results of training set S_{BCHI}	114
5.12b Test results of training set S_{BHIL}	116
5.13 Two-association training sets for the heteroassociative experiment	118
5.14a Learning rates, momentum terms, total errors and training cycles of two-layer BP	119
5.14b Learning rates, momentum terms, total errors and training cycles of three-layer BP	120
5.15 Verification results of BAM in the two-association experiment	121
5.16a Test results of training set S_{AB-IJ}	122

5.16b	Test results of training set S_{AH-EF}	122
5.16c	Test results of training set S_{EG-AI}	123
5.16d	Test results of training set S_{EI-GJ}	123
5.16e	Test results of training set S_{HI-EG}	124
5.16f	Test results of training set S_{IJ-EG}	124
5.17	Three-association training sets for the heteroassociative experiment	125
5.18a	Learning rates, momentum terms, total errors and training cycles of two-layer BP	126
5.18b	Learning rates, momentum terms, total errors and training cycles of three-layer BP	126
5.19a	Test results of training set $S_{BHI-ACJ}$	127
5.19b	Test results of training set $S_{BHI-AJC}$	128
5.20	The enhanced S_{AFH} training set	132
5.21	Verification test results of BAM on the S_{AFH} and S_{AFH}^* training sets	132
5.22	The enhanced S_{EFG} training set	133
5.23	Verification test results of BAM on the S_{EFG} and S_{EFG}^* training sets	133
5.24	The S_{IJ-EG} and S_{IJx-EG} training sets	134
5.25	Verification test result of BAM on the S_{IJ-EG} and S_{IJx-EG} training sets	135
6.1a	Learning rate, momentum term, total errors and training cycles of the two-layer BP	145
6.1b	Learning rate, momentum term, total errors and training cycles of the three-layer BP with 4 hidden neurons	146
6.1c	Learning rate, momentum term, total errors and training cycles of the three-layer BP with 60 hidden neurons	146
6.2a	The categories produced for vigilance 0.5	157
6.2b	The categories produced for vigilance 0.6	158
6.2c	The categories produced for vigilance 0.7	159
6.2d	The categories produced for vigilance 0.8	160
6.2e	The categories produced for vigilance 0.9	161
6.2f	The categories produced for vigilance 1.0	162
6.3a	The true versus predicted categories for vigilance 0.5	164
6.3b	The true versus predicted categories for vigilance 0.8	165
6.3c	The true versus predicted categories for vigilance 1.0	168

LIST OF ABBREVIATIONS AND ACRONYMS

ANN	Artificial Neural Network
ART	Adaptive Resonance Theory
ARTMAP	Predictive Adaptive Resonance Theory
BAM	Bidirectional Associative Memory
BP	Backpropagation
CABAM	Competitive Adaptive Bidirectional Associative Memory
CPN	Counterpropagation
E_R	Error Rate
FSCL	Frequency Sensitive Competitive Learning
GLN	Graded Learning Network
GUI	Graphical User Interface
IBAM	Intraconnected Bidirectional Associative Memory
LMS	Least Mean Square
LTM	Long Term Memory
PDP	Parallel Distributed Processing
RABAM	Random Adaptive Bidirectional Associative Memory
STM	Short Term Memory
TAM	Temporal Associative Memory
TCL	THINK Class Library
UON	Uniform Object Notation
XOR	Exclusive OR

CHAPTER I

INTRODUCTION

1.1 Objective

The objective of this thesis is to study the characteristic features of four representative artificial neural network models, namely, Bidirectional Associative Memory (BAM) [Kosk88], Backpropagation (BP) [RHWi86, McRu86], Counterpropagation (CPN) [Hech87], and Adaptive Resonance Theory 1 (ART-1) [CaGr88].

1.2 Motivation

An artificial neural network is a computational structure that is based on concepts derived from research into the nature of the brain [DARP88, MüRe90]. This new computing paradigm is becoming increasingly attractive not only in the study of intelligent machine behaviour, but also in solving a variety of practical problems [HuYK90]. Several studies have shown some advantages of neural networks to solve some practical problems as opposed to other approaches [DARP88, Souc89, WeKu91, and Kosk92].

The development of new theories in the past decade has led to a variety of artificial neural network models. Today, there are at least 26 distinct models of artificial neural networks, each of which has its own advantages as well as problems and limitations [MaHP90]. The unique characteristics of each distinct artificial neural network model lead to a question of how to select a proper model that matches a particular application. Such a selection is difficult because the models differ in their behaviour significantly. This thesis is an attempt to develop unified benchmarks through a comparative study of several

representative neural network models. The study is intended to improve our understanding of the capabilities of different neural network structures, and hence to provide a better insight into neural network behaviour.

Ideally, a comparative study should include all the existing neural network models. However, since the comparison is very time consuming, only a few models can be studied. Previous works on comparative studies of certain neural network models have been done and presented by this research group [Silv90, HuYK90, KiHu90, and KiIL90]. This thesis is an extension of the previous work by Kinsner, Indrayanto, and Langi [KiIL90], with emphasis on the characteristic features of the BAM, BP, CPN, and ART-1 neural network models. These models have been selected since they are representatives of distinct behaviours.

The study is done through (i) identifying the characteristic features of selected neural network models; (ii) comparing the characteristic features of the models; (iii) developing a software simulator of the models; (iv) doing experimental study on the models using the software.

A study of neural networks requires the availability of tools for simulating various neural network models. Although commercial neural network simulators are available, most of them come with only a few models and rarely allow the definition of arbitrary neural network parameters. What is even more detrimental, the software usually comes without a source code which could show exactly how the learning algorithms have been implemented in the selected models.

For practical applications, there is no need to have access into the details of the source code, as long as the software has a good user interface and some flexibility to change the learning parameters and the network architecture. However, in a comparative study, such an access is necessary to control all the implementation details. This is important, since to

study the characteristics of the models, one must make sure that the original learning algorithms are implemented properly (i.e., there are no modifications nor any improvements to the original learning algorithms). A study of the details usually reveals such alterations. Thus, it is preferable to implement each neural network model from the ground up, with all the known modifications. Furthermore, implementations of separate simulators for distinct neural network models may differ so much that a comparative study could be meaningless. Consequently, a unified framework for all the models of interest must be developed. Since the development of such a unified framework is a very involved process, an object-oriented approach yields good software that is expandable, maintainable and portable. These are the major reasons to include the development of a new neural network software simulator in the thesis. Without this new neural network simulator the comparative study could not have been possible.

The basic differences between the selected models may be studied through the topology and the learning algorithm of each model. However, this may not confirm or discover all their capabilities, and particularly, their abilities to solve specific problems. Therefore, an experimental study of the models is also included in the thesis together with some benchmark problems.

It is not easy to find an application that is suitable to all of the selected models, since each of the models has its own characteristics. For instance, BAM will suit an associative memory application but not a pattern classification. On the other hand, ART-1 will suit a pattern classification but not an associative memory application. Due to their specifications, separate experiments have been concluded. The first experiment, namely, the associative memory experiment includes only BAM, BP, and CPN models, while the second experiment, namely, the pattern classification experiment includes only BP, CPN, and ART-1 models. Another issue is the representation of the data samples to the networks.

BP and CPN can take both binary and analog data representations. However, BAM and ART-1 can only take the binary data representation. Therefore, a binary data representation is preferable. A set of 11 alphabetic characters represented by 7×5 binary pixels is selected as the main data samples in both experiments. This data set has been used in the previous work [HuYK90, KiHu90, and KiIL90].

1.2 Organization of Thesis

This thesis consist of seven chapters. Following an introduction to this thesis (Chapter I), Chapter II gives background information on artificial neural networks. This includes the discussion of the theoretical basis of artificial neural networks in general and their evolution. Chapter III discusses the essential features of selected neural network models. The discussion covers the network topologies, the learning procedures, and some limitations of the models, as well as a basis for the comparative study. Chapter IV describes the structure and the implementation details of the neural network software simulator. This includes the discussion of the system specifications, design methodology, the architecture of the system, and the verification technique of the system. Chapters V and VI present experimental results. Chapter V describes the associative memory experiment, and discusses the experimental results of BAM, BP, and CPN models. Chapter VI describes the pattern classification experiment, and discusses the experimental results of BP, CPN, and ART-1 models. Finally, Chapter VII gives conclusions and recommendations.

CHAPTER II

BACKGROUND ON ARTIFICIAL NEURAL NETWORKS

An artificial neural network is a computational structure that is based on concepts derived from research into the nature of the brain [DARP88, MüRe90]. That research has helped in the development of artificial neural networks. Therefore, it is natural to briefly review the biological neuron model in this chapter. The chapter also presents an overview of the developmental history and evolution of the artificial neural networks field in general.

2.1 The Biological Model

Neurons, or nerve cells, are the building blocks of the brain. In spite of the similarity in their biochemical apparatus with other cells, neurons have unique features, such as distinctive cell shapes, outer membranes capable of generating nerve impulses, and unique structures, called *synapses*, for transferring signals from one neuron to the next [Llin89]. Regardless of their unique forms, most neurons share certain structural features that make it possible to distinguish three regions of the cell, namely, the *soma* or the cell body, the *dendrites*, and the *axon* (see Fig. 2.1). The dendrites and axons extend from the cell body to other neurons via connection points, the *synapses*. The cell body receives incoming signals from other neurons through dendrites. In the state of inactivity, the interior of the neuron is negatively charged (about -70mV) against the surrounding neural liquid. Signals arriving from the synaptic connections result in a transient weakening, or *depolarization*, of the resting potential. The cell fires when the cumulative excitation of

these signals exceeds a threshold (i.e., when the total magnitude of the depolarization potential in the cell body exceeds the critical threshold, about 10mV). Then, the fired cell sends a signal (i.e., pulse trains ranging from about 1 to 100 pulses per second) down the axon to the other neurons. This is the basic mechanisms of how neurons communicate among themselves. Notice that the cumulative excitation of the incoming signals, to some extent, is determined by the types and the strengths of the synapses. Some synapses are excitatory in that they tend to promote firing, while others are inhibitory in that they are capable of canceling signals that would otherwise excite a neuron to fire. Moreover, the strengths of the synapses are not fixed once and for all. There is a mechanism of synaptic *plasticity* in the structure of the synapses, known as *Hebb's rule* [Hebb49, MüRe90], which is described in more detail in Section 2.3.2. A detailed description of the brain physiology can be found in [Lin89] and [Time90].

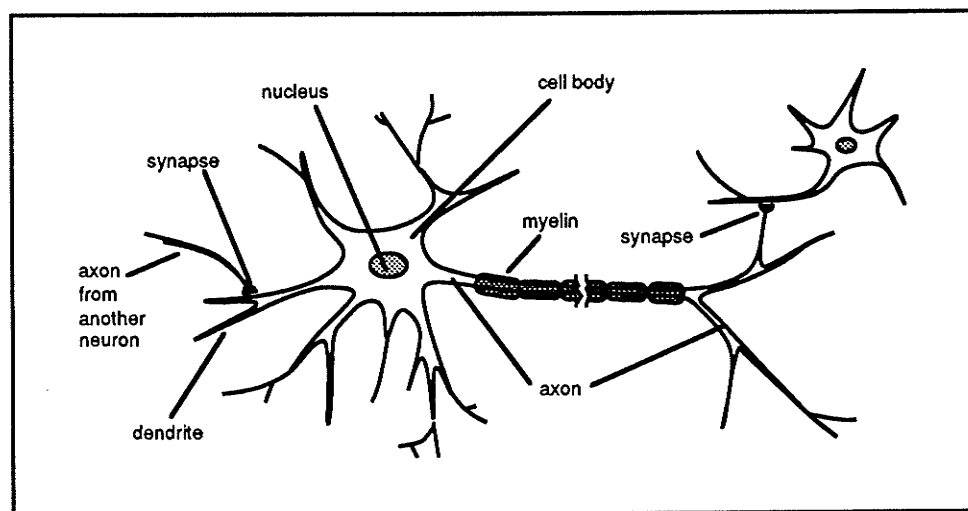


Fig. 2.1. A typical neuron in the human nervous system. After [Lin89]

2.2 Artificial Neural Networks (ANNs)

Neurons can be viewed as processing elements capable of at least summing operations; the axons and dendrites become the connections that establish the communication paths among neurons, and the synapses become the connection weights, which are changeable through some learning algorithms. Similar to the biological model, an artificial neuron can have any number of incoming connections as well as outgoing connections. While each incoming connection can receive any signal, the outgoing connections must transmit the same signals. In other words, a processing element has a single output connection that can branch or *fan out* into exact copies to form multiple output connections [Hech89].

In a more formal definition, a neural network model is defined as a *directed graph* [MüRe90, Hech89], a geometrical object consisting of a set of points (called *nodes*) along with a set of directed line segments (called *links*) between them, with the following properties (see Fig. 2.2) :

1. A state variable x_i is associated with each node i .
2. A real-valued connection strength or weight w_{ij} is associated with each link (ij) between two nodes i and j .
3. A real-valued bias θ_i is associated with each node i .
4. A transfer function $S_i[x_j, w_{ij}, \theta_i, (j \neq i)]$ is defined, for each node i , which determines its state as a function of the states of the nodes connected to it, the connection strengths or weights linking other nodes to it, and its bias. A general form of S_i usually is given as $S_i\left(\sum_j w_{ij} x_j - \theta_i\right)$, where $S(y)$ is a non-linear squashing function (e.g., sigmoid).

In the standard terminology, the nodes of the graph are called *neurons* or *processing elements*, the links are called *synapses* or *connections*, the states of the nodes are called the *activations* of neurons, and the biases are known as the *activation thresholds*.

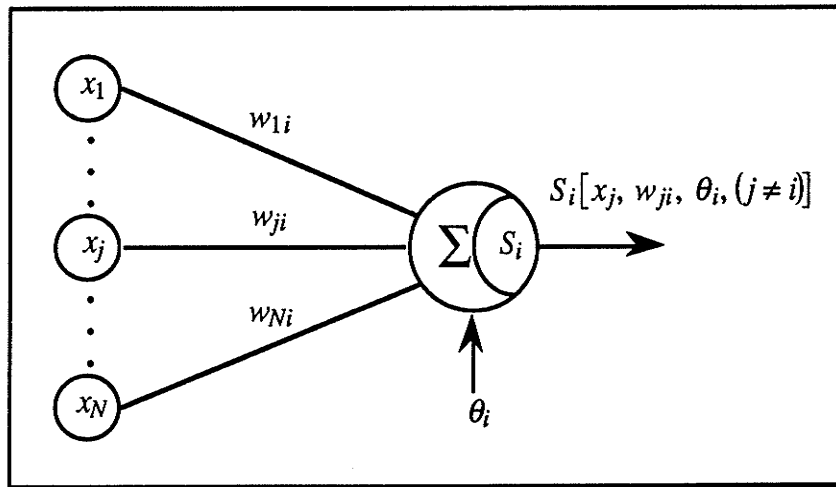


Fig. 2.2. A model of an artificial neuron.

The formal definition of the artificial neural networks, however, has only been introduced recently. In spite of this, the theories of the artificial neural networks are not new. They have been known for sometimes under different names.

2.3 The Early Foundations

The field of artificial neural networks appears to be a new discipline concerned with data processing system. However, the foundations have been established before the emergence of computers [MaHP90, Kurz90].

The field has interesting history since its appearance in the 1940s. The early progress, which culminated in the development of the first successful neurocomputer, Mark I

Perceptron, a two-layer *feedforward* neural network model built from the idea of Frank Rosenblatt [Hech89], was followed by a period of disinterest. This lack of enthusiasm was strengthened by the publication of the *Perceptrons* book of Marvin Minsky and Seymour Papert in 1969 [MiPa88], which exposed what appeared to be significant important limitations of the perceptron models of the time. During the period from 1967 to 1982, little progress in artificial neural network research was reported in the United States [Hech89]. Nevertheless, some researchers in this period, namely Bernard Widrow, Harry Klopf, James Anderson, Steven Grossberg, Paul Werbos, and others in Europe (Teuvo Kohonen) and Japan (Sun-ichi Amari and Kunihiko Fukushima), helped keep the field of artificial neural networks afloat by pursuing the research. By 1986, with the publication of the *Parallel Distributed Processing* (PDP) book by David Rumelhart, James McClelland and the PDP research group, the field exploded. Since then, the field of artificial neural networks has attracted a great deal of attention and funds for further research.

2.3.1 McCulloch-Pitts Neurons

The research in artificial neural networks had its first interesting results about forty-eight years ago, when Warren McCulloch and Walter Pitts showed in their 1943 paper that even simple types of neural networks could, in principle, compute any arithmetic or logical function [Hech89]. In their paper, they assumed that the activity of the biological neuron was an “all-or-none” process [McPi43]. Thus, their model neuron was somewhat similar to a binary device with a fixed threshold. Also, they assumed that the model included the effect of synaptic decay and that the inhibitory synapse absolutely prevented excitation of the neuron. However, at that time, the authors did not mention any practical use of their work. Nevertheless, the paper was widely read and had great influence on the development of the network models and learning paradigms that followed.

2.3.2 Hebb's Learning Law

In 1949, Donald Hebb, in his book entitled *The Organization of Behavior*, postulated that the strength of a synaptic weight between two neurons increases whenever an axon of neuron *A* is near enough to excite a neuron *B* and repeatedly or persistently takes part in firing it [Hebb49]. In this way, often-used paths in the network are strengthened, and the phenomena of habit and learning through repetition are explained. This proposal of a specific learning law for the synapses of neurons has become the basic learning law of current artificial neural networks.

The Hebbian learning law, also called *Hebb's law*, can be expressed in mathematical notation as

$$\Delta w_{ij} = \alpha x_i x_j, \quad \alpha > 0; x_i, x_j \geq 0 \quad (2.1)$$

where Δw_{ij} is the change in synaptic weight, α is the constant of proportionality representing the learning rate, and x_i and x_j represent the activations of neuron *i* and neuron *j*, respectively. Originally, *Hebb's law* assumed positive activation values. Nevertheless, learning that involves neurons with negative activation values has also been labeled as Hebbian [EbDo90].

2.3.3 The Perceptron

The first successful neurocomputer, the Mark I Perceptron, was developed during 1957 and 1958 by Frank Rosenblatt, Charles Wightman, and others [Hech89]. The perceptron includes simple neuron-like processing elements of McCulloch and Pitts' model neuron, which aggregates the incoming inputs and forwards the result to a simple threshold function (see Fig. 2.2). Note that the input signals to the processing elements are the input

signals to the network multiplied by the connection weights. A network with N inputs can be expressed by the following equations:

$$net_i = \sum_{j=1}^N x_j w_{ji}, \quad (2.2a)$$

$$x_i = \begin{cases} 1 & \text{if } net_i > \theta \\ 0 & \text{otherwise.} \end{cases} \quad (2.2b)$$

Consequently, net_i represents the total sum of the incoming signals, x_i is the activation of neuron i , x_j represents the input signal to the network, w_{ji} is the connection weight value, and θ is a constant value representing the threshold (usually, $\theta = 0$). Notice that (2.2b) expresses the threshold function of McCulloch and Pitts' model neuron.

In order to do classification, the perceptron needs to be "taught" by pairs of training patterns. The training patterns consist of input patterns to be recognized and the desired output patterns, namely the target patterns. During training, the perceptron modifies its connection weights according to *Rosenblatt's learning law*, given by

$$\Delta w_{ij} = \alpha (t_i - x_i) x_j, \quad \alpha > 0; x_i, x_j \geq 0 \quad (2.3)$$

where t_i represents the target signal, and Δw_{ij} , α , x_i and x_j are as defined by Eq. 2.1.

Rosenblatt has proved that, given training data with linearly separable classes, a network of simple neuron-like processing elements, such as the perceptron, can develop connection weight values that separate the classes [McRu88, Hech89]. However, the incapability of the perceptron to cope with non-linearly separable tasks, such as XOR and parity problems, had discouraged many researchers at the time. These shortcomings have been cautiously described through a very careful mathematical analysis in Minsky and

Papert's book *Perceptrons* [MiPa88]. The book had a dramatic effect, and practically all work on Perceptrons came to a halt. Recently, the problems of a two-layer perceptron have been overcome through adding more layers. The revised version of the perceptron, sometimes called a multilayer perceptron, was developed independently by several researchers such as Werbos in 1974, Parker in 1982, and, Rumelhart, Hinton, and Williams in 1986, under different names [Wass89, MaHP90].

2.4 New Computing Paradigm

The development of new theories in the past decade has led to a variety of artificial neural network models. Today, there are at least 26 distinct models of artificial neural networks [MaHP90], each of which has its own advantages as well as problems and limitations. However, most of them share the same basic architecture, in that the networks consist of neuron-like processing elements linked together through connection weights, and their learning algorithms evolve from the Hebb's law.

2.4.1 The Networks

The neurons in a neural network may be organized in a number of different ways. Generally, several neurons are grouped together forming a *layer*. Neurons in a layer usually work together to perform a specific function. For example, the neurons in an input layer, acquire the input signals from the outside world. A network may have one or more layers. A network with two layers is called a *two-layer* network, whereas a network with more than two layers is called a *multilayer* network.

Every neuron is connected to other neurons. However, the patterns of connectivity between neurons vary across neural network models. Neurons within a layer may be laterally connected, while neurons between layers may be fully or sparsely connected.

From the patterns of neuron connectivity between layers, there are two kinds of neuron connectivity, namely, *feedforward* connections and *feedback* connections. Base on this structure, a network can be called a *feedforward* network or a *feedback* network. The differences between them are described in the following section.

2.4.1.1 Feedforward Network

A feedforward network has connections through weights extending from the outputs of neurons in a layer to the inputs of neurons in the next layers; e.g., connections from the input layer to the output layer in a two-layer network. A two-layer or multilayer feedforward network operates by means of propagating the input signals from the first layer in the network, usually an input layer, up to the last layer in the network, which generally is the output layer. Some examples of neural network models in this category are the perceptron [MiPa88, McRu88], the *backpropagation* (BP) [RHWi86, McRu86], and the *counterpropagation* (CPN) networks [Hech87].

2.4.1.2 Feedback Network

A feedback network, besides having forward connections (i.e., connections from the input layer to the output layer), also includes backward connections (i.e., connections from the output layer to the input layer). Thus, for a two-layer network, this means that the network has two sets of connection weights, one going from the first layer to the second, and the other connecting the second layer back to the first. Two of the most popular models of this type are *bidirectional associative memory* (BAM) [Kosk87a, Kosk87b, Kosk88], and *adaptive resonance theory 1* (ART1) [CaGr88, Gros88a, Gros88b].

2.4.2 Learning Methods

A unique feature of artificial neural networks is their abilities to learn from examples. This capability is achieved by means of a learning algorithm. Every neural network model has its unique learning algorithm. However, most of the learning algorithms have evolved from *Hebb's law*. The learning procedures can be categorized into three distinct types, namely, *supervised learning*, *reinforcement learning*, and *unsupervised learning*.

2.4.2.1 Supervised Learning

A network employing a supervised learning type algorithm requires labeled input data and an external "teacher". The teacher knows the desired correct response to each input and thus provides a detailed error signal after each trial. A network with this type of learnings usually calls for two distinct sets of patterns from the same problem domain, one for the training set and the other for the testing set. Consequently, the learning phase, often called training phase, is separated from the recognizing or testing phase. Some neural network models in this category are BAM, BP, and CPN.

2.4.2.2 Reinforcement Learning

Reinforcement learning, also called *graded training* [Hech89], is similar to supervised learning; that is, it requires training data and a teacher. The only difference is that the teacher only indicates whether a response was correct or incorrect and does not provide detailed error information. In other words, the teacher gives a sort of "performance score" that tells the network how well it has done overall since the last time it was graded. An example of networks with this type of learning is the graded learning network (GLN) [Souc89].

2.4.2.3 Unsupervised Learning

A network with unsupervised learning uses unlabeled input data and requires no external teacher. The network demands no separate pattern sets (i.e., no separate training or testing sets). All data inputs are treated as testing patterns. Its weights change over time as new patterns are presented to the network. From just the presentation of inputs, the network is expected to organize its weights into some “useful” configuration, thus, the learning is also known as *self-organization*. The ART1 neural network model is an example of models employing the unsupervised learning algorithms. Note that, for convenience, the term learning is used to refer to the unsupervised learning mode, whereas training is associated with the supervised learning mode.

2.5 Summary

This chapter provides a review of artificial neural networks. The discussion begins with a review of the biological neuron, which inspires the development of neuron-like processing elements. This leads to a definition of artificial neural networks. The historical background of research in artificial neural networks is also presented. One of the most popular neural networks of the past, the perceptron, has been discussed along with the original model of an artificial neuron and the original learning rule of neural networks. Finally, a discussion of the developmental history and a description of neural network structures as well as their various learning paradigms are presented.

CHAPTER III

ESSENTIAL FEATURES OF SELECTED ANN MODELS

This chapter presents a comparison of the following four neural network models: bidirectional associative memories (BAM) [Kosk88], backpropagation (BP) [RHWi86, McRu86], counterpropagation (CPN) [Hech87], and adaptive resonance theory 1 (ART1) [CaGr88]. These models are representatives of different classes. For instance, BAM is considered to be a supervised feedback network, BP is supervised feedforward network, CPN is unsupervised feedforward network and ART1 is an unsupervised feedback network. Although BP and CPN are networks of the same class (using the classification mentioned in Chapter II), they have very different learning principles. These learning procedures as well as their limitations are the topics to discuss in this chapter.

3.1 Bidirectional Associative Memory (BAM)

A bidirectional associative memory (BAM) is an associative network [Kosk87a, Kosk87b, Kosk88]. It is used to store and to recall information by association with other information. To be more specific, a BAM is called an *auto-associative* network, if the stored pattern can be recalled from its partial pattern, whereas it is called a *hetero-associative* network, if the stored pattern can be recalled through another associated pattern.

There are numbers of variants and evolutions on BAM, including continuous and discrete BAMs [Kosk87b, Kosk88], intraconnected BAM [Simp90], competitive adaptive

BAM (CABAM) [Kosk87a], temporal associative memory (TAM) [Kosk88], random ABAM (RABAM) [Kosk89], and others. However, this study focuses on the less complex type of BAM, that is the discrete BAM.

3.1.1 Network Topology

The discrete BAM, introduced by B. Kosko, is a bilayered non-linear feedback network. The network has symmetric interconnections between layers; i.e. connections from the outputs of the neurons in the first layer, F_A , to the inputs of the neurons in the second layer, F_B , denoted by connection matrix W , and connections from the outputs of the neurons in F_B , to the inputs of the neurons in F_A , denoted by connection matrix V . In general W and V differ in structure. However, in BAM, W and V are assumed to have the same, or approximately the same, structure [Kosk91]. One way to impose an equivalent structure is to set $W = V^T$ or $V = W^T$, where W^T and V^T denote the matrix transposes of W and V respectively. In practice, it is often sufficient to use only W to represent the connection matrix from F_A to F_B and W^T for connection matrix from F_B to F_A (Fig. 3.1).

Information passes forward from F_A to F_B through the connection matrix W . Similarly, information passes backward from F_B back to F_A through the matrix transpose W^T . This process is repeated until the network arrives at a stable point; that is, when neither information at F_A nor at F_B is changing. Since every real matrix is both a discrete and continuous bidirectionally stable associative memory [Kosk88], it is expected that gradual changes due to learning in W will result in stability.

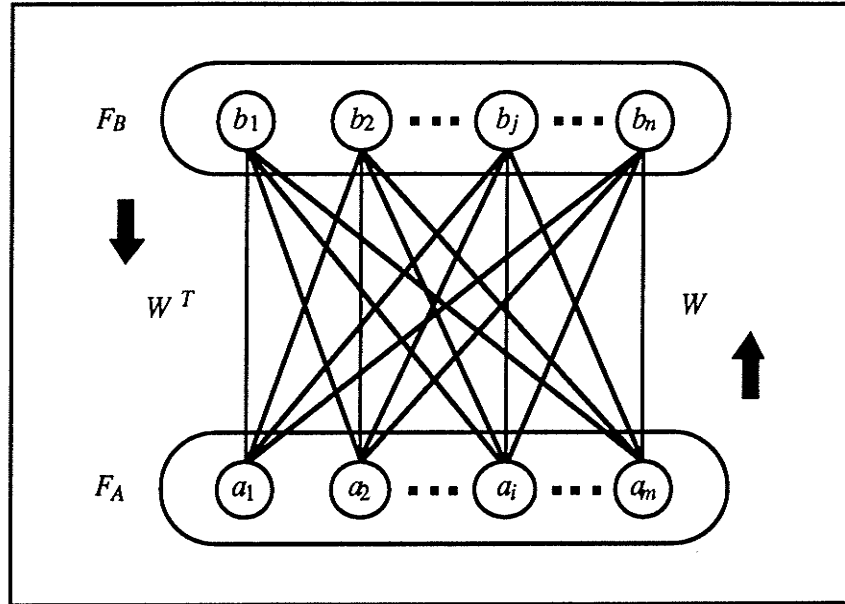


Fig. 3.1. Topology of a BAM network. After [Kosk88]

The neurons in the layers are two-valued, or *bivalent*, neurons with *hard-limit* threshold signal functions, and they process signals deterministically and synchronously. This type of neuron is similar to the Perceptron, which stems from the classical neural model of McCulloch and Pitts [McPi43]. If I_i represents the input signal to a neuron i ; $S(y_j)$ denotes the state of the neuron j in the other layer that is connected to the neuron i by a connection weight w_{ij} ; x_i denotes the sum of the input signals to a neuron i ; $S(x)$ is the threshold function; and index k indicates the discrete time step, then the state of the neuron i in a time step k is

$$x_i^{k+1} = \sum_{j=1}^N S(y_j^k) w_{ij} + I_i \quad (3.1a)$$

$$S(x_i^{k+1}) = \begin{cases} 1 & \text{if } x_i^{k+1} > \theta \\ S(x_i^k) & \text{if } x_i^{k+1} = \theta \\ 0 & \text{if } x_i^{k+1} < \theta \end{cases} \quad (3.1b)$$

for a number of neurons, N , connected to neuron i and an arbitrary real-valued threshold θ . A discrete BAM of which the threshold θ is equal to 0 is called a *homogeneous BAM*.

3.1.2 BAM Weight Modification Procedure

Neural networks, including BAM, store the information distributively in their connection weights through some learning algorithms. In discrete BAM, the connection weights, denoted by connection matrix W , are developed by the *outer-product* learning method [Kosk91]. This method sums weighted correlation matrices of the associations. For instance, let us assume that we wish to store N associations of binary vectors (A_i, B_i) for $i = 1, 2, \dots, N$. A_i denotes a binary vector of length m , and B_i denotes a binary vector of length n . Then, the sum of the N binary correlation matrices $A_i^T B_i$ is

$$W = \sum_{i=1}^N A_i^T B_i \quad (3.2a)$$

with dual BAM memory W^T given by

$$W^T = \sum_{i=1}^N B_i^T A_i. \quad (3.2b)$$

The expression in (3.2a) and (3.2b) is called the *binary outer-product law*. The associations can be encoded also through the following *bipolar outer-product law*. The bipolar outer-product law for W is

$$W = \sum_{i=1}^N X_i^T Y_i, \quad (3.3a)$$

and for W^T is

$$W^T = \sum_{i=1}^N Y_i^T X_i. \quad (3.3b)$$

X_i , a point in the bipolar m -cube $\{-1, 1\}^m$, is a bipolar vector transformed from binary vector A_i . Similarly, Y_i , a point in the bipolar n -cube $\{-1, 1\}^n$, is a bipolar vector transformed from binary vector B_i . Accordingly, (X_i, Y_i) is the bipolar vector association. Using the bipolar representation, more accurate recall is possible [Kosk88]. The superiority of bipolar to binary representation, in this matter, is due to the nature of the binary signals. Intuitively, binary signals implicitly favor 1s over 0s (i.e., $1+0 = 1$). In other words, there are only excitatory connections or zero-weight connections produced from multiplying and adding binary quantities. No inhibitory connections exist. On the contrary, bipolar signals are not biased in favor of 1s or -1 s (i.e., $-1+1 = 0$). Multiplying and adding bipolar quantities produces inhibitory connections as well as excitatory connections. These inhibitory connections prevent excitation of the unwanted pattern.

The learning method discussed so far is merely a process to encode the association (A_i, B_i) into the connection matrix W . To decode an association, in other words, to retrieve a stored association, the network uses its *resonance* characteristic. For example, suppose an association (A_1, B_1) has been stored in the network. Then, suppose we

provide vector A_I as an input to the neurons in one of its layer, say F_A , and vector B_I is the expected output pattern from the neurons in layer F_B . Vector A_I becomes the current state vector at F_A . Through the connection matrix W , the output signals from F_A are propagated to F_B producing a state vector B_1' . The output signals from F_B are then propagated through the transpose matrix W^T to F_A , producing a close replica of the original input vector A_I , say A_1' . The vector A_1' becomes the new state vector of F_A . This process is repeated until there is no more changes in F_A and F_B . In other words, the network reaches a stable point for the association (A_I, B_I) .

To show the stability of the network, Kosko uses *Lyapunov* or *energy* function E . This energy function represents each state (A_i, B_i) , and is given as

$$E(A, B) = - A W B^T \quad (3.4a)$$

for the binary vectors, or

$$E(X, Y) = - X W Y^T \quad (3.4b)$$

for the bipolar vectors. From this point of view, every association (A_i, B_i) (i.e., each stable point) is represented by a *local energy minimum*. Thus, storing an association is similar to “sculpting” the system energy surface. However, there is a maximum number of patterns that can be stored. This issue along with other limitations found in BAM are discussed in the following section.

3.1.3 Problems in BAM Model

The major drawback of the BAM model is its limited memory capacity; this is a restriction on the maximum number of associations it can accurately recall. If this limit is exceeded, the network may produce incorrect outputs; it “remembers” associations that it has not known before. This phenomenon is called *spurious memories* or *spurious attractors*. Spurious attractors tend to increase in frequency as the network dimensionality increases [Kosk91]. Some methods, such as the *unlearning* process [HoFP83] and encoding/decoding enhancement [WaCM89], are intended to reduce these spurious attractors. Kosko states that the rough estimate of the memory capacity of a BAM is less than the number of neurons in the smaller layer [Kosk87b], such as

$$N < \min(m, n) \tag{3.5}$$

where N is the maximum number of associations, m is the number of neurons in layer F_A , and n is the number of neurons in layer F_B . McEliece *et al.* [McE187] shows a different way to calculate the memory capacity bound, given by

$$N = \frac{m}{2 \log_2 m} \tag{3.6}$$

where m is the number of neurons in the smaller layer. Some researchers have proposed methods to overcome the problem of memory limitation in BAM. For example, Haines and Hecht-Nielsen introduced the *non-homogeneous BAM*, which uses non-zero thresholds [HaHe88]. Every neuron in the layers may have a different non-zero threshold. Using this technique, the new upper bound becomes $\min(2^m, 2^n)$. Yet another method is through

using *high-order BAM* [TWJo89, Simp90]. This technique can double the memory capacity and improve the error-correcting capability at the expense of greater connectivity. In software simulation, the *high-order BAM* requires more computational time.

In BAM, perfect recall of the associations requires mutually orthogonal input vectors. To show this, let us observe one step of the signal propagation from F_A to F_B , assuming that we use the bipolar version X_i of A_i . Thus,

$$\begin{aligned} X_i W &= (X_i X_i^T) Y_i + \sum_{j \neq i}^N (X_i X_j^T) Y_j \\ &= m Y_i + \sum_{j \neq i}^N (X_i X_j^T) Y_j, \end{aligned} \quad (3.7)$$

where m is the dimension of X_i . The first term in (3.7) shows that the desired pattern Y_i is given the maximum positive amplification factor $m > 0$. The second term shows the *crosstalk* or the *noise*. If the input patterns X_i s are orthogonal to each other, then the second term will be zero. However, in real applications, the input patterns usually are not orthogonal, and sometimes contain noise. This particular characteristic of BAM limits the applicability of BAM in real applications. A method, such as employing a pre-processor that transforms arbitrary input vectors into orthogonal vectors, has been introduced in [LiNu90]. Using this approach, an optimal recall can be achieved.

The discrete BAM encoding procedure has a side-effect, being such that it also encodes the complement patterns by default. For example, an association (A_i^c, B_i) , where A_i^c is the complement of vector A_i , cannot be stored in BAM if an association (A_i, B_i) exists. Consequently, this limits the combination of vectors that can be stored. Through adding some intralayer connections, the intraconnected BAM (IBAM) overcomes this limitation [Simp90].

Another limitation of BAM is that the pair of vectors (A_I, B_I) must be taken from a *continuous* function f . The function f must map small changes in inputs to small changes in outputs. In other words, similar input vectors, or close input vectors in the *Hamming distance* sense, are associated with similar output vectors or vice versa [Kosk91]. The *continuity assumption* in terms of Hamming distance in binary m -cubes and n -cubes can be expressed as

$$\frac{1}{m} H(A_i, A_j) \approx \frac{1}{n} H(B_i, B_j) \quad (3.8)$$

where $H(A_i, A_j)$ and $H(B_i, B_j)$ are the Hamming distances between vectors A_i and A_j , and between vectors B_i and B_j , respectively. The Hamming distance is defined as

$$H(A_i, A_j) = \sum_{k=1}^m |a_i^k - a_j^k|. \quad (3.9)$$

Fortunately, training sets derived from real-world problems tend to satisfy the continuity assumption, since most sampled processes are continuous [Kosk91].

3.2 Backpropagation (BP)

One of the most commonly used neural network models employing training procedure is backpropagation (BP). The BP training algorithm was formulated independently by several researchers, such as Werbos in 1974, Parker in 1982, and Rumelhart, Hinton, and Williams in 1986 [Wass89]. Backpropagation evolves from the two-layer perceptron model. However, backpropagation includes some hidden layers, which are not found in the original perceptron topology. Moreover, backpropagation employs continuous non-

linear transfer functions (e.g., sigmoid) in its neurons. These properties and an improved learning procedure give backpropagation more capabilities than the perceptron. For instance, BP is capable to cope with non-linearly separable problems, such as the XOR or the parity problem, which the perceptron fails to solve.

Most of the problems solved by the backpropagation model are mapping problems. In this particular case, backpropagation is used as a *mapping neural network*; i.e., a network in which the information processing operation is an approximation to some function or mapping $f: \mathcal{R}^n \rightarrow \mathcal{R}^m$ from vectors into vectors [Krei91]. It has been proven through Kolmogorov mapping neural network existence theorem that a three-layered feedforward neural network using any continuous and bounded neuron activation function, such as in backpropagation, is capable of approximating an arbitrary continuous mapping [Hech89, Funa89, HoSW90, Horn91, and Krei91].

3.2.1 Network Topology

Architecturally, BP consists of several layers of neurons: an input layer, one or more hidden layer(s), and an output layer. Neurons in each layer are connected to the next layer through connection weights. Since it is a *feedforward* network, no feedback connections exist in its structure. For example, a network with one hidden layer has some connections from neurons in the input layer to neurons in the hidden layer, and from neurons in the hidden layer to neurons in the output layer. No intraconnections among neurons within a same layer exist. Similarly, for a network with more than one hidden layer, the connections are established between the input layer and the first hidden layer, between the output layer and the last hidden layer, and between a hidden layer and the adjacent hidden layer up to the last hidden layer. This fully-connected scheme between the adjacent layers is not mandatory after all. It is possible for a BP to have sparsely connections between the

adjacent layers or even some connections between specific layers (e.g., some connections from the input layers to the second hidden layers). However, for simplicity, this discussion only covers the generic BP; that is, a network with full connections between its adjacent layers. A typical backpropagation network with one hidden layer is shown in Fig. 3.2.

A network using linear neurons cannot solve more problems in multiple layers than it can in a single layer [McRu88]. Therefore, to effect the advantage of having many layers, backpropagation uses non-linear neurons. It has been shown that any arbitrary smooth and bounded non-linear function can be used for the threshold function in the neurons of a multi-layer network [HoSW90, Horn91, Krei91]. In practice, the often used non-linear function is the sigmoid or *semi-linear* function.

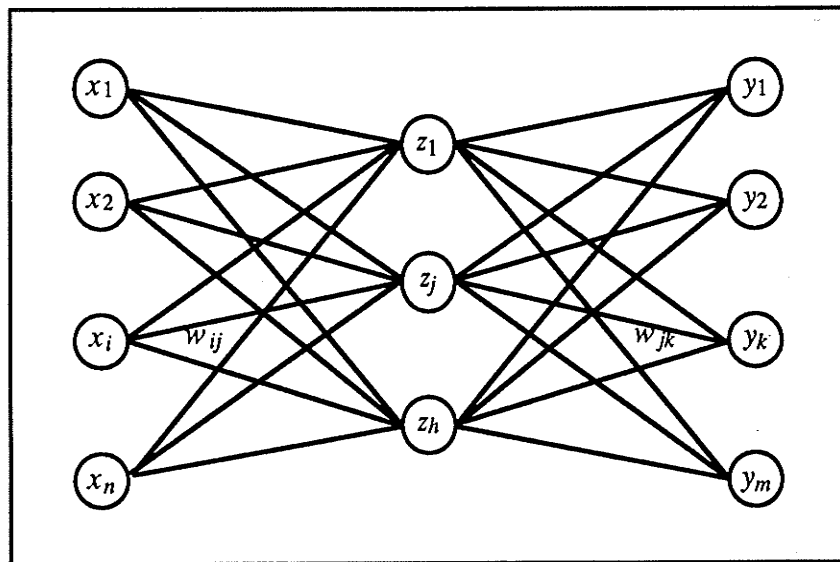


Fig. 3.2. Topology of a single hidden layer BP network.

A sigmoid function is a non-decreasing and differentiable function, which is expressed by

$$S(x) = \frac{1}{1 + e^{-x}} \quad (3.10)$$

The original BP uses the sigmoid function in all of its neurons in the layers, except for the neurons in the input layer. The input layer employs only linear neurons, which accept the component of the input vector and distribute them, without modification, to the next layer (i.e., to the hidden layer in a three-layer network).

Neurons in the hidden and the output layers include some *biases*. A bias gives an offset to the summed inputs to a neuron, thereby shifts the sigmoid function with a constant value θ (see Fig. 3.3). Thus, it performs similar to a threshold constant in (3.1b). In other words, a bias provides a means of scaling the average input into a useful range [MaHP90]. These bias values, θ s, are adjusted during training, and they are kept unchanged once the training process is done.

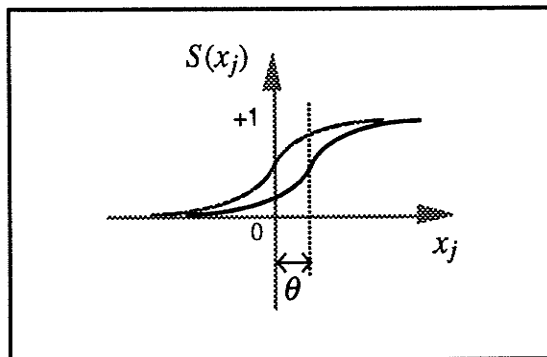


Fig. 3.3. Effect of a bias term θ to the sigmoid function.

It has been defined (in Section 2.2) that a neuron aggregates the incoming input signals and forwards the result to a threshold function. Similarly, every neuron in the hidden and the output layers of a BP network has those capabilities. These neurons are identical to the ones defined in (3.1a), except that they do not have feedback connections and they employ a different threshold function. For convenience, it is rewritten as follow

$$x_j = \sum_{i=1}^N S(x_i) w_{ji} + \theta_j \quad (3.11a)$$

$$S(x_j) = \frac{1}{1 + e^{-x_j}} \quad (3.11b)$$

where $S(x_i)$ represents the state of neuron i in the previous layer, which is connected to neuron j by a connection weight w_{ji} , x_j denotes the sum of the input signals to neuron j , θ_j is the bias of neuron j , S is the threshold signal function, and N is the number of neurons in that layer. For the neuron in the hidden layer of a three-layer network, the $S(x_i)$ is equal to I_i , the input signal. The index k , which indicates the discrete time step, in (3.1a) is omitted, since BP processes the data in one time step; that is, the states of neurons in a layer only depend on the states of the neurons in the previous layer, within a same time step. Data is propagated from a layer to the next layer via the connections, starting from the input layer to the last layer, which is the output layer. Although, BP processes the data in one time step, the training process, which is merely a *searching for appropriate weights* process, requires enormous time step. This training procedure, also called the *generalized delta rule*, is described in the following section.

3.2.2 BP Weight Modification Procedure

The procedure of training a BP network requires a set of pairs of input and target patterns. The target patterns are used as the “teacher” of the network during training. In backpropagation, the network first uses the input pattern to produce the output pattern. The input pattern is propagated through the layers, and an output is produced. The output pattern is compared with the target pattern, and the difference or the error between them is measured. There are several ways to measure the error, since each type of error has different costs in different situations [MiPa88, Hech89]. One of them is the *mean squared error* method. For a set of M pairs of input and target patterns, the error function is given by

$$E = \sum_{p=1}^M E_p = \frac{1}{2} \sum_{p=1}^M \sum_{j=1}^N (t_j^p - S(x_j^p))^2 \quad (3.12)$$

where the index p ranges over the set of input/target pattern pairs, j refers to the j th neuron in the output layer with N neurons, t_j^p is the p th target pattern for the output neuron j , $S(x_j^p)$ is the state or the actual output value of output neuron j for pattern p , x_j^p denotes the sum of the input signals to the output neuron j (3.11a) for pattern p , E_p represents the error on pattern p , and E is the total error of the entire set of patterns.

The training procedure is meant to make the error between the actual output and the target as small as possible. Through modifying the connection weights according to the error information, the error function is brought to its minimum, thus allowing the best approximation of the target. In BP, this scheme is performed by the *least-mean-square* (*LMS*) procedure of Widrow and Hoff [McRu88].

The LMS procedure makes use of the delta rule for adjusting the connection weights. It searches for the weight values that minimize the error function (3.12) using a method called *gradient descent*. The gradient descent adjusts the weight proportional to the negative of the derivative of the error with respect to each weight, as described by

$$\Delta_p w_{ji} = -c \frac{\partial E_p}{\partial w_{ji}} \quad (3.13)$$

where $\Delta_p w_{ji}$ represents the amount of change in weight w_{ji} for pattern p , and c is the constant of proportionality. The right part of the equation (3.13) can be rewritten as

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial x_j^p} \frac{\partial x_j^p}{\partial w_{ji}}, \quad (3.14a)$$

where the first term of (3.14a) can be partitioned further into

$$\frac{\partial E_p}{\partial x_j^p} = \frac{\partial E_p}{\partial S(x_j^p)} \frac{\partial S(x_j^p)}{\partial x_j^p}. \quad (3.14b)$$

Using the error function in (3.12), the first term of (3.14b) becomes

$$\frac{\partial E_p}{\partial S(x_j^p)} = -(t_j^p - S(x_j^p)), \quad (3.14c)$$

and the second term of (3.14b) becomes

$$\frac{\partial S(x_j^p)}{\partial x_j^p} = S'(x_j^p). \quad (3.14d)$$

From (3.11a), the second term of (3.14a) becomes

$$\frac{\partial x_j^p}{\partial w_{ji}} = S(x_i^p). \quad (3.14e)$$

Thus, replacing the right part of (3.14b) with (3.14c) and (3.14d), and then, substituting (3.13) with (3.14b) and (3.14e), we get

$$\begin{aligned} \Delta_p w_{ji} &= -c \frac{\partial E_p}{\partial w_{ji}} \\ &= c (t_j^p - S(x_j^p)) S'(x_j^p) S(x_i^p). \end{aligned} \quad (3.15)$$

However, equation (3.15) is only valid for adjusting the weights of the connections attached to the neurons in the output layer. For adjusting the other connection weights (i.e., the connections which are not attached to the output neurons), we need to calculate the error change with respect to the output of neuron j in the hidden layer, that is $\frac{\partial E_p}{\partial S(x_j^p)}$. Since the error information is back propagated from the output neurons towards the input neurons, then the error change with respect to the hidden neuron j is

$$\begin{aligned} \frac{\partial E_p}{\partial S(x_j^p)} &= \sum_{k=1}^N \frac{\partial E_p}{\partial x_k^p} \frac{\partial x_k^p}{\partial S(x_j^p)} \\ &= \sum_{k=1}^N \frac{\partial E_p}{\partial x_k^p} \frac{\partial}{\partial S(x_j^p)} \left(\sum_{j=1}^M S(x_j^p) w_{kj} + \theta_k \right) \\ &= \sum_{k=1}^N \frac{\partial E_p}{\partial x_k^p} w_{kj}, \end{aligned} \quad (3.16)$$

where $\frac{\partial E_p}{\partial x_k^p}$ is the error change with respect to the neuron k in the output layer (3.14b), and N is the number of output neurons. For a network with more than one hidden layer, equation (3.16) is also valid for calculating the error changes of the neurons in the other hidden layers, where $\frac{\partial E_p}{\partial x_k^p}$ is the error change with respect to the neuron k in the subsequent hidden layer. So, the amount of weight change for the connection of hidden neurons is calculated by

$$\Delta_p w_{ji} = c \left(S'(x_j^p) \sum_{i=1}^N \frac{\partial E_p}{\partial x_k^p} w_{kj} \right) S(x_i^p), \quad (3.17)$$

where j indicates the j th neuron of a hidden layer, k indicates the k th neuron of the next adjacent layer, and i indicates the i th neuron of the previous adjacent layer. In a three-layer network, i denotes the i th neuron of the input layer and k denotes the k th neuron of the output layer. If we define

$$\delta^p = -\frac{\partial E_p}{\partial x^p}, \quad (3.18)$$

then, (3.15) and (3.17) become

$$\Delta_p w_{ji} = \epsilon \delta_j^p S(x_i^p), \quad (3.19a)$$

where $c \approx \epsilon$ represents the learning rate, and δ_j^p is given by

$$\delta_j^p = (t_j^p - S(x_j^p)) S'(x_j^p) \quad (3.19b)$$

if j indicates the j th neuron of the output layer, or

$$\delta_j^p = S'(x_j^p) \sum_{k=1}^N \delta_k^p w_{kj} \quad (3.19c)$$

if j indicates the j th neuron of the hidden layer. A *momentum* term is introduced into the learning rule in (3.19a), so that (3.19a) becomes

$$\Delta_p w_{ji}(n+1) = \varepsilon \delta_j^p S(x_i^p) + \alpha \Delta_p w_{ji}(n), \quad (3.20)$$

where n is the index of the training cycle, and α is the *momentum* constant. The momentum term is meant to filter out high-frequency variations of the error-surface in the weight space. In other words, the momentum term suppresses oscillation.

The LMS training procedure requires an iterative process in order to find a solution of a particular problem. A solution is found if the system reaches the *global minimum* (i.e., an error minimum in error surface with respect to the weights that constitutes solutions to the problems in which the system reaches an errorless state [McRu88]). However, the gradient descent method used in the LMS learning procedure does not guarantee a solution [McRu86], and even if the global minimum can be found, the time required to reach it cannot be predicted. These “unique” characteristics of the BP model are elaborated further in the next section.

3.2.3 Problems in BP Model

The LMS procedure is a procedure to minimize an error function (3.12). It searches for the weight values where the error function takes on a minimum value. An extremum (a minimum point in our case) can be either *global* (truly the lowest) or *local* (the lowest in a

finite neighborhood) [PFTV88]. However, virtually nothing is known about finding global extrema in general. So far, there are two standard heuristics which are often used: (i) find local extrema starting from widely varying starting values of the independent variables, and choose the most extreme of these, or (ii) perturb a local minimum by taking a finite amplitude step away from it, and see if the routine returns to a better point, or “always” to the same one. The LMS procedure with the *gradient descent* uses the latter. However, there is no guarantee that the method will always reach a global minimum. This leads to the local minima problem. It is suspected that the LMS procedure (i.e., the BP algorithm) actually converges to a local minimum, if it converges at all [Kosk91]. From this point of view, the backpropagation is considered to reach a “global” minimum (i.e., it finds a solution of a problem), if it reaches an error minimum in error surface with respect to the weights which is less than a tolerance value. Some techniques to anticipate the local minima problem have been introduced in [Baba89, BuLu90, and Fere91].

It is known that the computational time taken for training the BP network (i.e., the process to find a global minimum) to learn a problem is unpredictable. Sometimes, the training may take a while, but often it requires an enormous computational time to converge. This phenomenon emerges due to the nature of the *gradient descent* or the *steepest descent*. For example, suppose the gradient descent is used to find a minimum on the error surface with respect to two weights w_1 and w_2 (see Fig. 3.4). This method performs many small steps in going down a long, narrow valley. A step starts off in the local gradient direction, perpendicular to the contour lines, and traverses a straight line until a local minimum is reached, where the traverse is parallel to the local contour lines. Consequently, it will take many steps to reach a local minimum, and hence more steps to reach a global minimum. The size of the step may be tailored through changing the learning rate ϵ in (3.20). A large value of ϵ makes the process run faster, but this also may

increase oscillation and the eventually of reaching a local minima. On the contrary, a small value prevents oscillation, but this makes it run slower, thus lengthening the training time. The choice of the learning rate ϵ is critical for the training speed. This explains why the backpropagation requires an excessive amount of time for its training. Moreover, it was found that training in backpropagation is an *NP-complete* problem; that is, the computational time grows exponentially with the size of the network [DARP88, Judd90]. Several methods to accelerate the convergence have been proposed in [AlKe90, Hagi90, Li90, WeMa91].

Theoretically, a BP network with as few as one hidden layer and a non-constant activation function (e.g., sigmoid function) is capable of approximating any continuous mapping $f: \mathfrak{X}^n \rightarrow \mathfrak{X}^m$ (f belongs to L_2), provided that sufficiently many hidden neurons are available [Funa89, HoSW90, Horn91, and Krei91]. Yet, how to determine the exact number of the hidden neurons is another issue. It has been proven in [SaAn91, HuHu91, MeMR91] that a network with one hidden layer can implement exactly an arbitrary training set with p training patterns, provided that $p-1$ hidden neurons are used. However, while this shows the least upper bound of the number of hidden neurons required to solve the training set only, not much is known about the optimum number of hidden neurons required in order that the network performs best (smallest error) on both the training and testing data. So far, this optimum number of neurons is determined by trial and error. There is some evidence [KiIL90, WeKu90, SiDo91] that a network with too many hidden neurons tends to *memorize* the task rather than *generalize* it (i.e., *overfitting* the data). In solving real-world problems, the learning process often involves massive training data. Following the least upper bound theory, it would require an impractically large number of hidden neurons to train the network with only a single hidden layer. For this purpose, a network with more hidden layers might be better applied. Nonetheless, multiple hidden

layers are more complicated to analyze, since there are two variables to adjust, namely, the number of layers and the number of neurons per layer.

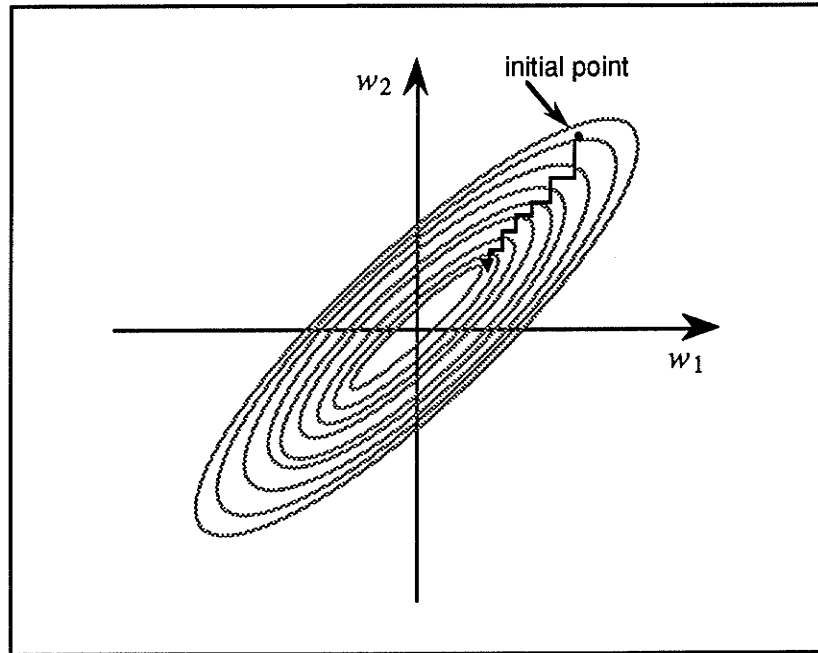


Fig. 3.4. Steepest descent method on the error surface with respect to w_1 and w_2 . After [PFTV88]

3.3 Counterpropagation (CPN)

The counterpropagation network [Hech87], invented by R. Hecht-Nielsen, is a unique neural network model. It is a combination of a portion of the *self-organizing map* of Kohonen [Koho90] and the *outstar* structure of Grossberg [Carp89]. Using this combination, the network self-organizes a near-optimal look-up table approximation to the mapping. In other words, it functions as a statistically optimal self-programming look-up table.

Counterpropagation can learn both binary and continuous vector mappings. To learn a mapping $f: \mathcal{R}^n \rightarrow \mathcal{R}^m$, the network requires pairs of input and target patterns. From this viewpoint, the counterpropagation network is considered as a supervised learning network, or a network with training procedure. However, from the way it encodes or learns pattern information in its synaptic topologies, the network may have an unsupervised learning procedure [Kosk91]. This unique learning procedure is described in the following section.

3.3.1 Network Topology

The full counterpropagation network comprises of five layers: two input layers, two output layers, and a single hidden layer. It is designed to approximate a continuous function $f: A \subset \mathcal{R}^n \rightarrow B \subset \mathcal{R}^m$, defined on a compact set A , where the inverse of the function $f^{-1}: B \subset \mathcal{R}^m \rightarrow A \subset \mathcal{R}^n$ exists and is continuous. However, for the case of a non-invertible continuous mapping, the forward-only version of the CPN network can be used. This network's topology is shown in Fig. 3.5.

The forward-only CPN network consists of three layers: an input layer, a single hidden layer, and an output layer. The input neurons serve only as fan-out points and perform no computation. The neurons in the input layer are fully connected to the neurons in the hidden layer (or *Kohonen* layer) by a weight matrix W . Similarly, all neurons in the hidden layer are connected to all neurons in the output layer (or *Grossberg* layer), by a weight matrix V . This architecture is similar to a fully connected feedforward BP network. However, there are some differences in their neurons. Backpropagation uses the sigmoid function for the activation of every neuron in its hidden and output layers, whereas counterpropagation employs linear activation for its neurons in the hidden and output layers. Furthermore, the neurons in the Kohonen layer are competitive, that is, only a single hidden neuron (in the *accretive* mode) can be activated (output of '1'). The

remaining Kohonen neurons are deactivated (output 0). This scheme is called *competitive learning* [Gros88b]. In physical implementations, the competitive learning scheme may use the *on-center off-surround* networks [Gross76, Gross88b]. Through this scheme, the non-linearity characteristic (which is essential for a neural network) is preserved.

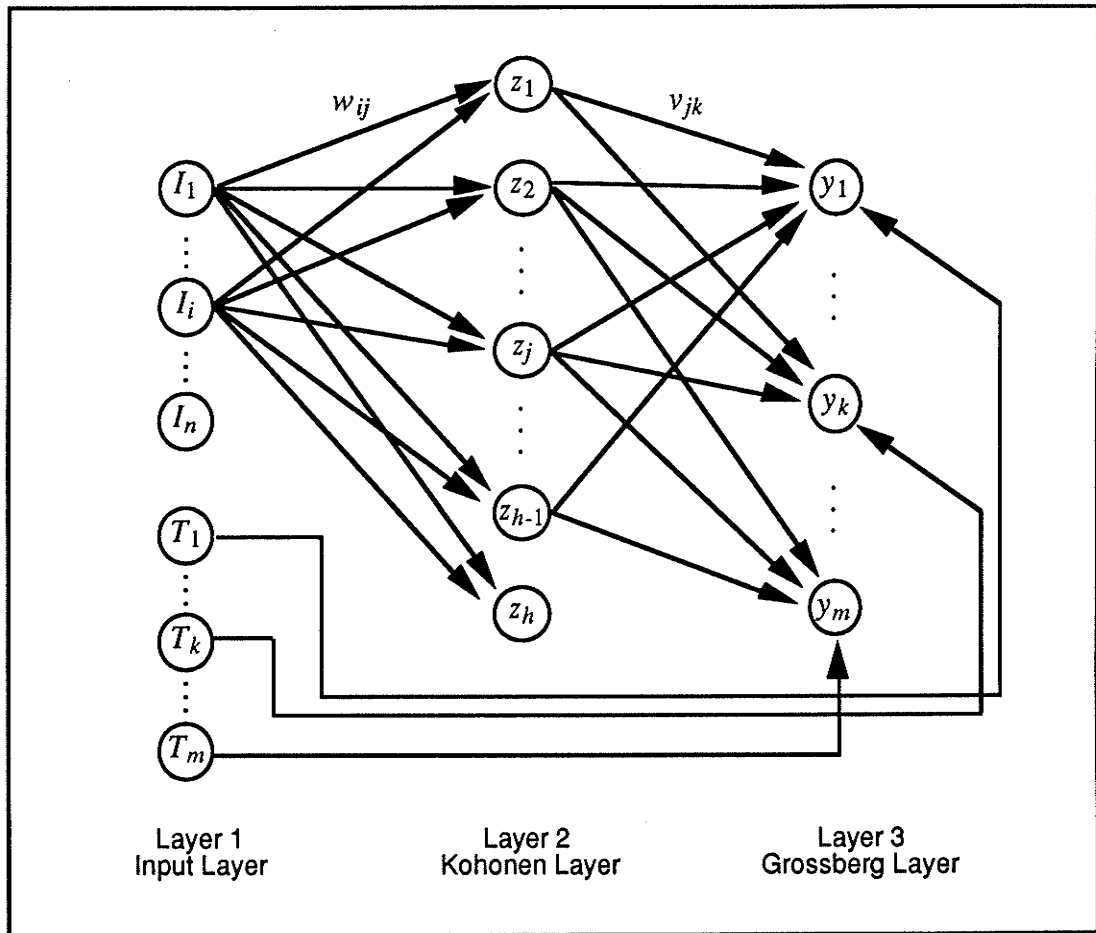


Fig. 3.5. Topology of a forward-only CPN. After [Hech89]

3.3.2 CPN Weight Modification Procedure

Like any other training network, a pair of input and target patterns is presented one by one to the network during the training process. The input pattern is propagated through the weight matrix W . At the Kohonen layer, every neuron competes with one another. The one with the biggest total sum of incoming signals is the “winner”. A winner neuron has an activation 1 at its output, while the rest have zero activations. This scheme is also called “winner-takes-all” [Gross88]. The propagation scheme is expressed by

$$x_j = \sum_{i=1}^n w_{ji} I_i, \quad (3.21a)$$

whereas the “winner-take-all” scheme is given by

$$z_j = \begin{cases} 1 & \text{if } j \text{ is the smallest integer for} \\ & \text{which } x_j \geq x_r, \forall r, j \neq r \\ 0 & \text{otherwise,} \end{cases} \quad (3.21b)$$

where I_i represents the value of the i th component of the input pattern (since the input neurons are merely fan-out points), w_{ji} is the connection weight from input neuron i to hidden neuron j , x_j is the total sum of the incoming signals to hidden neuron j , and z_j represents the activation of the hidden neuron j . Equation (3.21a) is simply a dot-product of the input vector $\mathbf{I} = (I_1, I_2, \dots, I_n)$ and the weight vector $\mathbf{w}_j = (w_{j1}, w_{j2}, \dots, w_{jn})$. The hidden neuron with the largest dot-product is declared the winner.

The propagation scheme can also be expressed as a distance measured (in *Euclidean* metric sense) of the two vectors \mathbf{w}_j and \mathbf{I} . From this point of view, the winner will be the neuron with the closest weight vector \mathbf{w}_j to the input vector \mathbf{I} . Such a scheme can also be

expressed by

$$d(\mathbf{w}_j, \mathbf{I}) = \|\mathbf{w}_j - \mathbf{I}\| = \left(\sum_{i=1}^n (w_{ji} - I_i)^2 \right)^{1/2}, \quad (3.22a)$$

$$z_j = \begin{cases} 1 & \text{if } j \text{ is the smallest integer for which} \\ & d(\mathbf{w}_j, \mathbf{I}) \leq d(\mathbf{w}_r, \mathbf{I}), \forall r, j \neq r \\ 0 & \text{otherwise,} \end{cases} \quad (3.22b)$$

where $d(\mathbf{w}_j, \mathbf{I})$ is the *Euclidean distance* between the weight vector \mathbf{w}_j and input vector \mathbf{I} .

The activations of the hidden neurons specify which weight vector needs to be changed. The weight adjustment follows the Kohonen learning rule:

$$\Delta w_{ji} = \alpha(t) (I_i - w_{ji}) z_j \quad (3.23)$$

where Δw_{ji} represents the amount of weight change, and $\alpha(t)$ is the learning rate which decreases with time to zero, $0 < \alpha(t) < 1$. From (3.23), it is shown that only the connection weight that attaches to the winner neuron j is updated. All the other weights remain unmodified. Notice that there is no target vector required to adjust the weights in the Kohonen layer. This explains why some literature [Kosk91] classify the CPN as an unsupervised learning network. Yet the CPN network has a second layer, namely the Grossberg layer, that makes CPN appear as a supervised learning network. The Grossberg layer requires a target pattern and the information from the Kohonen layer to adjust its weights. The weights that are related to the Grossberg layer are the values of the connections between the Kohonen layer and the Grossberg layer. The information from the Kohonen layer is propagated through this connection matrix V to the Grossber layer.

The propagation is simply another dot-product transformation given by

$$y_k = \sum_{j=1}^h v_{kj} z_j, \quad (3.24)$$

where y_k denotes the activation of neuron k in Grossber layer, and v_{kj} is the value of the connection between neuron j in Kohonen layer and neuron k in Grossber layer. Since there is only one neuron in the Kohonen layer (for the *accrative* mode) having an activation value different from zero, the output vector is nothing but the weight vector $\mathbf{v}_j = (v_{j1}, v_{j2}, \dots, v_{jm})$ where j is the index of the winning neuron in Kohonen layer (assuming that the allowable value different than zero is 1). From this scheme, it is clear now that in order to get a desired output vector, given a particular input vector to the network, a target vector $\mathbf{T} = (T_1, T_2, \dots, T_m)$ needs to be encoded into the weight vector \mathbf{v}_j . This encoding scheme uses the Grossberg learning rule:

$$\Delta v_{jk} = \beta (T_k - v_{jk}) z_j \quad (3.25)$$

where Δv_{jk} represents the amount of the weight change, and β is the Grossberg learning constant ($0 < \beta < 1$). From (3.25), it is shown that the weights of the Grossberg layer will converge to the average values of the target vectors, whereas the weights of the Kohonen layer (through Eq. 3.23) will self organize and distribute themselves in an almost equiprobable configuration. After training is done, all the weights are frozen and only Eq. 3.21 (or Eq. 3.22) and Eq. 3.24 are used. In this configuration, the network is ready to be used as a feedforward network.

3.3.3 Problems in CPN Model

The Kohonen learning procedure as in (3.23) has a problem. It sometimes leads to neurons which are under-utilized (i.e., neurons that never win). This problem appears especially when there are some initial weight vectors that are closer to the input vectors than others in the vector space. The latter weight vectors are likely to become “losers”. Moreover, there is some evidence that even though all the weight vectors are initialized with the same value, the under-utilization problem still exists [AKCM90]. This problem can be overcome through employing some “conscience” parameters [Hech87, DeSi88, AKCM90]. The conscience parameter biases the competition process so that each neuron in Kohonen layer can win the competition with close to the $\frac{1}{N}$ probability desired for an optimal vector quantization, where N is the number of neurons in Kohonen layer.

The normalization process (assuming the Euclidean norm is used) replaces the “gain” information of a vector to gain 1 (unit length). For instance, vector $\mathbf{x} = (0.1, 0.1)$ and vector $\mathbf{y} = (0.9, 0.9)$ are treated as the same vector. For some problems that consider the gain of the vector is important, the dot product operation as in (3.21a) and (3.21b) is inappropriate. To overcome this problem, the distance measure as in (3.22a) and (3.22b) may be used instead, since no normalization process is necessary for this procedure. However, this distance measure procedure requires a different network topology, since it cannot use the propagation scheme such as in (3.21a) [Koho90].

3.4 Adaptive Resonance Theory 1 (ART-1)

So far, the discussion has covered the supervised learning networks. These networks have failed to solve the *stability-plasticity* dilemma (i.e., the ability of a system to remain plastic, or adaptive, in response to unexpected changes and yet retain the stability to preserve previously learned knowledge) [CaGr88]. For example, learning a new pattern in

these networks erases the existing one, if the previous one is not retrained together with the new one. A network without this stability-plasticity characteristic is incapable to adapt itself autonomously in real time from unexpected changes in the real world.

As a solution to this problem, Grossberg and Carpenter introduced the Adaptive Resonance Theory (ART) model. This model maintains the plasticity required to learn new patterns, and still preserves the stability required to protect the knowledge that had been learned previously. The ART model requires no target patterns, so it is an unsupervised learning network, or we can say a learning network as opposed to a training network (i.e., a supervised learning network). The learning is achieved in real time through direct “confrontation” with its experiences [CaGr88]. Since there is no particular pattern for the output, the network is most useful as a pattern recognizer where only a single output neuron can be active at a time. Currently, there are three different models of ART networks, namely, ART-1, ART-2, and ART-3. However, only the ART-1 model, which recognizes binary patterns, is discussed in this chapter.

3.4.1 Network Topology

The ART-1 model has a slightly different topology than the previously discussed networks. It consists of two layers of neurons: the first layer or the *comparison* layer, denoted by F_1 , and the second layer or the *recognition* layer, denoted by F_2 (see Fig. 3.6). Between these two layers there are two connection weight arrays called a *bottom-up adaptive filter* that connects neurons in F_1 to neurons in F_2 , and a *top-down adaptive filter* that connects neurons in F_2 to neurons in F_1 . So far, from the network’s topology, ART-1 appears to be a similar network to a BAM network; that is, a feedback network. However, ART-1 uses the top-down pathways differently. Using these pathways, ART-1 employs a top-down learned expectation scheme that focuses attention upon bottom-up

information in a way that protects previously learned memories from being washed away by new learning. In order to achieve this, three additional parts, namely, the gain control of F_1 denoted by Gain-1, the gain control of F_2 denoted by Gain-2, and the STM (Short Term Memory) reset wave denoted by A provide a mechanism to control the neurons in both layers. These many different parts that works in harmony along with a unique learning procedure make the ART-1 behaves differently from the other feedback networks. The details on its learning procedure are discussed in the Section 3.4.2.

The neurons in F_1 are more complex than the ones usually encountered in the three models previously discussed. They have to keep track of two values in the comparison phase; the input pattern and the top-down expectation pattern. In addition, they also have to store the input pattern for a finite time, so that the input pattern will be available for the next matching if the first attempts at matching fail. The device that is capable of storing knowledge for a short period is often called a *short-term memory* (STM), as opposed to a *long-term memory* (LTM) found in the connection paths. Note that a linear activation function is employed in F_1 . The other process besides the comparison process is the normalization process of the input pattern as given by

$$x_i = \frac{I_i}{\sum_{k=1}^N I_k} \quad (3.26)$$

where x_i denotes the activation (or STM trace) of neuron i in F_1 , and I_i denotes the input signal to neuron i . From Fig. 3.6, we can see that there is an input coming into F_1 from the gain control Gain-1 and there is another one coming out from F_1 to the STM reset wave A . The Gain-1 signal is used to enable F_1 to distinguish between bottom-up input patterns and top-down *priming*, or expectation, patterns, and to match these bottom-up

and top-down patterns by the 2/3 Rule (which is described in the learning section). The result of the matching or comparison process will determine the state of the STM reset wave A .

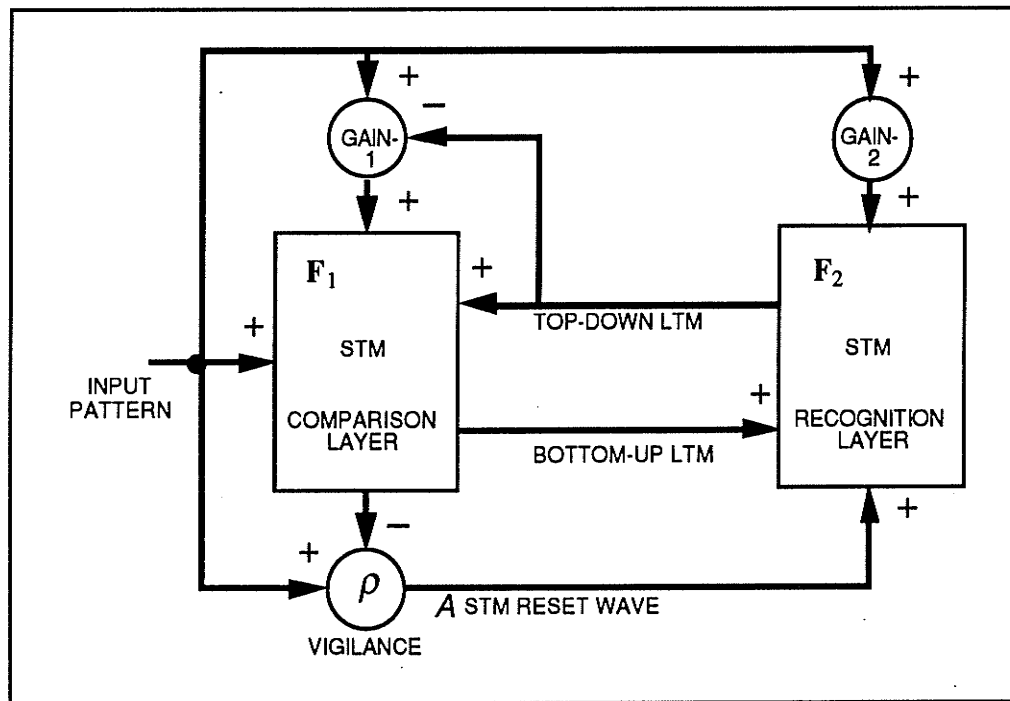


Fig. 3.6. Topology of an ART-1 network. After [CaGr88]

The ART-1 model grew from a simpler type of adaptive pattern recognition network, called a *competitive learning* model. Neurons in the recognition layer of the competitive learning model compete with each other in response to an input pattern. The neuron with the closest weight vector to the input vector becomes the winner. Consequently, every

neuron in the recognition layer becomes a representation of a particular class of the input patterns. In ART-1, the competitive learning scheme is implemented in the F_2 layer (the recognition layer). The winner neuron in F_2 will later enable a top-down expectation pattern that is required in the comparison process of F_1 . Notice that the control gain Gain-2 enables F_2 to react *supraliminally* to signals from F_1 while an input pattern is on.

3.4.2 ART-1 Weight Modification Procedure

In ART-1, there are two arrays of interconnection weights that need to be adjusted during the learning process, namely, the bottom-up connection weights and the top-down connection weights. These weights are modified according to the competitive learning procedure. For example, suppose that an input pattern I activates F_1 . The signals from F_1 are then propagated through the bottom-up connections to the F_2 layer. Since the F_2 layer employs a competitive learning scheme, a single neuron j in F_2 , which receives the largest total signal, becomes the winner. This neuron j quickly reads out its learned top-down expectation V to F_1 via the top-down connections. At F_1 , the top-down expectation pattern and the bottom-up input pattern are matched. If expectation V matches input I , the bottom-up weight vector b_j which corresponds to the winning neuron j is adjusted according to a learning rule (i.e., fast learning rule) given by

$$b_{ji} = \frac{L x_i}{L - 1 + \sum_{k=1}^N x_k} \quad (3.27)$$

where j is the index of the winning neuron in F_2 , i is index of a neuron in F_1 , b_{ji} represents the bottom-up connection weights, and L is a constant > 1 (typically 2).

Similarly, the top-down weight vector t_j is adjusted by

$$t_{ji}(n+1) = t_{ji}(n) x_i \quad (3.28)$$

where t_{ji} represents the top-down connection weights, and n is the index of the learning step. On the other hand, if expectation V mismatches input I , the mismatch event significantly inhibits STM activity across F_1 which then stimulates A to send a reset wave to F_2 . A parameter called the *vigilance parameter* determines how large a mismatch will be tolerated before A emits a reset wave. The vigilance test can be done using an inequality expressed by

$$\frac{\sum_{i=1}^N t_{ji} x_i}{\sum_{i=1}^N x_i} \geq \rho \quad (3.29)$$

where ρ is the vigilance parameter, and x_i and t_{ji} are the same notation as used in (3.27) and (3.28). If this inequality is satisfied, then A will emit a reset wave. From (3.29), it is shown that low vigilance value tolerates large mismatches, thus preventing A from emitting the reset wave. For now, let us assume that a large mismatch takes place at F_1 ; that is, the left hand side of the inequality (3.29) is less than the vigilance parameter ρ , so that the inequality (3.29) is false. Accordingly, A emits a reset wave to F_2 . The reset wave selectively inhibits the active population in F_2 , and this inhibition is long lasting. The inhibition of the winning neuron j in F_2 leads to removal of the top-down expectation V , and thus terminates the mismatch between I and V . Input pattern I can then activate F_1 for the second time, and again, the signal from F_1 is propagated through the bottom-up

connections to the F_2 layer. Due to the enduring inhibition of the previous winner, another neuron j^* (usually the neuron with the second largest total signal) becomes the current winner. Similarly, another top-down expectation pattern (i.e., the weight vector in the top-down pathway that corresponds to the neuron j^*) V^* is produced at F_1 . And again, the comparison process in F_1 is repeated. This procedure is repeated until one of three possibilities occurs: (i) a neuron j in F_2 is chosen whose top-down expectation matches with input I , (ii) a previously uncommitted neuron in F_2 is selected, or (iii) all the neurons in F_2 are committed and no one can accommodate input I (i.e., the system has reached its full capacity. No new category can be made).

The Gain-1 parameter controls the process in F_1 layer. For example, the first time an input pattern I is given to the input of F_1 , the input signal also enables the Gain-1. This makes F_1 *supraliminally* activated; that is, activated enough to generate output signals to other parts of the network and thereby to initiate the hypothesis testing cycle. When the top-down expectation is initiated, this signal disables the Gain-1 and thereafter F_1 becomes *subliminally* activated; that is, attentionally prime F_1 for future input pattern that may or may not generate an approximate match with the expectation pattern, but does not generate output signals. In other words, during this phase, F_1 is in comparison mode and no output signals are generated. This rule for matching a bottom-up input pattern with a top-down expectation at F_1 is called the 2/3 Rule. Likewise, the Gain-2 parameter controls the activation in F_2 . However, the Gain-2 does not use the 2/3 Rule. It simply activates all neurons in F_2 when an input pattern I is present. These two controls together with the STM reset wave A regulate both the hypothesis testing cycle and the self-stabilization of learning in an ART-1 system.

3.4.3 Problems in ART1 Model

The vigilance parameter ρ tunes the categorical coarseness. Using different vigilance values, the ART-1 network automatically rescales its sensitivity to patterns of variable complexity. However, choosing the value of ρ is critical, since it has significant effects on the number of pattern categories. For example, if ρ is too high, most patterns will fail to match those in storage and the network will create a new category for each of them. In other words, ART-1 stores all patterns it encountered into different categories. On the other hand, if ρ is too low, different patterns will be grouped together, distorting the stored patterns [Wass89, HuYK90, KilL90]. So far, there is no theory to guide the correct setting of the vigilance parameter. Trial-and-error is still the common method used to determine the proper vigilance parameter.

3.5 Summary

Four neural network models, namely BAM, BP, CPN, and ART-1 networks have been discussed in this chapter. Each model is viewed either as a supervised learning feedforward, an unsupervised learning feedforward, a supervised learning feed-back, or an unsupervised learning feedback network. The discussion covers the basic architectures and the weight modification procedures of the models, as well as some limitations or problems encountered in a particular model. For instance, the BAM model, which is a less complex model, suffers from its limited memory capacity and requires mutually orthogonal patterns to achieve a perfect recall, whereas BP suffers from its required enormous computational training time. Also, the CPN and the ART-1 networks have unique problems such as the underutilization problem of the Kohonen learning of the CPN model and the problem with selecting the proper vigilance parameter in the ART-1 model.

This chapter provides a basis for the comparative study. The basic differences on their characteristics may be studied through examining the topology and the weight modification procedure of each model. However, this may not confirm or discover all their capabilities, and particularly, their abilities to solve specific problems. This requires some experiments to be done in the study. The experiments are discussed in Chapter V and VI.

CHAPTER IV

SOFTWARE IMPLEMENTATION

This chapter describes a software simulation of the selected neural network models. An object-oriented design methodology has been used in the design process. The software has been implemented using an extended C programming language (THINK C 4.02 compiler) and ResEdit 2.1 resource editor on an Apple® Macintosh Plus computer with a minimum of 2 Mbyte memory and System 6.0.7. Note that the extended C is a standard C language with additional capabilities for object-oriented programming. The extended C is a subset of C++ programming language [Syma89].

The discussion begins with specification of the requirements followed by a description of the architecture of the system. Verification of the system will be explained afterward. The description of the software structure is achieved through the use of the Uniform Object Notation (UON) [PCWe90], a structure chart-like notation specially designed to model the structure of object-oriented software. The notation provides a comprehensive picture of the software's element interactions without the source code details. A complete listing of the source code is presented in a technical report [InKi91].

4.1 Specifications

The design process starts with specifying the requirements of the software. The first issue to address is software flexibility. The software has to be flexible enough to implement different kinds of neural network models. This calls for reusable components in

the software. Next, the software has to be implemented on a desktop computer such as the IBM PC (compatible) or Apple Macintosh. Another requirement is a good user interface for the software. A graphical user interface (GUI) is preferable, since it facilitates the user in examining the states of the neural network model under study. To achieve this, some examination tools are necessary. In the design stage, it is usually difficult to list all the tools needed for the study. Therefore, it is necessary to design a software that is maintainable. Note that the maintainability issue is also a requisite for designing a good program [DaMa88, Page88, Sodh90].

4.2 Design Methodology

Programming a graphical user interface (GUI) using a procedural-oriented programming language is difficult. Object-oriented programming reduces the complexity of a GUI by *encapsulating* [Krae89] standard windowing behaviour into predefined *objects* [Urlo90]. Different from the procedural-oriented approach, the object-oriented approach decomposes a system using the concept of an *object* [Sodh90]. Every *object* belongs to a *class*, which defines the implementation of a particular kind of *object* [Syma89]. The *object* contains data and procedures to manipulate that data. The data describe the local state of an *object* and are only accessible to the outside world through the object's procedures, called *method*. This characteristic ensures data encapsulation. Note that the *method* will be invoked by another *object* through the use of a *message*.

In the object-oriented approach, a new *class* may be defined through deriving an existing *class*. The technical term for defining a new *subclass* from the existing *class* is *inheritance*. The *inheritance* feature facilitates development of maintainable software which reusable components [Holl90]. Following specification of the requirements, with underlining the software reusability and maintainability issues, it seems that developing the

software using the object-oriented approach will meet such requirements. Therefore, it is preferable to choose the object-oriented approach as the design methodology.

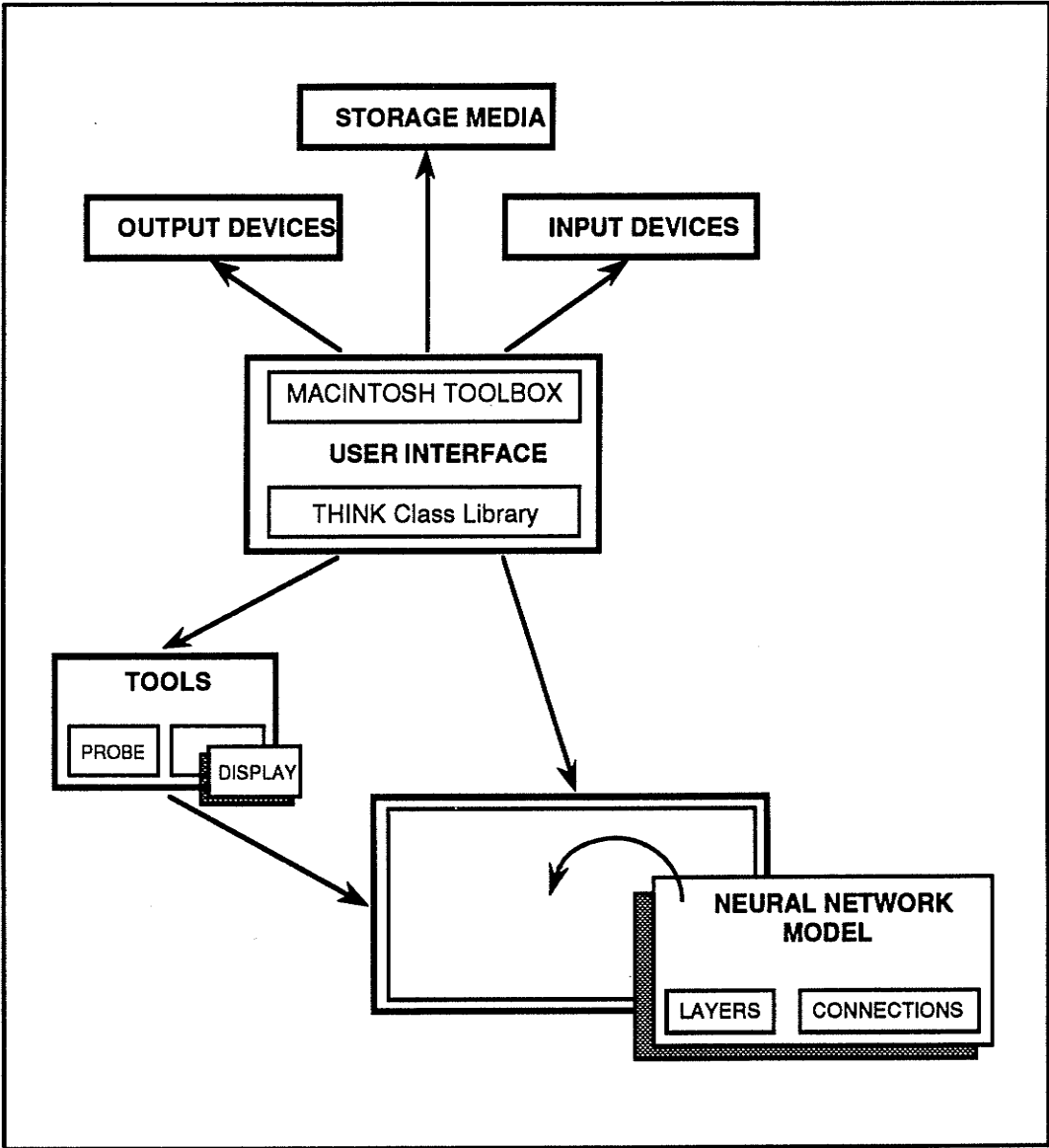


Fig. 4.1. Block diagrams of the main modules. The neural network module and the tool module can be replaced by similar modules.

4.3 Architecture

The software comprises of three main modules: the user interface module, the neural network module and the tool module, which are illustrated in Fig. 4.1.

The user interface module is the center of the program. Any task relating to a user command is managed by the user interface module. Through this module all other modules are connected to the user. This module also functions as an interface between the neural network module and other peripherals such as display windows and storage devices. This module has been developed through extensive use of the encapsulation feature of the object-oriented approach.

The critical part of the software is the neural network module. In this module, a specific neural network learning algorithm is implemented. Since more than one different models must be implemented, the *inheritance* feature has been intensely used to produce reusable components. To facilitate the experimentation, the software also includes some examination tools. These tools are implemented in the tool module. Furthermore, a text editor has also been embedded into the software as a part of the tools. A detail description of each module will be discussed in the next section.

4.3.1 User Interface

The user interface module consists of two parts: the Macintosh Toolbox and the THINK Class Library (TCL). The Macintosh Toolbox is a collection of functions of the standard Macintosh GUI. These functions operate as an interface between the application program and the operating system of the Apple Macintosh computer. A complete reference of the Macintosh Toolbox can be found in [App188]. The second part, the TCL, comprises of several objects that implement the entire Macintosh interface. It takes care of things like handling menu commands, updating windows, dispatching events, dealing with

MultiFinder, maintaining the Clipboard, and so on [Syma89]. With the help of the TCL, it is much easier to develop a standard Macintosh application, since there is no need to implement all the details of the GUI. The TCL is provided as a part of the THINK C development tool. It is organized into three distinct, but interacting structures: the class hierarchy, the visual hierarchy, and the chain of command.

The class hierarchy is a collection of all classes that make up the TCL. It describes the relationships among all the classes. All the classes are descendants of the root *class* *CObject*, and each descendant *class* inherits all the characteristic of its predecessor. Following the TCL class name notation, all *class* names begin with the letter 'C' for 'class'. Fig. 4.2 shows the class hierarchy (for convenience, the leading letter 'C' in each class name is omitted).

The visual hierarchy describes the organization of all visible entities. It is built around the idea of *enclosures*. At the top of the visual hierarchy resides the *desktop*. The *desktop* encloses all the windows in the application [Syma89].

The chain of command specifies which objects must handle commands. The chain of command is based on the idea of *supervisors*. If an object cannot handle a command, it passes the command on to its *supervisor*. The chain of command and the visual hierarchy receive *messages* from an object named *CSwitchboard*, which receives events from the Macintosh Event Manager and translates them into *messages*. Note also that the name of a *class* is usually used to indicate an *object* instantiated from that *class*.

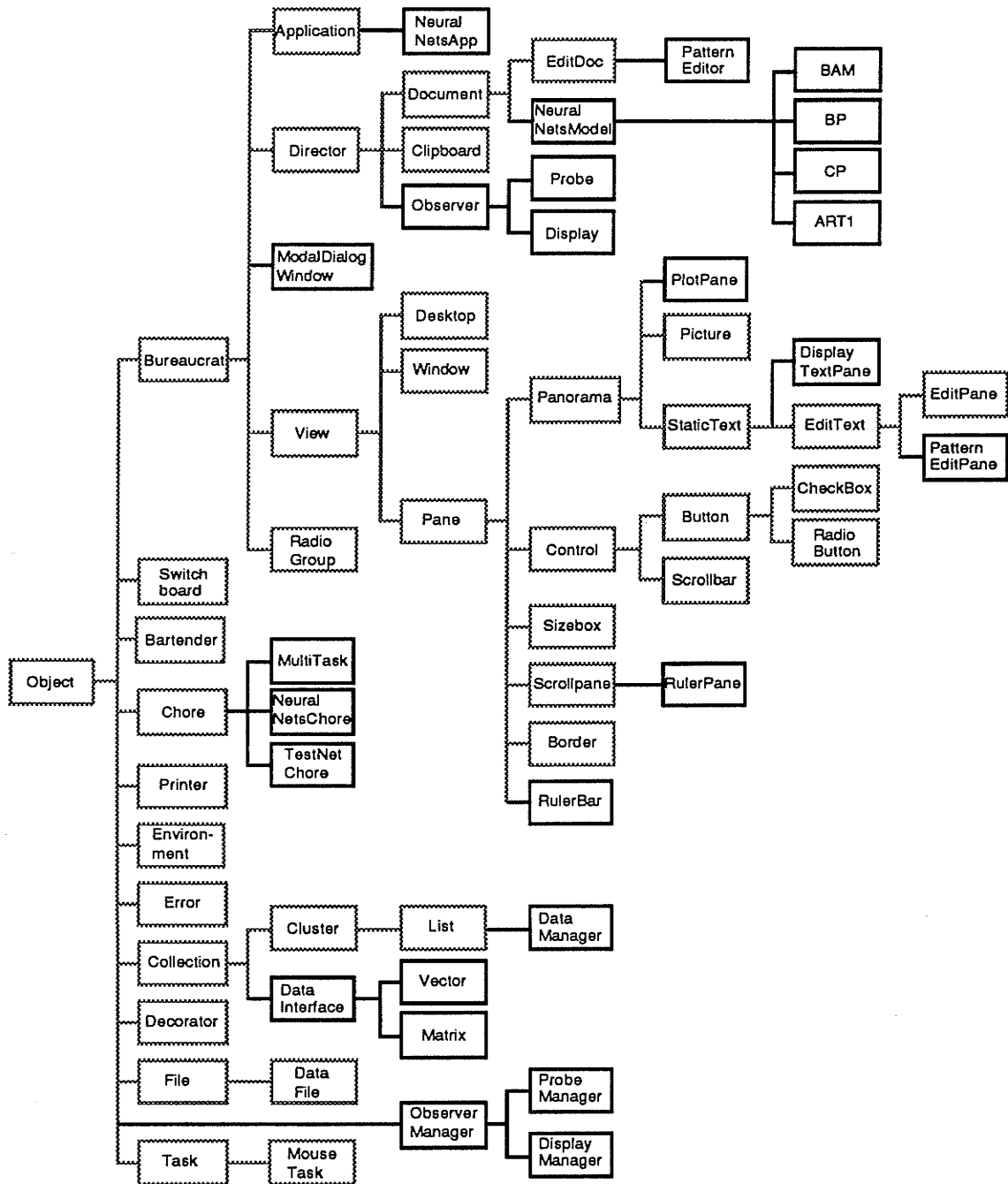


Fig. 4.2. The software class hierarchy. The dashed block diagrams represent all the TCL classes, whereas the solid block diagrams represent the application classes.

The TCL environment makes the connections between the application module and the user interface module much simpler. In the TCL, a unique object named *CApplication* functions as a mediator. Through this object, some objects in the user interface module can communicate with objects in the application module. This feature also offers an easy way to embed the user interface module into the application module. Notice also that *objects* communicate through sending *messages* [Syma89, Krae89, Mull89, and Mark90].

For our purpose, a new class called *CNeuralNetsApp* is inherited from the *CApplication* class. The *CNeuralNetsApp* object connects the user interface module to the neural network module as well as to the tool module. To get the picture of these interactions, it is preferable to show them in an object-cooperation diagram, as shown in Fig. 4.3. Note that the object-cooperation diagram is a notation to facilitate the object-oriented design methodology, similar to a structure chart. The object-cooperation diagram is a part of the Uniform Object Notation introduced by Page-Jones et al. [PCWe90].

Figure 4.3 shows a communication path between the *CNeuralNetsApp* object and the *CNeuralNetsModel* object as well as some communication paths between *CNeuralNetsApp* object and the tool module objects such as *CProbe*, *CDisplay*, and *CPatternEditor* objects. It also shows interactions between tool module objects, represented by *CProbe* and *CDisplay* objects, and neural network module objects, represented by *CMatrix* and *CVector* objects. Figure 4.3 can be seen as another detailed description of some interactions among the main modules, as shown in Fig. 4.1. However, some connections with the other parts of user interface are not displayed since all of those connections are achieved through the *CNeuralNetsApp* object. This kind of interaction is well explained through using an object-communication diagram, shown in Fig. 4.4. Notice that the object-communication diagram is also a part of the Uniform Object Notation [PCWe90].

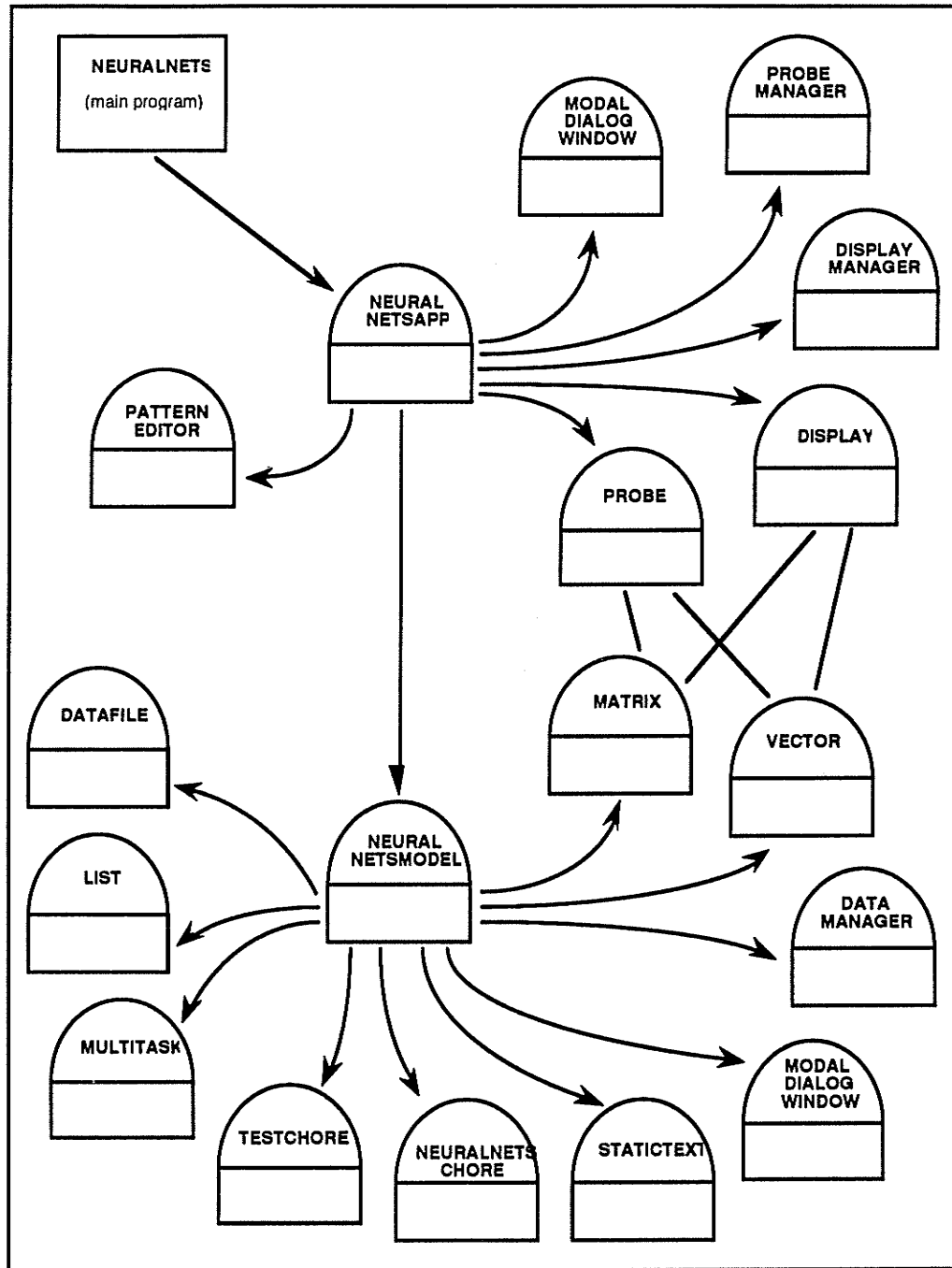


Fig. 4.3. Object-cooperation diagram showing interactions among objects in the main modules.

Following the C language procedure to begin the program execution, the program uses the function called *main()* as the first module [DaMa88]. The *main()* function of the program, as shown in Fig. 4.4, creates a *CNeuralNetsApp* object and initializes the object. During initialization, the object also sends some initialization messages to its superclass object, the *CApplication* object, and to other relevant objects. After initialization the *main()* function sends a *Run* message to *CNeuralNetsApp* object. Since this message is implemented by its superclass' method, a *CApplication* class name is used instead [PCWe90].

The *CApplication* object, through the *Run* method, sends a *ProcessEvent* message to *CSwitchboard* object repeatedly. This scheme performs a *main event loop*, since the *CSwitchboard* object calls the Macintosh Event Manager to get *events* and then translate them into messages. These messages are then sent to the relevant objects in the chain of command. Note that the *main event loop* is the "heart" of every application program running on the Machintosh [Appl88]. A method-structure diagram showing detailed structure of the *ProcessEvent* method of the *CSwitchboard* class is given in Fig. 4.5. Notice that the method-structure diagram is another part of the Uniform Object Notation [PCWe90].

The program remains in the *main event loop* until a "Quit" command from the menu is chosen by the user. This command causes an interruption in the loop. Then, control is returned to the *main()* function. Subsequently, the *main()* function sends an *Exit* message to *CApplication* object to stop the program. This *Exit* message is required to do some final tasks such as stoping the neural network learning process, closing files, and freeing some allocated memories, before terminating the program. The whole program ends after the *Exit* message is executed.

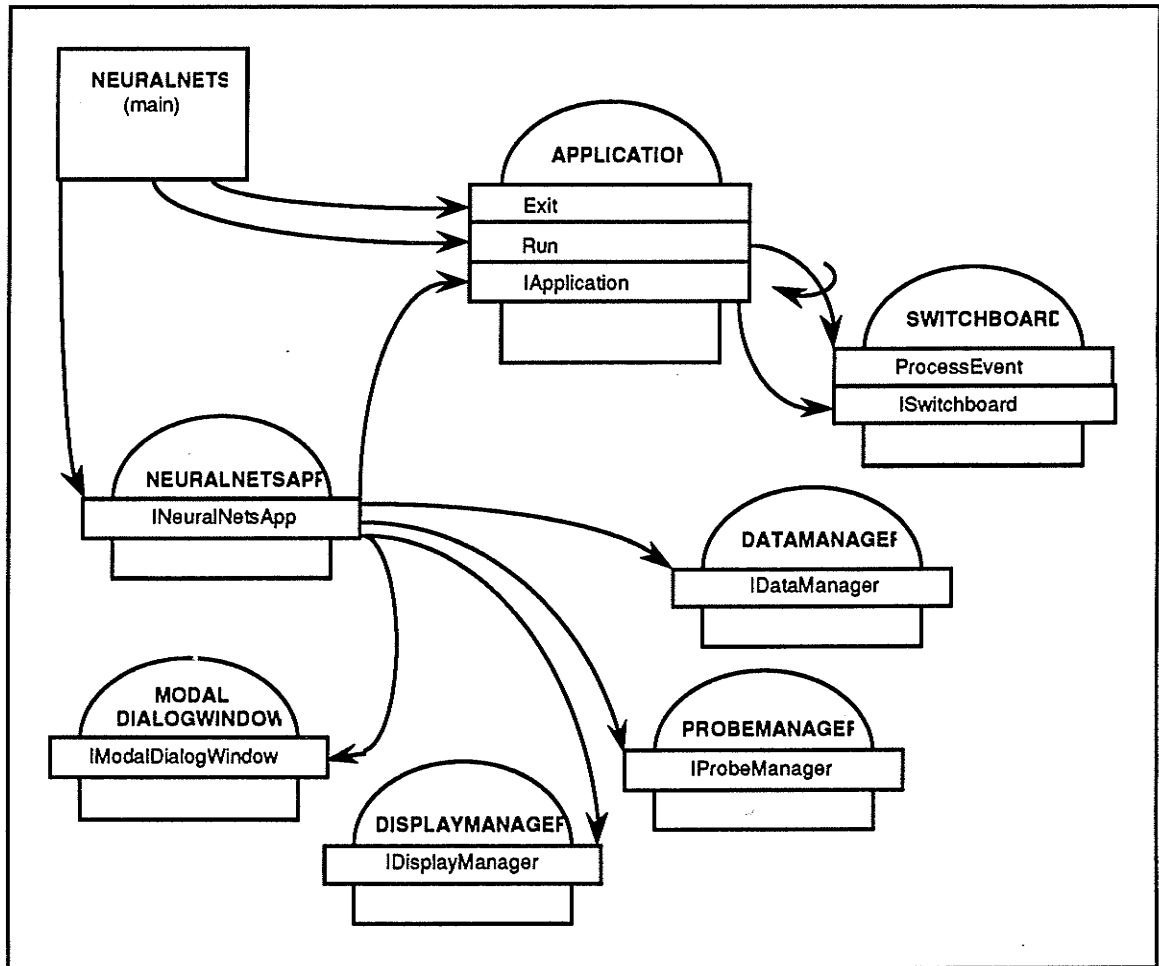


Fig. 4.4. Object-communication diagram showing the *main event loop* of the program.

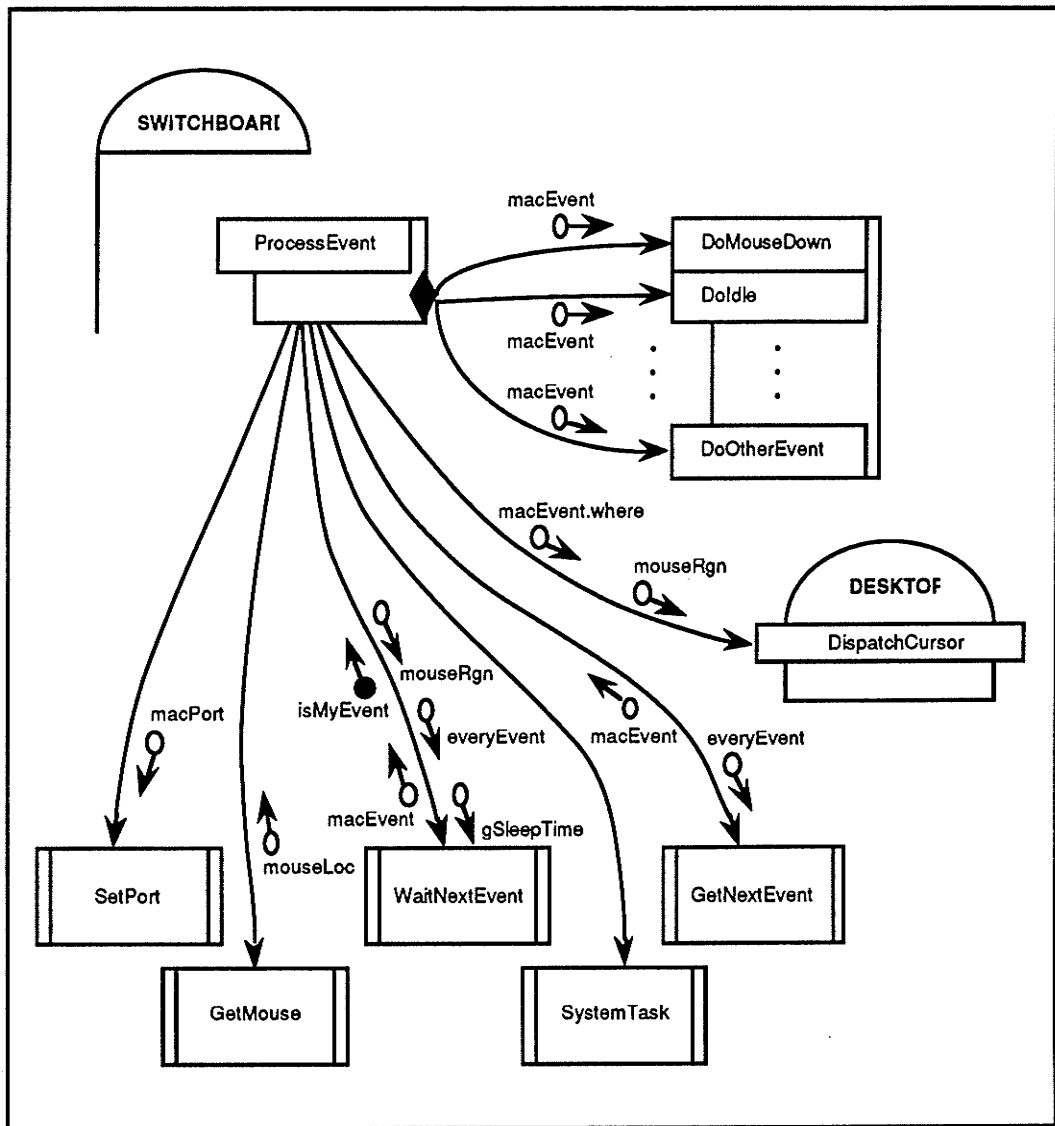


Fig. 4.5. Method-structure diagram showing the *ProcessEvent* method of the *CSwitchboard* class.

4.3.2 Neural Network Model

Each neural network paradigm has a unique network topology and learning algorithm. Nevertheless, they share some common features such as employing layers and weights in their networks, and involving learning and testing processes. These common

features are good candidates to obtain a general class, which is necessary to produce a reusable component [Mull89].

In object-oriented programming, a general class may be presented as an *abstract* class. Note that an abstract class is not truly complete enough to operate as an independent entity. It only serves to group together the member functions and data elements that are common to all of its subclasses [Mull89]. Therefore, each neural network model has to be implemented as an object of a subclass derived from this abstract class. The abstract class ensures a uniform interface for all the neural network model objects.

Every neural network model is an object of a unique subclass derived from an abstract class named *CNeuralNetsModel*. Each subclass will only implement specific tasks to a model. All common features such as the initialization process, learning and testing schemes, input/output data handling, and any communication interface to the user interface module are implemented in the *CNeuralNetsModel* class. However, this class is not working alone. There are several other objects that handle specific jobs, which are assigned by the *CNeuralNetsModel* class. This kind of team work has been previously illustrated in the object-cooperation diagram of Fig. 4.3.

The figure also shows interactions between some particular objects in the neural network module, namely *CMatrix* and *CVector* objects with objects such as *CProbe* and *CDisplay* in the tool module. Since the *CProbe* and the *CDisplay* objects are used to display the states of the *CMatrix* and the *CVector* objects, which always change during the learning process, it is preferable to have their own communication path. This establishes a direct link between them. Notice that the *CMatrix* and the *CVector* objects are employed to implement the weights and the layers of a network, respectively.

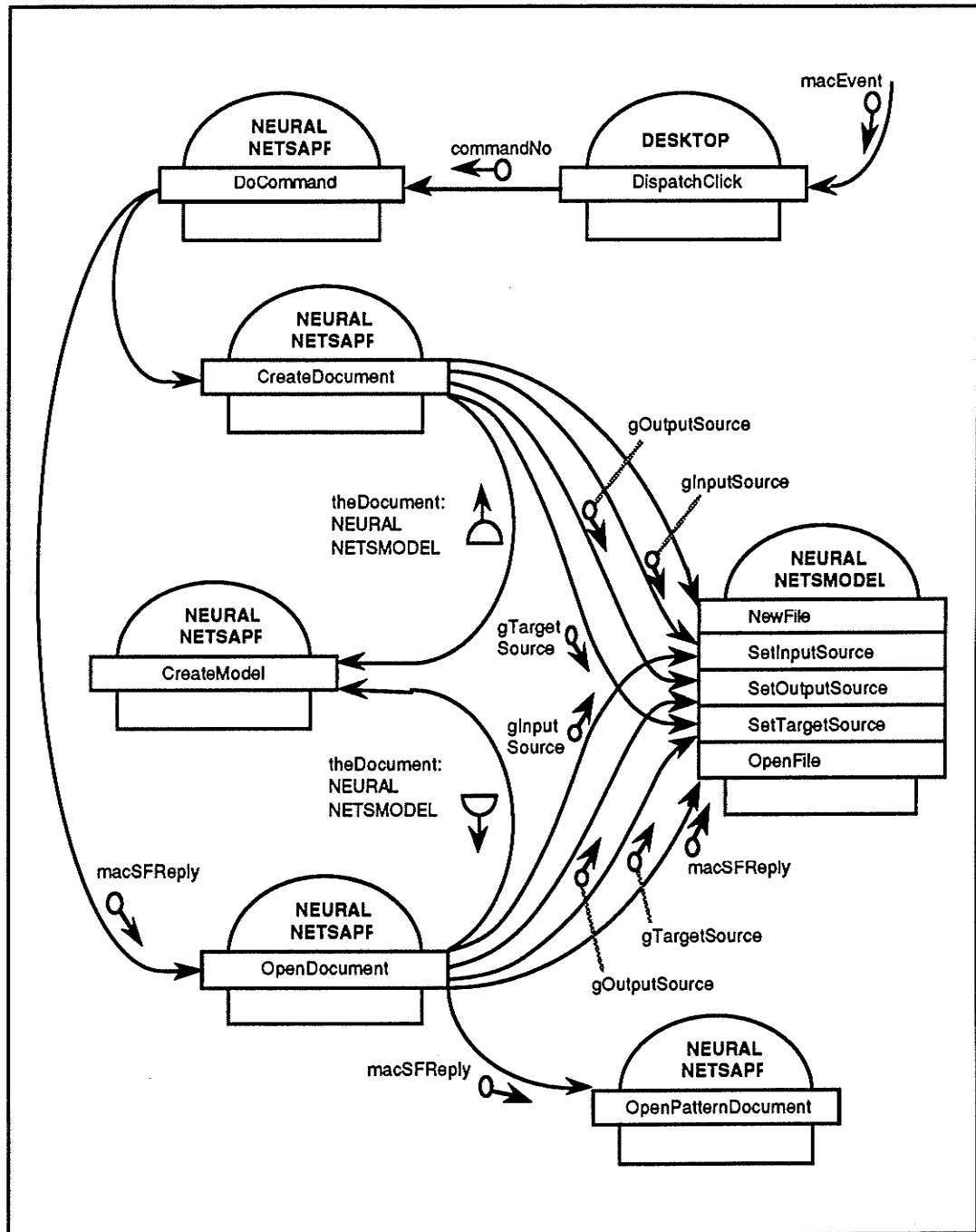


Fig. 4.6. Object-communication diagram showing the process of creating and initializing a *CNeuralNetsModel* object.

It is easier, perhaps, to describe the neural network module by showing how the module works. To begin, assume that a *CNeuralNetsApp* object has been created by the *main()* function, and the program is now in its *main event loop*.

The “tale” begins when the user chooses a “New” command or an “Open” command to create a model. The *CNeuralNetsApp* object receives a message relating to that command from the *CSwitchboard* object. Consequently, a new *CNeuralNetsModel* object is created. The *CNeuralNetsModel* object is assumed to be a normal object that can operate independently. In a real situation it will be replaced by an object of a subclass derived from it. The process also involves some initialization. An object-communication diagram showing this process is presented in Fig. 4.6. At this point, the user is able to modify either the topology of the model or the parameters of network. Since every model has its own architecture and learning algorithm, any task pertaining to a specific feature of a model is implemented in a subclass derived from the *CNeuralNetsModel* class.

The next step, prior to the learning process, is the initialization of the weights. For a certain model, all the weight values are initialized with some random values, and for others, the weights are initialized using small fixed values. The initialization option is left to the user. The learning process starts when the user selects a “Start Learning” command. Similarly, a message pertaining to the command is sent by the *CSwitchboard* object to the *CNeuralNetsModel* object. The *CNeuralNetsModel* object will then do some preparation for the learning process. A method, called *StartLearning*, completes the task through sending some messages to other objects. Notice that this is not the method that implements the learning algorithm. The learning algorithm is implemented in a method called *DoLearning*, which is discussed later. The *InitConnections* method, which implements the initialization process, and the *StartLearning* method are illustrated in Fig. 4.7.

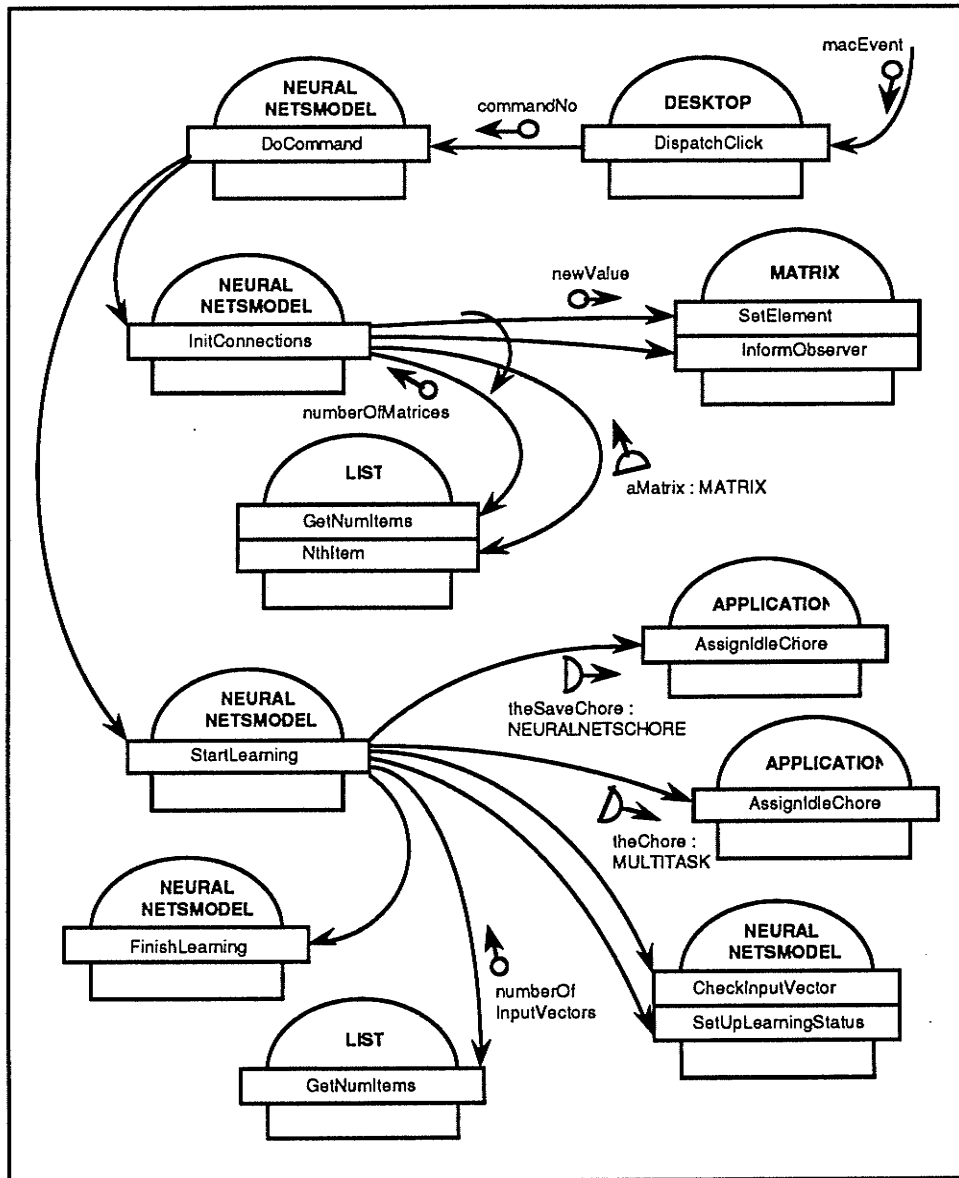


Fig. 4.7. Object-communication diagram showing the weights initialization and the start learning events of the *CNeuralNetsModel* object.

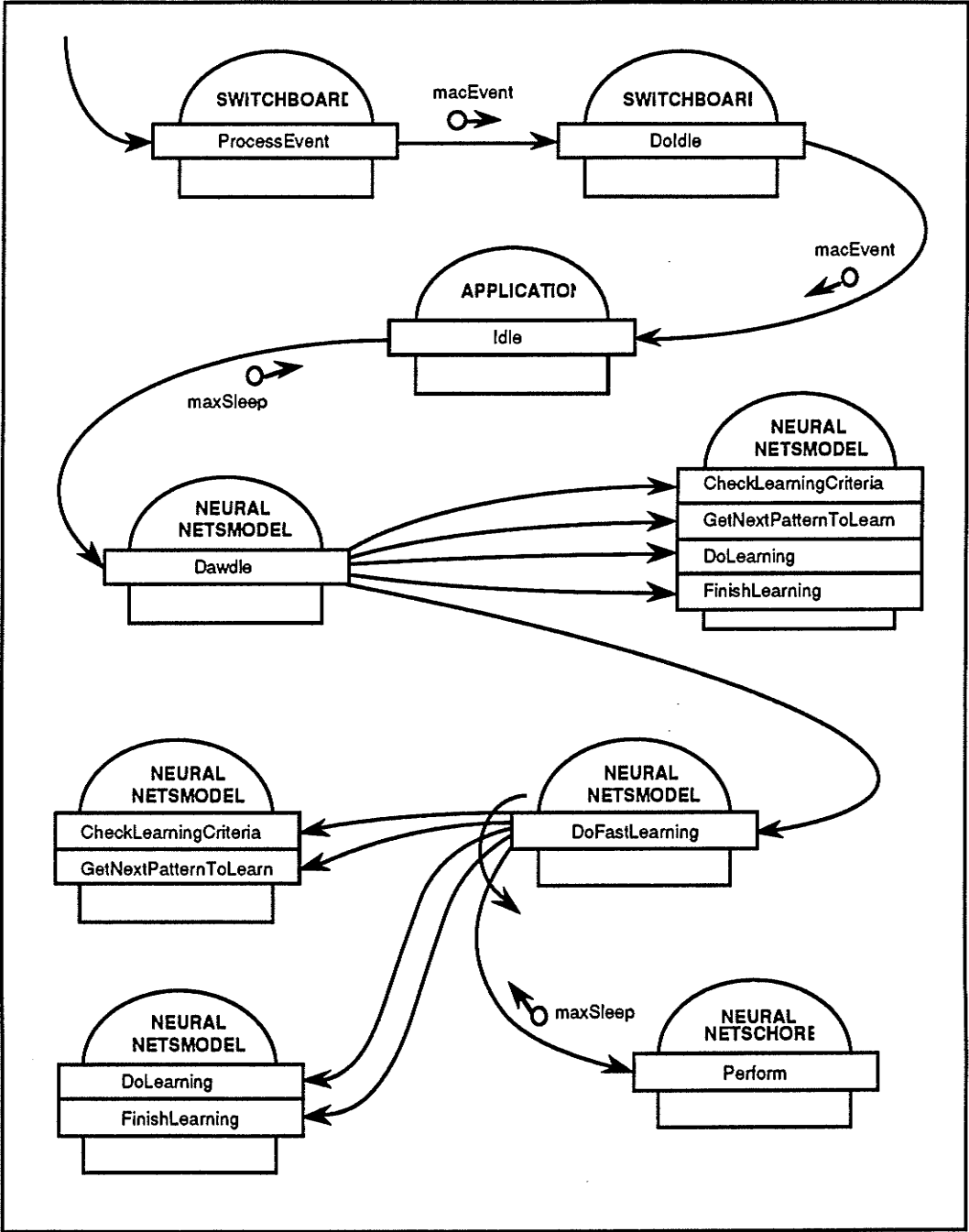


Fig. 4.8. Object-communication diagram showing the interaction of many objects during a learning session.

Each neural network model has its own learning algorithm. This commonality leads us to define a general message to do the learning. The technical term for this is *polymorphism* [Krae89, Syma89, Mull89]. For this purpose, a method named *DoLearning* has been defined in *CNeuralNetsModel* class to implement the learning algorithm. However, since a learning algorithm is unique for a neural network model, each subclass will *override* this method. Overriding a method means that the subclass responds to the same message as its superclass, but it uses its own method to respond to the message [Syma89]. For the *CNeuralNetsModel* class, since it is not implementing any specific model, the *DoLearning* method in this class will do nothing.

Fig. 4.8 shows some interactions among objects during execution of learning. A special method named *Dawdle* in the *CNeuralNetsModel* class will send messages to itself. The messages will be handled by related methods. The *Dawdle* method will be invoked by another object, namely the *CApplication* object, during idle time, that is, when there is no input/output task to handle. Using this kind of a scheme, the learning process can be performed as a background process (i.e., the program can do multitasking). In other words, more than one neural network model or other applications operate together simultaneously. This behavior can be realized through some help from a special object, *CMultiTask*, which is an object of subclass derived from the *CChore* class. Fig. 4.9 and Fig. 4.10 show the object-communication diagrams.

To prevent unfair time sharing, the *DoLearning* method must process only one pattern at a time. In other words, the corresponding learning algorithm is implemented so as to learn one pattern only. This is the main reason to send a *GetNextPatternToLearn* message prior to sending the *DoLearning* message. If, however, having a faster learning process is more important than implementing a multitasking process, a *DoFastLearning* method can be chosen instead. By enabling the "Fast Learning" command, the *Dawdle*

method will send a *DoFastLearning* message, which then performs an iterative process locally (see Fig. 4.8). The control does not return to the *main event loop* until either all the patterns are trained, or a keyboard/mouse event has been detected. In the fast learning mode, only one neural network model can operate at a time.

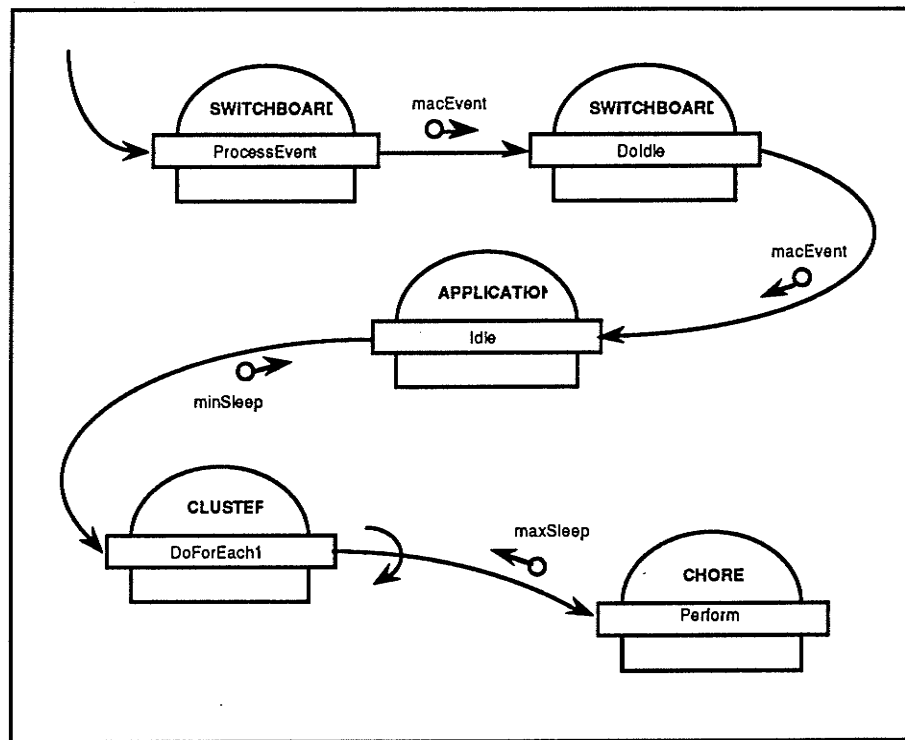


Fig. 4.9. Object-communication diagram showing a multitask event during an idle time.

In response to a “Start Learning” command, a *StartLearning* method sends an *AssignIdleChore* message to the *CApplication* object together with a pointer to a *CMultiTask* object. The *CMultiTask* object will be added into the *CApplication*’s list chore. During idle time, the *CApplication* object sends a *Perform* message to all the *CChore* objects or its inheritances in the list (see Fig. 4.9), which later initiate the learning process.

In a similar fashion, but with an opposite goal, the *FinishLearning* method sends a *CancellIdleChore* message to the *CApplication* object in order to remove the *CMultiTask* object from the *CApplication* list chore. This ends the learning session.

Other methods, which are complementary to one another, are the *StopLearning* and the *ResumeLearning* methods. Those two are aimed to interrupt the learning process for a specified time. Fig. 4.11 gives details of the object interactions.

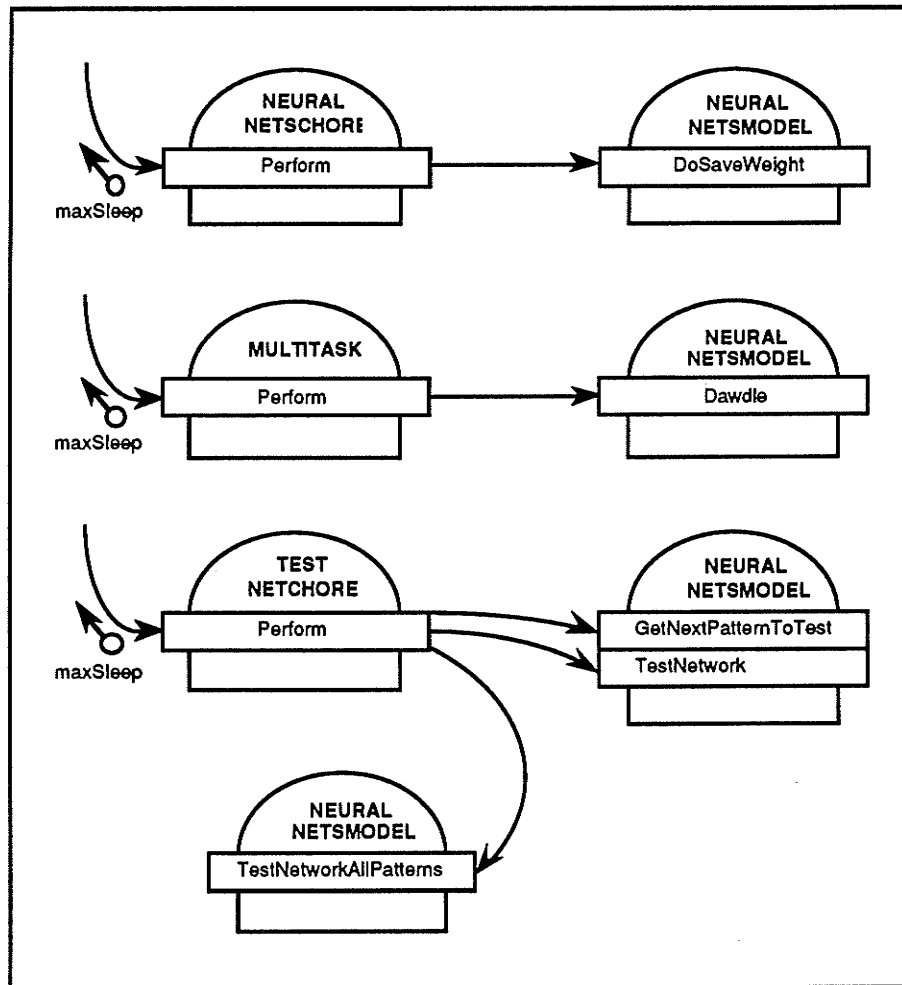


Fig. 4.10. Object-communication diagram illustrating the objects of subclasses of the *CChore* class.

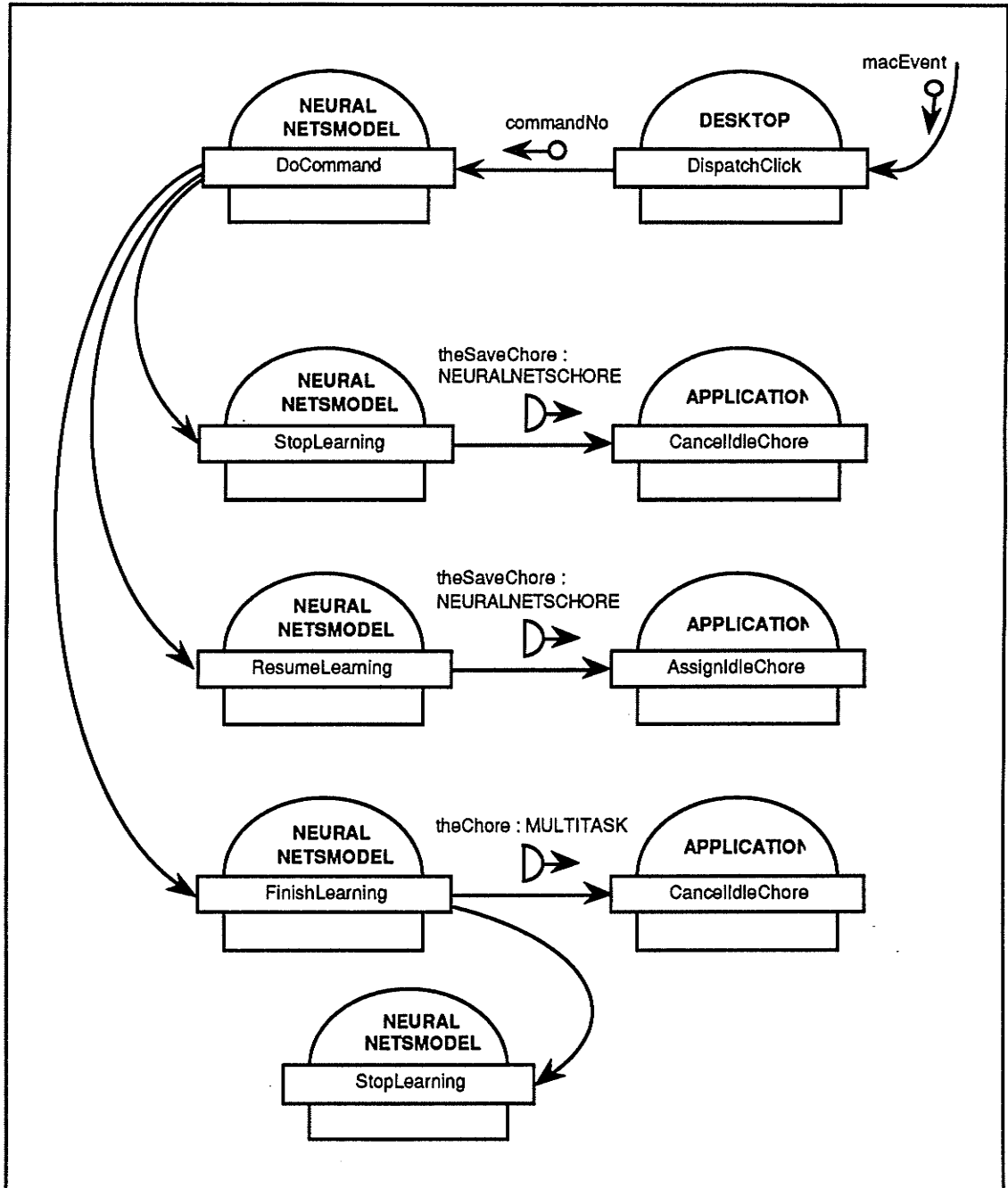


Fig. 4.11. Object-communication diagram showing the stop, resume and finish learning events of the *CNeuralNetsModel* object.

Finally, after a learning session has been completed, a testing session needs to be done. A "Test Network" command yields the *CNeuralNetsModel* object to send a *TestNetwork* message. This command will only process one pattern, which is selected from the menu. To have all patterns in a set processed, a *TestNetworkAllPatterns* message is sent instead. The testing algorithm, which is in some model represented as a forward calculation, is implemented in the *TestNetwork* method. The process is depicted in the object-communication diagram shown in Fig. 4.12.

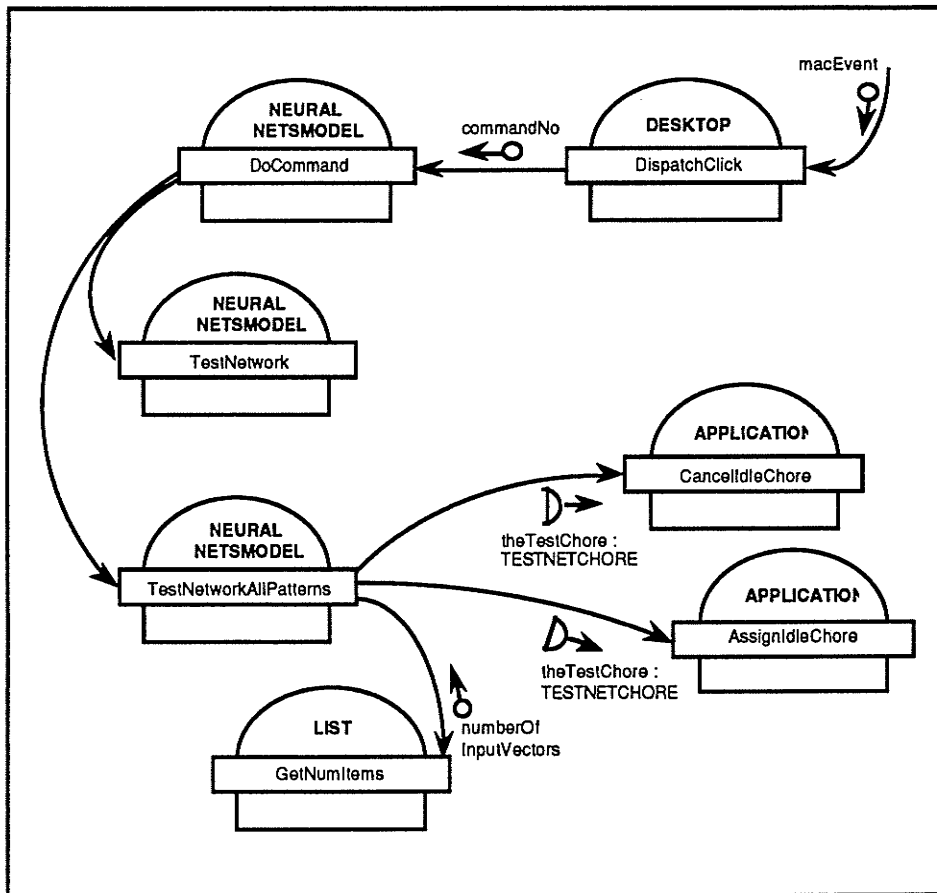


Fig. 4.12. Object-communication diagram showing the interaction of many objects during the testing process.

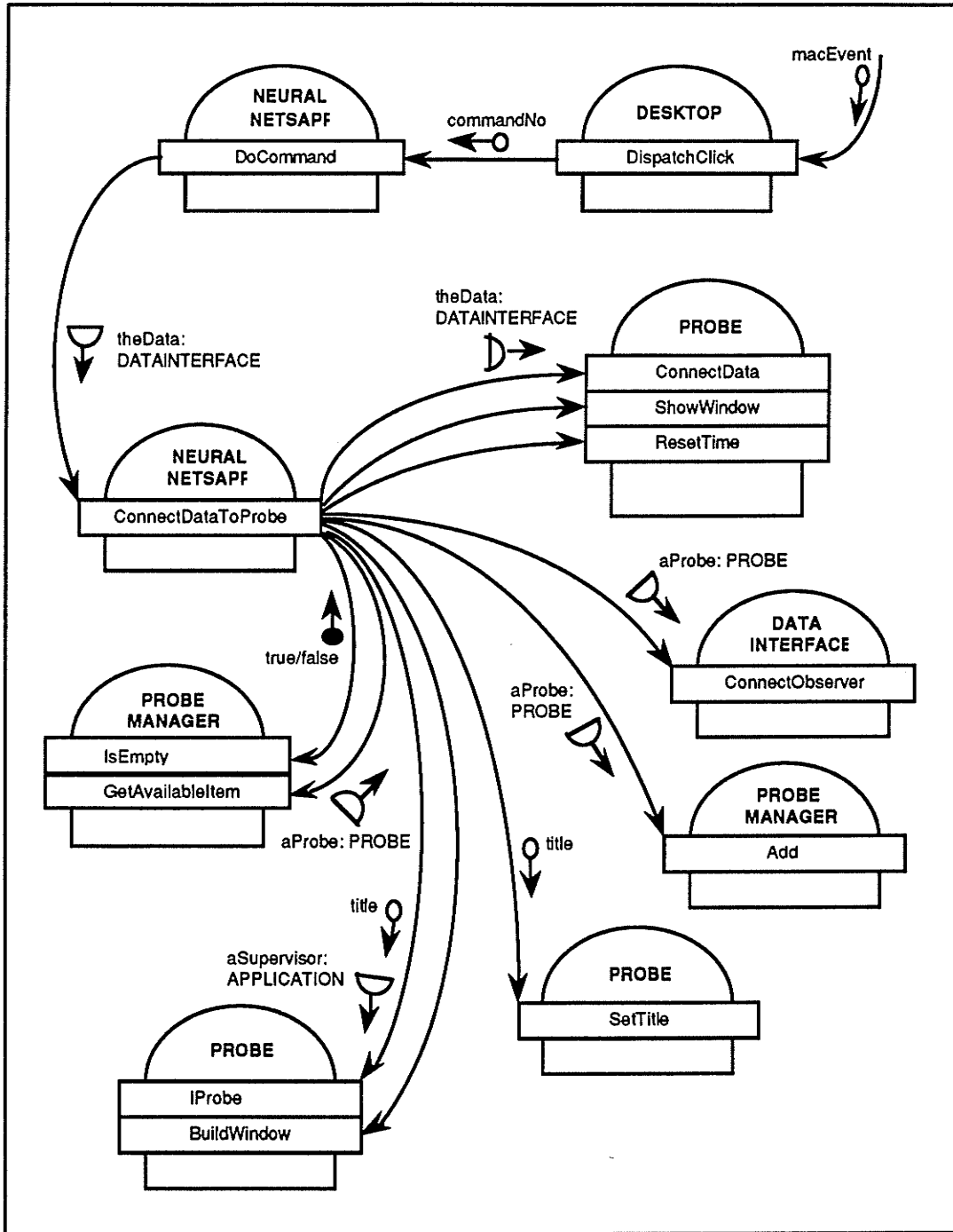


Fig. 4.13. Object-communication diagram showing the process of creating the *CProbe* object and establishing the communication path between the *CProbe* object and the *CDataInterface* object.

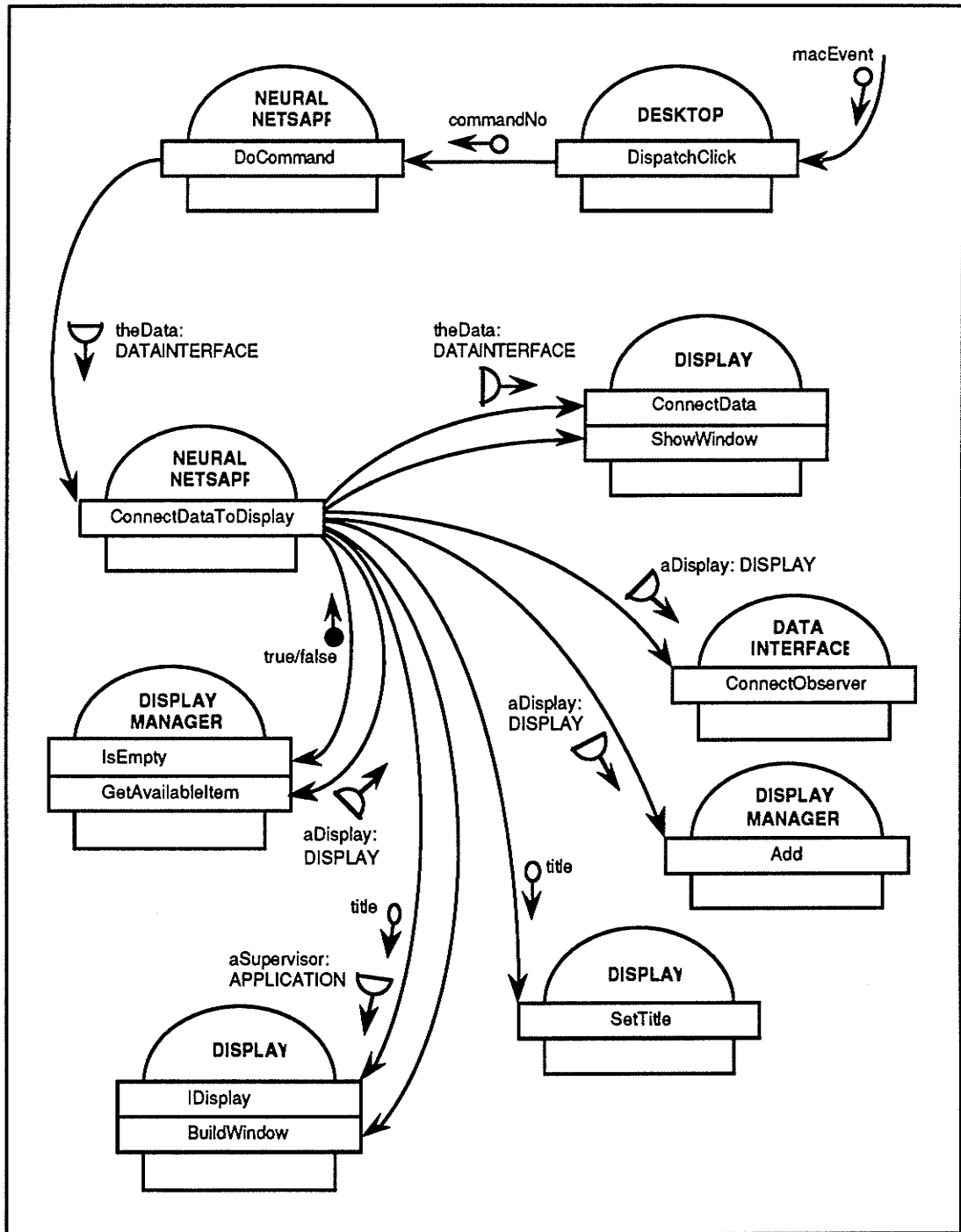


Fig. 4.14. Object-communication diagram showing the process of creating the *CDisplay* object and establishing the communication path between the *CDisplay* object and the *CDataInterface* object.

4.3.3 Tools

Two examination tool objects, namely the probe tool object and the display tool object, have been developed to facilitate the study of neural network models. The probe tool is used to plot several values of a vector or a matrix element within a specified period, for example, to plot the total sum of squared error of training patterns in the backpropagation (BP) model during its learning process. Similarly, the display tool is used to represent the element values of any vector or matrix in the model under study. The interaction between vector or matrix objects and display or probe objects have been discussed in previous section with the help from an object-cooperation diagram in Fig. 4.3. This section describes the process to establish the communication path between vector or matrix objects and display or probe objects.

The probe tool is implemented as an object of a *CProbe* class, which is an inheritance of an abstract class *CObserver*. When the user chooses "Probe" command, the *CNeuralNetsApp* object will receive a command to create a new *CProbe* object, and to establish a communication path between the *CProbe* object and a *CDataInterface* object or its inheritance. Accordingly, the *CProbe* will open a window, and set up a communication path to a selected *CDataInterface* object or an inheritance of it, particularly a *CMatrix* or a *CVector* object. Since each *CProbe* object can only communicate with a selected *CDataInterface* object, it is possible to create more *CProbe* objects to probe different vector or matrix objects. The only factor that limits the number of *CProbe* objects created is the total amount of available memories. The user has privilege to select which vector or matrix object to probe.

In a similar fashion, the display tool is implemented as an object of a *CDisplay* class derived from the *CObserver* class (see Fig. 4.2). The *CNeuralNetsApp* object will create a *CDisplay* object and establish a communication path between the *CProbe* object

and the *CDataInterface* object. Since the *CDisplay* class is derived from the same abstract class of the *CProbe* class, that is the *CObserver* class, basically, it has the same behaviour with the *CProbe* class. Fig. 4.13 and 4.14 show the process of creating the *CProbe* and the *CDisplay* objects, respectively.

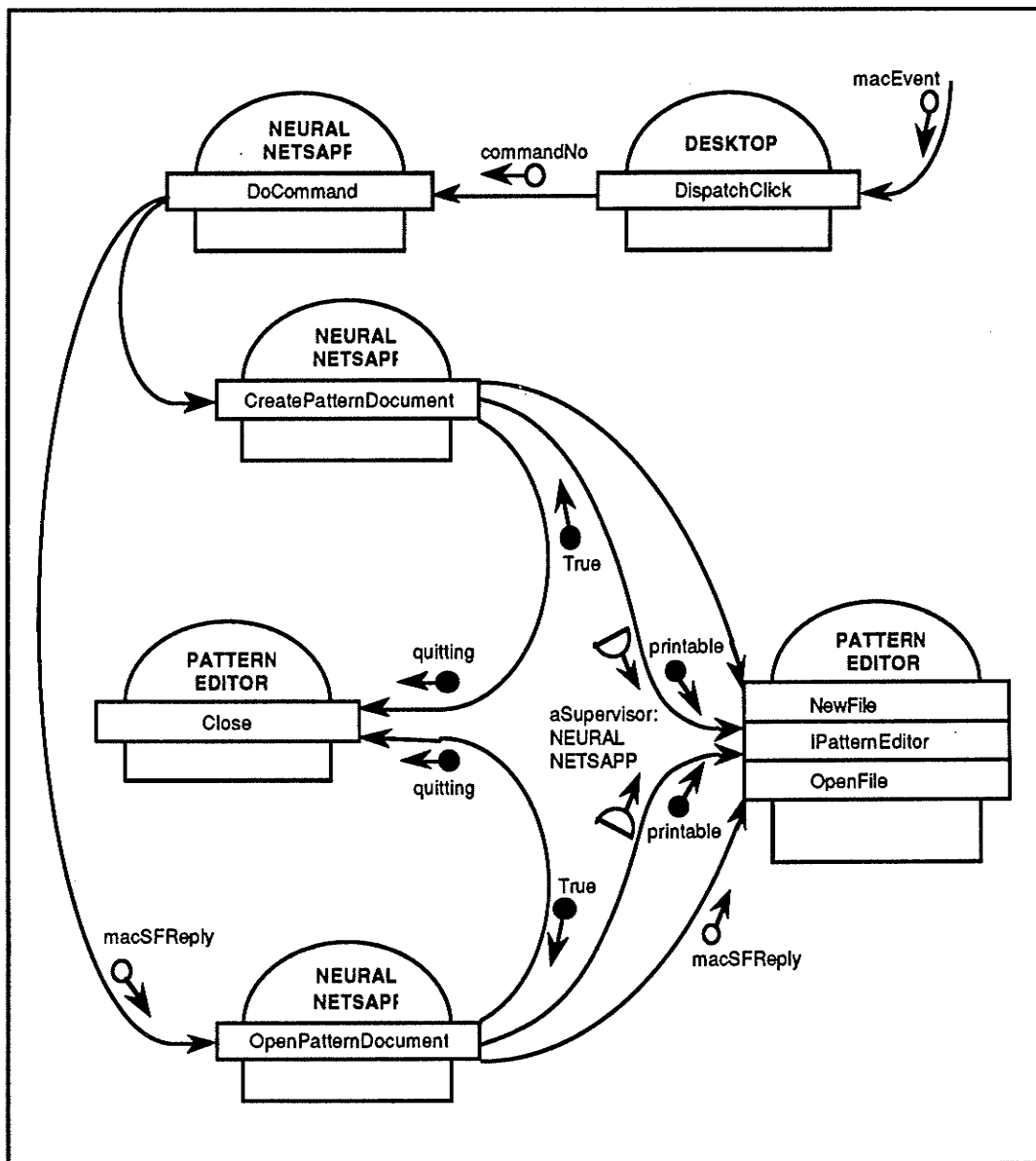


Fig. 4.15. Object-communication diagram showing the process of creating and initializing the *CPatternEditor* object.

Another useful tool to help preparing the input/target pattern sets is the pattern editor. Since the program will read the pattern data as a vector, the pattern editor tool that implemented as a *CPatternEditor* object is used to transform text data into vectors. Each input/target pattern is represented as a vector. The tool creates a list of pattern vectors and send it to the network. The network gets a pattern vector from this list during learning or testing process. An object-communication diagram showing the process to create a *CPatternEditor* object is given in Fig. 4.15.

4.4 Verification

A product is only good as its test system [DaMa88]. Therefore, it is necessary to test and to verify the software. Though, the process of testing and verifying software system is a discipline in its own right.

For our purpose, there are two verification processes: the software verification and the neural network model verification. The software verification is meant to verify the entire program with respect to its specification, whereas the neural network model verification is intended to verify the implementation of the model.

4.4.1 Software Verification

Software verification can be done in two ways: (i) single module verification and (ii) modular integration verification. The software component verification is done during the development process. This verification is done in a modular level. Each module is tested and verified during the development of the software. The process, however, only verifies the modules independently. It is only good for checking the syntax errors and the logical connection within a module.

The modular integration verification can only be done after the whole program has been completed. This is done in the system verification. For this kind of testing, it seems that the best way is to use the program. The verification process at this level is called *alpha test*. The *alpha test* consists of testing performed by the developer for the express purpose of turning up any bugs in the final product [DaMa88]. In this stage, there is no more new code implemented unless it is intended to fix the bugs.

4.4.2 Neural Network Model Verification

The model verification is necessary to ensure that there is no error in the implementation of the neural network models. An error appearing at this stage is no longer a software problem, but a problem in understanding the learning algorithm. This kind of problem cannot be detected in the software verification process, since there is nothing wrong with the coding. Instead, results obtained from the model simulation are not valid.

To do this kind of verification, one can use the same test problem that has been used by the author of the model in their original publication. Then, the verification process is simply an inconsistency test, that is, the simulation result of the implemented model has to show similar or nearly similar result as shown in the original papers, given a particular problem used in that publication. If an inconsistent result occurs, then the software needs further examination. Otherwise, the learning algorithm is assumed to be properly implemented. Another approach is to test the systems with known patterns and assess the quality of the results.

4.5 Summary

This chapter discusses the development process of the neural network software. An object-oriented design methodology has been used to design the software structure.

Throughout the discussion, a Uniform Object Notation has been utilized to model the system. The program is a software implementation of the four neural network models under study. The program is also equipped with a probe tool, a display tool, and a text editor. The entire program is implemented using an extended C language, which supports the object-oriented programming, on the Macintosh computer. An example of the neural network simulator's windows is shown in Appendix C.

CHAPTER V

ASSOCIATIVE MEMORY EXPERIMENT

The pattern associator is an example application of artificial neural networks. In this example, a network is used to store pair-data associations. In other words, the network is trained to associate pairs of patterns. Ideally, using this scheme, a stored pattern can be recalled completely from an incomplete pattern (*autoassociative memory*) or from a different pattern (*heteroassociative memory*) that is associated to it. This chapter studies the behaviour of selected artificial neural network models implemented as pattern associators. However, since the associative memory task requires pairs of labeled patterns (i.e., a particular output pattern is associated to a particular input pattern), only the models with supervised learning are involved, namely BAM, BP, and CPN models.

There are two different associative memory experimentations: the autoassociative experimentation and the heteroassociative experimentation. The autoassociative experiment is divided into three individual experiments. Each individual experiment stores different number of associations (pairs of patterns), such as two associations, three associations, and four associations. Similarly, the heteroassociative experiment is divided into two individual experiments, namely two-association experimentation and three-association experimentation.

In this chapter, the network configurations and the parameter settings of the models are discussed along with the training and testing results of each individual experiment. The test results for each individual experiment are shown as a list in a table. In this table, the test

patterns and the output patterns of the networks are depicted in 7×5 binary pixel projection. The results of the experiments are discussed after the experiment section, and the summary of the chapter is presented at the end of the chapter.

5.1 Pattern Sets

The patterns used in the experiments are 7×5 arrays of binary pixels representing the alphabet characters *A* through *J* and letter *L*. The first ten characters (i.e., letter *A* to letter *J*) are the same patterns used in [HuYK90] and [KiIL90]. These characters are grouped into several training sets consisting of two, three and four associations. Notice that this kind of grouping is used because of the memory capacity limitation of the BAM model. According to McEliece [McEl87] (see Eq. 3.6), a network with 35 input neurons and 35 output neurons (7×5 binary pixels) can only store up to 3 pairs of patterns to recall all the patterns perfectly. However, not every possible combination is used. The selection of patterns in a group is determined according to the Hamming distance between two patterns (see Eq. 3.9), the number of ON bits (1s) in the patterns, and the number of similar and distinct bits of the patterns. For example, letter *I* and letter *J* are grouped together as a training set in the two-association experiment because they have the smallest Hamming distance (i.e., 2 bits). However, letter *E* and letter *G*, which have the same Hamming distance with *I* and *J*, are also used because *E* and *G* have the same number of 1s (17 bits) whereas *I* and *J* have different number of 1s (*J* is a subset of *I*). The projections of these four letters represented by 7×5 binary pixels are shown in Fig. 5.1. For convenience, we use $H_d(I, J)$ to denote the Hamming distance (the total number of distinct bits) between *I* and *J*, I_{1s} to denote the total number of ON bits (1s) of a pattern *I*, and $I_{1s} \cap J_{1s}$ to denote the total number of similar ON (1) bits (intersection) between *I* and *J*. For simplification, $I_{1s} \cap J_{1s}$ is also written as $I \cap J$.

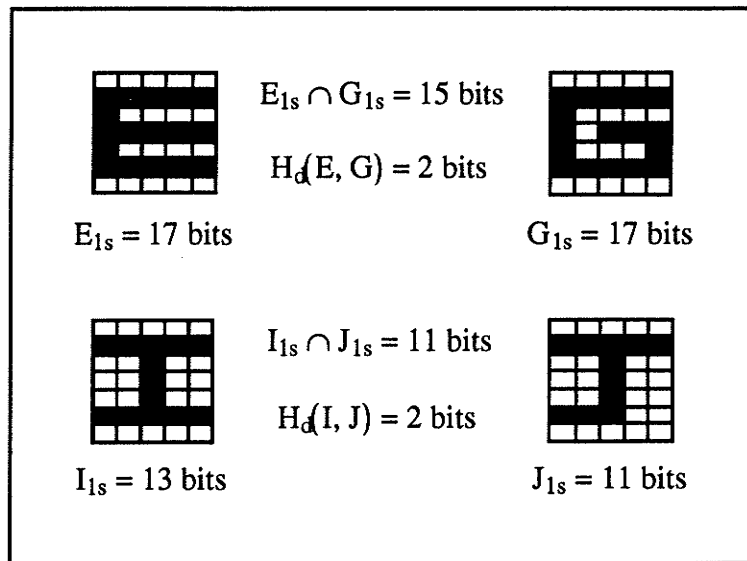


Fig. 5.1. 7x5 binary pixel projection of E & G and I & J

The training sets consist of 7 two-association sets and 4 three-association sets used in the autoassociative experiments, and 6 two-association sets and 2 three-association sets used in the heteroassociative experiments. Besides these training sets, 2 four-association sets are also used in the autoassociative experimentation. The aim is to study the performance of BAM when it is used to store more associations beyond the suggested limit. Notice that this memory limitation is less critical for BP and CPN since they have larger memory capacity than BAM (i.e., the memory capacity of BP without a hidden layer is twice the number of its weights [Nils90], while the memory capacity of CPN is equal to the number of its hidden neurons [Hech89]). All of these pattern sets are used as the training sets. The list of the training sets is given in Table 5.1. For convenience, we use S_{AB} to denote the set of two associations of characters A and B (i.e., $S_{AB} = \{(A, A), (B, B)\}$), and S_{AB-IJ} to denote the set of two characters A and B that are respectively associated with characters I and J (i.e., $S_{AB-IJ} = \{(A, I), (B, J)\}$). For testing purposes,

the incomplete or noisy versions of those patterns are used as inputs to the networks in the recalling process. The complete training patterns can be found in Appendix A.

Table 5.1. Training sets for the associative memory experimentation.

Autoassociative			Heteroassociative	
2-association	3-association	4-association	2-association	3-association
S _{AB} S _{AH} S _{AI} S _{BG} S _{EF} S _{EG} S _{HI} S _{IJ}	S _{ABI} S _{AFH} S _{BHI} S _{EFG}	S _{BCHI} S _{BHIL}	S _{AB-IJ} S _{AH-EF} S _{EG-AI} S _{EI-GJ} S _{HI-EG} S _{IJ-EG}	S _{BHI-ACJ} S _{BHI-AJC}

5.2 Measurement Technique

One method to examine the behaviour of a network is through examining the outputs of the network, given some various input patterns. The important point to be considered in this method is how we interpret the output pattern generated by the network in response to an arbitrary input pattern. In an associative memory, the output may be a pattern that has been stored previously, or it may be a totally new pattern that has never been learned before. If the output pattern is one of the stored patterns (in case of autoassociative memory) or a pattern that is associated with the input (in case of heteroassociative memory), then there is no problem to interpret the output pattern. In other words, the given input pattern leads to a perfect recall of a stored pattern. However, it is not so obvious if the output pattern is a totally different pattern than the one the network has

learned before. In this case, the Hamming distance (see Eq. 3.9) is used for measuring the similarity of an output pattern to a stored pattern. For instance, a new pattern is named *Anoisy* (a noisy version of pattern *A*) if that pattern imperfectly resembles pattern *A*. Notice that every noisy version that is presented more than once in the table has an index, e.g., *Anoisy*¹. However, there are several exceptional cases where the Hamming distance is not used. These are the cases when the output pattern is an intersection of two or more stored patterns, a unique feature of a stored pattern (a pattern that is part of a particular pattern), or a complement of a stored pattern. To show this, let us consider each pattern as a set of 35 ordered binary numbers (bits), that is, $A = \{a_1, a_2, \dots, a_{35}\}$, $a_i \in \{0, 1\}$. The order of the bits (i.e., the position of each bit in a set) is important. A pattern *A* is said to be equal to a pattern *A** if every bit, a_i , of *A* equals the corresponding bit, a_i^* , of *A**. In other words, *A* is comprised of all elements of *A** and vice versa. For convenience, we are only interested in the existence of the ON (1) bits in a set. From this point of view, a set $A \cap B$ exists (i.e., $A \cap B \neq \{\emptyset\}$) if there is at least one ON bit, $a_i = 1$, of pattern *A* that equals to the corresponding bit, $b_i = 1$, of pattern *B*. Using the same notation, an output pattern is named $A \cap B$ if its 1s are only the intersection bits of pattern *A* and *B*. Similarly, a pattern is named $A_{\mu-B}$ if its 1s are bits that are not in the set *B* (the unique bits of pattern *A*; i.e., $A - (A \cap B)$), and it is named A^c if it is a set of the first complement of pattern *A*. Some examples of these cases are illustrated in Fig. 5.2. There is also a condition for which we cannot use the notation defined above to classify the pattern. This is the case when the output pattern has the same distances toward two or more stored patterns. In this case, a question mark “?” is used as its name.

There are some adjustments to BP's output, since it uses real values for representing the output values. All the outputs of BP are rounded up, that is, an output value ≥ 0.5 becomes 1 and 0 otherwise. This adjustment is not necessary for BAM and CPN models,

since BAM uses only integer values and CPN stores only integer values at its Grossberg layer.

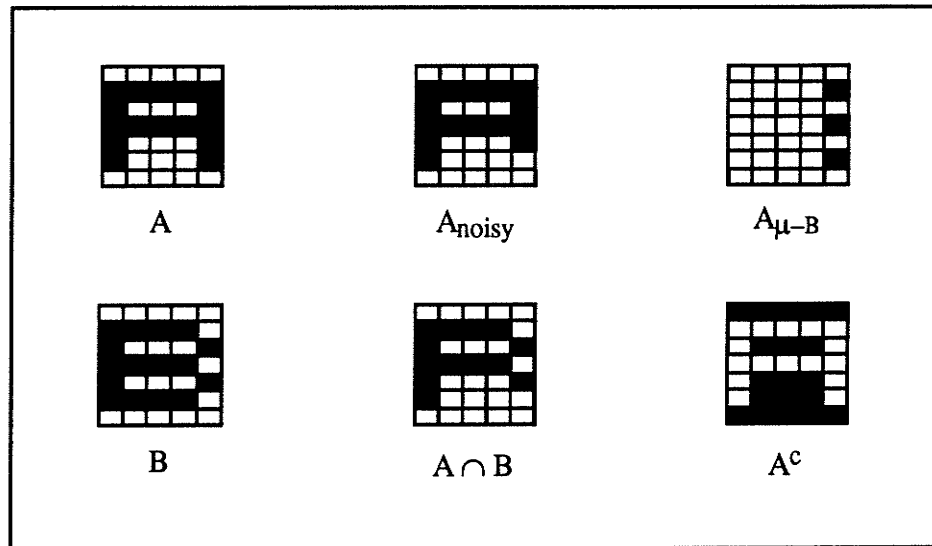


Fig. 5.2. Projections of A , B , A_{noisy} , $A \cap B$, $A_{\mu-B}$, and A^c .

5.3 Autoassociative Experimentation

This experiment is aimed to study the behaviour of BAM, BP, and CPN models as autoassociators. These models are used to store several associations, and then their behaviors are examined through analyzing the output given a particular test pattern to the input. The experiment is divided into four individual experiments. Each experiment is distinct by the number of associations used.

5.3.1 Two-association

In this experiment, 7 two-association sets (S_{AB} , S_{AH} , S_{AI} , S_{BG} , S_{EG} , S_{HI} , and S_{IJ}) are selected as the training sets. Each individual experiment uses each of the two-association set. Thus, each model is trained separately using each of the training sets. The trained networks are then tested using several incomplete versions of the training patterns. The training sets are listed in Table 5.2. This table includes the properties of the training patterns, such as the total number of ON bits (1s), and the number of similar bits and distinct bits (Hamming distance).

5.3.1.1 Network Configurations

The discrete BAM network consists of 35 input and 35 output neurons. This number of neurons is selected since the inputs and the outputs are 7×5 binary pixel patterns. The thresholds of the neurons are equal to 0, so this is a homogeneous BAM. The CPN network consists of 35 input neurons, 2 hidden neurons, and 35 output neurons. This number of hidden neurons is used since the memory capacity of CPN is equal to the number of its hidden neurons [Hech89]. The network is a forward-only CPN (refer to Section 3.3.1 for an explanation of a forward-only network) with accretive mode, that is, only a single hidden neuron can be activated at a time. All neurons in the adjacent layers are fully connected. The experiment also includes a two-layer BP (BP without hidden layer) and a three-layer BP (BP with one hidden layer) network. Both networks have 35 input and 35 output neurons. The three-layer BP uses only a single hidden neuron. This is sufficient for learning only two training patterns since a network with one hidden layer can exactly implement an arbitrary training set with p training patterns, provided that $p-1$ hidden neurons are used [SaAn91, HuHu91, MeMR91]. The neurons in the hidden and output layers have biases, and they use the sigmoid function as the threshold function. All

neurons in the adjacent layers are fully connected. Those network configurations are fixed for all two-association training sets.

Table 5.2. Two-association training sets for the autoassociative experiment.

Training set	1s (bits)	H _d (bits)	∩ (bits)
S _{AB} = {(A, A), (B, B)}	A _{1s} = 16 B _{1s} = 16	6	13
S _{AH} = {(A, A), (H, H)}	A _{1s} = 16 H _{1s} = 13	3	13
S _{AI} = {(A, A), (I, I)}	A _{1s} = 16 I _{1s} = 13	13	8
S _{BG} = {(B, B), (G, G)}	B _{1s} = 16 G _{1s} = 17	5	14
S _{EG} = {(E, E), (G, G)}	E _{1s} = 17 G _{1s} = 17	2	16
S _{HI} = {(H, H), (I, I)}	H _{1s} = 13 I _{1s} = 13	16	5
S _{IJ} = {(I, I), (J, J)}	I _{1s} = 13 J _{1s} = 11	2	11

5.3.1.2 Storing

The training procedure in BAM is straightforward. There is no learning parameter to adjust nor any error criterion to meet. Before training, all the weights are initialized to zero. On the contrary, BP requires some learning parameter adjustments and some error criterion

to meet. The total error of 0.03 is chosen as the stopping criterion. This means that the training process will terminate if the total error of the network is less than or equal to 0.03 (in the Euclidean metric). At this point, it is assumed that the network has reached its global minima. In this experiment, all the BP trainings are epoch trainings (batch-update trainings), and the training patterns are presented in random order. Notice that a uniform distribution of random values between -1.0 and $+1.0$ is used for the weight initialization. Table 5.3a and 5.3b show the learning rate (ϵ), the momentum term (α), the total error, and the number of training cycles completed on each BP training. Fig. 5.3 depicts a typical plot of the total error versus the number of epochs (training cycles) of a two-layer BP trained with the S_{AB} training set. Similar to BP, CPN requires some learning parameter adjustments. For all CPN trainings, the Kohonen learning rate is set to 0.1 and the Grossberg learning constant is set to 1.0. To prevent the under-utilization problem (see Section 3.3.3), a scheme called Frequency Sensitive Competitive Learning (FSCL) [AKCM90] is employed. Using this scheme, every hidden neuron is divided by the winning frequency of that neuron. This prevents a hidden neuron to become a winner more often than $\frac{1}{N}$ of the time, where N is the number of the hidden neurons. So, every hidden neuron can win the competition with approximately $\frac{1}{N}$ probability. Since CPN does not have any error criterion like BP, the number of training cycles becomes the CPN stopping criterion. In the experiment, 40 training cycles is to be completed. Prior to each training, all the weights are initialized to a fixed value of 0.1. During training, the training patterns are presented one by one in random order.

Table 5.3a. Learning rates, momentum terms, total errors and training cycles of the two-layer BPs.

Training set	ϵ	α	total error	epochs
S _{AB}	0.3	0.5	0.03	175
S _{AH}	0.3	0.5	0.03	205
S _{AI}	0.3	0.5	0.03	168
S _{BG}	0.3	0.5	0.03	178
S _{EG}	0.3	0.5	0.03	190
S _{HI}	0.3	0.5	0.03	170
S _{IJ}	0.3	0.5	0.03	223

Table 5.3b. Learning rates, momentum terms, total errors and training cycles of the three-layer BPs.

Training set	ϵ	α	total error	epochs
S _{AB}	0.3	0.5	0.03	2767
S _{AH}	0.3	0.5	0.03	2878
S _{AI}	0.3	0.5	0.03	2684
S _{BG}	0.3	0.5	0.03	2892
S _{EG}	0.3	0.5	0.03	7567
S _{HI}	0.3	0.5	0.03	3101
S _{IJ}	0.3	0.5	0.03	9860

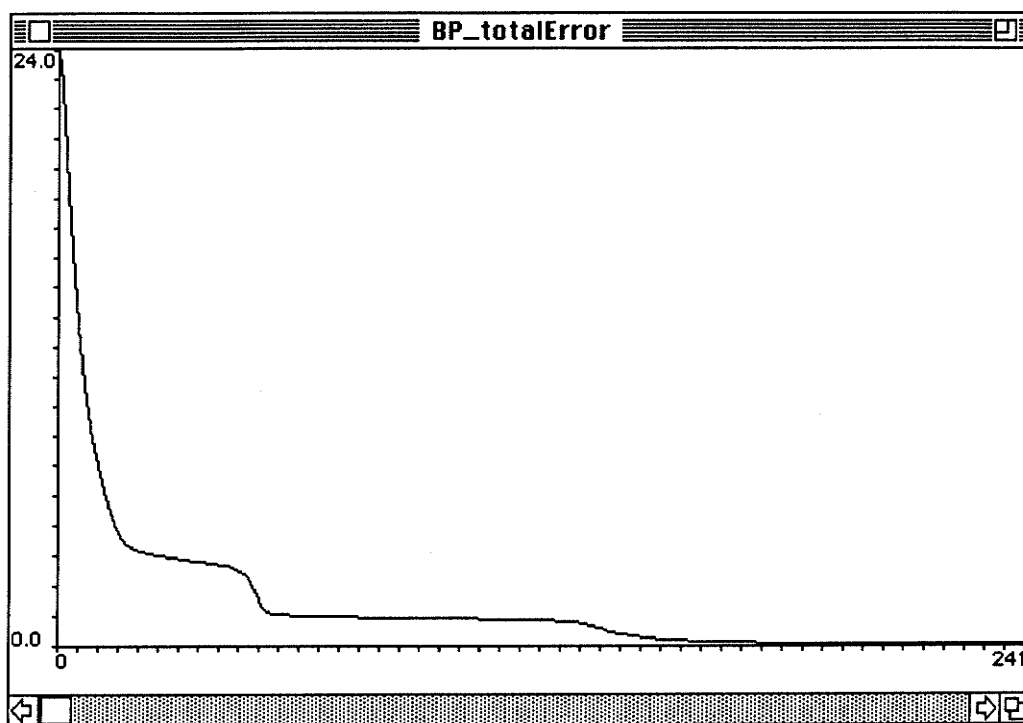


Fig. 5.3. Total error versus number of epochs of a two-layer BP on S_{AB} set.

5.3.1.3 Recalling

Every trained network is first tested using the training patterns as inputs to make sure that successful training is achieved. Successful training means that the network always outputs a copy or an approximate copy of the target pattern, given a corresponding input pattern used in the training. This verification test seems redundant for BP, since it uses the difference (the total error) between the actual and the desired (target) outputs as the stopping criterion of the training. It is obvious that successful BP training is achieved if the network can approximate the target pattern within a specified error criterion. This is not so obvious, however, for the BAM and CPN models. On the one hand, since there is no error criterion

nor any training cycles involved in BAM training, the test employing the training patterns is essential. On the other hand, CPN has some training cycles to complete. However, the number of training cycles is not an explicit indicator for successful training, unlike the error criterion in BP. To verify whether a CPN network is successfully trained, a test employing the training patterns is necessary. The verification test shows that all the two-association training sets are successfully trained by BAM, BP, and CPN networks. Moreover, this test also shows that 40 training cycles is sufficient for the CPN trainings. The detail lists of BP trainings, including the number of epochs completed, are given in Table 5.3a and 5.3b.

The next test is aimed to study the performance of the models giving some incomplete patterns. These incomplete patterns are not randomly generated nor selected from the noisy versions of the stored patterns. They are merely partial versions of the stored patterns. Notice that, we distinguish between a noisy version and a partial version of a pattern. A noisy version of a pattern is generated through distorting the pattern with some random noise, whereas a partial version of a pattern is generated by hand through considering that every ON bit is a part of a pattern. Some incomplete patterns used in the experiments are illustrated in Table 5.4a-g.

All the networks trained with two-association training sets are tested using patterns that are partial versions of the training patterns. The test results for the S_{AB} , S_{AH} , S_{AI} , S_{BG} , S_{EG} , S_{HI} , and S_{IJ} training sets are shown in Table 5.4a, 5.4b, 5.4c, 5.4d, 5.4e, 5.4f, and 5.4g, respectively. Since many of the test patterns are a superset/subset of the others and yield the same results, only representative elements are shown in the tables. Each table shows the test pattern used as input to the networks, the Hamming distance (H_d) between the test pattern and the stored patterns, and the output pattern of the networks. All the input (test) patterns in the table are represented by 7×5 binary pixel projections. Every input/output pattern has a name that follows the naming conventions defined in Section 5.2.

Table 5.4a shows the results of 14 out of 24 test patterns available for S_{AB} testing. The BP and CPN networks associate most of the input patterns with the closest (in Hamming distance) stored pattern. However, the BP and CPN networks respond differently if an ambiguous input pattern is presented. This can be seen at the output patterns in response to the input patterns $p02$, $p11$, $p14$, $p15$, $p20$, $p22$, $p23$, and $p24$ that have similar Hamming distances between all the stored patterns. The BAM network responds differently to these input patterns. Using the same definition of a pattern set as in Section 5.2, a spurious pattern that appears to be an intersection of the stored patterns becomes a stable output if the input is comprised of only intersection bits such as $p02$, or some intersection bits and some unique bits such as $p14$, $p15$. However, this is not so for the input patterns containing some unique bits of only one stored pattern besides the intersection bits. To see this, let us compare pattern $p14$ and $p15$ with pattern $p07$ and $p09$. Pattern $p14$ and $p15$ have an equal number of unique bits of both stored patterns, that is, 3 bits in $p14$ and 1 bit in $p15$. On the other hand, pattern $p07$ has 1 unique bit of B and none of A (besides the intersection bits) while pattern $p09$ has 3 unique bits of A and none of B . The network outputs a perfect pattern B and a perfect pattern A for the input pattern $p07$ and $p09$, respectively. Other spurious outputs are shown for pattern $p11$, $p20$, $p22$ and $p24$. The network outputs a zero vector (a vector of which all the elements are zero) in response to pattern $p11$ and $p22$, while it outputs a pattern that is a union of the first complements of the stored patterns in response to pattern $p20$ and $p24$. If the input pattern is comprised of only unique bits of a stored pattern such as $p08$ or $p10$, the network outputs a spurious pattern that is simply the input pattern itself.

Table 5.4a. Test results of training set S_{AB} .









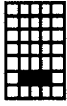
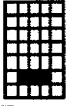

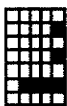




Input	H_d (bits)	BAM	BP	BP-1 hid.	CPN
p02 	p02-A = 11 p02-B = 11	 $A \cap B$	 ?	A	B
p07 	p07-A = 15 p07-B = 13	B	B	B	B
p08 	p08-A = 13 p08-B = 19	 $A \neq B$	 A_{noisy}	A	A
p09 	p09-A = 9 p09-B = 15	A	A	A	A
p10 	p10-A = 19 p10-B = 13	 $B \neq A$	 B_{noisy}	B	B
p11 	p11-A = 16 p11-B = 16	\emptyset	 A_{noisy}	B	A
p14 	p14-A = 3 p14-B = 3	$A \cap B$	A	B	B
P15 	P15-A = 15 p15-B = 15	$A \cap B$	 A_{noisy}	A	A

Table 5.4a (continued). Test results of training set S_{AB} .

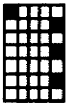



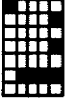
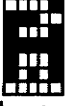
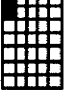
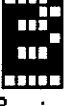


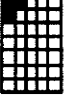

Input	H_d (bits)	BAM	BP	BP-1hid.	CPN
p18 	p18-A = 14 p18-B = 20	B^c	 A_{noisy}	A	A
p19 	p19-A = 20 p19-B = 14	A^c	 B_{noisy}	B	B
p20 	p20-A = 17 p20-B = 17	$A^c \cup B^c$	 A_{noisy}	B	A
p22 	p22-A = 16 p22-B = 16	\emptyset	 B_{noisy}	A	A
p23 	p23-A = 15 p23-B = 15	$A \cap B$	 B_{noisy}	A	A
p24 	p24-A = 17 p24-B = 17	$A^c \cup B^c$	 B_{noisy}	A	A

Table 5.4b shows the results of 9 out of 16 test patterns used for S_{AH} testing. As in the previous S_{AB} testing, BP and CPN associate most of the input patterns with the closest stored pattern (in Hamming distance). Although there is no ambiguous input pattern in

terms of a similar Hamming distance between the input and the stored patterns, a pattern such as $p12$ seems enough to make the networks disagree. This pattern is an ambiguous pattern for the human eye. However, it is not so ambiguous in terms of Hamming distance, considering that there is another stored pattern which is closer to the input pattern than the one that has been trained with the input pattern. Yet, if we closely examine pattern $p12$, we see that it is comprised of a bit that does not belong to either pattern A nor pattern H . On the other hand, pattern $p09$, which is also an ambiguous pattern (at least for the human eye), is associated with pattern H by all the networks. This pattern consists of a single bit that belongs to both pattern A and H . However, since H is a subset of A , the intersection of both patterns is simply the pattern H itself. This is also shown by the BAM result. There is only one spurious output, that is $A_{\mu-H}$. Notice that the complements of the stored patterns are not to be considered as spurious patterns since these patterns are, by default, encoded automatically in the BAM storing process (see Section 3.1.3). Another disagreement on the output patterns of the networks is shown for the input pattern $p02$. All BP and CPN networks associate the pattern with pattern H . Yet, BAM associates the pattern with pattern A , even though the pattern is closer to H than to A .

Table 5.4c shows the results of 9 out of 20 test patterns available for S_{AI} testing. There are several spurious patterns such as $A \cap B$, $A_{\mu-I}$, $I_{\mu-A}$, and the zero vector (\emptyset). From the BAM test results, two input patterns ($p04$ and $p05$) can be pointed out. These patterns contain only the unique bits of both stored patterns. The difference is that $p04$ has more unique bits of A than of I , while $p05$ has the same amount of unique bits of both stored patterns. This result shows that the zero vector is selected when the amount of unique bits of the stored patterns are equal (see also $p11$ in Table 5.4a, $p11$ in Table 5.4d, and $p09$ in Table 5.4f). This is also true if the numbers of the complement bits and the intersection bits are equal (see $p22$ in Table 5.4a and $p10$ in Table 5.4f).

Table 5.4b. Test results of training set S_{AH} .




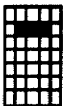

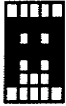
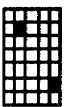
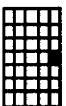

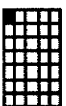

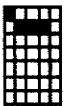

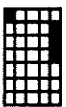
Input	H_a (bits)	BAM	BP	BP-1hid.	CPN
p01 	p01-A = 6 p01-H = 3	H	H	H	H
p02 	p02-A = 2 p02-H = 1	A	H	H	H
p05 	p05-A = 11 p05-H = 10	A	A	A	A
p06 	p06-A = 13 p06-H = 16	 $A_{\mu-H}$	 A_{noisy}	A	A
p08 	p08-A = 14 p08-H = 13	A	A	A	A
p09 	p09-A = 15 p09-H = 12	H	 H_{noisy}	H	H
p12 	p12-A = 17 p12-H = 14	A^c	 H_{noisy}	H	A
p13 	p13-A = 14 p13-H = 17	H^c	 A_{noisy}	A	A
p14 	p14-A = 14 p14-H = 11	H	H	H	H

Table 5.4c. Test results of training set S_{AI} .

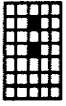


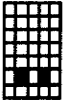
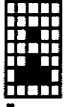

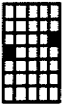
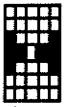






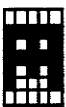

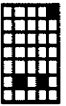
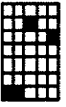
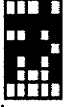
Input	H_d (bits)	BAM	BP	BP-1hid.	CPN
p01 	p01-A = 14 p01-I = 11	 $A \cap I$	 Inoisy	I	I
p02 	p02-A = 18 p02-I = 11	 $I \neq A$	 Inoisy	I	I
p03 	p03-A = 14 p03-I = 15	 $A \neq I$	 Anoisy	A	A
p04 	p04-A = 13 p04-I = 16	$A \neq I$	 Anoisy	A	A
p05 	p05-A = 16 p05-I = 13	\emptyset	 Inoisy	I	I
p06 	p06-A = 14 p06-I = 13	A	 Anoisy	A	A
p07 	p07-A = 16 p07-I = 11	I	I	I	I
p11 	p11-A = 19 p11-I = 12	A^c	I	I	I
p14 	p14-A = 16 p14-I = 15	I^c	 Anoisy	A	A

Table 5.4d. Test results of training set S_{BG} .













Input	H_d (bits)	BAM	BP	BP-1hid.	CPN
p07 	p07-B = 14 p07-G = 17	B	 B_{noisy}	B	B
p11 	p11-B = 16 p11-G = 17	\emptyset	 G_{noisy}	B	B
p14 	p14-B = 17 p14-G = 18	$B^c \cup G^c$	 G_{noisy}	B	B
p18 	p16-B = 18 p16-G = 17	B^c	 G_{noisy}	G	G
p19 	p19-B = 3 p19-G = 2	G	G	G	G
p20 	p20-B = 2 p20-G = 3	$B \cap G$	B	B	B
p21 	p21-B = 3 p21-G = 2	G	G	G	G
p22 	p22-B = 2 p22-G = 3	$B \cap G$	B	B	B

Table 5.4e. Test results of training set S_{EG} .








Input	H_a (bits)	BAM	BP	BP-1hid.	CPN
p04 	p04-E = 16 p04-G = 16	 $E \cap G$	G	E	E
p05 	p05-E = 16 p05-G = 16	$E \cap G$	 ?	E	G
p15 	p15-E = 2 p15-G = 2	$E \cap G$	 $E \cup G$	E	G
p16 	p16-E = 1 p16-G = 1	$E \cap G$	$E \cap G$	E	G

Table 5.4d shows 8 out of 22 S_{BG} test results, and Table 5.4e shows 4 out of 16 S_{EG} test results. Similarly, Table 5.4f and Table 5.4g show 8 out of 16 S_{HI} results and 5 out of 9 S_{IJ} results, respectively. The networks more or less show the same behaviour in response to the partial versions of the stored patterns. Some results that can be pointed out are the outputs of BAM, for the input pattern $p14$ in Table 5.4d, $p13$ and $p14$ in Table 5.4f. The common feature these input patterns have is the complement bits. Pattern $p14$ of Table 5.4d has a single bit of the complement bit and an equal number of unique bits of the stored patterns. The unique bits seem to cancel out each other, thus the complement bit appears to be dominant. Likewise, $p13$ of Table 5.4f and $p14$ of Table 5.4f have more complement bits than the intersection bits that make the BAM network outputs a spurious pattern which is a union of the complements of the stored patterns.

Table 5.4f. Test results of training set S_{HI} .






























Input	H_d (bits)	BAM	BP	BP-1hid.	CPN
p01 	p01-H = 8 p01-I = 8	 $H \cap I$	 H_{noisy}	I	I
p02 	p02-H = 12 p02-I = 12	$H \cap I$	 I_{noisy}	I	I
p09 	p09-H = 13 p09-I = 13	\emptyset	 I_{noisy}	H	H
p10 	p10-H = 13 p10-I = 13	\emptyset	 I_{noisy}	I	H
p13 	p13-H = 14 p13-I = 14	 $H^c \cup I^c$	 H_{noisy}	I	H
p14 	p14-H = 14 p14-I = 14	$H^c \cup I^c$	 I_{noisy}	I	H
p15 	p15-H = 8 p15-I = 8	$H \cap I$	 H_{noisy}	H	I
p16 	p16-H = 8 p16-I = 8	$H \cap I$	 I_{noisy}	I	I

Table 5.4g. Test results of training set S_{IJ} .

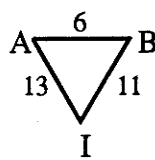
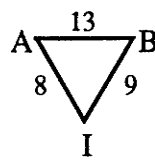
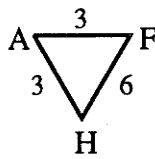
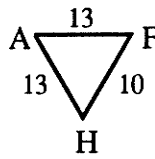
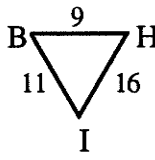
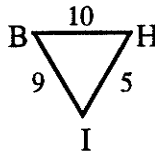
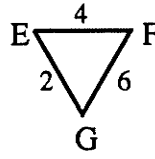
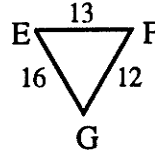
Input	H_d (bits)	BAM	BP	BP-1hid.	CPN
p02 	p02-I = 12 p02-J = 12	 $I_{p \neq J}$	 Inoisy	I	I
p03 	p03-I = 11 p03-J = 11	I	 Inoisy	I	I
p04 	p04-I = 12 p04-J = 10	J	 J_{noisy}	J	J
p07 	p07-I = 12 p07-J = 14	J_c	 Inoisy	I	I
p09 	p09-I = 14 p09-J = 12	I_c	 ?	J	I

5.3.2 Three-association

The next major experiment consists of 4 individual experiments employing different training sets. These training sets are S_{ABI} , S_{AFH} , S_{BHI} , and S_{EFG} . Similar to the previous experiment, each network is trained with each of these training sets and tested using some incomplete versions of the training patterns. The training sets and their properties are listed in Table 5.5. The first column of the table lists the training sets, the second column gives

the total number of ON bits of a pattern, the third column shows the number of different bits between patterns, and the fourth column shows the number of similar bits between patterns in the set.

Table 5.5. Three-association training sets for the autoassociative experiment.

Training set	$1s$ (bits)	H_d (bits)	\cap (bits)
$S_{ABI} = \{(A, A), (B, B), (I, I)\}$	$A_{1s} = 16$ $B_{1s} = 16$ $I_{1s} = 13$		 $A \cap B \cap I = 6$
$S_{AFH} = \{(A, A), (F, F), (H, H)\}$	$A_{1s} = 16$ $F_{1s} = 13$ $H_{1s} = 13$		 $A \cap F \cap H = 9$
$S_{BHI} = \{(B, B), (H, H), (I, I)\}$	$B_{1s} = 16$ $H_{1s} = 13$ $I_{1s} = 13$		 $B \cap H \cap I = 3$
$S_{EFG} = \{(E, E), (F, F), (G, G)\}$	$E_{1s} = 17$ $F_{1s} = 13$ $G_{1s} = 17$		 $E \cap F \cap G = 12$

5.3.2.1 Network Configurations

The three-association experiment employs the same networks used in the previous experiment, i.e., BAM, BP, and CPN networks with 35 input and 35 output neurons. However, the three-layer BP and CPN networks have slightly different configurations. Instead of using a single hidden neuron in the three-layer BP and two hidden neurons in the CPN, they use 2 hidden neurons and 3 hidden neurons respectively. This is due to the number of associations (patterns) to be stored, the same reason for choosing the number of hidden neurons as in the previous experiment. The other configurations such as the type of threshold function and the connectivities between layers remain the same.

5.3.2.2 Storing

The training procedure for each network is similar as in the previous experiment. BP uses the total error 0.03 for the stopping criterion of training and uniformly distributed random values between -1.0 and $+1.0$ for the initial weights. Table 5.6a and 5.6b show the learning rate (ϵ), the momentum term (α), the total error, and the number of training cycles completed for each BP training. CPN requires 40 training cycles to complete training, with the Kohonen learning rate set to 0.1, the Grossberg learning constant set to 1.0, and the FSCL option switched on.

Table 5.6a. Learning rates, momentum terms, total errors and training cycles of two-layer BP.

Training set	ϵ	α	total error	epochs
SABI	0.3	0.5	0.03	288
SAFH	0.3	0.5	0.03	301
SBHI	0.3	0.5	0.03	290
SEFG	0.3	0.5	0.03	315

Table 5.6b. Learning rates, momentum terms, total errors and training cycles of three-layer BP.

Training set	ϵ	α	total error	epochs
SABI	0.3	0.5	0.03	3314
SAFH	0.3	0.5	0.03	1703
SBHI	0.3	0.5	0.03	3716
SEFG	0.3	0.5	0.03	1658

5.3.2.3 Recalling

Similar to the previous experiment, the trained networks are verified using the training sets. From this training verification, it is found that two of the BAM networks trained unsuccessfully. These two networks use the SAFH, and SEFG sets. They can store only one training pattern and the complement, i.e., pattern A and A^c for the SAFH set, and pattern E and E^c for the SEFG set, given any of the training pattern as the input. The test also shows that all BP and CPN networks are trained successfully. The verification results are shown in Table 5.7.

Table 5.7. Verification results of the three-association experiment.

Training set	Training status			
	BAM	BP	BP-1hidden	CPN
SABI	ok	ok	ok	ok
SAFH	fail	ok	ok	ok
SBHI	ok	ok	ok	ok
SEFG	fail	ok	ok	ok

All the networks are tested with several incomplete patterns. This test also includes the two unsuccessfully trained BAM networks. The results are shown in Table 5.8a-d for the S_{ABI} , S_{AFH} , S_{EFG} , and S_{BHI} training sets, respectively. Table 5.8a shows only 12 out of 34 test results of S_{ABI} , and Table 5.8d shows only 8 out of 25 test results of S_{BHI} . Since BAM fails to learn the S_{AFH} and S_{EFG} training sets, there are only 7 and 4 test patterns used to test the trained networks. Tables 5.8b and 5.8c show 4 test results of S_{AFH} and 3 test results of S_{EFG} , respectively.

As in the two-association test, most of the networks respond differently to some ambiguous input patterns, such as $p07$, $p12$, and $p17$ in Table 5.8a. This can be seen also in Table 5.8b for input pattern $p03$ and in Table 5.8d for input pattern $p03$, $p04$, $p05$, $p15$, and $p19$. Most of the ambiguous input patterns have the same closest Hamming distances towards at least two of the stored patterns. However, there is an ambiguous pattern such as $p05$ in Table 5.8c that makes all the networks (except BAM that fails in the training process) to respond similarly. They associate the input pattern with pattern G . This is, perhaps, because of more ON bits in $p05$ that belong to pattern G . Another example that shows the importance of some specific bits in the input patterns can be seen also in $p17$, $p18$, and $p19$ of Table 5.8a. Pattern $p17$ contains only two ON bits; one bit that also belongs to patterns A , B , and I , and one bit that belongs to pattern A and B . Although pattern I is closer to the input pattern (refer to the second column of the table), the networks associate the input pattern with either pattern A or pattern B since there are more ON bits in the input pattern that belong to those stored patterns. If we add another bit that is a unique bit of a stored pattern to pattern $p17$, such as in pattern $p18$ and $p19$, then all the networks associate pattern $p18$ with pattern A and pattern $p19$ with pattern B . If we examine closely pattern $p18$, there are more ON bits that belong to pattern A than to the others. Likewise, pattern $p19$ contains more ON bits that are parts

of pattern *B*. In other words, *p19* is a subset of pattern *B*. These examples show that the networks favour a specific bit in the input pattern that reflects a feature of a stored pattern. The results also show that BAM outputs the complement of a stored pattern if the distance between at least two of the stored patterns and an input pattern greater than their total ON bits (1s). This is shown by the output patterns for input pattern *p22* and *p23* of Table 5.8a, and *p20* of Table 5.8d.

Table 5.8a. Test results of training set S_{ABI} .









Input	H_d (bits)	BAM	BP	BP-2hid.	CPN
p07 	p07-A = 3 p07-B = 3 p07-I = 14	 Anoisy ¹	Anoisy ¹	A	B
p11 	p11-A = 2 p11-B = 4 p11-I = 13	A	A	A	A
p12 	p12-A = 7 p12-B = 11 p12-I = 6	 Bnoisy ¹	 Inoisy	 Inoisy	A
p13 	p13-A = 4 p13-B = 2 p13-I = 13	B	B	B	B

Table 5.8a (continued). Test results of training set S_{ABI} .













Input	H_d (bits)	BAM	BP	BP-2hid.	CPN
p17 	p17-A = 14 p17-B = 14 p17-I = 13	Anoisy ¹	 Anoisy	Bnoisy ¹	B
p18 	p18-A = 13 p18-B = 15 p18-I = 12	A	A	A	A
p19 	p19-A = 15 p19-B = 13 p19-I = 12	B	B	B	B
p22 	p22-A = 15 p22-B = 17 p22-I = 14	I ^c	A	A	A
p23 	p23-A = 18 p23-B = 18 p23-I = 11	(Bnoisy ¹) ^c	I	I	I
p27 	p27-A = 14 p27-B = 18 p27-I = 13	I	 Anoisy	A	A
p28 	p28-A = 13 p28-B = 17 p28-I = 12	A	 Anoisy	 Anoisy	A
p29 	p29-A = 13 p29-B = 17 p29-I = 14	A	A	A	A

Table 5.8b. Test results of training set S_{AFH} .






Input	H_d (bits)	BAM	BP	BP-2hid.	CPN
p01 	p01-A = 11 p01-F = 8 p01-H = 14	A	F	F	F
p02 	p02-A = 11 p02-F = 14 p02-H = 8	A	H	H	H
p03 	p03-A = 7 p03-F = 4 p03-H = 4	A	 $A \cap F \cap H$	A	H
p04 	p04-A = 7 p04-F = 10 p04-H = 10	A	A	A	A

Table 5.8c. Test results of training set S_{EFG} .





















Input	H_d (bits)	BAM	BP	BP-2hid.	CPN
p01 	p01-E = 12 p01-F = 8 p01-G = 12	E	F	F	F
p03 	p03-E = 11 p03-F = 9 p03-G = 11	E	 ?	E	E
p05 	p05-E = 12 p05-F = 10 p05-G = 10	E	G	G	G

Table 5.8d. Test results of training set S_{BHI} .

Input	H_a (bits)	BAM	BP	BP-2hid.	CPN
p03 	p17-B = 15 p17-H = 12 p17-I = 12	B	 Inoisy	 B_{noisy}^1	I
p04 	p18-B = 18 p18-H = 11 p18-I = 11	I	 Inoisy ²	B_{noisy}^1	I
p05 	p05-B = 19 p05-H = 10 p05-I = 12	Inoisy ²	Inoisy ²	H	H
p07 	p07-B = 11 p07-H = 08 p07-I = 14	 H_{noisy}^1	 H_{noisy}^2	H	H
p12 	p12-B = 6 p12-H = 7 p12-I = 13	B	 B_{noisy}	B	B
p15 	p15-B = 16 p15-H = 9 p15-I = 9	B	Inoisy ²	B_{noisy}^1	I
p19 	p19-B = 16 p19-H = 13 p19-I = 13	∅	 Inoisy	B_{noisy}^1	I
p20 	p20-B = 17 p20-H = 14 p20-I = 12	H^c	 Inoisy	I	I

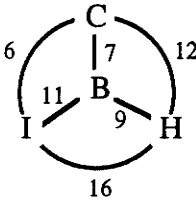
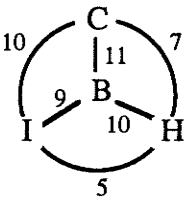
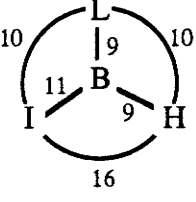
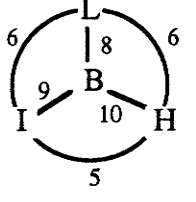
5.3.3 Four-association

From the previous three-association experiments on BAM, it is found that training sets containing patterns with a Hamming distance of 6 (or more) are successfully trained. This evidence leads to an assumption that, perhaps, a minimal Hamming distance between every two patterns in a set must be maintained in order to achieve perfect recall on all the stored patterns in BAM. Thus, the selection of patterns for four-association training is based on this assumption. The training sets are S_{BCHI} and S_{BHIL} . The S_{BCHI} training set contains one pattern with 6 bits of Hamming distance, whereas the S_{BHIL} training set contains only patterns with more than 6 bits of Hamming distance. They are listed in Table 5.9. The first column of the table lists the training sets, the second column shows the total number of ON bits (1s) of a pattern, the third column gives the number of different bits between patterns (i.e., the Hamming distances between two patterns), and the fourth column shows the number of similar bits (overlapping bits) between patterns in the set.

5.3.3.1 Network Configurations

The experiment uses the same networks from the previous experiment, except that the CPN network now has 4 hidden neurons instead of 3. Another three-layer BP network with 3 hidden neurons is also included in addition to the one with 2 hidden neurons. So, there are five networks with different configurations, that is, one BAM, one two-layer BP, two three-layer BPs with 2 and 3 hidden neurons respectively, and one CPN with 4 hidden neurons. All the other configurations such as the type of the threshold function and the connectivities between layers remain the same.

Table 5.9. Four-association training sets.

Training set	Is (bits)	Hd (bits)	\cap (bits)
$S_{BCHI} = \{ (B, B), (C, C), (H, H), (I, I) \}$	$B_{1s} = 16$ $C_{1s} = 13$ $H_{1s} = 13$ $I_{1s} = 13$		 $B \cap C \cap H = 5$ $B \cap C \cap I = 8$ $B \cap H \cap I = 3$ $C \cap H \cap I = 4$ $B \cap C \cap H \cap I = 2$
$S_{BHIL} = \{ (B, B), (H, H), (I, I), (L, L) \}$	$B_{1s} = 16$ $H_{1s} = 13$ $I_{1s} = 13$ $L_{1s} = 9$		 $B \cap H \cap I = 3$ $B \cap H \cap L = 5$ $B \cap I \cap L = 5$ $H \cap I \cap L = 3$ $B \cap H \cap I \cap L = 2$

5.3.3.2 Storing

As in the previous experiments, the BP networks use a total error of 0.03 for the stopping criterion and uniformly distributed random values between -1.0 and +1.0 for the initial weights. Tables 5.10a-c show the BP training results. In table 5.10b, it is shown

that one of the BP networks cannot reach a total error of 0.03 or less. This three-layer BP with 2 hidden neurons, which is trained with the S_{BHIL} set, fails to converge in training since BP uses the total squared error as an indicator of a successful training. The CPN network requires 40 training cycles to complete training with the Kohonen learning rate set to 0.1, the Grossberg learning constant set to 1.0, and the FSCL option on. The BAM and CPN training verifications are discussed in the next section.

Table 5.10a. Learning rates, momentum terms, total errors and training cycles of two-layer BP.

Training set	ϵ	α	total error	epochs
SBCHI	0.3	0.5	0.03	410
SBHIL	0.3	0.5	0.03	432

Table 5.10b. Learning rates, momentum terms, total errors and training cycles of three-layer BP with two hidden neurons.

Training set	ϵ	α	total error	epochs
SBCHI	0.1	0.5	0.0300	33185
SBHIL	0.1	0.5	0.8642	41500














Table 5.10c. Learning rates, momentum terms, total errors and training cycles of three-layer BP with three hidden neurons.

Training set	ϵ	α	total error	epochs
SBCHI	0.3	0.5	0.03	3530
SBHIL	0.3	0.5	0.03	4056

5.3.3.3 Recalling

The trained networks are verified using the training sets. From the verification test, it is found that all four-association training sets for CPN, two-layer BP, and three-layer BP with 3 hidden neurons are successfully trained. However, the three-layer BP with 2 hidden neurons can be successfully trained only with the S_{BCHI} set. The network fails to learn the S_{BHIL} training set, as indicated by its total error in Table 5.10b. The verification test for BAM shows that not all of the stored patterns can be recalled perfectly. The network produces the noisy versions of pattern C of the S_{BCHI} set and of pattern L of the S_{BHIL} set, given the original pattern C and L , respectively. These patterns can be seen in Table 5.11.

Table 5.11 The verification test results of BAM.

Input	Output		Energy	
	S_{BCHI}	S_{BHIL}	S_{BCHI}	S_{BHIL}
B 			-312	-284
C 			-312	
H 			-228	-234
I 			-252	-212
L 				-170

Recall tests through presentation of incomplete patterns are done for all trained BAM and for all successfully trained CPN and BP networks. Table 5.12a shows 8 out of 24 test results for the networks trained with the S_{BCHI} set. The first part of this table shows the Hamming distances between the input (test) pattern and each stored pattern, while the second part shows the output of each network. The test pattern set comprises of several ambiguous patterns such as pattern $p08$ and pattern $p10$, and several noisy versions of the stored patterns such as pattern $p13$, $p14$, $p15$, $p16$, $p22$, and $p23$. For some ambiguous input patterns, the networks give different outputs. For instance, pattern $p10$ can be associated with pattern H or I . However, most of the networks associate this pattern with pattern I . Yet, the BAM network associates the pattern with C since $p10$ is also a subset of the noisy version of C (see Table 5.11). If we examine the results of the two-layer BP, the three-layer BP with 3 hidden neurons and the CPN network in the table, they show almost the same responses to the input patterns. One difference is that some of the BP outputs are the noisy versions of the stored patterns, whereas the CPN always outputs one of the stored pattern. Unfortunately, the three-layer BP with 2 hidden neurons responds differently. A good example of this is the network's response to pattern $p14$ and $p15$. All the networks (except BP with 2 hidden neurons) associate these input patterns with pattern C . This is consistent with the Hamming distance concept of similarity, since these patterns are closer to pattern C than to the others. In terms of the number of ON bits that belong to a stored pattern, both patterns are subsets of pattern C . Still, the three-layer BP with 2 hidden neurons cannot associate these patterns consistently. On the other hand, it outputs a perfect C for input pattern $p22$, and a noisy version of C for input pattern $p23$. Table 5.12b shows the test results of the networks trained with the S_{BHIL} set. All the networks associate the noisy versions of B and H ($p06$ and $p07$) with the stored

pattern B and H , respectively. Similar to the results in Table 5.12a, the networks associate the ambiguous input patterns, such as pattern $p01$, $p02$, $p03$ and $p05$, differently.

Table 5.12a. Test results of training set S_{BCHI} .

Input	H_d (bits)			
p08	p08-B = 11	p08-C = 8	p08-H = 8	p08-I = 14
p10	p10-B = 15	p10-C = 10	p10-H = 8	p10-I = 8
p13	p13-B = 2	p13-C = 8	p13-H = 12	p13-I = 12
p14	p14-B = 5	p14-C = 2	p14-H = 11	p14-I = 8
p15	p15-B = 9	p15-C = 2	p15-C = 12	p15-I = 6
p16	p16-B = 9	p16-C = 14	p16-H = 2	p16-I = 18
p22	p22-B = 9	p22-C = 2	p22-H = 14	p22-I = 8
p23	p23-B = 6	p23-C = 1	p23-H = 13	p23-I = 7

Table 5.12a (continued). Test results of training set S_{BCHI} .



































Input	BAM	BP	BP-2hid.	BP-3hid.	CPN
p08 	 Bnoisy ¹	 Hnoisy	B	 Hnoisy	H
p10 	 Cnoisy ¹	 Inoisy ¹	Inoisy ¹	 Inoisy	I
p13 	B	B	 Bnoisy ²	B	B
p14 	Cnoisy ¹	 Cnoisy ²	Bnoisy ²	C	C
p15 	Cnoisy ¹	C	 Inoisy ²	C	C
p16 	 Hnoisy ¹	H	H	H	H
p22 	Cnoisy ¹	C	C	C	C
p23 	Cnoisy ¹	C	Cnoisy ²	C	C

Table 5.12b. Test results of training set S_{BHL} .

Input	H_d (bits)	BAM	BP	BP-3hid.	CPN
p01 	p01-B = 11 p01-H = 8 p01-I = 14 p01-L = 4	 Bnoisy ¹	 Lnoisy	 ?	L
p02 	p02-B = 13 p02-H = 14 p02-I = 8 p02-L = 12	I	 Inoisy	 Inoisy ¹	I
p03 	p03-B = 13 p03-H = 14 p03-I = 8 p03-L = 4	 Lnoisy	 Lnoisy	L	L
p05 	p05-B = 15 p05-H = 8 p05-I = 8 p05-L = 8	 Lnoisy ¹	 Inoisy	 Lnoisy	I
p06 	p06-B = 2 p06-H = 6 p06-I = 13 p06-L = 7	B	B	B	B
p07 	p07-B = 11 p07-H = 2 p07-I = 16 p07-L = 10	H	H	H	H

5.4 Heteroassociative Experimentation

This experiment is aimed to study the behaviour of BAM, BP, and CPN models as heteroassociators. Similar to the autoassociation experiment, the networks are trained to learn the associations, and then their behaviors are examined through analyzing the outputs in response to a particular test input pattern. The experiment consists of two individual experiments, namely the two-association and the three-association experiments.

5.4.1 Two-association

This experiment uses 6 two-association sets as the training sets. They are S_{AB-IJ} , S_{AH-EF} , S_{EG-AI} , S_{EI-GJ} , S_{HI-EG} , and S_{IJ-EG} . The networks are first trained using these sets and then tested using the incomplete versions of the training sets. A list of these training sets and their properties is given in Table 5.13. The organization of this table is similar to Table 5.2, 5.5, and 5.9 of Section 5.3. Notice that the line with double arrows in the table shows the Hamming distance (in the H_d column) or the intersection (in the \cap column) between the associated patterns.

5.4.1.1 Network Configurations

The networks have the same configurations as the networks used in the previous two-association experiment. They all have 35 input and 35 output neurons. Two BP networks are involved in the experiment: a two-layer BP and a three-layer BP with a single hidden neuron. All the neurons in the hidden and the output layers have biases and use the sigmoid function. The CPN network uses a forward-only architecture with 2 hidden neurons in its Kohonen layer. The neurons in all the networks are fully connected.

Table 5.13. Two-association training sets for the heteroassociative experiment.

Training set	I_s (bits)	H_d (bits)	\cap (bits)
$S_{AB-IJ} = \{(A, I), (B, J)\}$	$A_{1s} = 16$ $B_{1s} = 16$ $I_{1s} = 13$ $J_{1s} = 11$	$\begin{array}{ccc} & \xleftrightarrow{13} & \\ 6 & & 2 \\ & \xleftrightarrow{\quad} & \\ & \xleftrightarrow{11} & \\ & & \end{array}$	$\begin{array}{ccc} & \xleftrightarrow{8} & \\ 13 & & 11 \\ & \xleftrightarrow{\quad} & \\ & \xleftrightarrow{8} & \\ & & \end{array}$
$S_{AH-EF} = \{(A, E), (H, F)\}$	$A_{1s} = 16$ $H_{1s} = 13$ $E_{1s} = 17$ $F_{1s} = 13$	$\begin{array}{ccc} & \xleftrightarrow{5} & \\ 3 & & 4 \\ & \xleftrightarrow{\quad} & \\ & \xleftrightarrow{6} & \\ & & \end{array}$	$\begin{array}{ccc} & \xleftrightarrow{14} & \\ 13 & & 13 \\ & \xleftrightarrow{\quad} & \\ & \xleftrightarrow{10} & \\ & & \end{array}$
$S_{EG-AI} = \{(E, A), (G, I)\}$	$E_{1s} = 17$ $G_{1s} = 17$ $A_{1s} = 16$ $I_{1s} = 13$	$\begin{array}{ccc} & \xleftrightarrow{5} & \\ 2 & & 13 \\ & \xleftrightarrow{\quad} & \\ & \xleftrightarrow{8} & \\ & & \end{array}$	$\begin{array}{ccc} & \xleftrightarrow{14} & \\ 16 & & 8 \\ & \xleftrightarrow{\quad} & \\ & \xleftrightarrow{11} & \\ & & \end{array}$
$S_{EI-GJ} = \{(E, G), (I, J)\}$	$E_{1s} = 17$ $I_{1s} = 13$ $G_{1s} = 17$ $J_{1s} = 11$	$\begin{array}{ccc} & \xleftrightarrow{2} & \\ 8 & & 10 \\ & \xleftrightarrow{\quad} & \\ & \xleftrightarrow{2} & \\ & & \end{array}$	$\begin{array}{ccc} & \xleftrightarrow{16} & \\ 11 & & 9 \\ & \xleftrightarrow{\quad} & \\ & \xleftrightarrow{11} & \\ & & \end{array}$
$S_{HI-EG} = \{(H, E), (I, G)\}$	$H_{1s} = 13$ $I_{1s} = 13$ $E_{1s} = 17$ $G_{1s} = 17$	$\begin{array}{ccc} & \xleftrightarrow{8} & \\ 16 & & 2 \\ & \xleftrightarrow{\quad} & \\ & \xleftrightarrow{8} & \\ & & \end{array}$	$\begin{array}{ccc} & \xleftrightarrow{11} & \\ 5 & & 16 \\ & \xleftrightarrow{\quad} & \\ & \xleftrightarrow{11} & \\ & & \end{array}$
$S_{IJ-EG} = \{(I, E), (J, G)\}$	$I_{1s} = 13$ $J_{1s} = 11$ $E_{1s} = 17$ $G_{1s} = 17$	$\begin{array}{ccc} & \xleftrightarrow{8} & \\ 2 & & 2 \\ & \xleftrightarrow{\quad} & \\ & \xleftrightarrow{10} & \\ & & \end{array}$	$\begin{array}{ccc} & \xleftrightarrow{11} & \\ 11 & & 16 \\ & \xleftrightarrow{\quad} & \\ & \xleftrightarrow{9} & \\ & & \end{array}$

5.4.1.2 Storing

The training procedure for each network is similar to the preceding experiments. The only difference is the type of association the network is made to learn. Instead of learning to associate identical patterns, the networks learn the association between two different patterns. Before training, all the weights in BAM are set to zero, while the weights in CPN are set to 0.1. Also, the weights in all BP networks are initialized with uniformly distributed random values between -1.0 and 1.0 . The training stopping criterion remains the same, namely, a total error of 0.03 for the BP network and 40 training cycles for the CPN network. During training, the networks read the training pattern one by one in random order. The parameter set-ups of the CPN network are 0.1 for the Kohonen learning rate and 1.0 for the Grossberg learning constant, whereas the parameter set-ups of the BP networks are shown in Table 5.14a-b. These tables also show the number of epochs completed for each training.

Table 5.14a. Learning rates, momentum terms, total errors and training cycles of two-layer BP.

Training set	ϵ	α	total error	epochs
SAB-IJ	0.3	0.5	0.03	94
SAH-EF	0.3	0.5	0.03	247
SEG-AI	0.3	0.5	0.03	855
SEI-GJ	0.3	0.5	0.03	209
SHI-EG	0.3	0.5	0.03	101
SII-EG	0.3	0.5	0.03	220

Table 5.14b. Learning rates, momentum terms, total errors and training cycles of three-layer BP.

Training set	ϵ	α	total error	epochs
S _{AB-IJ}	0.3	0.5	0.03	7957
S _{AH-EF}	0.3	0.5	0.03	3823
S _{EG-AI}	0.3	0.5	0.03	4360
S _{EI-GJ}	0.3	0.5	0.03	2355
S _{HI-EG}	0.3	0.5	0.03	2628
S _{IJ-EG}	0.3	0.5	0.03	5717

5.4.1.3 Recalling

The verification test results of BP and CPN show successful training of the BP and CPN networks. Each of the stored patterns can be recalled completely through presentation of the associated pattern. However, BAM fails to learn one association, namely the association (J, G) in the S_{IJ-EG} set. It does not give the correct output, pattern G , given the input pattern J . Instead, a spurious pattern, which has similar distances between E and G , replaces the stored pattern G . This pattern is shown in Table 5.15.

The next test employs some incomplete patterns as the test patterns. The test sets are the same ones used in the autoassociative experimentation. For instance, the test set that is used to test the network trained with the S_{AB} set in the autoassociative experiment is used again to test the network trained with the S_{AB-IJ} set. Some test results of the S_{AB-IJ}, S_{AH-EF}, S_{EG-AI}, S_{EI-GJ}, S_{HI-EG}, and S_{IJ-EG} training sets are shown in Table 5.16a to 5.16f, respectively.

The test results of BAM and CPN seem consistent with the previous test results from the autoassociation experiment. Let us compare the BAM outputs in Table 5.16a with the BAM outputs in Table 5.4a, for the input pattern $p02$, $p07$, $p08$, and $p09$. The

networks give similar responses, except that the one trained as a heteroassociator outputs the associated pattern. For the input pattern $p02$, the heteroassociator outputs pattern J since this pattern is also an intersection pattern of pattern I and J . This is also shown by the results in Table 5.16b, Table 5.16c, Table 5.16e, and Table 5.16f. However, this is not the case for the BP networks, as shown by their outputs in response to pattern $p02$ of Table 5.16a (see also Table 5.4a), pattern $p15$ and $p16$ of Table 5.16c (see also Table 5.4e), and pattern $p02$, $p09$, and $p13$ of Table 5.16e (see also Table 5.4f).

Table 5.15. Verification results of BAM in the two-association experiment.


Training set	Input	Output	Energy
$S_{AB-IJ} = \{(A, I), (B, J)\}$	A	I	-298
	B	J	-286
$S_{AH-EF} = \{(A, E), (H, F)\}$	A	E	-362
	H	F	-338
$S_{EG-AI} = \{(E, A), (G, I)\}$	E	A	-272
	G	I	-266
$S_{EI-GJ} = \{(E, G), (I, J)\}$	E	G	-294
	I	J	-206
$S_{HI-EG} = \{(H, E), (I, G)\}$	H	E	-176
	I	G	-176
$S_{IJ-EG} = \{(I, E), (J, G)\}$	I	E	-356
	J		-352

Table 5.16a. Test results of training set S_{AB-IJ} .









Input	H_d (bits)	BAM	BP	BP-1hid.	CPN
p02 	p02-A = 11 p02-B = 11	J	 ?	J	J
p07 	p07-A = 15 p07-B = 13	J	 Jnoisy	J	J
p08 	p08-A = 13 p08-B = 19	 $I_{\mu-J}$	 Inoisy	I	I
p09 	p09-A = 9 p09-B = 15	I	I	I	I

Table 5.16b. Test results of training set S_{AH-EF} .






Input	H_d (bits)	BAM	BP	BP-1hid.	CPN
p02 	p02-A = 2 p02-H = 1	E	F	F	F
p06 	p06-A = 13 p06-H = 16	 $E_{\mu-F}$	E	E	E
p12 	p12-A = 17 p12-H = 14	E^c	 F_{noisy}	F	E

Table 5.16c. Test results of training set S_{EG-AI} .








Input	H_d (bits)	BAM	BP	BP-1hid.	CPN
p04 	p04-E = 16 p04-G = 16	 $A \cap I$	 I_{noisy}	A	A
p15 	p15-E = 2 p15-G = 2	$A \cap I$	 I_{noisy}	I	I
p16 	p16-E = 1 p16-G = 1	$A \cap I$	 A_{noisy}	I	I

Table 5.16d. Test results of training set S_{EL-GJ} .









Input	H_d (bits)	BAM	BP	BP-1hid.	CPN
p01 	p01-E = 14 p01-I = 16	 $G_{\mu-J}$	G	G	G
p02 	p02-E = 19 p02-I = 11	 $J_{\mu-G}$	J	J	J
p03 	p03-E = 12 p03-I = 8	 $J \cap G$	 J_{noisy}	J	J
p04 	p04-E = 2 p04-I = 6	G	G	G	G

Table 5.16e. Test results of training set S_{HI-EG} .
















Input	H_d (bits)	BAM	BP	BP-1hid.	CPN
p01 	p01-H = 8 p01-I = 8	 $E \cap G$	 $E \cup G$	G	G
p02 	p02-H = 12 p02-I = 12	$E \cap G$	E	G	G
p09 	p09-H = 13 p09-I = 13	\emptyset	$E \cap G$	G	E
p13 	p13-H = 14 p13-I = 14	 $E^c \cup G^c$	 E noisy	E	E

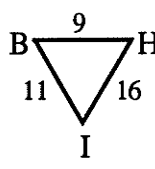
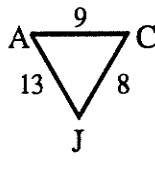
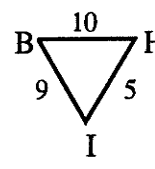
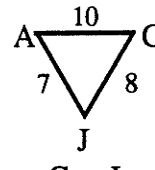
Table 5.16f. Test results of training set S_{IJ-EG} .

Input	H_d (bits)	BAM	BP	BP-1hid.	CPN
p01 	p01-I = 11 p01-J = 13	 $E_{\mu} - G$	 E noisy	E	E
p04 	p04-I = 12 p04-J = 10	 $E \cap G$	G	G	G
p07 	p07-I = 12 p07-J = 14	G^c	 E noisy	E	E

5.4.2 Three-association

The next experiment employs 2 three-association training sets: $S_{BHI-ACJ}$ and $S_{BHI-AJC}$ sets. Both sets contain the same patterns, except that they have a different formation for each pair. Table 5.17 shows the training sets and their properties.

Table 5.17. Three-association training sets for the heteroassociative experiment.

Training set	$1s$ (bits)	H_d (bits)	\cap (bits)
$S_{BHI-ACJ} = \{(B, A), (H, C), (I, J)\}$ $S_{BHI-AJC} = \{(B, A), (H, J), (I, C)\}$	$B_{1s} = 16$ $H_{1s} = 13$ $I_{1s} = 13$ $A_{1s} = 16$ $C_{1s} = 13$ $J_{1s} = 11$	 	 $B \cap H \cap I = 3$  $A \cap C \cap J = 6$

5.4.2.1 Network Configurations

The networks are the same ones used in the three-association experiment of the autoassociation experimentation. The BAM is a homogeneous network, whereas the CPN is a forward-only type with the accretive learning mode. Two BPs are used: a two-layer and a three-layer with 2 hidden neurons. All the networks have 35 input and 35 output neurons, and their neurons are fully connected. Biases are also used in the hidden and output neurons of the BP networks.

5.4.2.2 Storing

The total error 0.03 is, again, used as the stopping criterion of the BP trainings, while 40 training cycles is the number of cycles to complete in the CPN trainings. Similar to the previous experiments, the CPN network uses the FSCL to prevent the under-utilization problem. Prior to training, the weights in BAM are set to zero, the weights in CPN are set to 0.1, and the weights in BP are initialized with uniformly distributed random values between -1.0 and 1.0 . The learning parameters of the CPN are 0.1 for the Kohonen learning rate and 1.0 for the Grossberg learning constant. These parameters are kept unchanged throughout the learning process. Similarly, the learning parameters of the BP network are 0.3 for the learning rate and 0.5 for the momentum rate, and they are kept unchanged. Tables 5.18a-b show the number of epochs completed for each training set.

Table 5.18a. Learning rates, momentum terms, total errors and training cycles of two-layer BP.

Training set	ϵ	α	total error	epochs
SBHI-ACJ	0.3	0.5	0.03	319
SBHI-AJC	0.3	0.5	0.03	330







Table 5.18b. Learning rates, momentum terms, total errors and training cycles of three-layer BP.

Training set	ϵ	α	total error	epochs
SBHI-ACJ	0.3	0.5	0.03	3204
SBHI-AJC	0.3	0.5	0.03	3237

5.4.2.3 Recalling









All the networks learn the training sets well. This is shown from the verification test. For every network, each training pattern is recalled through the associated pattern. The following test uses some incomplete input patterns. Since the two training sets have the same patterns, only one test pattern set is used. This test set is the one used to test the S_{BHI} set in the three-association experiment of the autoassociative experimentation. Some test results are given in Table 5.19a and Table 5.19b for the $S_{BHI-ACJ}$ and $S_{BHI-AJC}$ sets, respectively.

Table 5.19a. Test results of training set $S_{BHI-ACJ}$.

Input	H_d (bits)	BAM	BP	BP-2hid.	CPN
p05 	p05-B = 19 p05-H = 10 p05-I = 12	J	 Jnoisy	J	C
p07 	p07-B = 11 p07-H = 08 p07-I = 14	A	 Cnoisy	A	C
p20 	p20-B = 17 p20-H = 14 p20-I = 12	 (C^c) noisy	J	J	J

The test on BAM shows some inconsistent results. Let us compare the BAM outputs in Table 5.19a and Table 5.19b, for the input pattern $p05$ and $p20$. The result for the input pattern $p05$ in Table 5.19a shows that this input pattern is attracted to pattern I , since J is the associated pattern of I (see Table 5.17 for the association pairs). However, the result in Table 5.19b, for the same input pattern, shows that this input pattern is attracted to pattern H since J is now the associated pattern of H . This is also shown by the BP results for that input pattern. On the other hand, the test results show that the CPN gives consistent responses (see also Table 5.8d).

Table 5.19b. Test results of training set $S_{BHI-AJC}$.

Input	H_d (bits)	BAM	BP	BP-2hid.	CPN
p05 	p05-B = 19 p05-H = 10 p05- I = 12	J	 J_{noisy}	J	J
p06 	p06-B = 17 p06-H = 14 p06- I = 10	C	C	C	C
p07 	p07-B = 11 p07-H = 08 p07- I = 14	A	 J_{noisy}	J	J
p20 	p20-B = 17 p20-H = 14 p20- I = 12	 $(C^c)_{noisy}$	 C_{noisy}	J	C

5.5 Discussion

The experimental results show that BAM stores some spurious patterns. If we view the stored patterns as attractors in BAM [Kosk91], these spurious patterns are also stable attractors. It is suspected that these spurious patterns are created when the original patterns are learned [HoFP83]. From the autoassociation experiment employing two-association training sets, results show that these spurious patterns have some regularities. For instance, we can see the output of BAM for the input pattern *p02* in Table 5.4a as the intersection pattern of the stored patterns. This output pattern contains bits that belong to both stored patterns. Similarly, the output pattern of BAM for the input pattern *p08* in Table 5.4a contains all ON (1) bits that belong to only pattern *A* (in the experiments this pattern is called a unique pattern of *A*). These cases are also true for the other two-association training sets. This experiment shows that BAM also stores the intersection patterns and the unique patterns of the stored patterns besides their originals and their complements. The experiment also shows that BAM does not use the closeness in Hamming distance to select the stored pattern. This is shown from the fact that BAM associates pattern *p02* of Table 5.4b with pattern *A* even though pattern *H* is closer to the input pattern than pattern *A* (compare to the other networks that prefer pattern *H*). The selection of the stored pattern in BAM depends on the number of ON bits that belong to the stored pattern. Five conditions can be pointed out from the two-association experiment:

- (i) If the input pattern contains only the intersection bits of a stored pattern (e.g., *p02* in Table 5.4a or *p01* in Table 5.4c), the BAM network will output the intersection pattern;
- (ii) If the input pattern contains only the unique bits of a stored pattern (e.g., *p08* in Table 5.4a or *p06* in Table 5.4b), the BAM network will output a spurious pattern that comprises of all the unique bits of that pattern;

-
- (iii) If the input pattern contains only (an) intersection bit(s) and (a) unique bit(s) of a stored pattern (e.g., $p07$ in Table 5.4a or $p07$ in Table 5.4d), the output will be that stored pattern;
 - (iv) If the input pattern contains the same numbers of unique bits of both stored patterns (e.g., $p11$ in Table 5.4a or $p05$ in Table 5.4c), or it contains the same numbers of intersection bits and complement bits (e.g., $p22$ in Table 5.4a or $p10$ in Table 5.4f), the output will be the zero pattern. However, if the number of bits is different (e.g., $p15$ in Table 5.4a and $p04$ in Table 5.4c) the selection favours the one with more bits;
 - (v) If one of the stored pattern is a subset of the other (e.g., pattern H is a subset of pattern A in S_{AH} set), and the input pattern contains only (an) intersection bit(s) and (a) unique bit(s) of a stored pattern (e.g., $p02$ and $p08$ in Table 5.4b), the BAM network will output the superset pattern.

The three-association experiment shows different results. The spurious patterns do not represent the intersection patterns nor the unique patterns. However, BAM still gives similar responses for conditions (iii) and (iv) of the five conditions mentioned above, for certain inputs such as pattern $p18$ and $p19$ in Table 5.8a or $p19$ in Table 5.8d.

The BAM training of the S_{AFH} and S_{EFG} sets are unsuccessful. This may be due to the strong correlations among the stored patterns in the set. From Table 5.5, we can see that both sets have patterns that are separated only by 6 or less Hamming bits. In the S_{AFH} set, pattern A has the closest Hamming distances between the other two patterns. Similarly, pattern E in the S_{EFG} set has the closest Hamming distances between the other two patterns. If we view the problem in terms of the unique features those patterns have, pattern A in S_{AFH} has all unique features of pattern F and pattern H . From this point of view, it seems that the features that make the distinction among those patterns disappear or

at least weaken. If this conjecture is true, then through strengthening the unique features, BAM is expected to learn the set well. To show this, an additional pattern that contains only the unique bits are included in the set. Table 5.20 shows the enhanced S_{AFH} training set, and Table 5.21 shows the verification test result. The S_{AFH}^* denotes the enhanced S_{AFH} set, and pattern hf is the additional training pattern. The same technique is also employed to the S_{EFG} training set. The network is retrained using the enhanced S_{EFG} set (S_{EFG}^*). The training set is given in Table 5.22, and the results are shown in Table 5.23.

The verification test results of retraining show that BAM recalls the stored patterns perfectly. It seems that the failures of BAM to store the S_{AFH} and S_{EFG} sets are due to the deterioration of the unique features. One possibility that caused this condition might be because the patterns in the set are too close to each other. If the patterns are close to each other (the Hamming distance is small), the more akin the patterns would be. This implies that there is a minimal Hamming distance between the patterns that has to be maintained in order to recall all the stored patterns perfectly. However, it seems that the minimal Hamming distance depends on the number of stored patterns. As the number of stored patterns increases, the minimal Hamming distance also increases. The verification results for the S_{AH} set, the S_{AFH} and S_{EFG} sets, and the S_{BCHI} and S_{BHIL} sets support this conjecture. Hence, it seems possible to control the minimal Hamming distance through reinforcing the unique features of the patterns. Another method for achieving perfect recall of the stored patterns is through reinforcing the “weak” pattern using multiple training. The training set would contain more weak patterns than stronger ones. A more detailed discussion of this method can be found in [WaCM90] and [WaCM91].

Table 5.20. The enhanced S_{AFH} training set.





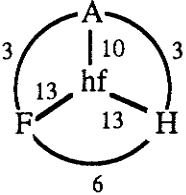
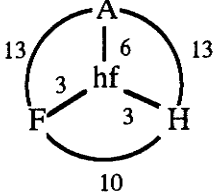
Training set	1s (bits)	H_d (bits)	\cap (bits)
$S_{AFH}^* = \{ (A, A), (F, F), (H, H), (hf, hf) \}$			
A  F  H  hf 	$A1s = 16$ $F1s = 13$ $H1s = 13$ $hf1s = 6$		

Table 5.21. Verification test results of BAM on the S_{AFH} and S_{AFH}^* training sets.













Input	Output		Energy	
	S_{AFH}	S_{AFH}^*	S_{AFH}	S_{AFH}^*
A 			-456	-472
F 			-456	-436
H 			-456	-436
hf 			-456	-72

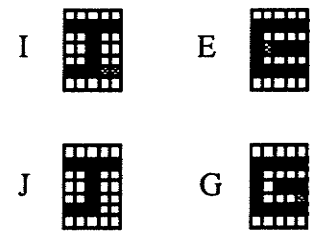
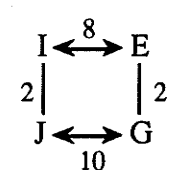
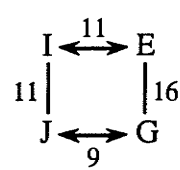
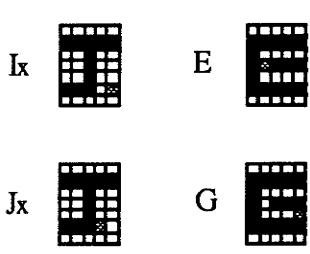
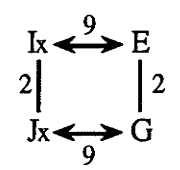
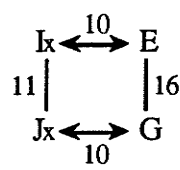
Table 5.22. The enhanced S_{EFG} training set.

Training set	1s (bits)	H_d (bits)	\cap (bits)
$S_{EFG}^* = \{ (E, E), (F, F), (G, G), (gf, gf) \}$	<p> $E_{1s} = 17$ $F_{1s} = 13$ $G_{1s} = 17$ $gf_{1s} = 5$ </p>		
<p> E F </p> <p> G gf </p>			

Table 5.23. Verification test results of BAM on the S_{EFG} and S_{EFG}^* training sets.

Input	Output		Energy	
	S_{EFG}	S_{EFG}^*	S_{EFG}	S_{EFG}^*
E			-595	-644
F			-595	-580
G			-595	-644
gf			-595	-68









Table 5.24. The S_{IJ-EG} and S_{IJx-EG} training sets.

Training set	I_s (bits)	H_d (bits)	\cap (bits)
$S_{IJ-EG} = \{(I, E), (J, G)\}$ 	$I_{1s} = 13$ $J_{1s} = 11$ $E_{1s} = 17$ $G_{1s} = 17$		
$S_{IJx-EG} = \{(I_x, E), (J_x, G)\}$ 	$I_{x1s} = 12$ $J_{x1s} = 12$ $E_{1s} = 17$ $G_{1s} = 17$		

The heteroassociation experiment results for BAM show that the unique feature of a pattern is also associated to the unique feature of the associated pattern. This explains the failure of training the S_{IJ-EG} set. To show this, we introduce another training set, S_{IJx-EG} , for comparison. This S_{IJx-EG} set has almost identical properties compared to the S_{IJ-EG} set, except for the unique features of the patterns. The S_{IJx-EG} training set and the verification results are shown in Table 5.24 and Table 5.25, respectively. The results in Table 5.25 show that BAM can recall all the patterns of the S_{IJx-EG} set. If we closely examine the unique features of I against J and the unique features of E against G , then there is an

inconsistency of bit flipping of the unique bits (each shown as a grey pixel in Table 5.24). In case of I and J , the two unique bits (represented by the grey pixels in the I projection of Table 5.24) are flipped together in the same direction to change from I to J . In case of E and G , the two unique bits (represented by the grey pixels in the E and G projections of Table 5.24) are flipped in the opposite direction to change from E to G . On the contrary, both training pairs in the S_{IJx-EG} set have the opposite direction of bit flipping to change from I_x to J_x and from E to G .

Table 5.25. Verification test result of BAM on the S_{IJ-EG} and S_{IJx-EG} training sets.

Input	Output		Energy	
	S_{IJ-EG}	S_{IJx-EG}	S_{IJ-EG}	S_{IJx-EG}
I 			-356	
J 			-352	
I_x 				-354
J_x 				-354

The experiments show that both the two-layer and the three-layer BP networks produce some spurious patterns. However, the two-layer BP has more spurious output patterns than the three-layer BP, especially for ambiguous patterns. The three-layer BP has fewer spurious patterns since it employs some hidden neurons. These hidden neurons perform as

feature detectors. Usually, the number of hidden neurons is less than the dimension of the input vector. In this case, all the information of an input pattern is squeezed through a narrow bottleneck, i.e., the hidden layer. The output layer of the three-layer BP takes the squeezed information to build the association at the output. Compared to the two-layer BP of which the output layer receives the information directly from the input layer, the output layer of the three-layer BP receives less information from the hidden layer. Thus, the output patterns of the three-layer BP differ to a lesser degree.

The three-layer BP experiment on the S_{BCHI} training set shows different results for different numbers of hidden neurons used. This is shown by the output patterns in response to the input pattern $p08$, $p14$, and $p15$ in Table 5.12a. The two-layer and three-layer BP with 3 hidden neurons as well as the CPN show similar results, while the three-layer BP with 2 hidden neurons disagrees, especially for pattern C (see $p14$, $p15$, $p22$, and $p23$ in Table 5.12a). If we examine $p14$ in Table 5.12a, it is possible that this pattern is associated with pattern B since $p14$ is also a subset of pattern B . However, in terms of the closest Hamming distance, this input pattern is closer to pattern C (see the first table of Table 5.12a). From the result we can see that the other networks associate the input pattern $p14$ with pattern C . One possible explanation why the three-layer BP with 2 hidden neurons gives a different result is that because this network uses less hidden neurons. If we view the hidden neurons in a three-layer BP network as feature detectors, this means that the network has fewer feature detectors used to discriminate the training patterns. Moreover, the highly correlated training patterns make training using less hidden neurons more difficult. The failure of training the S_{BHIL} set supports this explanation.

From the experimental results of the CPN, it is shown that CPN performs similar to a look-up table in which each stored pattern has a "radius of association" [SmSt90]. This means that the network associates the input pattern based on the closest distance between

the stored patterns. If the input pattern is within a radius of association of a stored pattern X then that input pattern will be associated with that stored pattern X . Another significant result of the CPN compared to the other networks is that the CPN does not produce any spurious output pattern, with the configuration used in the experiment. Since the accretive mode only permits one winner at a time, this limits the number of the output patterns. Thus, the output pattern is always the pattern stored in the Grossberg layer that is associated with the stored pattern in the Kohonen layer through the winning hidden neuron. If during training the target patterns are perfectly stored in the Grossberg layer, then the output patterns are always the perfect target patterns. The selection of the learning constants at the Grossberg layer may affect the storing of the target patterns.

5.6 Summary

This chapter describes and analyzes experiments and results of three selected neural networks, namely, BAM, BP, and CPN. The network configurations, the learning parameters and the training procedures are discussed. In the experiments, all the networks have been used as pattern associators of 7×5 binary pixel alphabet characters. The experiments consist of two individual experiments, namely, the autoassociation experiment and the heteroassociation experiment.

The first experiment uses the networks as autoassociators. Each network is trained using 7 sets of two-association training sets, 4 sets of three-association training sets, and 2 sets of four-association training sets. The trained networks are tested with several incomplete versions of the training patterns. The training and testing results are listed in table form.

The second experiment uses the networks as heteroassociators. The networks are trained using 6 sets of two-association training sets and 2 sets of three-association training

sets. Each trained network is tested and the results are listed in tables.

Finally, the training and testing results are discussed following the experiment description sections. Some failures in the training are addressed, and this is followed by a discussion of some methods to overcome these problems. Results indicate that the closeness (in Hamming distance sense) of the patterns to be stored in a BAM is important. The pattern closeness property becomes increasingly critical for the BAM as the number of the patterns to be stored increases. The experimental results for BP networks show that different network configurations give different outputs in response to a new input pattern. Results show that they produce some spurious patterns. However, three-layer BP produces less spurious patterns than two-layer BP. This may be attributable to hidden neurons in three-layer BP. The discussion also covers the experimental results of the CPN network. Results show that the CPN resembles a look-up table whose entries are separated by a radius of associations.

CHAPTER VI

CLASSIFICATION EXPERIMENT

Pattern classification is one of the most widely used applications of artificial neural networks. Such an application usually uses artificial neural networks for mapping patterns represented by points in a pattern space into the category numbers, $1, \dots, R$. For instance, suppose that the pattern space is an n -dimensional Euclidean space, E^n , and let the symbol C_i denote the set of points in E^n that are mapped into the category number i . Then, for each category number, there is a set of points in E^n denoted by one of the symbols C_1, C_2, \dots, C_R .

In this experiment, three neural network models, namely, BP, CPN, and ART-1 have been used as pattern classifiers. The networks are trained with 10 alphabet characters represented by 7×5 binary pixels, and they are tested with several noisy versions of those alphabet characters. The main objective of the experiment is to study the effects of varying critical parameters of the selected networks. Since each network has different critical parameters, the experiment is divided into three individual experiments, each for a specific model. The first experiment uses a two-layer BP and several three-layer BPs with different numbers of hidden neurons. The performance of the network with different configurations as well as different error criteria is studied. The second experiment employs the CPN model. The performance of the model for various numbers of training cycles is examined. The last experiment deals with the ART-1 model. This experiment focuses on the number of categories produced by the network for various vigilance parameters.

6.1 Pattern Sets

The basic pattern set for input vectors includes 10 Latin letters, i.e., from letter *A* to letter *J* in alphabetical order. These patterns, which are represented by 7×5 binary pixels [HYKi89], are put together in a set. From this basic pattern set, 5 additional sets of noisy patterns are produced through randomly flipping some of the pixels in each of the basic patterns. For instance, the first noisy pattern set contains basic patterns with 1-pixel distorted (flipped), while the last noisy pattern set contains basic patterns with 5-pixel distorted. For convenience, we use S_0 to denote the set of the 10 basic patterns, where $S_0 = \{A, B, C, D, E, F, G, H, I, J\}$, and S_1 to S_5 to denote the five noisy pattern sets. Notice that the subscripts indicate the number of pixels flipped. For example, the set with two pixels flipped is denoted by $S_2 = \{A_2, B_2, C_2, D_2, E_2, F_2, G_2, H_2, I_2, J_2\}$. Each noisy pattern set contains 50 noisy patterns, i.e., five noisy patterns of each basic pattern. Thus, there are 250 noisy patterns obtained from the 10 basic (undistorted) patterns.

For the BP and CPN models, the basic pattern set containing the ten undistorted patterns is used as the training set, while the five noisy pattern sets are used as the test pattern sets. Since BP and CPN are supervised networks, they require a target pattern for each training pattern. The target pattern has a 10-element vector format because the networks are used as pattern classifiers of 10 distinct classes. Each of the target vectors is defined as a vector with one element of value 1, and value 0 for the remaining elements. Using this fashion, we can assume that each output neuron represents a distinct class. Through training, the networks are forced to map an input pattern into one of the 10 categories represented by the output neurons.

The target patterns are not used in the ART-1 experiments. This model does not require predetermined output vectors because the categories are formed during the learning phase.

Since there is no training nor testing phases, all the pattern sets (the ten basic pattern set and the noisy pattern sets) are included in the learning phase.

6.2 Measurement Technique

The main task of a pattern classifier is to classify correctly any input pattern into a predetermined class. In our case, the input patterns include the 10 basic patterns and all the noisy versions. A correctly classified input means that a basic pattern and its noisy versions are classified into one of the 10 categories. However, misclassification can occur. For instance, a noisy version of a specific pattern, say A , can be misclassified into a class (category) that belongs to a different pattern, say B . The number of misclassifications a classifier makes is used as the basis for measuring the performance of the classifier. The empirical *error rate* can be defined as the ratio of the number of errors (misclassifications) to the number of cases (patterns) examined, as expressed by

$$E_R = \frac{\text{number of errors}}{\text{number of cases}} \quad (6.1)$$

where E_R is the error rate. For an asymptotically large number of cases that converges in the limit to the actual population distribution, the error rate given by (6.1) is statistically defined as the *true error* of the classifier. However, since the experiment only uses 260 cases, which is relatively small, the major question is then whether the true error can be extrapolated from the empirical error rates calculated from small sample results. This question has been addressed and discussed comprehensively in [WeKu90].

A method that seems fit for measuring the error rate using a limited number of samples is the *train-and-test error rate estimation*. The method splits the samples into a training set and a testing set. The training set is used to design the classifier, and the testing set is used

for testing only. The error rate measured on the test cases is called the *test sample* error rate. This method is applicable to BP and CPN models, since they are supervised models which require two phases; training phase and testing phase. Note that the *train-and-test error rate* method will give an estimation within 5% error tolerance for 250 samples [WeKu90].

The *train-and-test error rate estimation* is inappropriate for measuring the error rate of the ART-1 model. The reason is that the ART-1 model is an unsupervised network, and has only a learning phase. It does not know or use class-membership information [Kosk92], hence the total number of categories produced is not known a priori. Moreover, each of the categories (denoted by numbers) is not explicitly associated to one of the true classes (i.e., the classes that have been defined based on the 10 basic patterns). This gives some difficulties to measure the error rate. One way to measure the performance of ART-1 is through examining the clusters it produces. For instance, a category created by ART-1 is associated to a specific class, say *A*, if the majority of the members in that category are also members of class *A*. This approach can be done through applying a *confusion matrix*. The *confusion matrix* lists the correct classification (true classification) against the predicted classification (i.e., the actual classes that are produced by the network) for each class. Typically, the number of predicted classes (categories) is the same as the number of true classes. Hence, the *confusion matrix* is a square matrix. The number of correct predictions for each class falls along the diagonal of the matrix. All other numbers are the number of errors for a particular type of misclassification error. However, since the number of predicted categories produced by ART-1 can be either smaller or larger than the number of the true categories, an extension of the *confusion matrix* is used. Instead of having the same number of categories between the predicted class and the true class, the *confusion matrix* may have a different number of predicted categories while it still has a

fixed number of true categories (10 categories in this experiment). The measurement is done for different vigilance parameters.

6.3 BP Experiment

The BP experiment focuses on the performances of two-layer and three-layer BPs. The performance is measured based on the *test sample* error rate. The error rate is measured for different stopping error criteria used in the training and for different number of hidden neurons used in the three-layer BP.

6.3.1 Network Configuration

All the BP networks used in this experiment have 35 input neurons and 10 output neurons since the input pattern is represented by 7×5 arrays of binary pixels and there are 10 different categories (one category for each basic pattern). While there are no hidden neurons in the two-layer BP, the three-layer BP uses different numbers of hidden neurons: from 2 to 20 hidden neurons in increments of 2, and then 9, 24, 28, 30, 34, 35, 36, 60, 100, 102, and 104 hidden neurons. Note that a three-layer BP with 9 hidden neurons is also included following the proof [SaAn91, HuHu91, MeMR91] that a network with one hidden layer (three-layer BP) can exactly implement an arbitrary training set with p training patterns, provided that $p-1$ hidden neurons are used. Since there are 10 patterns in the training set (i.e., the basic pattern set), thus 9 hidden neurons is sufficient to learn the training set. Every neuron in the hidden layer and output layer uses a sigmoid function and a bias. Each neuron in a layer is connected to every neuron in the next layer, thus the connections between the adjacent layers are fully connected.

6.3.2 Training

There are two kinds of training. The first one is training using a fixed stopping criterion. This training is applied to every network. A total error 0.03 is chosen for the fixed stopping error criterion. The second one is training using various stopping error criteria. This kind of training is only applied to the two-layer network and the three-layer network with 4, 9, and 60 hidden neurons. The stopping error criterion varies from 0.9 to 0.1 in decrements of 0.2, from 0.09 to 0.01 in decrements of 0.02, and from 0.009 to 0.003 in decrements of 0.002. This training is continuous for a specific network. For instance, the network is first trained for the largest stopping error criterion, that is 0.9. When the total error of the network is less than or equal to this number, the training stops, and the weights of the network are saved. The training is then continued for the next stopping error criterion, that is 0.7. Again, the weights of the network are saved when the total error of the network reaches this number. This is done continuously until the last stopping error criterion, that is 0.003. This continuous training forms a series of training. Using this scheme, the initialization is done only once at the very beginning of the training series. Note that a uniform distribution of random values between -0.1 and $+1.0$ is used for the weight initialization. This is done for all the network before each training (or each training series).

The weights are updated after all the training patterns are presented (i.e., after each epoch). For every epoch, the patterns are presented in random order. Throughout the training, the learning and momentum parameters are kept constant. Most of the BP networks in the experiment use 0.3 for the learning rate and 0.5 for the momentum parameter. However, larger networks such as the three-layer BP with 60, 100, 102, and 104 hidden neurons tend to oscillate using this learning rate. Therefore, a smaller learning rate, 0.1, is used instead.

Most of the training is completed in about 10,000 or less epochs, using the fix stopping error criterion 0.03. This includes the larger networks with a learning rate 0.1. However, the three-layer BP networks with 2 and 4 hidden neurons complete their training in about 33,000 epochs. Training using various stopping error criteria require proportional number of epochs. For smaller stopping error criterion, more training time is required, thus a larger number of epochs is required. Table 6.1a-c list the numbers of epochs completed by two-layer and three-layer BP with 4, 9, and 60 hidden neurons.

Table 6.1a. Learning rate, momentum term, total errors and training cycles of the two-layer BP.

Training set	ϵ	α	total error	epochs
S ₀	0.3	0.5	0.900	902
			0.700	908
			0.500	914
			0.300	926
			0.100	981
			0.090	991
			0.070	1023
			0.050	1092
			0.030	1305
			0.010	2755
			0.009	3010
			0.007	3742
			0.005	5064
0.003	8148			

Table 6.1b. Learning rate, momentum term, total errors and training cycles of the three-layer BP with 4 hidden neurons.

Training set	ϵ	α	total error	epochs
S ₀	0.3	0.5	0.900	636
			0.700	715
			0.500	814
			0.300	997
			0.100	1722
			0.090	1833
			0.070	2145
			0.050	2701
			0.030	3998
			0.010	10528
			0.009	11616
			0.007	14722
			0.005	20306
0.003	33317			

Table 6.1c. Learning rate, momentum term, total errors and training cycles of the three-layer BP with 60 hidden neurons.

Training set	ϵ	α	total error	epochs
S ₀	0.1	0.5	0.900	168
			0.700	189
			0.500	222
			0.300	288
			0.100	554
			0.090	594
			0.070	705
			0.050	895
			0.030	1312
			0.010	3187
			0.009	3483
			0.007	4314
			0.005	5770
0.003	9041			

We can use the total error in BP to indicate convergence. If the training leads the total error to a value equal or less than the specified stopping error criterion, it is said that the training converges. The selection of the stopping error criterion determines the error tolerance the network may have. Typically, for a classification problem such as mapping a binary pattern into a category, a small stopping error criterion (less than 1.0) will lead to a successful training. In other words, the trained network will produce no false categories given the training set as input. This is also supported by the verification results on the trained networks. All the BP networks (including the networks trained using the stopping error as large as 0.9) correctly classify all the training patterns.

6.3.3 Testing

There are three tests: (i) test for measuring the error rate of different network configurations; (ii) test for measuring the error rate of various stopping error criteria; (iii) test for measuring the error rate of the network in response to various amounts of noise in the input patterns (different numbers of bits flipped). The first and the third tests measure the error rates of the networks trained using the fix stopping error criterion, 0.03. The tests use all five test sets containing noisy versions. Thus, there are 250 samples involved in the test. The results are given as graphs, and are shown in Fig. 6.1, Fig. 6.2, and Figs. 6.3a-e. The vertical axis of the graph represents the error rate calculated from (6.1), while the horizontal axes in Fig. 6.1, 6.2, and 6.3a-e represent the numbers of hidden neurons of the network, the stopping error criteria, and the numbers of bits flipped (noise) of the input patterns, respectively. In Fig. 6.1, the network with 0 (zero) hidden neurons is actually the two-layer BP, while the remaining are three-layer BP networks. The horizontal axis of the graph in Fig. 6.2 should actually represent the increment of the training time or the training cycles. However, since completion of training is indicated by the stopping error criterion, and the measurement of the error rate is done for each stopping

error criterion, the stopping error criteria is used for marking the horizontal axis. Accordingly, the arrangement of the stopping error criteria is (from left to right on the horizontal axis) from the largest value to the smallest value.

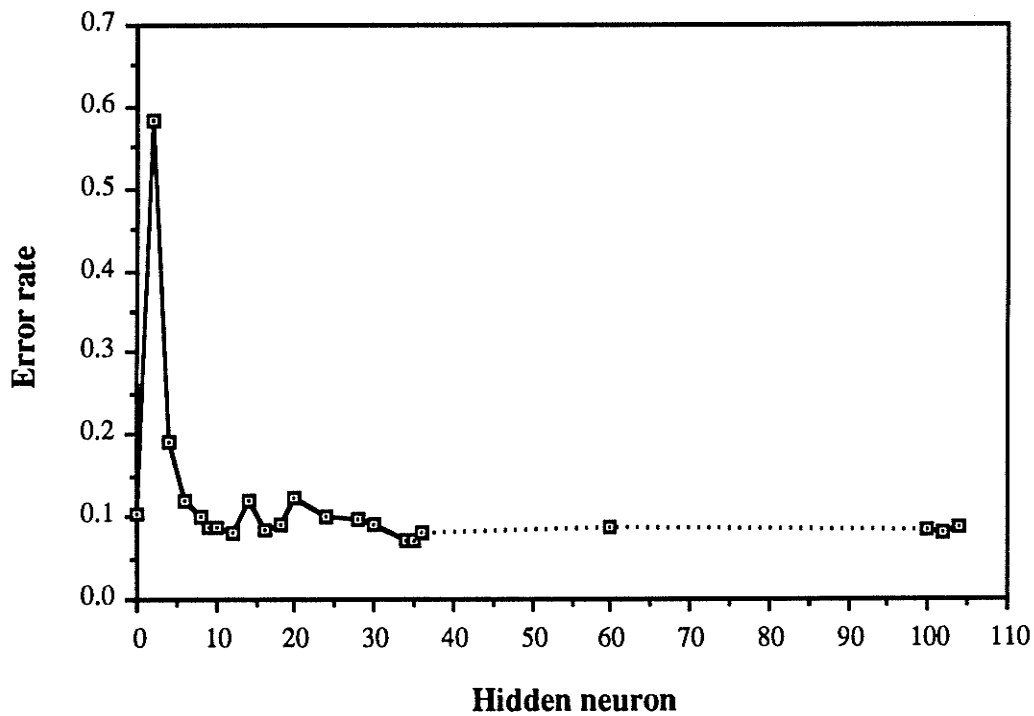


Fig. 6.1. Error rate versus number of hidden neurons.

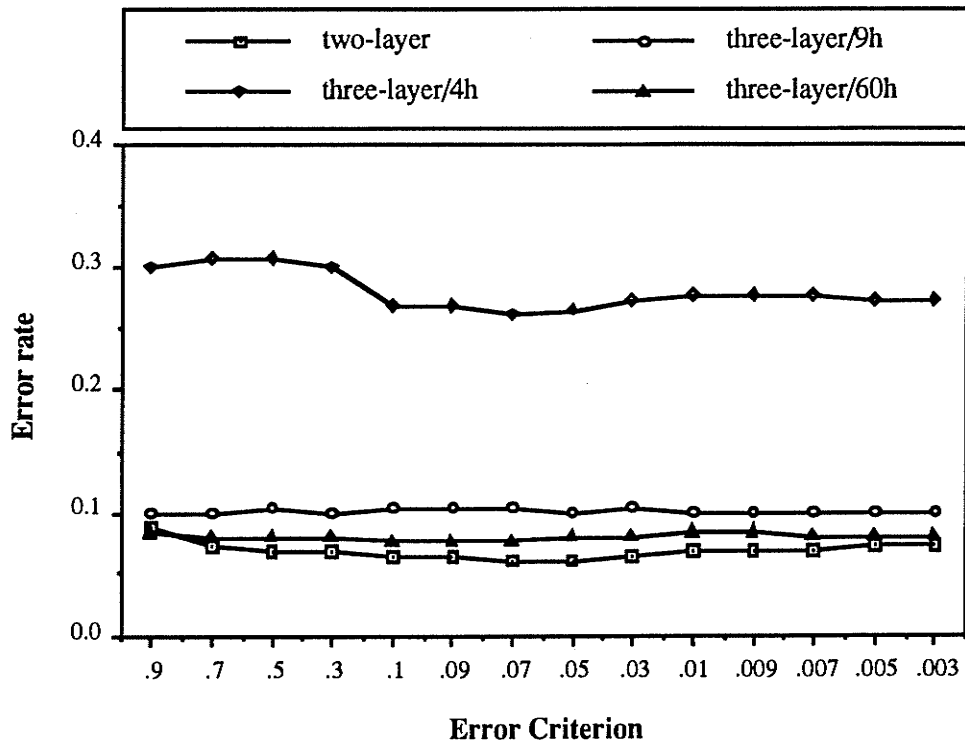


Fig. 6.2. Error rate versus stopping error criterion.

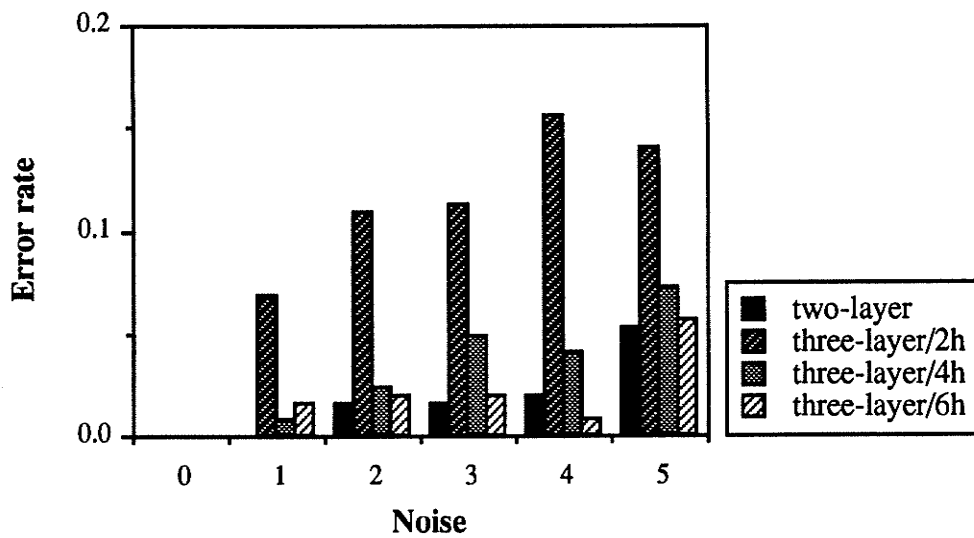


Fig. 6.3a. Error rate versus number of bits flipped (noise).

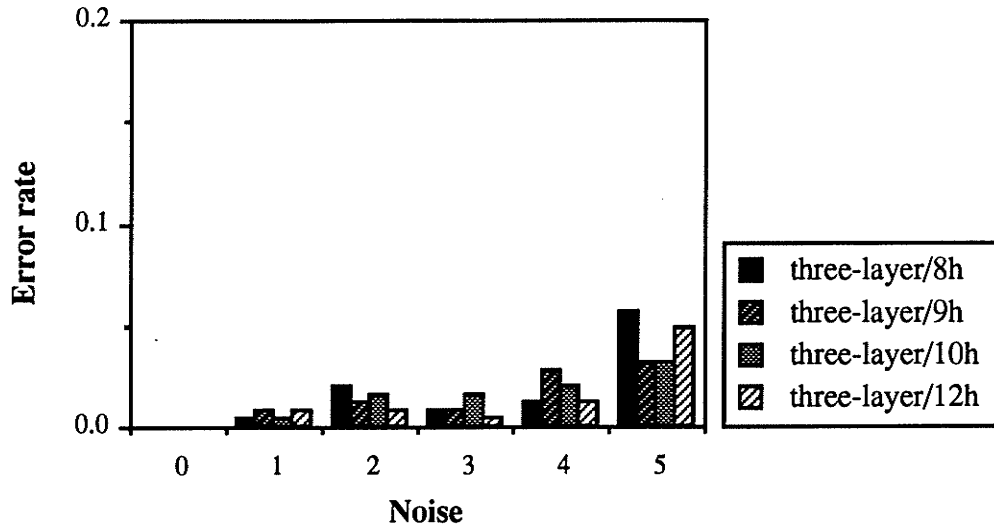


Fig. 6.3b. Error rate versus number of bits flipped (noise).

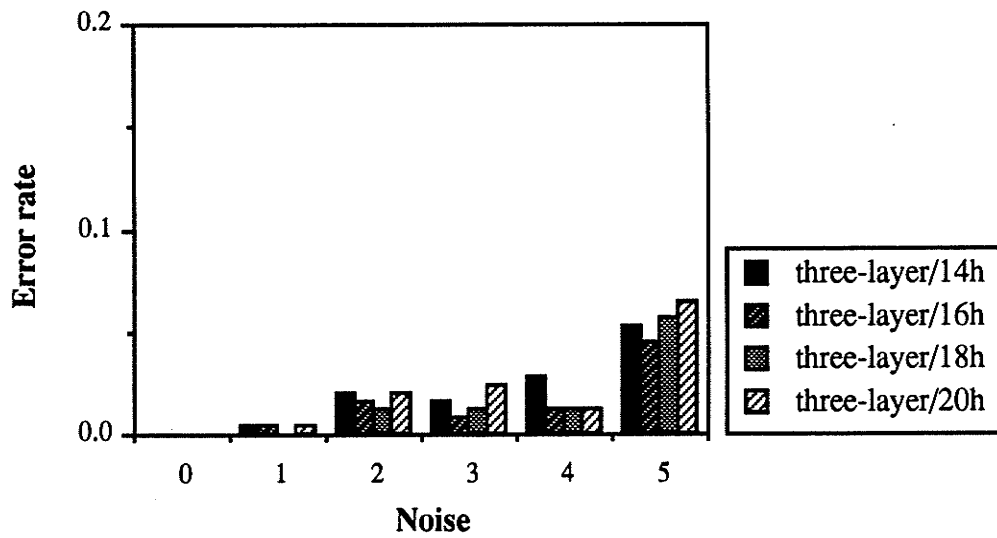


Fig. 6.3c. Error rate versus number of bits flipped (noise).

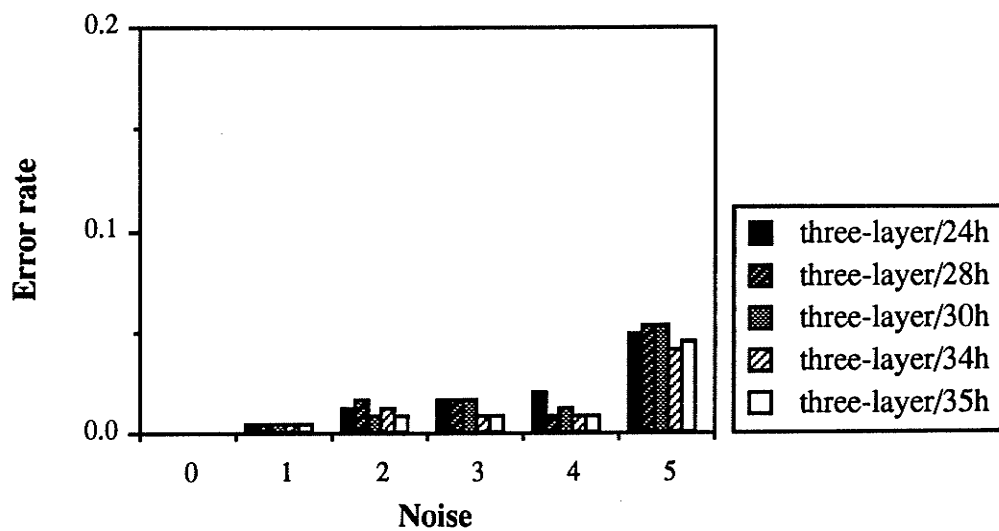


Fig. 6.3d. Error rate versus number of bits flipped (noise).

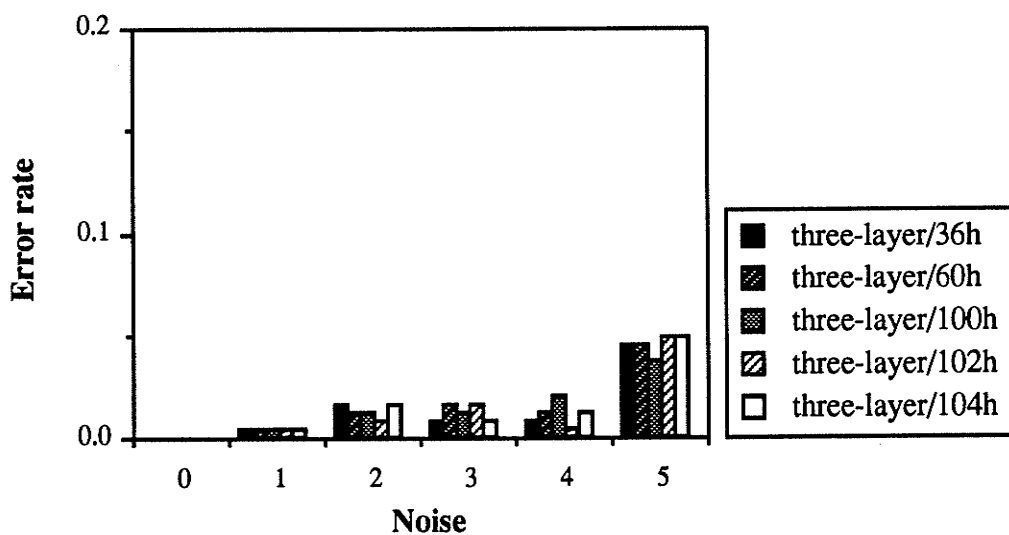


Fig. 6.3e. Error rate versus number of bits flipped (noise).

6.4 CPN Experimentation

The experiment is designed to measure the test sample error rate of the CPN network trained on different training cycles.

6.4.1 Network Configuration

The network is a forward-only CPN. It has 35 input neurons of which each input neuron corresponds to each binary pixel of the 7×5 binary projection, 10 output neurons that represent the categories, and 10 hidden neurons. There is some evident that the number of hidden neurons of a CPN running on the *accretive* mode corresponds to the memory capacity of the CPN [Hech89, KiIL90]. Since the network is trained to classify the input patterns into 10 distinct classes, at least 10 hidden neurons are required. The connections between adjacent layers are full connections. A typical forward-only CPN topology is shown in Fig. 3.5 (Chapter III).

6.4.2 Training

The training uses eight different training cycles (epochs), i.e., from 10 to 80 in increments of 10. In each training, the network employs three different Kohonen learning rate settings, namely 0.02, 0.1, and 0.5, while the Grossberg parameters are kept fix, 1.0. All the training uses a random order presentation of the input patterns. For the network with the Kohonen learning rate 0.1, a sequential order presentation is also used besides the random order presentation. Prior trainings, the weights of the network are initialized with a constant value, 0.1.

6.4.3 Testing

Two tests have been done on all the trained networks, namely, (i) test for measuring the error rate of the network trained with a different number of epochs (training cycles), (ii) test for measuring the error rate of the network in response to various amounts of noise in the input patterns (different numbers of bits flipped). The second test is done only for the network trained using the Kohonen learning rate of 0.1, and employs the random order presentation. In this test, the error rate is measured for different epochs. Both tests use all the 250 test patterns, which are the noisy versions of the training patterns. Fig 6.4 and Fig. 6.5a-b, show the results of the first and the second test, respectively.

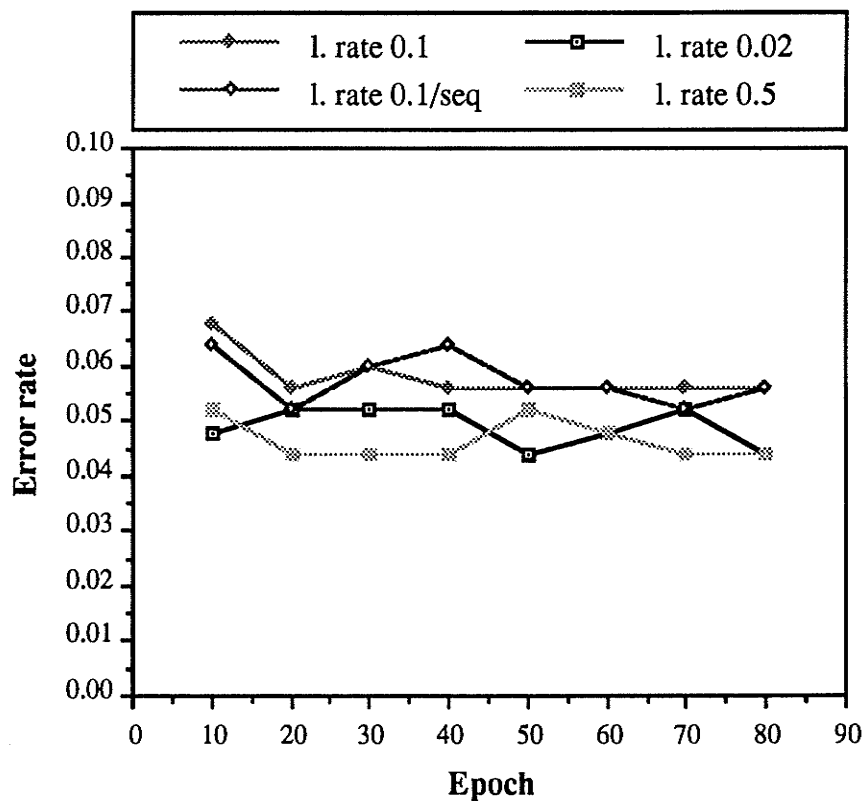


Fig. 6.4. Error rate versus number of training cycles (epochs).

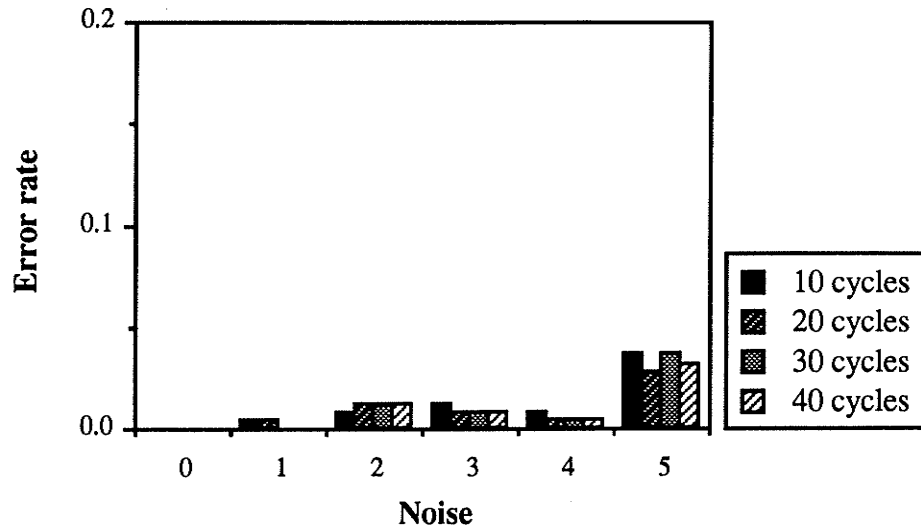


Fig. 6.5a. Error rate versus number of bits flipped (noise), for the network trained using learning rate 0.1 with random order presentation.

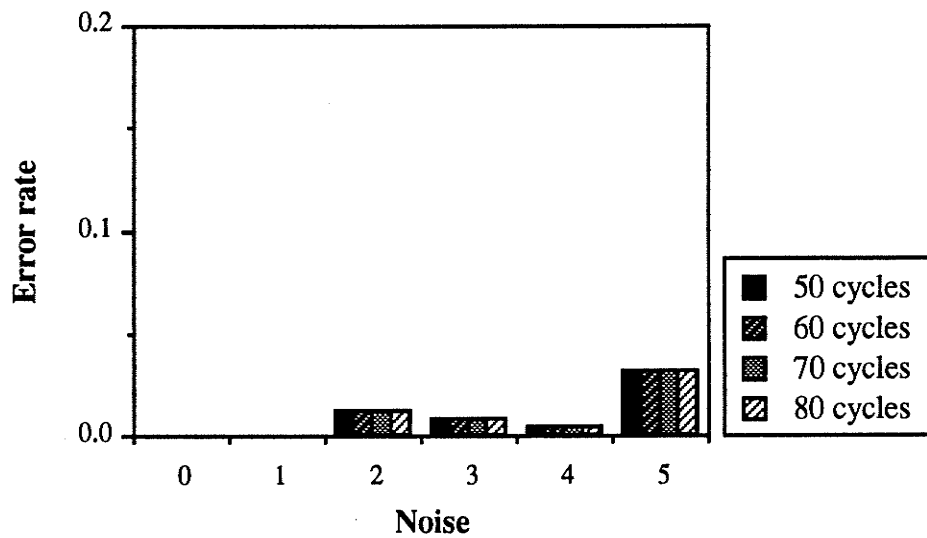


Fig. 6.5b. Error rate versus number of bits flipped (noise), for the network trained using learning rate 0.1 with random order presentation.

6.5 ART-1 Experiment

The objective of the ART-1 experiment is to examine the categories produced by the network for different vigilance values. The stability of the learning is also examined through observing how the category number varies against the repetition steps of the basic set applied to the network.

6.5.1 Network Configuration

A typical ART-1 network has two layers: the *comparison* layer and the *recognition* layer (see Fig. 3.6 in Chapter III). The *comparison* layer is responsible for receiving data from outside world, and hence it is meant also as the input layer. Therefore, the number of neurons in the *comparison* layer must match the input vector dimension. For this experiment, the *comparison* layer employs 35 neurons since the input vector is a 7×5 binary pixel. The *recognition* layer is meant also as the output layer. This *recognition* layer is designed as a competitive layer. So, there is only one activated neuron in this layer in response to a given input vector to the network (assuming that the maximum capacity has not been reached). In other words, the network will categorize (cluster) the input vectors, so that each cluster is represented by one of the neurons in the *recognition* layer. However, there is no definite guide to estimate how many categories the network will make from a certain population of input data. Ideally, the number of categories may grow infinitely, for unrestricted input data. Practically, the number of categories grows to the maximum capacity of the network, that is, the number of neurons in the *recognition* layer. For this experiment, since there are 260 input samples, at most 260 categories may be created by the network (assuming that all the input samples are unique, and each unique category is assigned to a unique input sample). Hence the network has 260 neurons in the

recognition layer. The other components of the network (except the vigilance value) remain intact.

6.5.2 Learning

There are two main points to be examined in the experiment: the stability of the learning and the categories produced by the network. Examination of the stability of the learning is done through observing how the category number varies against the repetition presentations of the basic pattern set (S_0) to the network. The basic pattern set (S_0) is applied to the network five times in a sequential order, and the categories produced are examined for different vigilance values (i.e., 0.5, 0.6, 0.7, 0.8, 0.9, and 1.0). Tables 6.2a-f show the categories produced, represented by number 0 to 9 (for 10 classes) in the second column, and the members (the input vectors) are listed in the same row with each category. The first column of the table contains the index of the presentation, denoted as cycle. The number inside the parenthesis beside every name of the input vector represents the number of times the network searches for a proper *top-down expectation* (refer to section 3.5.2 in Chapter III). A zero means that no search has taken place; that is, the input pattern directly activates the category that best represents it.

Table 6.2a. The categories produced for vigilance 0.5.

Cycle	Category	Input Pattern				
1	0	A ₀ (0)	B ₀ (0)	C ₀ (0)	D ₀ (0)	F ₀ (0)
	1	E ₀ (1)	G ₀ (1)	H ₀ (0)		
	2	I ₀ (2)	J ₀ (0)			
2	0	B ₀ (0)	C ₀ (0)	D ₀ (0)	F ₀ (0)	
	1	A ₀ (0)	E ₀ (0)	G ₀ (0)	H ₀ (0)	
	2	I ₀ (0)	J ₀ (0)			
3	0	B ₀ (0)	C ₀ (0)	D ₀ (0)	F ₀ (0)	
	1	A ₀ (0)	E ₀ (0)	G ₀ (0)	H ₀ (0)	
	2	I ₀ (0)	J ₀ (0)			
4	0	B ₀ (0)	C ₀ (0)	D ₀ (0)	F ₀ (0)	
	1	A ₀ (0)	E ₀ (0)	G ₀ (0)	H ₀ (0)	
	2	I ₀ (0)	J ₀ (0)			
5	0	B ₀ (0)	C ₀ (0)	D ₀ (0)	F ₀ (0)	
	1	A ₀ (0)	E ₀ (0)	G ₀ (0)	H ₀ (0)	
	2	I ₀ (0)	J ₀ (0)			

Table 6.2b. The categories produced for vigilance 0.6.

Cycle	Category	Input Pattern
1	0	A ₀ (0) B ₀ (0) C ₀ (0) F ₀ (0)
	1	D ₀ (1) E ₀ (1) G ₀ (0) I ₀ (0) J ₀ (0)
	2	H ₀ (2)
2	0	C ₀ (0) F ₀ (0)
	1	J ₀ (0)
	2	A ₀ (0) B ₀ (2) H ₀ (0)
	3	D ₀ (3) E ₀ (2) G ₀ (0) I ₀ (1)
3	0	C ₀ (0) F ₀ (0)
	1	J ₀ (0)
	2	A ₀ (0) B ₀ (0) H ₀ (0)
	3	I ₀ (0)
	4	D ₀ (4) E ₀ (3) G ₀ (0)
4	0	C ₀ (0) F ₀ (0)
	1	J ₀ (0)
	2	A ₀ (0) B ₀ (0) H ₀ (0)
	3	I ₀ (0)
	4	D ₀ (0) E ₀ (0) G ₀ (0)
5	0	C ₀ (0) F ₀ (0)
	1	J ₀ (0)
	2	A ₀ (0) B ₀ (0) H ₀ (0)
	3	I ₀ (0)
	4	D ₀ (0) E ₀ (0) G ₀ (0)

Table 6.2c. The categories produced for vigilance 0.7.

Cycle	Category	Input Pattern
1	0	A ₀ (0) B ₀ (0) F ₀ (0)
	1	C ₀ (1) D ₀ (0)
	2	E ₀ (2) G ₀ (1) H ₀ (1)
	3	I ₀ (3) J ₀ (0)
2	0	F ₀ (0)
	1	C ₀ (0) D ₀ (0)
	2	H ₀ (0)
	3	I ₀ (0) J ₀ (0)
	4	A ₀ (4) B ₀ (2)
	5	E ₀ (5) G ₀ (2)
3	0	F ₀ (0)
	1	C ₀ (0) D ₀ (0)
	2	H ₀ (0)
	3	I ₀ (0) J ₀ (0)
	4	A ₀ (0) B ₀ (0)
	5	E ₀ (0) G ₀ (0)
4	0	F ₀ (0)
	1	C ₀ (0) D ₀ (0)
	2	H ₀ (0)
	3	I ₀ (0) J ₀ (0)
	4	A ₀ (0) B ₀ (0)
	5	E ₀ (0) G ₀ (0)
5	0	F ₀ (0)
	1	C ₀ (0) D ₀ (0)
	2	H ₀ (0)
	3	I ₀ (0) J ₀ (0)
	4	A ₀ (0) B ₀ (0)
	5	E ₀ (0) G ₀ (0)

Table 6.2d. The categories produced for vigilance 0.8.

Cycle	Category	Input Pattern
1	0	A ₀ (0) B ₀ (0) F ₀ (0)
	1	C ₀ (1)
	2	D ₀ (2)
	3	E ₀ (3) G ₀ (1) I ₀ (1) J ₀ (0)
	4	H ₀ (4)
2	0	F ₀ (0)
	1	C ₀ (0)
	2	B ₀ (1) D ₀ (0)
	3	J ₀ (0)
	4	A ₀ (0) H ₀ (0)
5	E ₀ (5) G ₀ (2) I ₀ (2)	
3	0	F ₀ (0)
	1	C ₀ (0)
	2	B ₀ (0) D ₀ (0)
	3	J ₀ (0)
	4	A ₀ (0) H ₀ (0)
	5	I ₀ (0)
6	E ₀ (6) G ₀ (3)	
4	0	F ₀ (0)
	1	C ₀ (0)
	2	B ₀ (0) D ₀ (0)
	3	J ₀ (0)
	4	A ₀ (0) H ₀ (0)
	5	I ₀ (0)
6	E ₀ (0) G ₀ (0)	
5	0	F ₀ (0)
	1	C ₀ (0)
	2	B ₀ (0) D ₀ (0)
	3	J ₀ (0)
	4	A ₀ (0) H ₀ (0)
	5	I ₀ (0)
6	E ₀ (0) G ₀ (0)	

Table 6.2e. The categories produced for vigilance 0.9.

Cycle	Category	Input Pattern
1	0	A ₀ (0) F ₀ (0)
	1	B ₀ (1) D ₀ (1)
	2	C ₀ (2)
	3	E ₀ (3) G ₀ (1)
	4	H ₀ (4)
	5	I ₀ (5) J ₀ (0)
2	0	F ₀ (0)
	1	D ₀ (0)
	2	C ₀ (0)
	3	E ₀ (0) G ₀ (0)
	4	H ₀ (0)
	5	J ₀ (0)
	6	A ₀ (6)
	7	B ₀ (7)
8	I ₀ (8)	
3	0	F ₀ (0)
	1	D ₀ (0)
	2	C ₀ (0)
	3	E ₀ (0) G ₀ (0)
	4	H ₀ (0)
	5	J ₀ (0)
	6	A ₀ (0)
	7	B ₀ (0)
8	I ₀ (0)	
4	0	F ₀ (0)
	1	D ₀ (0)
	2	C ₀ (0)
	3	E ₀ (0) G ₀ (0)
	4	H ₀ (0)
	5	J ₀ (0)
	6	A ₀ (0)
	7	B ₀ (0)
8	I ₀ (0)	
5	0	F ₀ (0)
	1	D ₀ (0)
	2	C ₀ (0)
	3	E ₀ (0) G ₀ (0)
	4	H ₀ (0)
	5	J ₀ (0)
	6	A ₀ (0)
	7	B ₀ (0)
8	I ₀ (0)	

Table 6.2f. The categories produced for vigilance 1.0.

Cycle	Category	Input Pattern
1	0	A ₀ (0) F ₀ (0)
	1	B ₀ (1)
	2	C ₀ (2)
	3	D ₀ (3)
	4	E ₀ (4)
	5	G ₀ (5)
	6	H ₀ (6)
	7	I ₀ (7) J ₀ (0)
2	0	F ₀ (0)
	1	B ₀ (0)
	2	C ₀ (0)
	3	D ₀ (0)
	4	E ₀ (0)
	5	G ₀ (0)
	6	H ₀ (0)
	7	J ₀ (0)
	8	A ₀ (8)
	9	I ₀ (9)
3	0	F ₀ (0)
	1	B ₀ (0)
	2	C ₀ (0)
	3	D ₀ (0)
	4	E ₀ (0)
	5	G ₀ (0)
	6	H ₀ (0)
	7	J ₀ (0)
	8	A ₀ (0)
	9	I ₀ (0)

Table 6.2f. (continued) The categories produced for vigilance 1.0.

Cycle	Category	Input Pattern
4	0	F ₀ (0)
	1	B ₀ (0)
	2	C ₀ (0)
	3	D ₀ (0)
	4	E ₀ (0)
	5	G ₀ (0)
	6	H ₀ (0)
	7	J ₀ (0)
	8	A ₀ (0)
	9	I ₀ (0)
5	0	F ₀ (0)
	1	B ₀ (0)
	2	C ₀ (0)
	3	D ₀ (0)
	4	E ₀ (0)
	5	G ₀ (0)
	6	H ₀ (0)
	7	J ₀ (0)
	8	A ₀ (0)
	9	I ₀ (0)

The objective of the second experiment is to examine the categories produced. The input patterns are the 10 basic patterns (from letter *A* to *J*) and the 250 noisy versions of them. The categories produced after learning (the predicted categories) are listed against the true categories in the *confusion matrix* shown in Tables 6.3a-c. The predicted categories are shown in the rows of the matrix, while the 10 true categories (denoted by letter *A* to *J* in alphabetical order) are shown in the columns of the matrix. The members of a category are sorted according to the true classes. For example, the input pattern A_5 (which is a noisy version of the basic pattern *A*) that belongs to a category, say category 1, is placed under the row denoted by the predicted category 1 and the column denoted by the true class *A*. However, for our convenience, the complete names of the patterns are not listed

VI. CLASSIFICATION EXPERIMENT

explicitly in the matrix. Simply, each element of the matrix represents the total number of patterns that belong to a specific true class and a unique predicted category. The unsorted results that contain the name of each input vector can be found in Appendix B.

Table 6.3a. The true versus predicted categories for vigilance 0.5.

Predicted Category	True Category									
	A	B	C	D	E	F	G	H	I	J
0			1							
1								1		
2										2
3			2							
4								2		
5				3		1	1			
6						1				
7			2							
8						2				4
9		1		1				1		
10			1							
11				1						
12									5	12
13			2	1						
14	1	2								
15			1	2	1					
16	3	1			1	17				
17						1		4		
18			1							
19								1		
20									1	4
21			4						3	
22									1	1
23		3								
24		1		1			1		2	
25				16	1		1			
26	1					1		4		
27					1	1	1			
28										
29					1				9	
30	1									
31							1			
32	1					1	1			
33					2					
34								1		
35	1					1			3	3
36					1					
37					1					
38		8	12	1	7		9		2	
39	4	2						12		
40		5					4			
41		3			3		4			
42	14				7		3			

VI. CLASSIFICATION EXPERIMENT

Table 6.3b. The true versus predicted categories for vigilance 0.8.

Predicted Category	True Category									
	A	B	C	D	E	F	G	H	I	J
0						1				
1			1							
2				1						
3										1
4								1		
5						1				
6	2									
7										1
8						2				
9		1								
10			2							
11							2			
12								1		
13	1									
14	1	2								
15				1						
16									1	
17										1
18				1						
19									1	
20										1
21										
22								1		
23				1						
24			1							
25				1						
26					3					
27		1								
28				1						
29								1		
30								1		
31										1
32		1								
33			1							
34			1							
35										1
36	1									
37									2	
38	1									
39		1								
40		1								
41				1						
42				1						
43					1					
44						1				
45						2				
46							1			
47			1							

VI. CLASSIFICATION EXPERIMENT

Table 6.3b. (continued) The true versus predicted categories for vigilance 0.8.

Predicted Category	True Category									
	A	B	C	D	E	F	G	H	I	J
48										
49								2		
50								1		
51										1
52										1
53										2
54						1				
55		1			1					
56									1	
57	1									
58								1		
59		1								
60		1		1						
61				1						
62			1							
63			1				1			
64			3							
65				3						
66				1						
67					2					
68					1					
69						1				
70						1				
71						1				
72						1				
73							1			
74					1		1			
75								1		
76							1			
77								1		
78								1		
79										1
80									1	
81									2	
82									1	
83										1
84										1
85		5								
86			2							
87								2		
88								1		
89				7						
90					6					
91							1			
92										2
93				1						
94			8							
95	2									

VI. CLASSIFICATION EXPERIMENT

Table 6.3b. (continued) The true versus predicted categories for vigilance 0.8.

Predicted Category	True Category									
	A	B	C	D	E	F	G	H	I	J
96						1				
97					1					
98							1			
99								7		
100								1		
101									7	
102										8
103		2								
104			1							
105							1			
106						1				
107		1								
108				2						
109						2				
110		1					1			
111									1	
112									2	
113										1
114	1				1	10				
115										1
116	1									
117								2		
118			2							
119				1						
120				1						
121					2		1			
122							1			
123							2			
124					1		5			
125									1	
126									1	
127									4	
128							2			
129		7								
130	8									
131					1		1			
132										1
133							1	1		
134					1				1	
135	1									
134										
137			1							
138					3					
139					1		1			
140							1			
141	3									
142	2									
143	1									

VI. CLASSIFICATION EXPERIMENT

Table 6.3c. The true versus predicted categories for vigilance 1.0.

Predicted Category	True Category									
	A	B	C	D	E	F	G	H	I	J
0						1				
1		1								
2			1							
3				1						
4					1					
5							1			
6								1		
7										1
8						1				
9			1							
10				1						
11								1		
12										1
13	1									
14								1		
15		1								
16		1								
17		1								
18			1							
19			1							
20				1						
21				1						
22				1						
23						1				
24						1				
25			1							
26						1				
27							1			
28							1			
29							1			
30								1		
31								1		
32										1
33									1	
34										1
35										1
36										1
37										1
38	1									
39						1				
40		1								
41			1							
42				1						
43				1						
44			1							
45					1					
46						1				
47						1				
48							1			
49								1		
50								1		
51										1

Table 6.3c. (continued) The true versus predicted categories for vigilance 1.0.

Predicted Category	True Category									
	A	B	C	D	E	F	G	H	I	J
52									1	
53										1
54	1									
55	1									
56	1									
57		1								
58		1								
59			1							
60				1						
61				1						
62					1					
63					1					
64					1					
65						1				
66							1			
67								1		
68								1		
69										1
70									1	
71										1
72	1									
73	1									
74		1								
75		1								
76		1								
77			1							
78			1							
79				1						
80				1						
81				1						
82					1					
83						1				
84						1				
85						1				
86						1				
87							1			
88			1							
89							1			
90								1		
91								1		
92								1		
93										1
94									1	
95										1
96										1
97	1									
98								1		
99		1								
100			1							
101				1						
102					1					
103			1							

Table 6.3c. (continued) The true versus predicted categories for vigilance 1.0.

Predicted Category	True Category									
	A	B	C	D	E	F	G	H	I	J
104						1				
105							1			
106								1		
107									1	
108										1
109									1	
110										1
111										1
112								1		
113	1									
114	1									
115		1								
116		1								
117						1				
118		1								
119			1							
120				1						
121				1						
122				1						
123					1					
124					1					
125					1					
126					1					
127						1				
128						1				
129						1				
130							1			
131							1			
132							1			
133							1			
134								1		
135								1		
136								1		
137									1	
138									1	
139									1	
140									1	
141										1
142										1
143										1
144						1				
145		1								
146			1							
147				1						
148					1					
149						1				
150							1			
151								1		
152									1	
153										1
154	1									
155	1									

Table 6.3c. (continued) The true versus predicted categories for vigilance 1.0.

Predicted Category	True Category									
	A	B	C	D	E	F	G	H	I	J
156	1									
157	1									
158	1									
159		1								
160		1								
161		1								
162		1								
163		1								
164			1							
165			1							
166			1							
167			1							
168			1							
169				1						
170				1						
171				1						
172				1						
173				1						
174					1					
175					1					
176					1					
177					1					
178					1					
179						1				
180						1				
181						1				
182						1				
183						1				
184							1			
185							1			
186							1			
187							1			
188								1		
189								1		
190								1		
191									1	
192									1	
193									1	
194									1	
195									1	
196										1
197										1
198										1
199							1			
200	1									
201			1							
202				1						
203								1		
204									1	
205	1									
206	1									
207		1								

Table 6.3c. (continued) The true versus predicted categories for vigilance 1.0.

Predicted Category	True Category									
	A	B	C	D	E	F	G	H	I	J
208		1								
209			1							
210				1						
211					1					
212							1			
213							1			
214							1			
215									2	
216										1
217	1									
218			1							
219				1						
220							1			
221							1			
222								1		
223									1	
224									1	
225	1									
226	1									
227		1								
228		1								
229					1					
230					1					
231								1		
232									1	
233									1	
234										1
235	1									
236			1							
237				1						
238					1					
239						1				
240							1			
241								1		
242									1	
243									1	
244	1									
245					1					
246									1	
247	1									
248		1								
249	1									
250			1							
251	1									
252					1					
253					1					
254									1	
255		1								
256					1					
257			1							
258					1					

6.6 Discussion

The BP experimental results (Fig. 6.1) show that the performance of the three-layer BP network (in terms of the error rate) is a function of the number of hidden neurons. The smallest error rate (0.072, i.e., 7.2% of the total test samples are misclassified) is achieved by the three-layer BP with 34 and 35 hidden neurons, while the three-layer BP with 2 hidden neurons has the largest error rate (0.584, i.e., 58.4% of the total test samples are misclassified). The graph shows that the error rate decreases exponentially as the number of hidden neurons increases. In other words, a three-layer BP network with many hidden neurons tends to generalize better than networks with few hidden neurons. This result is consistent with the results of other researchers [SiDo91]. The two-layer BP network gives an error rate of 0.104 (10.4% errors), which is similar to the error rate of three-layer BP with 8 hidden neurons (i.e., 0.1). The average error rate of two-layer and three-layer BP with a number of hidden neurons above 6 is 0.1. In other words, these networks correctly classify 90% of the total test samples. From Fig.6.3a-e, it is shown that most of the errors are due to the test patterns with 5 bits flipped.

The results imply that the problem of *overspecialization* or *overfitting* does not occur. The *overfitting* problem is assumed to occur when a network with many hidden neurons tends to memorize the training data instead of generalizing [WeKu90]. The network with an *overfitting* problem performs well with the training data, but it performs poorly with the new data (testing data). However, this problem does not occur in our case. In fact, the results show the opposite. The performance of the three-layer BP network increases (shown by the decreasing error rate in Fig. 6.1) as the number of hidden neurons increases from 2 to 9, and the performance is relatively constant as the number of hidden neurons increases beyond 9. One possible explanation why the *overfitting* problem does not occur in our experiment is that because the task is a classification of binary patterns.

To show this let us view the hidden neurons as hyperplanes that partition d -dimensional space (hyperspace) into various regions [Nils90]. It has been proved that a network with one hidden layer can exactly implement an arbitrary training set with p training patterns, provided that $p-1$ hidden neurons are used [SaAn91, HuHu91, MeMR91]. Since there are only 10 training patterns used (each pattern is a representative of a category), 9 hidden neurons and hence 9 hyperplanes are sufficient to partition the d -dimensional space into 10 regions. Thus, there will be redundant partitions created by the excess hyperplanes in a three-layer network with a number of hidden neurons greater than 9. In other words, there can be more than one hyperplane that separates two regions. Since the inputs are binary patterns (i.e., the pattern points are the vertices of a hypercube), the positions of the hyperplanes in the hyperspace are not critical. Furthermore, since the output is also binary with each neuron representing a unique category and it follows a “winner-takes-all” fashion (i.e., the selection of a category is based on the biggest value of the output neuron), the exact match of the output value to the target value is unnecessary. This is also supported by the experimental result for various stopping error criteria, as shown in Fig. 6.2.

The results (Fig. 6.2) show that the problem of *overtraining* does not occur. *Overtraining* problem occurs when a network is presented with the same set of training patterns many times, and the network tends to memorize the training data [Hech89]. This problem is similar to *overfitting* problem, except that the variable is now the stopping error criteria instead of the number of hidden neurons. Fig. 6.2. shows no significant increase in the error rate.

The results of the CPN experiments (Fig. 6.4) show a smaller error rate compare to the error rate of BP. On average, the error rate is 0.05 (i.e., 5% errors from the total test samples). In other words, the CPN network correctly classifies 95% of the total test

samples. Fig. 6.4 shows that the error rates for different learning rates vary within 4% of the average error rate, 0.05. Similarly, the error rates for different epochs also vary. This shows that the network still makes some adjustments to its weight vectors. For the network with learning rate of 0.1 and using random order pattern presentation, learning is stable after 40 epochs. On the other hand, for the other CPN networks, learning is unstable. If we view each weight vector $w_i^{(l)}$ at the Kohonen layer (first layer) as a point in \mathbf{R}^n approaching the centroid \bar{x}_j of a pattern class, this weight vector is wandering about the centroid instead of reaching the centroid. This particular phenomenon has been discussed thoroughly in [CIRa90]. For CPN, one possibility to prevent this problem is through utilizing a learning rate that is gradually reduced towards zero during training [Hech89, Wass89].

Figs. 6.5a-b show that most of the errors are due to the test patterns with 5 bits flipped. This result is similar to the result of the BP experiment (Figs. 6.3a-e), except that CPN produces less errors than BP. The error rate is constant after 40 epochs. For 2, 3, and 4 bits flipped (noise), the error rates differ by 1 test sample. This difference becomes significant as the number of bits flipped increases to 5 bits.

In supervised learning, the performance of the model is determined by the result of the training. For instance, if the network does not converge during training, then it is likely that the network will perform poorly in the test case. For an unsupervised model such as ART-1, this boundary between the training phase and the testing phase is not as clear. ART-1 learns the pattern as the network is faced with the pattern at its input. The question is how well the network learns the pattern.

One major point relating to this issue is the learning stability of the network. Tables 6.2a-f show the categories produced by the network with different vigilance settings. Through examining the members of each category, we can see that the members of each

category remain constant after the third cycle of the presentation. In other words, the network learns the set after presented with the same set for three times.

The tables also show the amount of time the network has to search for a proper category, as shown by the number in parenthesis beside each pattern's name. For higher vigilance values such as 0.9 and 1.0, the searching time for certain patterns is close to the number of categories produced. This is expected since for a high vigilance, say 1.0 (the extreme case), the network will search for a perfect match between the input pattern and the *top-down expectation*. This forces the network to search the entire available categories, and may lead to a creation of a new category, if the input pattern is totally a different one from the ones that have been learned previously. Nevertheless, this condition occurs only for the first 3 cycles of the pattern presentations. The next time the pattern is presented, the network will recognize the pattern directly without any searching process (denoted by the zero in the parenthesis).

The results show that the subset pattern always replaces the superset pattern. For example, this is shown in Table 6.2f for a vigilance of 1.0. Pattern A_0 is replaced by pattern F_0 , and pattern I_0 is replaced with pattern J_0 . For the second presentation, patterns F_0 and J_0 remain stable in categories 0 and 7, respectively. On the other hand, the network has to search new categories for patterns A_0 and I_0 that are categories 8 and 9, respectively. Similar cases can be found also for lower vigilance values. This phenomenon is due to the way the network stores the template in the *top-down LTM* (refer to Eq. 3.28 in Chapter III). Using Eq. 3.28, the network always stores the intersection of the input pattern and the previous template as the new template in the selected *top-down LTM*. If the new input pattern is a subset of the template pattern, then this new pattern becomes the template pattern for the next trial. In other words, the learning of the network is stable only at the subset pattern. This characteristic becomes significant for higher

vigilance values (the extreme case is for vigilance 1.0).

Tables 6.3a-c show the distribution of the input patterns among the categories created for vigilance 0.5, 0.8, and 1.0, respectively. One significant result for different vigilances is the number of categories created by the network. The number of categories increases to the maximum number of input patterns as the vigilance value increases to 1.0. This is a typical characteristic of ART-1 since the vigilance determines the tolerance of a mismatch between the input pattern and the *top-down expectation*. High vigilance forces the network to search for new categories in response to small tolerance of a mismatch. To the extreme, the network will classify each unique input pattern into each unique category.

Ideally, the network should result in the same number of true categories, which is 10 categories. However, this requirement is never accomplished for the problem given. The reason is that a small number of categories (obtained for a small vigilance value) directly competes with the number of overlaps. An overlap occurs when more than one pattern of distinct category is classified into the same category. To show this let us compare the result for vigilance of 0.5 and 0.8 in Table 6.3a and 6.3b, respectively. Most of the predicted categories for vigilance of 0.5 contain more patterns from different categories, while the predicted categories for vigilance of 0.8 have less patterns from different categories. However, each predicted category created by the network with vigilance of 0.8 contains less patterns too, even for patterns that belong to the same true category. Hence, the minimum number of categories with no overlaps (i.e., 10 categories) is never accomplished by the ART-1 network without some modifications. An improved version of ART-1, which is called ARTMAP, can be found in [CaGR91].

6.7 Summary

This chapter describes and analyzes experiments and results of three selected neural networks, namely, BP, CPN, and ART-1. The network configurations, the learning parameters and the training procedures are discussed. The networks have been used as pattern classifiers of 7×5 binary pixels of 10 alphabet characters. The performances and the learning characteristics of the networks are studied.

The experimental results of the BP experimentation show that a three-layer BP network with many hidden neurons tends to generalize better than networks with few hidden neurons. For the given problem, the two-layer and three-layer BP with a number of hidden neurons above 6 classify 90% of the total test samples correctly, while the CPN network shows a 95% classification. Results also show that no *overfitting* nor *overtraining* problem is detected in the BP training. The experimental results for ART-1 show that the minimum number of categories with no overlaps (i.e., 10 categories) is never accomplished for the given problem. The number of categories increases to the maximum number of input patterns as the vigilance value increases to 1.0. Furthermore, results also show that the learning of the network is stable only at the subset pattern, and this characteristic becomes significant for higher vigilance values.

CHAPTER VII

CONCLUSIONS AND RECOMMENDATIONS

The work described in this thesis was motivated by the need for developing unified benchmarks and a unified methodology of benchmarking in order to facilitate better understanding and applicability of artificial neural network models for particular applications. The thesis places emphasis on the comparative study of the characteristic features of the BAM, BP, CPN, and ART-1 artificial neural network models. This study has led to the development of a neural network software simulator on a Macintosh computer. An object-oriented design methodology has been used in the design process. This design approach reduces the programming complexity and provides a reusable and maintainable system for further expansion. The program has been extensively used in the experimental study of the selected models.

The study has two parts: a literature study and an experimental study. The objective of the literature study is to identify the characteristic features of selected neural network models and to provide the basis for the experimental study. For completeness, the characteristic features of the selected models are listed below.

A Bidirectional Associative Memory (BAM):

- is a two-layer nonlinear feedback network with supervised learning;
- the layers are fully connected;
- employs neurons with a nonlinear function such as sigmoid, threshold;

- requires bipolar (no analog) data representation;
- functions as an associative memory;
- stores the association of the data pair at local energy minima; and
- trainings through summing the outer-product of the data pairs.

The BAM has the following advantages:

- it requires no training parameters;
- no iterative process during training; and
- requires one training trial.

The BAM has the following disadvantages:

- it has limited memory capacity; $N = \frac{m}{2 \log_2 m}$, where m is the number of neurons in the smaller layer;
- requires mutually orthogonal of the data pairs for perfect recall;
- produces *spurious* patterns;
- bipolar BAM encodes the complements of the patterns automatically; and
- can be confused if like inputs are associated with unlike outputs (vice versa).

A Backpropagation (BP):

- is a multilayer nonlinear feedforward network with supervised learning;
- the layers may be fully connected or sparsely connected;
- employs biases and a nonlinear function in the hidden and output neurons;
- functions as a mapper; and
- training through minimizing the error function (search for a *global minimum*).

The BP has the following advantages:

- it accepts analog and binary data representations; and
- is capable of approximating any continuous mapping using a network with as few

as one hidden layer and an arbitrary bounded and non-constant activation function.

The BP has the following disadvantages:

- the training can be trapped in *local minima*;
- it requires some adjustment of the training parameters;
- requires iterative process during training;
- the training time grows exponentially with the size of the network;
- has a tendency to forget previously learned patterns;
- may become *overtrained* or *overfitted*; and
- difficult to select the optimal configuration for a specific task.

A Counterpropagation (CPN):

- is a three-layer nonlinear feedforward network with unsupervised learning;
- the layers are fully connected;
- the architecture is a combination of Kohonen's *self-organizing feature map* (SOFM) of and the Grossberg's *outstar* structure;
- uses the *competitive learning* scheme at the hidden layer; and
- employs a linear function in the hidden and output neurons.

The CPN has the following advantages:

- it accepts analog and binary data representations;
- functions as a statistically optimal self-programming look-up table;
- training through self-organizes the weights to approximate the mapping; and
- it requires a finite number of training trials.

The CPN has the following disadvantages:

- it requires some adjustment of the training parameters;

- requires iterative process during training;
- has a tendency to forget previously learned patterns; and
- has a problem of *under-utilization*.

An Adaptive Resonance Theory 1 (ART-1):

- is a two-layer nonlinear feedback network with unsupervised learning;
- the layers are fully connected;
- functions as a binary classifier;
- uses the *competitive learning* scheme at the second layer; and
- uses *top-down expectation* scheme (active attentional focus).
- has a more complex architecture;

The ART-1 has the following advantages:

- real-time (on-line) learning;
- learning through self-organizes the weights according to the past experiences;
- it is capable to preserve previously learned patterns while continuing to learn new patterns; and
- requires a finite number of learning trials.

The ART-1 has the following disadvantages:

- requires binary data representation;
- requires some adjustment of the learning parameters; and
- has some difficulty to adjust the *vigilance* parameter.

The objective of the experimental study is to confirm or to discover some capabilities, and particularly, their abilities to solve specific problems. Since there are two distinct classes of neural network models, the experimental study must employ two benchmark

problems, namely, the associative memory and the pattern classification problems.

The first experiment, the associative memory experiment, includes the BAM, BP, and CPN models. This experiment confirms some of the characteristics of the selected models and reveals other characteristics that have not been identified a priori. The experimental results of BAM indicate that the closeness (in Hamming distance sense) of the patterns to be stored becomes increasingly critical as the number of the patterns increases. However, the selection of the stored patterns seems inconsistent with the selection based on the closest Hamming distance. Results show that the spurious patterns produced by two associations have some regularities. These spurious patterns may form as either unique patterns, the intersection patterns, zero patterns, or the union of the complements of the stored patterns. Some of the BAM training is unsuccessful (i.e., the training results in imperfect recall of the stored patterns). This problem is likely to occur when some of the patterns are subsets of the others. Furthermore, the results show that the unique bits of a pattern at the input layer are associated with the unique bits of the associated pattern at the output layer. The experimental results of BP show that different network configurations give different outputs in response to a new input pattern. The results also show that the networks produce some spurious patterns. However, a three-layer BP network produces fewer spurious patterns than a two-layer BP network. This may be attributable to hidden neurons in the three-layer BP network. The failure of some of the BP training indicates that the training of a three-layer network with less than $N-1$ hidden neurons (where N is the number of distinct patterns) is likely to fail. The experimental results of CPN show that no spurious patterns occurs. This may be attributable to the *competitive learning* scheme used in the CPN model. The CPN performs similar to a look-up table whose entries are separated by a radius of association.

The second experiment, the classification experiment, includes the BP, CPN, and ART-1 models. The experimental results of BP show that a three-layer BP network with many hidden neurons tends to generalize better than networks with few hidden neurons. For the given binary classification problem, the two-layer BP gives 10.4% error rate, while the three-layer BP with 2 hidden neurons gives 58.4% error rate and the one with 35 hidden neurons gives 7.2% error rate. This means that, on the average, the two-layer BP and the three-layer BP with a number of hidden neurons above 6 classify around 90% of the total test samples correctly. Results also show that no *overfitting* nor *overtraining* problem is detected in the BP training. The experimental results of CPN show that the network with 10 hidden neurons gives 5% error rate. In other words, the CPN network shows a 95% classification. The results show that the learning can be unstable for certain learning rate settings. This problem, however, may be overcome through utilizing a learning rate that is gradually reduced towards zero during training. The experimental results for ART-1 show that the minimum number of categories with no overlaps (i.e., 10 categories) is never accomplished for the given problem. The number of categories increases to the maximum number of input patterns as the vigilance value increases to 1.0. Furthermore, results also show that the learning of the network is stable only at the subset pattern, and this characteristic becomes significant for higher vigilance values.

In summary, the thesis has contributed to technical knowledge by:

- (a) providing a better understanding of the selected models through the development of a unified benchmarking and an experimental study;
- (b) the use of object-oriented design methodology to develop a reusable and expandible artificial neural network software simulator;
- (c) development of an integrated software simulator of selected artificial neural

network models, namely BAM, BP, CPN, and ART-1;

- (d) providing a basis for the comparative study of selected artificial neural networks through experimental study; and
- (e) introducing the use of an extension of the *confusion matrix* to measure the performance of the ART-1 model.

Further research is required to either improve or extend this work, including:

- (a) the study of other artificial neural network models with the implementation of the models based on the existing software;
- (b) the development of some additional tools for the neural network software simulator, such as graphic displays for input and output patterns, and Hinton diagram to display the weight values; and
- (c) modification of the existing software to incorporate an artificial neural network co-processor (accelerator) board.

REFERENCES

- [AKCM90] S. C. Ahalt, A. K. Krishnamurthy, P. Chen, and D. E. Melton, "Competitive learning algorithms for vector quantization," *Neural Networks*, vol. 3, pp. 277-290, 1990.
- [AlKe90] L. G. Allred and G. E. Kelly, "Supervised learning techniques for backpropagation networks," *Proc. Second Int. Joint Conf. on Neural Networks*, vol. I, pp. 721-728, 1990.
- [Appl88] Apple Computer, Inc., *Inside Macintosh*. Cupertino, CA: Addison-Wesley, vol. 1-5, 1988.
- [Baba89] N. Baba, "A new approach for finding the global minimum of error function of neural networks," *Neural Networks*, vol. 2, pp. 367-373, 1989.
- [Bulm91] D. Bulman, "Refining candidate objects." *Computer Language*, vol. 8, No. 1, pp. 30-39, January, 1991.
- [BuLu90] P. Burrascano and P. Lucci, "A learning rule eliminating local minima in multilayer perceptrons," *Proc. Second Int. Joint Conf. on Neural Networks*, vol. I, pp. 865-868, 1990.
- [CaGr88] G. A. Carpenter and S. Grossberg, "The ART of adaptive pattern recognition by a self-organizing neural network," *IEEE Computer*, vol. 21, No. 3, pp. 77-88, March, 1988.
- [CaGR91] G. A. Carpenter, S. Grossberg, and J. H. Reynolds, "ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network," *Neural Networks*, vol. 4, pp. 565-588, 1991.

-
- [Carp89] G. A. Carpenter, "Neural network models for pattern recognition and associative memory," *Neural Networks*, vol. 2, pp. 243-257, 1989.
- [Card90] H. Card, Artificial Neural Networks. *Lecture Notes*, University of Manitoba, Canada, 1989.
- [ClRa90] D. M. Clark and K. Ravishankar, "Acquisition and decay rates in synaptically coded memory," *Neural Networks*, vol. 3, pp. 525-533, 1990.
- [Colo90] E. Colombini, "C Workshop: Designing an object with THINK C," *MacTutor: The Macintosh Programming Journal*, vol. 6, No. 8, August, 1990.
- [CrHJ91] Y. Crama, P. Hansen, and B. Jaumard, "Detection of spurious states of neural networks," *IEEE Trans. on Neural Networks*, vol. 2, No. 1, pp. 165-168, Jan., 1991.
- [DaMa88] P. A. Darnell and P. E. Margolis, *Software Engineering in C*. New York: Springer-Verlag, 1988, pp. 612.
- [DARP88] DARPA, *DARPA Neural Network Study*. Virginia: AFCEA International Press, 1988, 629 pp.
- [DeSi88] D. DeSieno, "Adding a conscience to competitive learning," *Proc. of IEEE Int. Conf. on Neural Networks*, vol. 1, pp. 117-124, 1988.
- [EbDo90] R. C. Eberhart and R. W. Dobbins, *Neural Network PC Tools: A Practical Guide*. San Diego, CA: Academic Press, 1990, 414 pp.
- [Fere91] K. Ferens, et. al., "A neural network Hamming encoder and decoder," *IASTED Int. Conf. Computers, Electronics, Communication & Control*, Calgary, AB; Apr. 8-10, 1991.
- [Funa89] K. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, vol. 2, pp. 183-192, 1989.

-
- [GaJo79] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: Freeman, 1979, 340 pp.
- [Gross76] S. Grossberg, "Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors," *Biol. Cybernetics*, vol. 23, pp. 121-134, 1976.
- [Gros88a] S. Grossberg, *Neural Networks and Natural Intelligence*. Cambridge (MA): MIT Press, 1988.
- [Gros88b] S. Grossberg, "Nonlinear neural networks: Principles, mechanisms, and architectures," *Neural Networks*, vol.1, pp. 17-61, 1988.
- [Hagi90] M. Hagiwara, "Novel back propagation algorithm for reduction of hidden units and acceleration of convergence using artificial selection," *Proc. Second Int. Joint Conf. on Neural Networks*, vol. I, pp. 625-630, 1990.
- [HaHe88] K. Haines, and R. Hecht-Nielsen, "A BAM with increased information storage capacity," *Proc. Second IEEE Int. Conf. on Neural Networks*, vol. I, pp. 181-190, 1988.
- [Hebb49] D.O. Hebb, *The Organization of Behavior: A Neurophysiological Theory*. New York: Wiley, 1949.
- [Hech87] R. Hecht-Nielsen, "Counterpropagation networks," *Applied Optics*, vol. 26, pp. 4979-4984, Dec. 1, 1987.
- [Hech88] R. Hecht-Nielsen, "Neurocomputing : picking the human brain," *IEEE Spectrum*, pp. 36-41, March 1988.
- [Hech89] R. Hecht-Nielsen, *Neurocomputing*. Menlo Park, CA; Addison-Wesley, 1989, 433 pp.

-
- [HoFP83] J. J. Hopfield, D. I. Feinstein, and R. G. Palmer, "'Unlearning' has a stabilizing effect in collective memories," *Nature*, vol. 304, pp. 158-159, 1983.
- [Holl90] I. M. Holland, "Cutting through the buzz words," *The C++ Insider*, vol. 1, No. 1, October, 1990.
- [Horn91] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, pp. 251-257, 1991.
- [HoSW90] K. Hornik, M. Stinchcombe, and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks," *Neural Networks*, vol. 3, pp. 551-560, 1991.
- [HuHu91] S. Huang and Y. Huang, "Bounds on the number of hidden neurons in multilayer perceptrons," *IEEE Trans. on Neural Networks*, vol. 2, No. 1, pp. 47-55, Jan., 1991.
- [Hump90] B. Humpert, "Bidirectional associative memory with several patterns," *Proc. Second Int. Joint Conf. on Neural Networks*, vol. I, pp. 741-750, 1990.
- [HuYK90] S. Hudon, Y. Yan and W. Kinsner, "A comparative study of neural network models," *Proc. 7th Intern. Conf. Math. and Computer Modelling* (Chicago, Aug. 2-5, 1989), vol. 14, pp. 300-304, 1990.
- [Ill89] W. T. Illingworth, "Beginners guide to neural networks," *IEEE AES Magazine*, pp. 44-49, September, 1989.
- [InKi91] A. Indrayanto and W. Kinsner, "Object oriented C implementation of BAM, BP, CPN and ART-1 neural network models," *Technical Report, DEL91-7*; University of Manitoba, Aug. 29, 1991.
- [JoHo87] W. P. Jones and J. Hoskins, "Back-Propagation : A generalized delta learning rule," *Byte*, pp. 155-162, October, 1987.

-
- [Judd90] J. S. Judd, *Neural Network Design and the Complexity of Learning*. Cambridge, MA: The MIT Press, 1990, 149 pp.
- [KiHu90] W. Kinsner and S. Hudon, "Benchmark patterns and a projection program for neural networks," *Technical Report, DEL90-2*; University of Manitoba, Mar., 1990.
- [KiIL90] W. Kinsner, A. Indrayanto, & A. Langi, "A study of Backpropagation, Counterpropagation, and Adaptive Resonance Theory neural network models," *Proc. 12th Int. Conf. IEEE Engineering in Medicine & Biology Soc.*, IEEE CH2936-3/90, vol. 3, pp. 1471-1473, 1990.
- [KiYa89] W. Kinsner and Y. Yan, "A model of the carotid vascular system with stenosis at the carotid bifurcation," *Proc. 7th Intern. Conf. Mathl. Comput. Modelling*, Chicago, MI, Aug. 2-5, 1989.
- [Koho88] T. Kohonen, "An introduction to neural computing," *Neural Networks*, vol. 1, pp. 3-16, 1988.
- [Koho90] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464-1480, Sept. 1990.
- [Kosk87a] B. Kosko, "Competitive adaptive Bidirectional Associative Memories," *Proc. of IEEE Int. Conference on Neural Networks*, vol. II, pp. 759-766, 1987.
- [Kosk87b] B. Kosko, "Constructing an associative memory," *Byte*, pp. 137-144, September, 1987.
- [Kosk88] B. Kosko, "Bidirectional associative memories," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 18, No. 1, pp. 49-60, Jan/Feb., 1988.
- [Kosk89] B. Kosko, "Unsupervised learning in noise," *Proc. Int. Joint Conf. on Neural Networks*, vol. I, pp. 7-17, 1989.

-
- [Kosk91] B. Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*. Englewood Cliffs, NJ: Prentice Hall, 1991, 480 pp.
- [Kosk92] B. Kosko, *Neural Networks for Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1992, 415 pp.
- [Krae89] T. F. Kraemer, "Product development using object-oriented software technology," *Hewlett-Packard Journal*, vol. 40, No. 4, pp. 87-100, August, 1989.
- [Krei91] V. Y. Kreinovich, "Arbitrary nonlinearity is sufficient to represent all function by neural networks: A theorem," *Neural Networks*, vol. 4, pp. 381-383, 1991.
- [Kurz90] R. Kurzweil, *The Age of Intelligent Machines*. Cambridge, MA: The MIT Press, 1990, 565 pp.
- [LaIK91] A. Langi, A. Indrayanto, and W. Kinsner, "Stochastic codebook parameter searching using neural network for CELP speech coding," *IASTED Int. Conf. Computers, Electronics, Communication & Control*, Calgary, AB; April 8-10, 1991.
- [Li90] S. Li, "An optimized backpropagation with minimum norm weights," *Proc. Second Int. Joint Conf. on Neural Networks*, vol. I, pp. 697-702, 1990.
- [LiNu90] P. Li, and R. S. Nutter, "A bidirectional associative memory used in a pattern recognition system," *Proc. Second Int. Joint Conf. on Neural Networks*, vol. I, pp. 815-820, 1990.
- [Llin89] R. R. Llinas (ed.), *The Biology of the Brain: From Neurons to Networks*. New York: Freeman, 1989, 170 pp.

-
- [Lipp87] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, pp. 4-22, April, 1987.
- [MaHP90] A. J. Maren, C. T. Harston, and R. M. Pap, *Handbook of Neural Computing Applications*. San Diego, CA: Academic Press, 1990, 470 pp.
- [Mark90] D. Mark, *Macintosh C Programming Primer, Volume II: Mastering the Toolbox Using THINK C*. Reading, MA: Addison-Wesley, 1990, 507 pp.
- [MaUp90] G. Mathai, and B. R. Upadhyaya, "Performance analysis and application of the bidirectional associative memory to industrial spectral signatures," *Proc. Second Int. Joint Conf. on Neural Networks*, vol. I, pp. 33-37, 1990.
- [MeMR91] K. G. Mehrotra, C. K. Mohan, and S. Ranka, "Bounds on the number of samples needed for neural learning," *IEEE Trans. on Neural Networks*, vol. 2, No. 6, pp. 548-558, Nov., 1991.
- [McEl87] R. J. McEliece et. al., "The capacity of the Hopfield associative memory," *IEEE Trans. on Information Theory*, vol. IT-33, pp. 461-482, July, 1987.
- [McPi43] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, No. 5, pp. 115-133, 1943.
- [McRu86] J. McClelland and D. Rumelhart, *Parallel Distributed Processing. Explorations in the Microstructure of Cognition, Volume I: Foundations*. Cambridge, MA: The MIT Press, 1986, 568 pp.
- [McRu88] J. L. McClelland and D. E. Rumelhart, *Explorations in Parallel Distributed Processing : a Handbook of Models, Programs, and Exercises*. Cambridge, MA: The MIT Press, 1988, 344 pp.
- [MiPa88] M. L. Minsky and S. A. Papert, *Perceptrons*. (expanded edition), Cambridge, MA: The MIT Press, 1988, 307 pp.

-
- [Mull89] M. Mullin, *Object Oriented Program Design with Examples in C++*. Reading, MA: Addison-Wesley, 1989, 303 pp.
- [MüRe90] B. Müller and J. Reinhardt, *Physics of Neural Networks. Neural Network: An Introduction*. Berlin, GE: Springer-Verlag, 1990, 266 pp.
- [Nils90] N. J. Nilsson, *The Mathematical Foundations of Learning Machines*. San Mateo, CA: Morgan Kaufmann, 1990, 138 pp.
- [Page88] M. Page-Jones, *The Practical Guide To Structured Systems Design, second edition*. New Jersey: Yourdon, 1988.
- [Pao89] Y. Pao, *Adaptive Pattern Recognition and Neural Networks*. Reading, MA: Addison-Wesley, 1989, 312 pp.
- [PCWe90] M. Page-Jones, L. L. Constantine and S. Weiss, "Modeling object-oriented systems: The uniform object notation," *Computer Language*, vol. 7, No. 10, pp. 69-87, October, 1990.
- [PFTV88] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge: Cambridge University Press, 1988, 735 pp.
- [RHWi86] D. E. Rumelhart, G. E. Hinton, & R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533-536, October, 1986.
- [RyWi87] T. W. Ryan and C. L. Winter, "Variations on adaptive resonance," *Proc. of IEEE International Conference on Neural Networks*, vol. II, pp. 767-775, 1987.
- [SaAn91] M. A. Sartori and P. J. Antsaklis, "A simple method to derive bounds on the size and to train multilayer neural networks," *IEEE Trans. on Neural Networks*, vol. 2, No. 4, pp. 467-471, Jul., 1991.

-
- [SiDo91] J. Sietsma and R. J. F. Dow, "Creating artificial neural networks that generalize," *Neural Networks*, vol. 4, pp. 67-79, 1991.
- [Silv90] J. N. Silva, "A study of learning in backpropagation," *Thesis*. University of Manitoba; Canada, Mar., 1990, 190 pp.
- [Simp90] P. K. Simpson, "Higher-ordered and intraconnected bidirectional associative memories," *IEEE Transactions on Systems, Man, Cybernetics*, vol. 20, No. 3, pp. 637-653, May/June, 1990.
- [SmSt90] D. Smith and P. Stanford, "A random walk in Hamming space," *Proc. of Int. Joint Conf. on Neural Networks*, vol. II, pp. 465-470, 1990.
- [Sodh90] J. Sodhi, *Computer System Techniques: Development, Implementation & Software Maintenance*. Blue Ridge Summit, PA: TAB BOOKS Inc., 1990, pp. 164.
- [Souc89] B. Soucek, *Neural and Concurrent Real-Time Systems: The Sixth Generation*. New York: John Wiley & Sons, 1989, 405 pp.
- [Syma89] Symantec Corp., *Think C*, Cupertino, CA: Symantec Corp., 1989, 511 pp.
- [Time90] Time-Life Books. *How Things Work: The Brain*. Virginia: Time-Life Books Inc., 1990, 144 pp.
- [TWJo89] H. M. Tai, C. H. Wu, & T. L. Jong, "High-order Bidirectional Associative Memory," *Electronics Letters*, vol. 25, No. 21, pp. 1424-1425, 12th October, 1989.
- [WaCM89] Y. F. Wang, J. B. Cruz, and J. H. Mulligan, "An enhanced bidirectional associative memory," *Proc. Int. Joint Conf. on Neural Networks*, vol. I, pp. 105-110, 1989.

-
- [WaCM90] Y. F. Wang, J. B. Cruz, and J. H. Mulligan, "On multiple training for bidirectional associative memory," *IEEE Trans. on Neural Networks*, vol. 1, No. 3, pp. 275-276, 1990.
- [WaCM91] Y. F. Wang, J. B. Cruz, and J. H. Mulligan, "Guaranteed recall of all training pairs for bidirectional associative memory," *IEEE Trans. on Neural Networks*, vol. 2, No. 6, pp. 559-567, 1991.
- [WaPi91] A. I. Wasserman and P. A. Pircher, "Object-oriented structured design and C++," *Computer Language*, vol. 8, No. 1, pp. 41-52, January, 1991.
- [Wass89] P. D. Wasserman, *Neural Computing: Theory and Practice*. New York: Van Nostrand Reinhold, 1989, 230 pp.
- [WeKu90] S. M. Weiss and C. A. Kulikowski, *Computer Systems that Learn: Classification and Prediction Methods from Statistics, Neural Nets, Machine Learning, and Expert Systems*. San Mateo, CA: Morgan Kaufmann, 1990, 223 pp.
- [WeMa91] N. Weymaere and J. Martens, "A fast and robust learning algorithm for feedforward neural networks," *Neural Networks*, vol. 4, pp. 361-369, 1991.
- [Whit90] H. White, "Connectionist nonparametric regression: Multilayer feedforward networks can learn arbitrary mappings," *Neural Networks*, vol. 3, pp. 535-549, 1990.
- [Wood88] D. Woods, "Back and Counter Propagation aberrations," *Proc. of IEEE Int. Conf. on Neural Networks*, vol. 1, pp. 473-479, July 1988.

APPENDIX A

TRAINING PATTERNS

A.0 Introduction

This appendix lists the training patterns used in the experimental study.

A.1 Patterns Used in the Associative Memory Experiment

S_{AB} :

A
0 0 0 0 0
1 1 1 1 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
0 0 0 0 0

B
0 0 0 0 0
1 1 1 1 0
1 0 0 0 1
1 1 1 1 0
1 0 0 0 1
1 1 1 1 0
0 0 0 0 0

S_{AH} :

A
0 0 0 0 0
1 1 1 1 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
0 0 0 0 0

A. TRAINING PATTERNS

H
0 0 0 0 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
0 0 0 0 0

S_{AI}:

A
0 0 0 0 0
1 1 1 1 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
0 0 0 0 0

I
0 0 0 0 0
1 1 1 1 1
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
1 1 1 1 1
0 0 0 0 0

S_{BG}:

B
0 0 0 0 0
1 1 1 1 0
1 0 0 0 1
1 1 1 1 0
1 0 0 0 1
1 1 1 1 0
0 0 0 0 0

G
0 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 0 1 1 1
1 0 0 0 1
1 1 1 1 1
0 0 0 0 0

S_{EF} :

E
0 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 1 1 1 1
0 0 0 0 0

F
0 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 0 0 0 0
0 0 0 0 0

S_{EG} :

E
0 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 1 1 1 1
0 0 0 0 0

G
0 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 0 1 1 1
1 0 0 0 1
1 1 1 1 1
0 0 0 0 0

S_{HI} :

H
0 0 0 0 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
0 0 0 0 0

A. TRAINING PATTERNS

```
I
0 0 0 0 0
1 1 1 1 1
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
1 1 1 1 1
0 0 0 0 0
```

S_{II} :

```
I
0 0 0 0 0
1 1 1 1 1
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
1 1 1 1 1
0 0 0 0 0
```

```
J
0 0 0 0 0
1 1 1 1 1
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
1 1 1 0 0
0 0 0 0 0
```

S_{IX} :

```
Ix
0 0 0 0 0
1 1 1 1 1
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
1 1 1 0 1
0 0 0 0 0
```

```
Jx
0 0 0 0 0
1 1 1 1 1
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
1 1 1 1 0
0 0 0 0 0
```

S_{ABI} :

A
0 0 0 0 0
1 1 1 1 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
0 0 0 0 0

B
0 0 0 0 0
1 1 1 1 0
1 0 0 0 1
1 1 1 1 0
1 0 0 0 1
1 1 1 1 0
0 0 0 0 0

I
0 0 0 0 0
1 1 1 1 1
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
1 1 1 1 1
0 0 0 0 0

 S_{AFH} :

A
0 0 0 0 0
1 1 1 1 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
0 0 0 0 0

F
0 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 0 0 0 0
0 0 0 0 0

A. TRAINING PATTERNS

H
0 0 0 0 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
0 0 0 0 0

S_{BHI} :

B
0 0 0 0 0
1 1 1 1 0
1 0 0 0 1
1 1 1 1 0
1 0 0 0 1
1 1 1 1 0
0 0 0 0 0

H
0 0 0 0 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
0 0 0 0 0

I
0 0 0 0 0
1 1 1 1 1
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
1 1 1 1 1
0 0 0 0 0

S_{EFG} :

E
0 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 1 1 1 1
0 0 0 0 0

A. TRAINING PATTERNS

F
0 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 0 0 0 0
0 0 0 0 0

G
0 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 0 1 1 1
1 0 0 0 1
1 1 1 1 1
0 0 0 0 0

S_{BCHI}:

B
0 0 0 0 0
1 1 1 1 0
1 0 0 0 1
1 1 1 1 0
1 0 0 0 1
1 1 1 1 0
0 0 0 0 0

C
0 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 0 0 0 0
1 0 0 0 0
1 1 1 1 1
0 0 0 0 0

H
0 0 0 0 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
0 0 0 0 0

A. TRAINING PATTERNS

I
0 0 0 0 0
1 1 1 1 1
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
1 1 1 1 1
0 0 0 0 0

S_{BHIL} :

B
0 0 0 0 0
1 1 1 1 0
1 0 0 0 1
1 1 1 1 0
1 0 0 0 1
1 1 1 1 0
0 0 0 0 0

H
0 0 0 0 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
0 0 0 0 0

I
0 0 0 0 0
1 1 1 1 1
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
1 1 1 1 1
0 0 0 0 0

L
0 0 0 0 0
1 0 0 0 0
1 0 0 0 0
1 0 0 0 0
1 0 0 0 0
1 1 1 1 1
0 0 0 0 0

A. TRAINING PATTERNS

S_{AB-IJ}:

A
 0 0 0 0 0
 1 1 1 1 1
 1 0 0 0 1
 1 1 1 1 1
 1 0 0 0 1
 1 0 0 0 1
 0 0 0 0 0

I
 0 0 0 0 0
 1 1 1 1 1
 0 0 1 0 0
 0 0 1 0 0
 0 0 1 0 0
 1 1 1 1 1
 0 0 0 0 0

B
 0 0 0 0 0
 1 1 1 1 0
 1 0 0 0 1
 1 1 1 1 0
 1 0 0 0 1
 1 1 1 1 0
 0 0 0 0 0

J
 0 0 0 0 0
 1 1 1 1 1
 0 0 1 0 0
 0 0 1 0 0
 0 0 1 0 0
 1 1 1 0 0
 0 0 0 0 0

S_{AH-EF}:

A
 0 0 0 0 0
 1 1 1 1 1
 1 0 0 0 1
 1 1 1 1 1
 1 0 0 0 1
 1 0 0 0 1
 0 0 0 0 0

E
 0 0 0 0 0
 1 1 1 1 1
 1 0 0 0 0
 1 1 1 1 1
 1 0 0 0 0
 1 1 1 1 1
 0 0 0 0 0

H
 0 0 0 0 0
 1 0 0 0 1
 1 0 0 0 1
 1 1 1 1 1
 1 0 0 0 1
 1 0 0 0 1
 0 0 0 0 0

F
 0 0 0 0 0
 1 1 1 1 1
 1 0 0 0 0
 1 1 1 1 1
 1 0 0 0 0
 1 0 0 0 0
 0 0 0 0 0

S_{EG-AI}:

E
 0 0 0 0 0
 1 1 1 1 1
 1 0 0 0 0
 1 1 1 1 1
 1 0 0 0 0
 1 1 1 1 1
 0 0 0 0 0

A
 0 0 0 0 0
 1 1 1 1 1
 1 0 0 0 1
 1 1 1 1 1
 1 0 0 0 1
 1 0 0 0 1
 0 0 0 0 0

A. TRAINING PATTERNS

G
 0 0 0 0 0
 1 1 1 1 1
 1 0 0 0 0
 1 0 1 1 1
 1 0 0 0 1
 1 1 1 1 1
 0 0 0 0 0

I
 0 0 0 0 0
 1 1 1 1 1
 0 0 1 0 0
 0 0 1 0 0
 0 0 1 0 0
 1 1 1 1 1
 0 0 0 0 0

S_{EI-GJ} :

E
 0 0 0 0 0
 1 1 1 1 1
 1 0 0 0 0
 1 1 1 1 1
 1 0 0 0 0
 1 1 1 1 1
 0 0 0 0 0

G
 0 0 0 0 0
 1 1 1 1 1
 1 0 0 0 0
 1 0 1 1 1
 1 0 0 0 1
 1 1 1 1 1
 0 0 0 0 0

I
 0 0 0 0 0
 1 1 1 1 1
 0 0 1 0 0
 0 0 1 0 0
 0 0 1 0 0
 1 1 1 1 1
 0 0 0 0 0

J
 0 0 0 0 0
 1 1 1 1 1
 0 0 1 0 0
 0 0 1 0 0
 0 0 1 0 0
 1 1 1 0 0
 0 0 0 0 0

S_{HI-EG} :

H
 0 0 0 0 0
 1 0 0 0 1
 1 0 0 0 1
 1 1 1 1 1
 1 0 0 0 1
 1 0 0 0 1
 0 0 0 0 0

E
 0 0 0 0 0
 1 1 1 1 1
 1 0 0 0 0
 1 1 1 1 1
 1 0 0 0 0
 1 1 1 1 1
 0 0 0 0 0

I
 0 0 0 0 0
 1 1 1 1 1
 0 0 1 0 0
 0 0 1 0 0
 0 0 1 0 0
 1 1 1 1 1
 0 0 0 0 0

G
 0 0 0 0 0
 1 1 1 1 1
 1 0 0 0 0
 1 0 1 1 1
 1 0 0 0 1
 1 1 1 1 1
 0 0 0 0 0

A. TRAINING PATTERNS

S_{IJ-EG} :

I	E
0 0 0 0 0	0 0 0 0 0
1 1 1 1 1	1 1 1 1 1
0 0 1 0 0	1 0 0 0 0
0 0 1 0 0	1 1 1 1 1
0 0 1 0 0	1 0 0 0 0
1 1 1 1 1	1 1 1 1 1
0 0 0 0 0	0 0 0 0 0

J	G
0 0 0 0 0	0 0 0 0 0
1 1 1 1 1	1 1 1 1 1
0 0 1 0 0	1 0 0 0 0
0 0 1 0 0	1 0 1 1 1
0 0 1 0 0	1 0 0 0 1
1 1 1 0 0	1 1 1 1 1
0 0 0 0 0	0 0 0 0 0

$S_{BHI-ACJ}$:

B	A
0 0 0 0 0	0 0 0 0 0
1 1 1 1 0	1 1 1 1 1
1 0 0 0 1	1 0 0 0 1
1 1 1 1 0	1 1 1 1 1
1 0 0 0 1	1 0 0 0 1
1 1 1 1 0	1 0 0 0 1
0 0 0 0 0	0 0 0 0 0

H	C
0 0 0 0 0	0 0 0 0 0
1 0 0 0 1	1 1 1 1 1
1 0 0 0 1	1 0 0 0 0
1 1 1 1 1	1 0 0 0 0
1 0 0 0 1	1 0 0 0 0
1 0 0 0 1	1 1 1 1 1
0 0 0 0 0	0 0 0 0 0

I	J
0 0 0 0 0	0 0 0 0 0
1 1 1 1 1	1 1 1 1 1
0 0 1 0 0	0 0 1 0 0
0 0 1 0 0	0 0 1 0 0
0 0 1 0 0	0 0 1 0 0
1 1 1 1 1	1 1 1 0 0
0 0 0 0 0	0 0 0 0 0

$S_{BHI-AJC}$:

<p>B</p> <p>0 0 0 0 0</p> <p>1 1 1 1 0</p> <p>1 0 0 0 1</p> <p>1 1 1 1 0</p> <p>1 0 0 0 1</p> <p>1 1 1 1 0</p> <p>0 0 0 0 0</p>	<p>A</p> <p>0 0 0 0 0</p> <p>1 1 1 1 1</p> <p>1 0 0 0 1</p> <p>1 1 1 1 1</p> <p>1 0 0 0 1</p> <p>1 0 0 0 1</p> <p>0 0 0 0 0</p>
<p>H</p> <p>0 0 0 0 0</p> <p>1 0 0 0 1</p> <p>1 0 0 0 1</p> <p>1 1 1 1 1</p> <p>1 0 0 0 1</p> <p>1 0 0 0 1</p> <p>0 0 0 0 0</p>	<p>J</p> <p>0 0 0 0 0</p> <p>1 1 1 1 1</p> <p>0 0 1 0 0</p> <p>0 0 1 0 0</p> <p>0 0 1 0 0</p> <p>1 1 1 0 0</p> <p>0 0 0 0 0</p>
<p>I</p> <p>0 0 0 0 0</p> <p>1 1 1 1 1</p> <p>0 0 1 0 0</p> <p>0 0 1 0 0</p> <p>0 0 1 0 0</p> <p>1 1 1 1 1</p> <p>0 0 0 0 0</p>	<p>C</p> <p>0 0 0 0 0</p> <p>1 1 1 1 1</p> <p>1 0 0 0 0</p> <p>1 0 0 0 0</p> <p>1 0 0 0 0</p> <p>1 1 1 1 1</p> <p>0 0 0 0 0</p>

A.2 Patterns Used in the Classification Experiment

S_{AJ} :

A01

0 0 0 0 0

1 1 1 1 1

1 0 0 0 1

1 1 1 1 1

1 0 0 0 1

1 0 0 0 1

0 0 0 0 0

B02
0 0 0 0 0
1 1 1 1 0
1 0 0 0 1
1 1 1 1 0
1 0 0 0 1
1 1 1 1 0
0 0 0 0 0

C03
0 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 0 0 0 0
1 0 0 0 0
1 1 1 1 1
0 0 0 0 0

D04
0 0 0 0 0
1 1 1 1 0
1 0 0 0 1
1 0 0 0 1
1 0 0 0 1
1 1 1 1 0
0 0 0 0 0

E05
0 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 1 1 1 1
0 0 0 0 0

F06
0 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 0 0 0 0
0 0 0 0 0

G07
0 0 0 0 0
1 1 1 1 1
1 0 0 0 0
1 0 1 1 1
1 0 0 0 1
1 1 1 1 1
0 0 0 0 0

A. TRAINING PATTERNS

H08
0 0 0 0 0
1 0 0 0 1
1 0 0 0 1
1 1 1 1 1
1 0 0 0 1
1 0 0 0 1
0 0 0 0 0

I09
0 0 0 0 0
1 1 1 1 1
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
1 1 1 1 1
0 0 0 0 0

J00
0 0 0 0 0
1 1 1 1 1
0 0 1 0 0
0 0 1 0 0
0 0 1 0 0
1 1 1 0 0
0 0 0 0 0

APPENDIX B

EXPERIMENTAL RESULTS OF ART-1

B.0 Introduction

This appendix lists the unsorted predicted categories of ART-1.

B.1 Predicted Categories for vigilance 0.5

0	C14(0)					
1	H53(0)					
2	J11(0)	J52(0)				
3	C23(0)	C34(0)				
4	H31(0)	H54(0)				
5	F11(0)	D21(0)	D24(0)	D33(0)	G54(0)	
6	F55(0)					
7	C32(0)	C44(0)				
8	J21(0)	F32(0)	J31(0)	F54(0)	J51(0)	J54(0)
9	B34(0)	D41(0)	H52(0)			
10	C43(0)					
11	D43(0)					
12	J01(0)	J12(0)	J13(0)	J14(0)	J15(0)	J22(0)
	J24(0)	J25(0)	I33(0)	J32(0)	J33(0)	I35(0)
	J34(0)	I43(0)	J41(0)	I51(0)	I52(0)	
13	D13(0)	C41(0)	C52(0)			
14	A43(0)	B42(0)	B52(0)			
15	D32(0)	E42(0)	C55(0)	D51(0)		
16	F01(0)	A14(0)	F12(0)	F13(0)	F14(0)	F15(0)
	F21(0)	B24(0)	F23(0)	F24(0)	F25(0)	A32(0)
	F31(0)	F33(0)	F34(0)	E43(0)	F41(0)	F43(0)
	F44(0)	F45(0)	A55(0)	F53(0)		
17	H23(0)	H32(0)	F35(0)	H34(0)	H35(0)	
18	C45(0)					
19	H55(0)					
20	I23(0)	J23(0)	J42(0)	J44(0)	J55(0)	
21	C12(0)	C21(0)	C24(0)	C31(0)	I42(0)	I45(0)
	I54(0)					
22	J45(0)	I53(0)				

B. EXPERIMENTAL RESULTS OF ART-1

23	B14(0)	B22(0)	B53(0)			
24	G23(0)	I32(0)	B55(0)	D52(0)	I55(0)	
25	D01(0)	D11(0)	D12(0)	D14(0)	D15(0)	D22(0)
	D23(0)	D31(0)	D34(0)	D35(0)	D42(0)	D44(0)
	G43(0)	D45(0)	D53(0)	D54(0)	D55(0)	E52(0)
26	H12(0)	H21(0)	A44(0)	H42(0)	F52(0)	H51(0)
27	F42(0)	E51(0)	G55(0)			
28						
29	I01(0)	I11(0)	I13(0)	I14(0)	I21(0)	I24(0)
	I25(0)	E31(0)	I31(0)	I44(0)		
30	A52(0)					
31	G42(0)					
32	A42(0)	G45(0)	F51(0)			
33	E34(0)	E55(0)				
34	H44(0)					
35	I12(0)	I15(0)	F22(0)	J35(0)	I41(0)	J43(0)
	A45(0)	J53(0)				
36	E45(0)					
37	E54(0)					
38	B01(0)	C01(0)	G01(0)	B11(0)	C11(0)	G11(0)
	B12(0)	B13(0)	B15(0)	C13(0)	C15(0)	E12(0)
	E15(0)	G12(0)	G13(0)	G14(0)	G15(0)	C22(0)
	E22(0)	I22(0)	B23(0)	B25(0)	C25(0)	D25(0)
	E23(0)	E25(0)	B33(0)	C33(0)	E33(0)	G33(0)
	C35(0)	G34(0)	I34(0)	C42(0)	E41(0)	C51(0)
	C53(0)	C54(0)	G53(0)			
39	H01(0)	H11(0)	H13(0)	H14(0)	H15(0)	A21(0)
	H22(0)	H24(0)	H25(0)	B31(0)	B32(0)	H33(0)
	A34(0)	A41(0)	H41(0)	H43(0)	H45(0)	A54(0)
40	B21(0)	G31(0)	G32(0)	B35(0)	G35(0)	B41(0)
	B43(0)	G44(0)	B51(0)			
41	E11(0)	G22(0)	E32(0)	B44(0)	E44(0)	G41(0)
	B45(0)	B54(0)	G51(0)	G52(0)		
42	A01(0)	E01(0)	A11(0)	A12(0)	A13(0)	A15(0)
	E13(0)	E14(0)	A22(0)	E21(0)	G21(27)	A23(0)
	A24(0)	A25(0)	E24(0)	G24(0)	G25(0)	A31(0)
	A33(0)	A35(0)	E35(0)	A51(0)	A53(0)	E53(0)

B.2 Predicted Categories for vigilance 0.8

0	F11(0)
1	C32(0)
2	D13(0)
3	J31(0)
4	H53(0)

B. EXPERIMENTAL RESULTS OF ART-1

5	F32(0)		
6	A14(0)	A43(0)	
7	J52(0)		
8	F12(0)	F25(0)	
9	B34(0)		
10	C23(0)	C34(0)	
11	G22(0)	G23(0)	
12	H25(0)		
13	A32(0)		
14	B11(0)	B24(0)	A55(0)
15	D24(0)		
16	I23(0)		
17	J41(0)		
18	D33(0)		
19	I42(0)		
20	J11(0)		
21			
22	H42(0)		
23	D35(0)		
24	C44(0)		
25	D32(0)		
26	E11(0)	E31(0)	E43(0)
27	B45(0)		
28	D43(0)		
29	H34(0)		
30	H31(0)		
31	J44(0)		
32	B43(0)		
33	C43(0)		
34	C42(0)		
35	J54(0)		
36	A44(0)		
37	I33(0)	I35(0)	
38	A42(0)		
39	B42(0)		
40	B44(0)		
41	D41(0)		
42	D51(0)		
43	E42(0)		
44	F54(0)		
45	F21(0)	F43(0)	
46	G42(0)		
47	C45(0)		
48			
49	H43(0)	H45(0)	
50	H55(0)		
51	J21(0)		
52	J23(0)		
53	J35(0)	J53(0)	
54	F45(0)		

B. EXPERIMENTAL RESULTS OF ART-1

55	E45 (0)	B55 (0)				
56	I45 (0)					
57	A52 (0)					
58	H23 (0)					
59	B52 (0)					
60	D45 (0)	B53 (0)				
61	D52 (0)					
62	C14 (0)					
63	G45 (0)	C52 (0)				
64	C33 (0)	C53 (0)	C55 (0)			
65	D23 (0)	D34 (0)	D55 (0)			
66	D21 (0)					
67	E51 (0)	E55 (0)				
68	E54 (0)					
69	F51 (0)					
70	F52 (0)					
71	F53 (0)					
72	F55 (0)					
73	G55 (0)					
74	G11 (0)	E12 (0)				
75	H54 (0)					
76	G54 (0)					
77	H51 (0)					
78	H52 (0)					
79	J42 (0)					
80	I52 (0)					
81	I14 (0)	I22 (0)				
82	I54 (0)					
83	J55 (0)					
84	J51 (0)					
85	B14 (0)	B22 (0)	B31 (0)	B32 (0)	B33 (0)	
86	C24 (0)	C25 (0)				
87	H12 (0)	H21 (0)				
88	H35 (0)					
89	D01 (0)	D11 (0)	D12 (0)	D14 (0)	D15 (0)	D22 (0)
	D25 (0)					
90	E01 (0)	E15 (0)	E22 (0)	E23 (0)	E25 (0)	E41 (0)
91	G33 (0)					
92	J33 (0)	J34 (0)				
93	D31 (0)					
94	C01 (0)	C11 (0)	C12 (0)	C13 (0)	C15 (0)	C21 (0)
	C22 (0)	C31 (0)				
95	A31 (0)	A34 (0)				
96	F35 (0)					
97	E34 (0)					
98	G44 (0)					
99	H01 (0)	H11 (0)	H14 (0)	H15 (0)	H22 (0)	H24 (0)
	H32 (0)					
100	H44 (0)					
101	I01 (0)	I11 (0)	I12 (0)	I13 (0)	I15 (0)	I21 (0)

B. EXPERIMENTAL RESULTS OF ART-1

	I41(0)					
102	J01(0)	J12(0)	J13(0)	J14(0)	J15(0)	J22(0)
	J24(0)	J25(0)				
103	B21(0)	B35(0)				
104	C41(0)					
105	G41(0)					
106	F41(0)					
107	B51(0)					
108	D42(0)	D44(0)				
109	F42(0)	F44(0)				
110	G43(0)	B54(0)				
111	I43(0)					
112	I32(0)	I44(0)				
113	J43(0)					
114	F01(0)	A15(0)	E14(0)	F13(0)	F14(0)	F15(0)
	F22(0)	F23(0)	F24(0)	F31(0)	F33(0)	F34(0)
115	J45(0)					
116	A53(0)					
117	H13(0)	H33(0)				
118	C35(0)	C54(0)				
119	D53(0)					
120	D54(0)					
121	E13(0)	G31(0)	E52(0)			
122	G53(0)					
123	G14(0)	G25(0)				
124	G01(0)	G12(0)	G13(0)	G15(0)	E33(0)	G52(0)
125	I51(0)					
126	I53(0)					
127	I24(0)	I25(0)	I31(0)	I55(0)		
128	G24(0)	G34(0)				
129	B01(0)	B12(0)	B13(0)	B15(0)	B23(0)	B25(0)
	B41(0)					
130	A01(0)	A11(0)	A12(0)	A13(0)	A22(0)	A24(0)
	A25(0)	A41(0)				
131	E32(0)	G32(0)				
132	J32(0)					
133	G35(0)	H41(0)				
134	I34(0)	E44(0)				
135	A45(0)					
136						
137	C51(0)					
138	E21(0)	E24(0)	E53(0)			
139	E35(0)	G51(0)				
140	G21(0)					
141	A21(0)	A23(0)	A54(0)			
142	A35(0)	A51(0)				
143	A33(0)					

B.3 Predicted Categories for vigilance 1.0

0	F11(0)
1	B11(0)
2	C32(0)
3	D13(0)
4	E31(0)
5	G11(0)
6	H12(0)
7	J11(0)
8	F12(0)
9	C14(0)
10	D01(0)
11	H01(0)
12	J01(0)
13	A15(0)
14	H15(0)
15	B22(0)
16	B24(0)
17	B34(0)
18	C23(0)
19	C12(0)
20	D12(0)
21	D35(0)
22	D43(0)
23	F01(0)
24	F15(0)
25	C24(0)
26	F14(0)
27	G14(0)
28	G22(0)
29	G23(0)
30	H23(0)
31	H14(0)
32	J31(0)
33	I23(0)
34	J12(0)
35	J13(0)
36	J14(0)
37	J15(0)
38	A21(0)
39	F23(0)
40	B21(0)
41	C01(0)
42	D24(0)
43	D22(0)
44	C43(0)
45	E22(0)
46	F25(0)

B. EXPERIMENTAL RESULTS OF ART-1

47	F35(0)
48	G54(0)
49	H21(0)
50	H42(0)
51	J21(0)
52	I14(0)
53	J22(0)
54	A23(0)
55	A14(0)
56	A32(0)
57	B14(0)
58	B01(0)
59	C25(0)
60	D23(0)
61	D25(0)
62	E15(0)
63	E11(0)
64	E25(0)
65	F24(0)
66	G24(0)
67	H34(0)
68	H25(0)
69	J23(0)
70	I33(0)
71	J44(0)
72	A31(0)
73	A43(0)
74	B31(0)
75	B32(0)
76	B33(0)
77	C31(0)
78	C34(0)
79	D15(0)
80	D32(0)
81	D33(0)
82	E32(0)
83	F13(0)
84	F21(0)
85	F32(0)
86	F33(0)
87	G31(0)
88	C44(0)
89	G33(0)
90	H31(0)
91	H32(0)
92	H13(0)
93	J54(0)
94	I01(0)
95	J32(0)
96	J33(0)

B. EXPERIMENTAL RESULTS OF ART-1

97	A34(0)
98	H55(0)
99	B35(0)
100	C35(0)
101	D34(0)
102	E34(0)
103	C41(0)
104	F34(0)
105	G34(0)
106	G35(0)
107	H35(0)
108	J52(0)
109	I35(0)
110	J34(0)
111	J35(0)
112	H54(0)
113	A42(0)
114	A44(0)
115	B41(0)
116	B42(0)
117	F43(0)
118	B44(0)
119	C42(0)
120	D41(0)
121	D42(0)
122	D44(0)
123	E41(0)
124	E42(0)
125	E43(0)
126	E44(0)
127	F41(0)
128	F42(0)
129	F44(0)
130	G41(0)
131	G42(0)
132	G43(0)
133	G44(0)
134	H41(0)
135	H43(0)
136	H44(0)
137	I41(0)
138	I42(0)
139	I43(0)
140	I44(0)
141	J41(0)
142	J42(0)
143	J43(0)
144	F22(0)
145	B45(0)
146	C45(0)

B. EXPERIMENTAL RESULTS OF ART-1

147	D45(0)
148	E45(0)
149	F45(0)
150	G45(0)
151	H45(0)
152	I45(0)
153	J45(0)
154	A51(0)
155	A52(0)
156	A53(0)
157	A54(0)
158	A55(0)
159	B51(0)
160	B52(0)
161	B53(0)
162	B54(0)
163	B55(0)
164	C13(0)
165	C52(0)
166	C53(0)
167	C54(0)
168	C55(0)
169	D51(0)
170	D52(0)
171	D53(0)
172	D54(0)
173	D55(0)
174	E51(0)
175	E52(0)
176	E53(0)
177	E54(0)
178	E55(0)
179	F51(0)
180	F52(0)
181	F53(0)
182	F54(0)
183	F55(0)
184	G51(0)
185	G52(0)
186	G53(0)
187	G55(0)
188	H51(0)
189	H52(0)
190	H53(0)
191	I51(0)
192	I52(0)
193	I53(0)
194	I54(0)
195	I55(0)
196	J51(0)

B. EXPERIMENTAL RESULTS OF ART-1

197	J53 (0)	
198	J55 (0)	
199	G01 (0)	
200	A01 (0)	
201	C11 (0)	
202	D11 (0)	
203	H11 (0)	
204	I11 (0)	
205	A12 (0)	
206	A13 (0)	
207	B13 (0)	
208	B15 (0)	
209	C21 (0)	
210	D14 (0)	
211	E14 (0)	
212	G12 (0)	
213	G13 (0)	
214	G15 (0)	
215	I12 (0)	I15 (0)
216	J25 (0)	
217	A22 (0)	
218	C22 (0)	
219	D21 (0)	
220	G25 (0)	
221	G21 (0)	
222	H22 (0)	
223	I13 (0)	
224	I22 (0)	
225	A24 (0)	
226	A25 (0)	
227	B23 (0)	
228	B12 (0)	
229	E23 (0)	
230	E12 (0)	
231	H24 (0)	
232	I24 (0)	
233	I25 (0)	
234	J24 (0)	
235	A33 (0)	
236	C15 (0)	
237	D31 (0)	
238	E33 (0)	
239	F31 (0)	
240	G32 (0)	
241	H33 (0)	
242	I31 (0)	
243	I32 (0)	
244	A35 (0)	
245	E35 (0)	
246	I34 (0)	

B. EXPERIMENTAL RESULTS OF ART-1

247	A41(0)
248	B43(0)
249	A45(0)
250	C51(0)
251	A11(0)
252	E01(0)
253	E21(0)
254	I21(0)
255	B25(0)
256	E24(0)
257	C33(0)
258	E13(0)

APPENDIX C

NEURAL NETWORK SIMULATOR'S WINDOWS

File Option Configuration Operation Tools Pattern Edit

S-function.BP

Total Training Passes : 370 Current Pass : 364
 Error Criteria : 0.010000 Total Error : 0.010001
 Total Input Patterns : 11 Current Pattern : 11
 Weight Update Mode : Batch Name : x_10
 Presentation Mode : Sequential Error

Weight

+8.32824

The Probe for totalError

The Probe for Output

Output
+0.965831