ANALYSIS OF ART NETWORK CLUSTERING PERFORMANCE

PROJECT REPORT

Submitted by

K.KARTHIKEYAN
9927K0129

V.MURALI KRISHNA
9927K0141

V.K.VENKATASAMY
9927K0172

P-863

Under the guidance of

Mrs. D. CHANDRAKALA M.E.

In partial fulfillment of the requirements for the award of the Degree of
Bachelor of Engineering
in Computer Science and Engineering of Bharathiar University, Coimbatore.

April 2003

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
KUMARAGURU COLLEGE OF TECHNOLOGY,
Coimbatore – 641 006.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
KUMARAGURU COLLEGE OF TECHNOLOGY,
Coimbatore – 641 006.

PROJECT REPORT
April 2003

CERTIFICATE

This is to certify that the project work entitled **"ANALYSIS OF ART NETWORK CLUSTERING PERFORMANCE"** submitted by the following students:

K.KARTHIKEYAN	**V.MURALI KRISHNA**	**V.K.VENKATASAMY**
9927K0129	9927K0141	9927K0172

In partial fulfillment of the requirements of the award of the Degree of Bachelor of Engineering in Computer Science and Engineering of Bharathiar University, is a record of bonafide work carried out by them.

D. Chandrika 14/3/'03
Guide

Prof. S. Thangasamy
Head of the Department

Place : Coimbatore

Certified that the candidate with Register no _____

was examined by us in the project viva voice held on 17.03.03

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

We owe our deepest gratitude to our patron and our Principal, **Dr K.K. Padmanabhan**, Kumaraguru College of Technology, Coimbatore for having permitted us to use the facilities in the college to do the project.

We also express our sincere thanks to the Head of the Department, **S. Thangasamy,** B.E (hons) PhD, who has been a constant source of support and encouragement, providing us with the materials necessary for the successful completion of the project.

We are also extremely thankful to our project guide, **Mrs. D.Chandrakala**, M.E. Senior Lecturer, project coordinator, **Ms. S.Rajini** and our class advisor **Mr. M.N. Guptha,** Department of Computer Science and Engineering, for their invaluable guidance, support, continuous encouragement, confident words, constructive criticisms and timely suggestions rendered to us.

We also wish to all non-teaching staff members and student friends of Computer Science and Engineering department for their constant support.

ABSTRACT

Self-organized clustering is a powerful tool, whenever huge sets of data have to be divided into separate categories. Adaptive resonance (ART) architectures are neural networks, that self organize stable recognition codes in real time in response to arbitrary sequence of input patterns. Many different types of ART networks have been developed to improve clustering capabilities.

In this project titled, " Analysis of ART network Clustering Performance", clustering performance of different typesof ART networks: ART-1,ART-2A, Fuzzy ART is compared. Some outstanding features of ART besides its clustering capabilities include performance, economic usage of memory resources and temporal stability of stored knowledge. This project concentrates on the comparative analysis of clustering properties for several variants of ART networks on input patterns of arbitrary dimensions, which illustrate the geometric characteristics of ART clustering and internal representation of knowledge by prototypes.

Keywords: Adaptive Resonance Theory, clustering analysis, neural networks, self-organization, unsupervised.

CONTENTS

CHAPTER I

INTRODUCTION

1 PROBLEM DEFINITION

Pattern clustering is the problem of grouping a giving set of patterns according to the similarity of the patterns. The pattern clustering by any ART network can be modeled by the simple sequence of operations: preprocessing, choice, match and adaptation. The central part of the ART network computes the matching score reflecting the degree of similarity of the present input to the previously encoded clusters.

A combination neural networks consisting of feed forward and feedback parts can perform pattern recognition tasks. The network is also called the competitive learning network. There are two types of problems here. The first type of problem is that the network displays an accretive behavior. That is, if an input pattern not belonging to any group is presented, then the network will force it into one of the groups. The input pattern space is typically a continuous space. The test input patterns could be the same as the ones used in training or could be different. The output pattern space consists of a set of cluster centre or labels. The second type of problem is that the network displays an interpolative behavior. In this case, a test input pattern not belonging to any group produces an output which is some form of interpolation of the output patterns or cluster centre depending on the proximity of the test input pattern to the input pattern groups formed during training.

1.2 RELEVANCE AND IMPORTANCE OF THE TOPIC

Adaptive Resonance theory(ART) that learns in an unsupervised fashion is of the interpolative type. ART structure is a neural network approach for cluster formation. ART allows the user to control the degree of similarity of patterns placed in the same cluster.

Many pattern mapping networks can be transformed to perform pattern classification or category learning tasks. However these networks have the disadvantage that during learning the weight vectors tend to encode the presently active pattern, thus weakening the traces of patterns it had already learnt. Moreover, any new patterns that does not belong to the categories already learnt is still forced into one of them using the best matched strategy, without taking into account how good the best match is. The lack of stability of weights as well the inability to accommodate
patterns belonging to new categories, led to the proposal of new architectures for pattern classification. The various conventional algorithms for clustering are K-means, vector quantization (VQ) techniques, ISODATA algorithms, which are examples of decision theoretical approaches.

In these conventional neural networks, the stability-plasticity dilemma encountered. The architecture of Adaptive Resonance Theory is specially designed to take care of the so called stability-plasticity dilemma in pattern classification. The common algorithm used for clustering in any kind of ART network is closely related to well known K-means algorithm. Both use single prototype to internally represent and dynamically adapt clusters. The K-means algorithm clusters a given set of input patterns into K-groups. The parameter K thus specifies the coarseness of the partition. In contrast ART uses a minimum required similarity between patterns that are grouped within one cluster. The resulting number K of clusters then depends on the distances (in terms of the applied metric) between all input patterns, presented to the network during training cycles. This similarity parameter is called vigilance.

1.2.1 STABILITY PLASTICITY DILEMMA

The human brain has the ability to learn and memorize many new things in a fashion that does not necessarily cause the existing ones to be forgotten. In order to design a truly intelligent pattern recognition machine, compatible with the human brain, it would be highly desirable to impart this ability to our models.

In real world applications, though, the network can be exposed to a constantly changing environment, such that training that does not evolve will ultimately become inaccurate. The ability of a network to adapt and learn a new pattern well at any stage of operation is called plasticity.

Grossberg describes the stability-plasticity dilemma as follows: "How can a learning system be designed to remain plastic, or adaptive, in response to significant events and yet remain stable in response to irrelevant events? How does the system know how to switch between its stable and its plastic modes to achieve stability without chaos? In particular how can it preserve its previously learned knowledge while continuing to learn new things? What prevents the new learning from washing away the memories of prior learning?"

As such ART provides a mechanism by which the network can learn new patterns without forgetting old knowledge. For example, in the context of character recognition problem, this could be useful in contexts such as training writer specific handwriting in an on-line system or in adding new fonts to an existing off-line system needing to retrain the network from scratch.

The incorporation of a tolerance measure (vigilance test) allows ART architecture to solve the stability-plasticity dilemma. New patterns from the environment can create additional classification categories, but they cannot cause an existing memory to be changed unless the two match closely.

1.2.2 ANALYSIS OF PATTERN CLUSTERING NETWORKS

A competitive learning network with nonlinear output functions for units in the feedback layer can produce at equilibrium larger activation on a single unit and small activations on other units.This behavior leads to a winner-take-all situation, where, when the input pattern is removed, only one unit in the feedback layer will have non-zero activation. If the feed forward weights are suitably adjusted, each of the units in the feedback layer can be made to win for a group of similar input patterns. The corresponding learning is called competitive learning. The units in the feedback layer have non-linear $f(x)=xn$,$n>1$ output functions. Other non-linear output functions such as hard-limiting threshold function or semi linear sigmoid function can also be used.

These units are connected among themselves with fixed weights in an on-centre off-surround manner. Such networks are called competitive learning networks. Since they are used for clustering or grouping of input patterns, they can also be called pattern clustering networks. In the pattern clustering task, the pattern classes are formed on unlabelled input data, and hence the corresponding learning is unsupervised. In the competitive learning the weights in the feed forward path are adjusted only after the winner unit in the feed back layer is identified for a given input pattern.

This project deals with the comparative analysis of the clustering performance of various ART architectures which are competitive learning models.

1.2.3 STAGES OF PROCESSING IN AN ART ARCHITECTURE

The various stages of processing in any ART architecture is as shown in Fig. 1.2

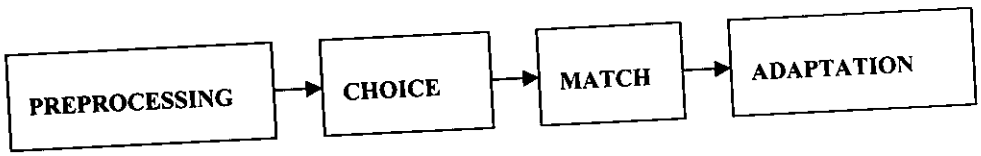


Figure 1.2

Preprocessing is the creation of input pattern as an array with constant number of elements. Once the input pattern is formed, it is compared with the n-stored prototypes. The neuron with the maximum net value i.e. maximum bottom-up net activity in recognition layer is fired. The top down net activity from F2 layer is matched with the input pattern. Depending on the degree of similarity, either the input pattern is fed again or adaptation occurs. The weights are accordingly changed so as to accommodate the input pattern in the cluster which it matches.

CHAPTER II

LITERATURE SURVEY

This Project gets inspired by the article presented in the IEEE Transaction on Neural networks titled, "Comparative Analysis of Fuzzy ART and ART-2A Network Clustering Performance".General Clustering Capabilities of ART network are briefly illustrated in conceptual manner with general idea about how to compare their clustering capabilities. The conceptual approach has been developed as a software program in this project which follows the algorithmic notion presented in the article.

And also, ART1 clustering capabilities are included to illustrate the performance analysis to begin from the basis. The main inspiration for this project is from the Text Book, "Simulating Neural Networks with Mathematica" by James A. Freeman. The processing equations that are dealt in this project are caught hold from the esteemed columns of the same.And also, it will be worth while to mention about the IEEE Transaction, "FUZZY ARTMAP : A Neural network architecture for Incremental Supervised Learning of Analog Multidimensional Maps" in which processing equations of Fuzzy ART are derived.

CHAPTER III

SOFTWARE AND HARDWARE REQUIREMENTS

3.1 USER INTERFACE

3.1.1 GUI

The system under design does not uses any Graphical User interfaces. This is since because the input to the system is mainly binary and real-valued analog input patterns.

3.1.2 CLI

Yes, the system under design invokes by itself on the command line with input file names as arguments and the system will be a powerful clustering tool under DOS platform.

3.1.3 API

Application programming interfaces are not required by the system under design and there is no need for invoking the routines using public interfaces.

3.1.4 DIAGNOSTICS

Any error encountered during run-time will be properly addressed to the users by invoking appropriate debugging routines.

3.2 HARDWARE SPECIFICATION

The system under design requires no additional hardware interfaces to be attached and it do need at least 64MB RAM as a major requirement and it can be run on any Celeron or Intel

Processors.

3.3 COMMUNICATIONS INTERFACES

The system do not require any Network interfaces.

3.4 SOFTWARE INTERFACES

The product under design require Turbo C or Borland C as a only software interface to be present.

CHAPTER IV

PROPOSED APPROACH TO THE PROBLEM

One of the nice features of human memory is its ability to learn many new things without necessarily forgetting things learned in the past. It is easier for humans to recognize the input patterns on viewing them and they will be able to judge correctly whether the input pattern was seen previously or it is a new one and they can update their memory accordingly.

Is there an intelligent learning system, to program computers to do the same things that we humans do? This is where, the implementation of "Neural Networks" architecture comes into effect. Humans can do this by developing the algorithm and writing the program that enables a computer to perform its function.

Many Neural Network architectures are available to do the above function. The flaw in those systems is, there is no built-in mechanism for the network to be able to recognize the novelty of the input. The Neural Network doesn't know that it doesn't know the input pattern. On the other hand, suppose that an input pattern is simply distorted, the network treats this pattern as a totally new pattern, then it may be over-working itself to learn that it has already learned in a slightly different form. This dilemma is known as stability-plasticity dilemma.

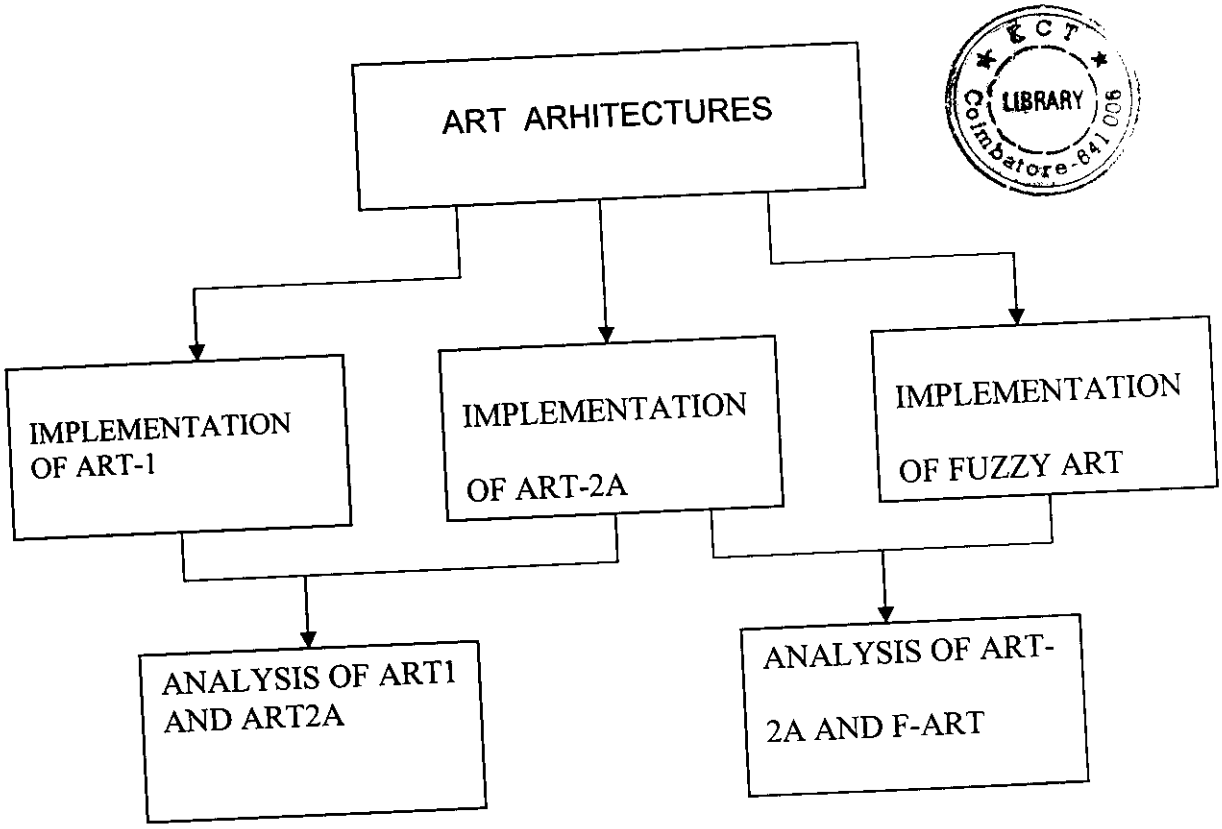
Adaptive resonance theory (ART) attempts to address the stability-plasticity dilemma. ART architectures are Neural Networks, that self organize stable recognition codes in real time in response to the arbitrary sequence of input patterns.

In this project titled, "Analysis of ART Network Clustering Performance", clustering performance of different types of ART networks: ART-1, ART-2A, Fuzzy ART is compared and their performances are analyzed.

Also this project concentrates on the comparative analysis of clustering properties for several variants of ART networks on input patterns of arbitrary dimensions, which illustrate the geometric characteristics of ART clustering and internal representation of knowledge by prototypes.

BLOCK DIAGRAM OF THE APPROACH

The block diagram of the approach is as shown in the fig 4.1. The first step is to implement ART-1. Secondly, ART-2A is implemented and its clustering performance for binary input vectors is compared with that of ART-1. Finally Fuzzy ART is implemented and its clustering performance for real-valued input vectors is compared with ART-2A.



4.1 Block Diagram

ART-1 : This is a binary version of ART, i.e. it can cluster binary input vectors.

ART-2A : This is an analog version of ART, i.e. it can cluster binary and real valued input vectors.

Fuzzy ART : This method is a synthesis of ART and fuzzy logic. It is also used to cluster real-valued input vectors

CHAPTER V

DESIGN DETAILS

5.1 PRODUCT FUNCTIONS

The product that is being designed will be helpful to impart the pattern clustering problem in any real time situation. Also, ART1 implementation which is being done here will be helpful to address "Optical Character recognition problem".

5.2 SIMILAR SYSTEM INFORMATION

The proposed system in the IEEE article is the main source of inspiration and it purely follows the conceptual approach described there.

5.3 USER CHARACTERISTICS

The user community should be able to run this program on DOS or Windows(9x) platforms either using Turbo C or Borland C compilers.

5.4 USER PROBLEM STATEMENT

Self-organized clustering is a powerful tool, whenever huge sets of data have to be divided into separate categories. Adaptive resonance (ART) architectures are neural networks, that self organize stable recognition codes in real time in response to arbitrary sequence of input patterns.

CHAPTER VI

ART-1 AND ART-2A

6.1 ART-1

ART is an extension of the competitive learning schemes. A key to solving the stability-plasticity dilemma in ART is to add a feedback mechanism between the competitive layer and the input layer of a network. This feedback mechanism facilitates the learning of new information without destroying old information automatic switching between stable and plastic modes, and stabilization of the encoding of the classes done by the nodes. This results in the nature of their input patterns. ART 1 networks require that the input vectors be binary, thus it can cluster binary input vectors.

6.2 ART-1 ARCHITECTURE

The ART1 architecture is as shown in figure 6.1. The overall architecture of the ART1 network is divided into two subsystems namely.

Attentional Subsystem and Orienting Subsystem

6.2.1 ATTENTIONAL SUBSYSTEM

The attentional subsystem is characterized by feedforward and feedback connections. This system determines whether the input pattern matches with one of the prototypes stored. If a match occurs a resonance is established.

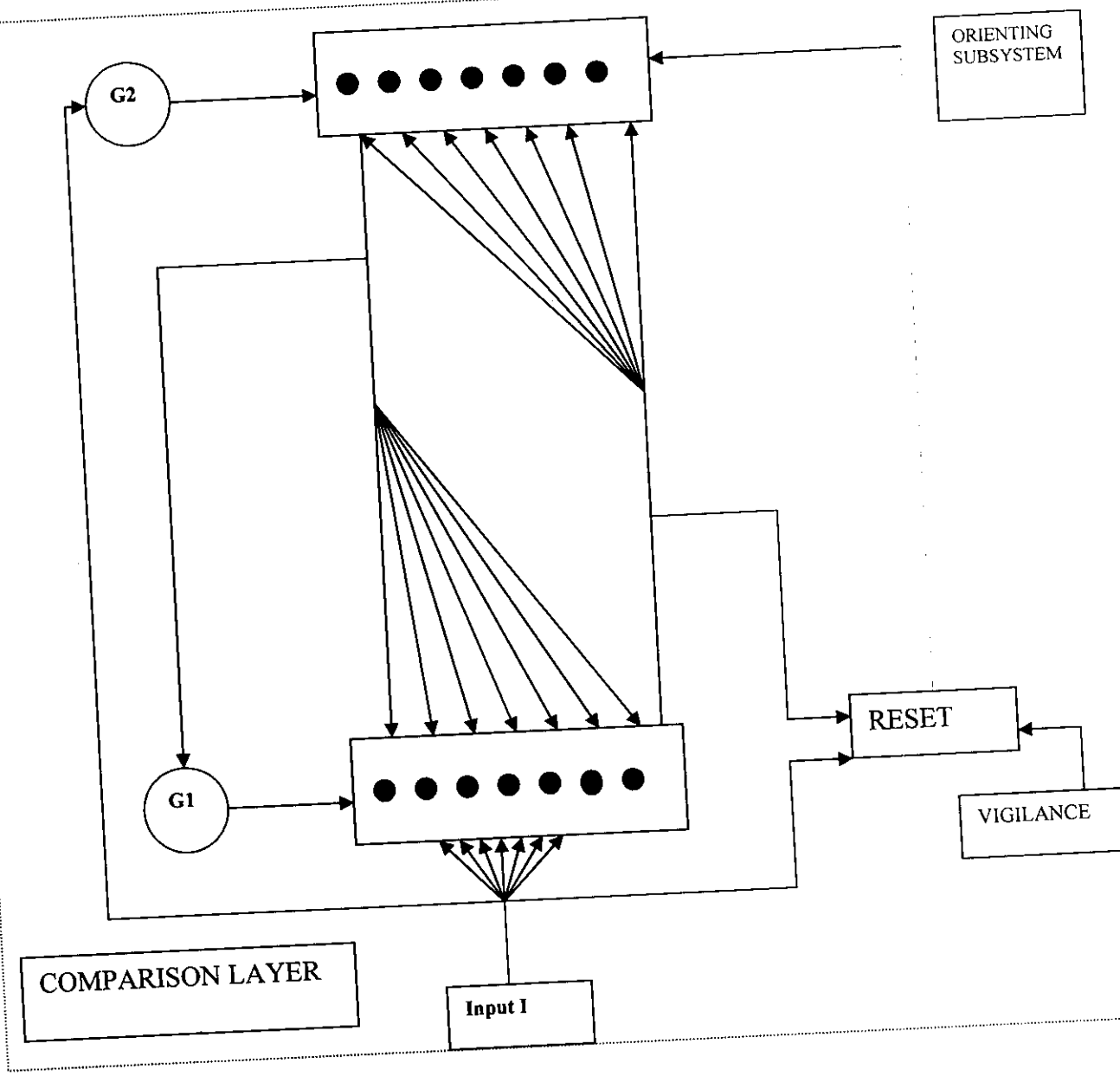


Fig. 6.1 ART-1 Architecture

The attentional subsystem consists of two layers of neurons called the comparison layer and recognition layer. The neurons in the comparison layer respond to the features in the input pattern. The synaptic connections (weights) between the two layers are modifiable in both directions. The recognition layer neurons have inhibitory connections that allow for a competition. The classification decision is indicated by a single neuron in the recognition layer that is fired.

6.2.2 ORIENTING SUBSYSTEM

The orienting system is responsible for sensing mismatches between bottom-up and top-down patterns on the comparison layer. The value of vigilance parameter measures the degree to which the system discriminates between different classes of input pattern. Depending on this vigilance parameter the orienting subsystem may or may not cause a reset.

The orienting subsystem can be modeled as a single processing, A, with an output to each unit on the recognition layer. The inputs to A are the outputs of comparison units, S, and the input vector are all equal to a value P: those on the connections from the comparison layer are all equal to a value-Q. The net input to A is then $P|I| - Q|S|$. The output of A switches on if the net input becomes nonzero :

$$P|I| - Q|S| > 0$$

Or

$$P|I| > Q|S|$$

$$P/Q > |S|/|I|$$

The quantity P/Q is given the name vigilance parameter and is usually identified by a symbol,ρ Thus, activation of the orienting subsystem is prevented if

$$|S|/|I| >= \rho$$

The recognition layer is inactive when $|S| = |I|$. The orienting subsystem must not set a reset signal to recognition layer at that time. From equation above. we get a condition on the vigilance parameter:

$$\rho <= 1$$

A subsequent condition on P and Q :

$$P <= Q$$

The value of vigilance parameter measures the degree to which the system discriminates between different classes of input patterns. Because of the way it is defined, it implements a self-scaling pattern match. By self-scaling we mean that the presence or absence of a certain feature in defining the pattern class.

The value of ρ determines the granularity with which input patterns are classified by the network. For a given set of input patterns to be classified, a large value of ρ will result in finer discrimination between classes than a smaller value of ρ.

6.3. ART-1 ALGORITHM

Step 1- Preprocessing : The input patterns which is a binary vector consisting of 1's and 0's is fed to the network. The input pattern I = (i1, i2, i3,.....i_n). The feedback from recognition layer is set to zero initially. Every neuron of an output layer receives a bottom-up net activity tj, built from all the comparison layer outputs S = I. The vector elements of T= t1,t2,t3,....tn) can be seen as results of comparisons between input pattern I and prototypes W_1 = (W_{11}, W_{12},.....W_{1m}),.....,Wn = (W_{n1}, W_{n2},.....W_{nm}). These Prototypes are stored in synaptic weights of the connections between comparison and recognition layer neurons.

Step 2- Choice: The bottom-up net activities lead to the choice of winning neuron in the recognition layer. The recognition layer neuron J receiving the highest net activity tj, sets its output to 1 while all other neurons,

$$U_j = \begin{cases} 1 \text{ if } t_j > \max(t_k : k!=j) \\ 0 \text{ otherwise} \end{cases}$$

One possible way to compute net activities t_j, and by that measure the similarity between I and W_J, is weighted as

$$Tj = \Sigma\, w_{ij}, i_i$$

The pattern again fed back as top-down activity to the comparison layer

Step 3-Match:The comparison layer checks for a match between the pattern fed back and the input pattern. If the degree of similarity is greater than vigilance value a match occurs and adaptation takes place. Else the reset signal is set and the winning neuron is inhibited.The input pattern is again fed and the process continues until all the neuron in the recognition layer has been fired after which a new cluster is formed for the input pattern fed.

Step 4- Adaptation: After the winning neuron J if the recognition layer has been found the corresponding prototype $W_j = (w_{1j}, w_{2j}, \ldots\ldots\ldots w_{mj})$ is adapted to input pattern I. Adaptation takes place by altering the weights suitably to accommodate the input pattern in the already stored pattern. One suitable method for adaptation is to slightly move the weight W_j towards the input pattern I

$$W_j^{(new)} = \eta.I + (1 - \eta)W_j^{(old)}$$

The constant learning rate ηέ [0,1] is chosen to prevent prototype Wj from moving too fast and therefore destabilizing the learning process.

6.4 ART-2A

ART-2 is an analog version of ART, that is it can cluster real valued input vectors. The ART-2A is a fast version of ART-2. ART-2 can categorize both binary and analog input vectors.

ART-2 differs from ART-1 only in the nature of the input patterns ART-2 accepts analog (or gray-scale) vector components as well as binary .components. The capability represents a significant enhancement to the system. Beyond the surface difference between ART-1and ART-2 lie architectural differences that give ART-2 its ability to deal with analog patterns.These differences are sometimes less complex than the corresponding ART-1 structures.

6.5 ART-2 ARCHITECTURE

The comparison layer of ART-2 is complex because continued valued input vectors are used. Additional nodes are designed for the following reasons :

1. Allowance for noise suppression
2. Normalization.
3. Comparison of top-down and bottom-up signals used for the reset mechanism.
4. Dealing with real-valued data that may be arbitrarily close to one another.

The recognition layer consist of six types of units w,x,u,v,p,q. There are N units in each of these units where N is the dimension of input pattern. Supplemental units are used to compute the norm of all input patterns which it receives and send to the inhibitory signal to the output unit to which it connects. The action of the recognition layer is essentially unchanged when compared to ART-1 .The units compute in a winner-take-all mode for the right to learn input patterns.

5.6 ART-2A ALGORITHM

Step 1- Preprocessing: No negative values are allowed and all uncoded input vectors. A are normalized to unit Euclidean length,

$$I = A / (\Sigma a_i^2)^{1/2} = A / \|A\|$$

$$A_i > 0 \text{ for all I, } \|A\| > 0.$$

Step 2- Choice: Bottom-up net activities leading to the choice of a prototype, are determined by

$$T_j = \begin{cases} I \,.\, W_j \text{ if j indexes a committed prototype} \\ \acute{\alpha} \,.\, \Sigma I_i \text{ otherwise} \end{cases}$$

$$\text{where } 0 <= \acute{\alpha} <= I / m^{1/2}$$

Bottom-up net activities are determined differently for previously committed and uncommitted prototypes. The choice parameter $\acute{\alpha} >= 0$ again defines the maximum depth of search for a fitting cluster. With $\acute{\alpha}=0$, all committed prototypes are checked before an uncommitted prototype is chosen as winner.

Step 3- Match: Resonance and adaptation occurs either if J is an index of an uncommitted prototype or if J is a committed prototype and

$$\rho <= I.W_j = T_j$$

Step 4- Adaptation: Adaptation of the final winning prototype requires a shift toward the current input pattern

$$W_j^{(new)} = \eta.I + (1 - \eta)W_j^{(old)} \quad 0 <= \eta <= 1$$

ART-2A type networks always use fast-commit slow-recode mode therefore the learning rate is set to $\eta=1$, if J is an uncommitted prototype and to lower values for further adaptation.

Since match and choice do not evaluate the values of uncommitted prototypes, there is no need to initialize them with specific values ART-2A related networks should not be used in fast-learning mode with $\eta = 1$, because prototypes then begin to "jump" between all patterns assigned to their cluster, instead of converging toward their mean.

CHAPTER VII

FUZZY ART

7.1 DEFINITION

Fuzzy ART achieves a synthesis of ART and Fuzzy Logic by analyzing the close similarities between the computations of fuzzy subset hood and ART category choice resonance and learning.

7.2 ARCHITECTURE

The architecture of fuzzy ART is as shown in figure 7.1

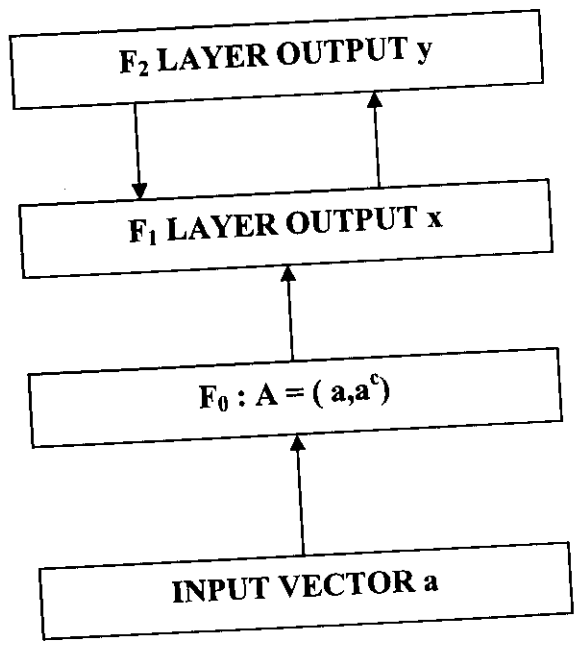


Fig. 7.1 Block Diagram of Fuzzy ART Architecture

The various notations used in Fuzzy ART given as follows

7.3 ART FIELD ACTIVITY VECTORS

Each ART system includes

1. A field F_0, of nodes that represent a current input vector
2. a field F_1, that receives both bottom-up input from F_0 and top-down input from a field F_2,
3. A field F_2,that represents a active code or category.

The activity vector of each field is as follows:

1. The F_0 activity vector is denoted as $I=(I_1,\ldots\ldots I_m)$, with each component I_i in the interval [0,1],i=1,2,....M.
2. The F_1 activity vector is denoted $X=(x_1,\ldots.x_m)$ and
3. The activity vector is denoted $Y=(y_1,\ldots\ldots y_n)$. The number of nodes in each field is arbitary.

7.3.1 WEIGHT VECTOR

Associated with each F_2 category node j (j=1,2,...N) is a vector $w_j=(w_{j1},\ldots,w_{jm})$ of adaptive weights or LTM traces.

Initially,

$$W_{j1}(o)=\ldots=w_{jm}(o)=1$$

Then each category is said to be uncommitted. After a category is selected for coding, it becomes commited. The fuzzy ART weight vector wj subsumes both the bottom-up and top-down weight vectors of ART1.

7.3.2 PARAMETERS

Fuzzy ART dynamics are determined by

i) Choice parameter $\alpha > 0$

ii) Learning rate parameter $\beta \in [0,1]$ and

iii) Vigilance parameter $\rho \in [0,1]$.

7.3.3 COMPLEMENT CODING

Complement coding is a normalization rule that preserves amplitude information. Complement coding represents both the on-response and off-response to an input vector a. Let a itself represent the on-response. The complement of a denoted By a^c represents the off-response where $a^c{}_i = 1 - a_i$.

The complement coded input I to the field F1 is the 2M dimensional vector:

$$I=(a,a^c)=(a_1,.....a_m,a_1{}^c,....a_m{}^c)$$

$$| I | = | a, ac| = M$$

So inputs preprocessed into complement coding form are automatically Normalized. Where complement coding is used, the initial condition is Replaced by

$$W_{ji}(0)=....W_{j2M}(0)=1$$

The weight vector W_j can be written in complement coding form

$$W_j = (U_j.V_j{}^c).$$

7.4 FUZZY ART ALGORITHM

Step1 – Preprocessing: Proliferation of categories is avoided in Fuzzy ART if inputs are normalized. A normalization procedure coding leads to a systematic theory in which the AND operator($\wedge$) and OR operator($\vee$) of fuzzy logic play complement roles. Complement coding I=a/|a|, uses on cells and off cells to represent the input pattern, and preserves individual feature amplitudes while normalizing, the total on cell-off cell vector. Normalization can be achieved by preprocessing each incoming vector

$$|a| = \sum I = 1 \ |ai, ai \in [0,1] \text{ for all i.}$$

Step 2- Category Choice : For each input I and F2 node j, the choice Function T_j is defined by

$$T_j(I) = |I \wedge W_j| / (\alpha + |W_j|)$$

Where the fuzzy AND operator $\wedge$ is defined by

$$(p \wedge q) = \min(p_i, \wedge q_i)$$

and the norm |.| is defined by

$$|p| = \sum |p_i|$$

For any M-dimensional vectors p and q,

$$\text{And } |w_j| = \sum (u_j \wedge a_i) + \sum [1 - (v_j \vee a_i)$$

$$= M - | \vee a - \wedge a|$$

where $(p \vee q) = \max(pi, qi)$

$$\alpha >= 0.001$$

The system is to make a category choice when at most one F2 node can become active at a given time. The category choice is indexed by J, where

$$\mathbf{Tj = max\{ Tj : j = 1,..N\}}$$

If more than one T_j is maximal, the category j with the smallest index is chosen. In Particular , nodes become committed in the order j=1,2,3,..... When the Jth category is chosen $Y_1 = 1$ and $Y_j = 0$ for j != J. In a choice system, the F1 activity vector obeys the equation :

$$\mathcal{X} = \begin{cases} I \text{ if } F2 \text{ is inactive} \\ I \cap Wj \text{ if } Jth \text{ node} \\ \quad F2 \text{ is chosen} \end{cases}$$

Step 3 – Match: The similarity of input I and current winning prototype Wj is measured by the degree of I being a fuzzy subset of Wj. Resonance occurs if the match function,

$$| I \wedge Wj | / | I | >= \rho$$

That is, when the Jth category is chosen.

Mismatch reset occurs if

$$| I \wedge Wj | / | I | >= \rho$$

Then the value of Tj is set to 0 for the duration of the input presentation to prevent the Persistent selection of the same category during search. A new index J is then chosen by eqn. The search process continues until the chosen J satisfies eqn.

Step 4 – Adaptation : Once the search ends, the weight vector Wj is updated according to the equation

$$W_j = \beta(I \wedge W_j) + (1- \beta)W_j$$

Fast learning corresponds to setting $\beta = 1$. In contrast lower learning rates lead to a slow learning mode.

Fast-commit Slow Recode Option : For efficient coding of noisy input sets, it is useful to set $\beta = 1$ when J is an uncommitted node, and then to set $\beta<1$ after the category is active. Proliferation problem can occur in certain analog ART systems when a large number of inputs erode the norm of weight vectors. Complement coding solves the problem.

CHAPTER VIII

EXPERIMENTATION AND RESULTS

8.1 INPUT PATTERNS PRESENTED :

The following input patterns were presented for training

(i) Numerals

(ii) Alphabets

(iii) Tamil Characters

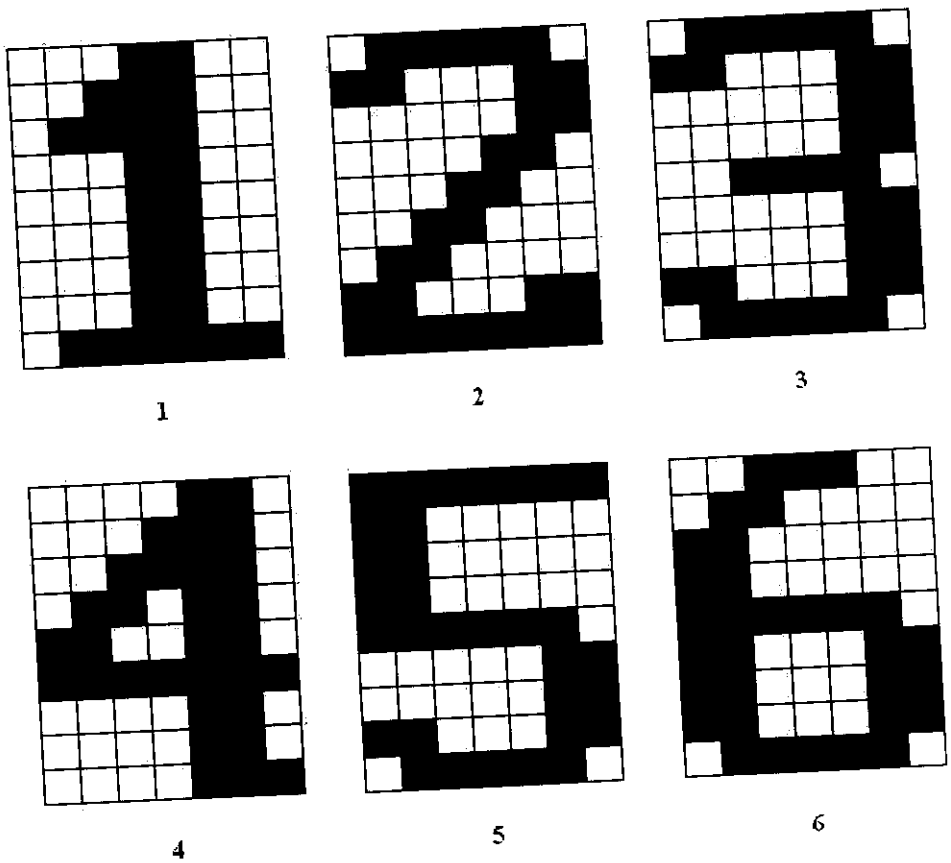
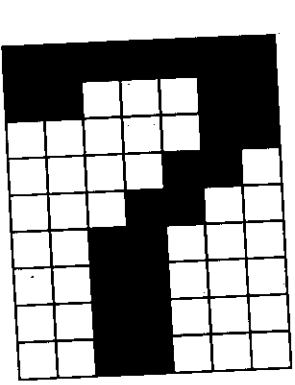
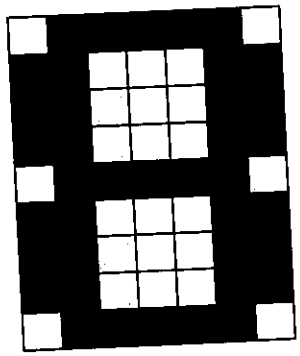


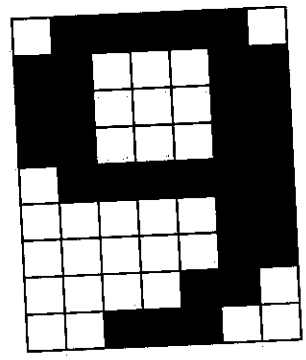
Fig. 8.1 Training Pattern (Numerals)



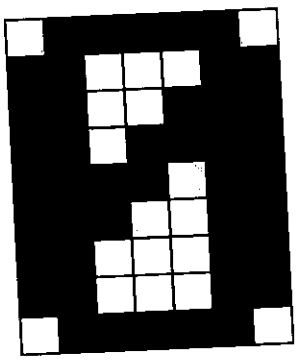
7



8

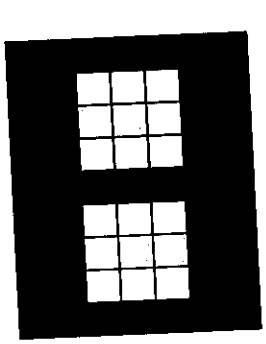


9

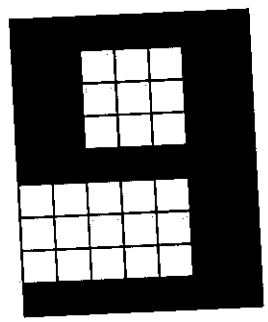


10

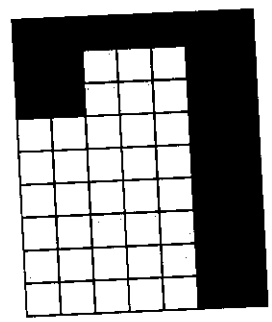
Fig. 8.1 Training Patterns of Fig. 8.1(continued)



1



2



3

Fig 8.2 Test Patterns with noise

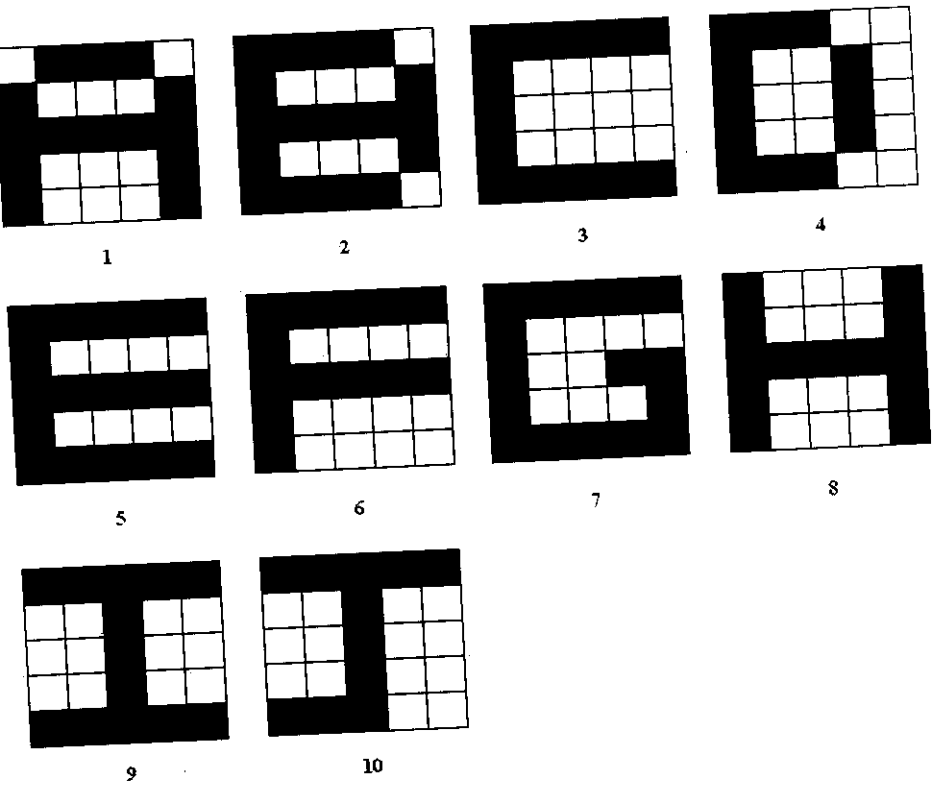


Fig 8.3 Training Patterns(alphabets)

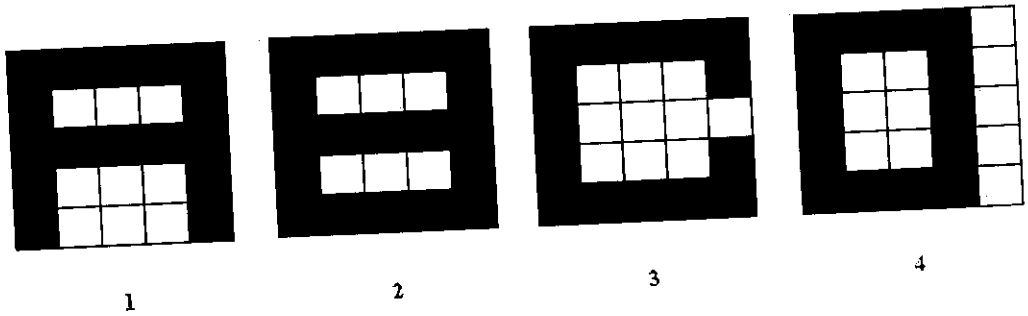


Fig 8.4 Test Patterns with noise

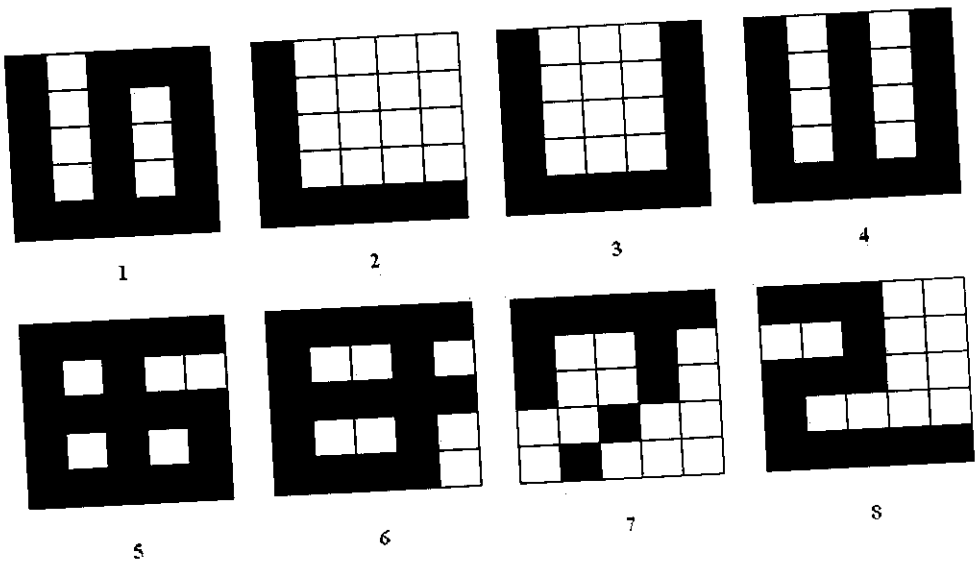


Fig 8.5 Training Patterns (Tamil)

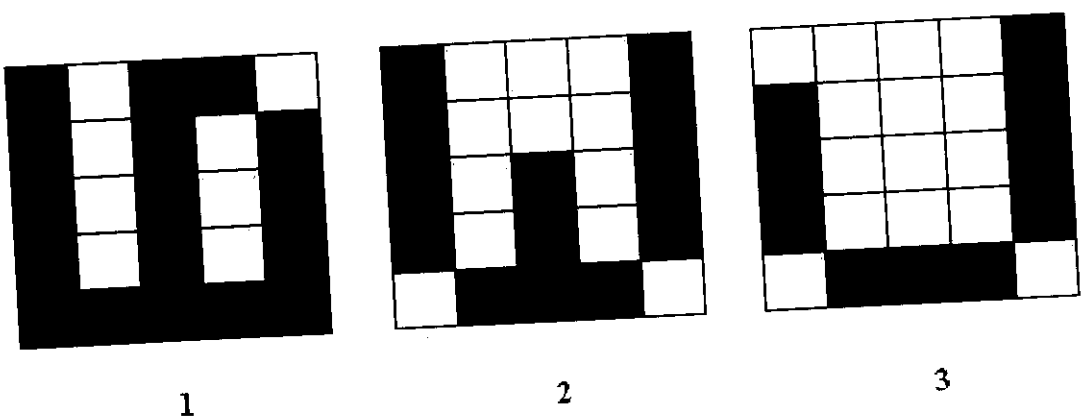


Fig 8.6 Test patterns with noise

8.2 PERFORMANCE ANALYSIS:

The clustering performance of ART-1 and ART-2A for numerals is tabulated in Table 8.1 and that of ART-2A and Fuzzy ART in Table 8.2.

Table 8.1 Comparison of ART-1 and ART-2A for numerals

PATTERN NUMBER	ρ=0.85		ρ=0.92		ρ=0.95	
	ART-1	ART-2A	ART-1	ART-2A	ART-1	ART-2A
1	1	1	1	1	1	1
2	2	2	2	6	2	6
3	3	3	3	3	3	3
4	4	4	4	4	4	4
5	5	2	5	2	5	8
6	6	5	6	5	6	5
7	7	6	7	7	7	9
8	8	2	9	2	8	9
9	9	5	8	5	9	5
10	9	3	10	3	10	10
PERFORMANCE	90%	60%	100%	70%	100%	80%

Table 8.2 Comparison of ART-2A and Fuzzy ART for Numerals

PATTERN NUMBER	ρ=0.85		ρ=0.92		ρ=0.95	
	ART-2A	F-ART	ART-2A	F-ART	ART-2A	F-ART
1	1	1	1	1	1	1
2	2	1	6	2	6	2
3	3	1	3	2	3	1
4	4	1	4	3	4	3
5	2	1	2	1	8	2
6	5	1	5	3	5	3
7	6	1	7	3	9	4
8	2	1	2	1	9	5
9	5	1	5	2	5	5
10	3	1	3	4	10	4
PERFORMANCE	60%	10%	70%	40%	80%	50%

The clustering performance of ART-1 and ART-2A for alphabet is tabulated in Table 8.3 and that of Fuzzy ART and ART-2A in Table 8.4

Table 8.3 Comparison of ART-1 and ART-2A for alphabet

PATTERN NUMBER	ρ=0.85		ρ=0.92		ρ=0.95	
	ART-1	ART-2A	ART-1	ART-2A	ART-1	ART-2A
1	1	1	1	1	1	1
2	2	1	2	3	2	3
3	3	2	3	5	3	5
4	4	2	4	5	4	5
5	6	2	6	5	9	5
6	5	3	5	3	5	3
7	6	2	6	2	6	2
8	7	4	7	6	7	6
9	9	2	9	2	10	2
10	8	4	8	4	8	4
PERFORMANCE	90%	40%	90%	60%	100%	60%

Table 8.4 Comparison of ART-2A and Fuzzy ART for alphabet

PATTERN NUMBER	ρ=0.85		ρ=0.92		ρ=0.95	
	ART-2A	F-ART	ART-2A	F-ART	ART-2A	F-ART
1	1	1	1	1	1	1
2	1	1	3	1	8	2
3	2	1	2	2	2	3
4	2	2	2	2	2	4
5	2	2	2	1	2	5
6	3	1	3	3	9	6
7	4	1	4	3	4	7
8	5	1	7	4	10	8
9	6	3	6	5	6	9
10	5	4	5	6	5	10
PERFORMANCE	60%	40%	70%	80%	80%	100%

The clustering performance of ART-1 and ART-2A for Tamil characters is tabulated in Table 8.5 and that of ART-2A and F-ART in Table 8.6.

Table 8.5 Comparison of ART-1 and ART-2A for Tamil Characters

PATTERN NUMBER	ρ=0.85		ρ=0.92		ρ=0.95	
	ART-1	ART-2A	ART-1	ART-2A	ART-1	ART-2A
1	3	1	3	1	7	1
2	1	2	1	2	1	2
3	2	3	2	3	2	3
4	3	4	3	4	3	4
5	7	5	7	5	8	5
6	4	6	5	6	5	6
7	5	2	7	2	6	2
8	6	5	4	5	4	7
PERFORMANCE	87.5%	75%	75%	75%	100%	87.5%

Table 8.6 Comparison of ART-2A and Fuzzy ART for Tamil characters

PATTERN NUMBER	ρ=0.85		ρ=0.92		ρ=0.95	
	ART-2A	F-ART	ART-2A	F-ART	ART-2A	F-ART
1	1	1	1	1	1	1
2	2	1	2	2	2	2
3	3	1	3	2	3	3
4	4	1	4	1	4	1
5	5	2	5	1	5	4
6	6	2	6	1	6	5
7	7	2	7	3	7	6
8	8	3	8	4	8	7
PERFORMANCE	100%	37.5%	100%	50%	100%	87.5%

3.3 RESULT

On comparing the clustering performance of the different types of ART architectures:

ART-1, ART-2A,Fuzzy ART from the tables, it is found that clustering in ART-1 is better than that of ART-2A with respect to binary input vectors.

Further, the clustering performance of ART-2A is better than that of Fuzzy ART for real valued input vectors.

CHAPTER IX

CONCLUSION

9.1 CONCLUSION

Leaving aside the biologically motivated aspects, ART turns out to be an effective, transparent clustering algorithm. Three different types of ART networks, ART-1,ART-2A, Fuzzy ART were inspected. Each variant is characterized by its preprocessing-, choice- match- and adaptation rule.

Arbitrary dimensional pattern sets illustrated the geometric nature of ART clusters. Fuzzy ART uses the degree of an input pattern being fuzzy subset of a stored prototype to measure the similarity between two patterns When using complement encoded input patterns, prototypes converge towards the common MIN and MAX values of all patterns assigned to the according cluster. Properties of ART networks depend on two main parameters ρ and η. Vigilance ρ defines the minimum similarity between patterns in one cluster in terms of the applied distance metric. Higher vigilance increase the total number of clusters set upon a static pattern set. If geometric preferences are given for a specific pattern set, as in our project, the number of clusters does not depend on the order of pattern presentation. Learning rate η regulates adaptation of stored prototypes towards input patterns. Fuzzy ART networks reach a state of temporarily stable prototypes, indicating the end of a training cycle on a fixed set of patterns.

All network weights are fixed, when all training Patterns are enclosed by the MIN- and MAX- bounds defined by the prototypes. The extension of the prototypes is limited by the vigilance parameter ρ. Once the maximum extension of a prototype has reached, no further patterns are assigned to the according cluster not lying completely within the MIN- and MAX bounds. This makes fuzzy ART highly sensitive to additional noise on trained input patterns and its output is unpredictable.

In most applications, where pure self-organized clustering is required, ART-1 is more appropriate when the binary input vectors in real world applications, where the input is real valued, ART-2A is more appropriate.

9.2 FUTURE EXTENSION

This Project can be extended by comparing the clustering performance of same architectures for higher dimensional input vectors and also for other architectures such as ART-3, Fuzzy ARTMAP, ARTMAP, Distributed ART, SMART etc. Clustering performances can also be tested with images, hand-written characters, face recognition , etc.

CHAPTER X

SOURCE CODE

```
/*IMPLEMENTATION OF ART-1*/
#include <conio.h>
#include <dir.h>
#include <ctype.h>
#include <dos.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <values.h>
#define BUFSIZE 2048

enum BOOL { FALSE = 0, TRUE = 1 };

struct F2
{
float T, *up, *dn;
};

struct VEC
{
char *name;
int *v;
};

int breed, hoog, scr_width, veclen, n_vec = 0, max_vec = 0, n_f2 = 0, max_f2 = 0,
n_used = 0, input_line;
```

```
float A1, B1, C1, D1, L, rho = 0, *S, *V;
struct VEC *vec = NULL;
struct F2 *f2 = NULL;
char *programname, buffer [BUFSIZE + 1], *no_mem_buffer, *infile = "patt.inp",
*no_mem_buffer, Out_of_memory [] = "Out of memory";
void show(int v), get_programname (char const *argv0), process_args (int argc, char
*argv []), setup(void), make_f2_unit (void), read_file (void), syntax (void), errit (char
const *format, ...), *s_malloc (size_t size), *s_realloc (void *block, size_t size);
char *s_strdup (char const *s), *getline (FILE *fp, enum BOOL required, char const
*filename);
char const *get_arg (int argc, char *argv[], int *index);
float *alloc_vector_float (int i1, float value);
int main(int argc, char *argv [])

{
int i,j,v,coded,f2win;
float max,sum,nwup,nwdn,S_mag,I_mag;
no_mem_buffer = (char *) malloc (1024);
get_programname (argv[0]);
process_args (argc, argv);
read_file ();
setup ();
coded = 0;
v = -1;
while (coded++ < n_vec) {
if (++v == n_vec)
v = 0;
printf ("%s:", vec [v].name);
if (n_used == n_f2)
make_f2_unit ();
I_mag = 0.0;
```

```
for (i = 0; i < veclen; i++) {
 S[i] =(((float) vec [v].v [i]) / (1.0 + A1 * (((float) vec [v].v [i]) + B1) + C1) > 0.0) ?
.0 : 0.0;
 I_mag += vec [v].v [i] ? 1.0 : 0.0;
 }
 for (j = 0; j< n_f2; j++) {
  sum = 0.0;
  for (i = 0; i < veclen; i++)
   sum += S [i] * f2 [j].up [i];
  f2 [j].T = sum;
  }
  for (;;) {
  max = -MAXFLOAT;
  for (i=0; i < n_f2; i++) {
   if (f2 [i].T > max) {
    max = f2 [i].T;
    f2win = i;
   }
  }
  if (max <= 0.0)
   errit ("Really weird!");
  S_mag = 0.0;
  for (i = 0; i < veclen; i++) {
  V[i] = f2 [f2win].dn [i];
  S_mag += ( S[i] = ((((float) vec [v].v [i]) + D1 * V[i] - B1) / (1.0 + A1 * (((float) vec
 [v].v [i]) + D1 * V[i]) + C1) > 0.0) ? 1.0 : 0.0);
   }

   printf ( "%i", f2win + 1);
   if (S_mag / I_mag < rho)
    f2 [f2win].T = -MAXFLOAT;
```

```
else
  break;

printf ("\n");
if (f2win >= n_used)
 n_used = f2win + 1;
for (i = 0; i < veclen; i++) {
 if (S [i] == 1.0) {
  nwup = L / (L - 1.0 + S_mag);
  nwdn = 1.0;
  } else
  nwup = nwdn = 0.0;
  if (nwup != f2 [f2win].up [i]) {
   coded = 0;
   f2 [f2win].up [i] = nwup;
   }
   if (nwdn != f2 [f2win].dn [i]) {
    coded =0;
    f2 [f2win].dn [i] = nwdn;
    }
   }
   if (coded == 0)
    show (v);
    }
   printf ("\n rho: %f\n\n", rho);
   getch();
   return 0;
   }

   void show (int nr)
   {
```

```
nt i,j,k,line,v,lines,v_per_line;
v_per_line = (scr_width - breed - 5) / (breed + 2);
lines = n_used / v_per_line;
if (lines * v_per_line < n_used)
 lines++;
for(line = 0; line < lines; line++) {
  for (j = 0; j < hoog; j++) {
   if (line == 0)
   {
   for (k = 0; k < breed; k++)
    printf (vec [nr].v [j * breed + k] ? "@" : ".");
    printf (" : ");
    }else
    printf ( "%*s  ", breed, "");
    for (i = 0; i < v_per_line; i++) {
     v = line * v_per_line + i;
    if (v < n_used) {
  for (k = 0; k < breed; k++)
 printf ((f2 [v].dn [k + breed * j] > 0.5) ? "@" : ".");
           sleep(2);
          printf ( "  ");
       }
      }
      printf ("\n");
      }
      printf ("\n");
    }
    }

    void setup ()
    {
```

```
truct text_info ti;
gettextinfo ( &ti);
scr_width = ti.screenwidth;
A1 = 1.0;
B1 = 1.5;
C1 = 5.0;
D1 = 0.9;
L  = 3.0;
if (rho <= 0.0)
 rho = 0.5;
S = alloc_vector_float (veclen, 0.0);
V = alloc_vector_float (veclen, 0.0);
 }

 void make_f2_unit()
 {
  while (n_f2 >= max_f2) {
   max_f2 += 64;
   f2 = (struct F2 *) s_realloc (f2, max_f2 * sizeof(struct F2));
   }
   f2 [n_f2].up = alloc_vector_float(veclen, 0.8 * (L / (L - 1.0 + (float) veclen)));
   f2 [n_f2++].dn = alloc_vector_float(veclen, 1.4 * ((B1 - 1.0) / D1));
   }
   void read_file ()
   {
    int i,j,k;
    char const *file_error = "File does not exist\"%s\",%i", *illegal = "Illegal";
    FILE *fp;
    if( ! infile)
     errit ("Illegal file name");
```

```
p = fopen (infile, "r");
f (! fp)
errit ("Cannot open \"%s\" file does not exist", infile);
input_line = 0;
getline (fp, TRUE, infile);
if (sscanf (buffer, "%i %i", &breed, &hoog) !=2)
 errit(file_error, infile, input_line);
 if (breed < 1)
  errit (illegal, "breedte");
 if (hoog < 1)
  errit (illegal, "hoogte");
 veclen = breed * hoog;
 while (getline (fp, FALSE, NULL)) {
  while (n_vec >= max_vec) {
   max_vec += 64;
   vec = (struct VEC *)
    s_realloc (vec, max_vec * sizeof (struct VEC));
    }
    vec [n_vec].name = s_strdup (buffer);
    vec [n_vec].v = (int *) s_malloc (veclen * sizeof(int));
    for (i = 0; i < hoog; i++) {
     getline (fp, TRUE, infile);
     j = 0;
     k = -1;
      while (j < breed) {
       switch (buffer [++k]) {
     case ' ':
     case '\t':
      break;
     case '-':
```

```
ase '0':
vec [n_vec].v[i * breed + j++] = 0;
break;
case '+':
case '1':
 vec [n_vec].v[i * breed + j++] =1;
 break;
default:
 errit (file_error, infile, input_line);
 }
 }
    }
   n_vec++;
    }
    fclose (fp);
    }

     char *getline (FILE *fp, enum BOOL required, char const *filename)
     {
      int i;
      for(;;) {
          if (fgets (buffer, BUFSIZE, fp) == NULL) {
           if (required)
            errit ("Unexpected end of file in %s", filename);
            else
             return NULL;
            }
            input_line++;
            i = strlen (buffer);
            while (i && isspace (buffer [i-1]))
             buffer [--i] = '\0';
```

```
  i = 0;
  while (buffer [i] && isspace (buffer [i]))
   i++;
  if (buffer [i] == '#')
   continue;
  if (buffer [i]) {
   memmove (buffer, buffer+i, strlen (buffer) + 1);
   return buffer;
   }
  }
}
    void syntax ()
    {
     fprintf (stderr, "\nAdaptive Resonance Theory 1\n\n" "Syntax: %s -i string [-r
float]\n\n" "-i : inputfile\n" "-r : vigilance: 0.0 < rho <= 1.0\n\n", programname);
      exit (1);
      }


      void process_args (int argc, char *argv [])
      {
       int i;

       rho=.85;
       if (argc == 1)
        syntax ();
       for (i = 1; i < 2; i++) {
        if (argv [i][0] != '-' && argv [i][0] != '/')
         errit ("Illegal argument");
         switch (argv [i][1]) {
```

```
case 'i':
  infile = s_strdup (get_arg (argc, argv, &i));
 break;



 case 'r':
  rho = atof (get_arg (argc, argv, &i));
  if (rho <= 0.0 || rho > 1.0)
   errit ("Illegal value for rho");
   break;
  default:
   errit ("Illegal option '%s'", argv [i]);
    }
   }
  }
  char const *get_arg (int argc, char *argv [], int *index)
  {
   if (argv [*index][2])
    return argv [*index] + 2;
   if (*index == argc - 1)
    errit ("Argument not enough '%s'", argv [*index]);
    return argv [++*index];
   }
   void get_programname (char const *argv0)
   {
    char name [MAXFILE];
    fnsplit (argv0, NULL, NULL, name, NULL);
    programname = s_strdup (name);
    }
    float *alloc_vector_float (int i1, float value)
    {
```

```
int i;
float *f;
f = (float *) s_malloc (i1 * sizeof (float));
for (i = 0; i < i1; i++)
 f [i] = value;
return f;
 }
void *s_malloc (size_t size)
{
 void *p;
 p = malloc (size);
 if (! p) {
  free (no_mem_buffer);
  errit (Out_of_memory);
  }
  return p;
  }
  void *s_realloc (void *block, size_t size)
  {
    void *p;
    p = realloc (block, size);
    if (! p) {
     free (no_mem_buffer);
     errit (Out_of_memory);
     }
     return p;
     }
     char *s_strdup (char const *s)
     {
      char *s1;
      if (s) {
```

```
s1 = (char *) s_malloc (strlen (s) + 1);
strcpy (s1, s);
} else{
 s1 = (char *) s_malloc (1);
 s1 [0] = '\0';
 }
 return s1;
 }

 void errit(char const *format, ...)
 {
  va_list list;
  fprintf (stderr, "\nError %s: ", programname);
  va_start (list, format);
  vfprintf (stderr, format, list);
  fprintf (stderr, "\n\n");
  exit (1);
  }
        // End of ART1
```

```
*IMPLEMENTATION OF ART-2A*/
#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<math.h>
int N,M;
double *X, *Xp, *W, *Wp, *U, *Up, *P, *Pp, *V, *Vp, *Q, *Qp, *R, *Y, *mis;
double **zij, **zji;
double *old1M, *old2M;
int n_mis=0;
double rho=0.9, a=5.0, b=5.0, c=0.225, d=0.8, theta=0.3, ac_dif=0.001, z_dif=0.01;
FILE *fout, *flog;
char fname[80], out_name[80], log_name[80];
double h=0.1;
int nsteps=100;
double mag_x;
int F2_on;
int l_flag;
int rsnc=0;
double rect();
double HVS();
double f();
double fab();
double L2_norm();
double vec_diff();
double RungeKutta();
FILE *datain, *fopen();
int p_dim,p_num,*IL;
double **I, *p_mag;
int ppc, *c_p;
```

```
oid get_inputs();
oid select_one();
oid select_cycle();
oid init_all();
oid BU_reset();
oid TD_reset();
oid new_learning();
void free_all();
void p_normalize();
void p_display();
void get_inputs()
{
 int i,j;

 char in_name[80];
 if (fname[0] == '\0') {
 fprintf(stderr, "Please enter input file name (no extension) --> ");
 scanf("%s",fname);
 }
 sprintf(in_name,"%s.inp",fname);
 if((datain=fopen(in_name, "r")) == NULL) {
 fprintf(stderr,"Fatal Error: Cannot open %s\n",in_name);
 exit(1);
 }
 fprintf(stderr, "\n**************************** reading %s\n\n", in_name);
 fscanf(datain,"%d %d", &p_dim,&p_num);
 fprintf(stderr, "Allocating and reading %d %d-D vectors\n",p_num,p_dim);
 if ((I=(double **) calloc(p_num,sizeof(double *))) == NULL) {
 fprintf(stderr, "Error: could not allocate %d %d-D patterns!\n", p_num,p_dim);
 exit();
 }
```

```
or (i=0; i<p_num; i++)
f ((I[i]=(double *)
calloc(p_dim,sizeof(double))) == NULL) {
fprintf(stderr, "Error: could not allocate %d %dD patterns!\n", p_num,p_dim);
exit();
}
if((p_mag=(double *) calloc(p_num, sizeof(double))) == NULL) {
fprintf(stderr, "Error: could not allocate %d %d-D patterns!\n", p_num,p_dim);
exit();
}
new_learning();
for (i=0; i<p_num; i++)

{
p_mag[i] = 0.0;
for (j=0; j<p_dim; j++)
if (fscanf(datain, "%lf",I[i]+j) == EOF) {
fprintf(stderr,"Error: Unexpected End-of-file in %s\n", in_name);
exit(1);
}
p_mag[i] = L2_norm(I[i],p_dim);
if (p_mag[i] <= 0.0)
{
fprintf(stderr, "(Warning: pattern %d has mag=0.0)",i);
p_mag[i] = 1.0;
}
printf(". ");
}
if (fscanf(datain,"%d %d", &M,&N) != EOF) {
if(M < p_dim)
    fprintf(stderr, "Warning: Too few F0-1 nodes: changed M to %d\n",(M=p_dim));
```

```
if (N <= M)
 fprintf(stderr, "Warning, requested only %d total nodes. Assigning %d\n", N,
=M+p_num);
 }
 else {
  M=p_dim;
  N=M+p_num;
  fprintf(stderr, "No values for M and N found. Assigning M=%d and
N=%d\n",M,N);
  }
  fclose(datain);
  printf("done.\n");
  }
  void select_one(i)
  int i;
  {
   do
   {
    fprintf(stderr, "select pattern for cycle %d -->",i);
    scanf("%d", &c_p[i]);
    if (c_p[i] >= p_num || c_p[i] < 0)
     fprintf(stderr, "Invalid choice. try again.\n");

 } while (c_p[i] >= p_num || c_p[i] < 0);
      }
      void select_cycle(num)
      short num;
      {
       int i;
       if (num< 1)
       {
```

```
fprintf(stderr, "How many patterns per cycle?");
scanf("%d", &ppc);
}
else
ppc = num;
if (c_p != NULL)
free(c_p);
if ((c_p=(int *) calloc(ppc,sizeof(int))) == NULL)
{
fprintf(stderr,"Error: could not allocate %d integers.\n",ppc);
exit();
}
for (i=0; i<ppc; i++)
select_one(i);
}
void init_all()
{
int i,j;
void free_all(), get_inputs();
free_all();
BU_reset();
TD_reset();
get_inputs();
}
void new_learning()
{
int i;

for(i=0; i<p_num; i++)
IL[i] = -1;
}
```

```
void free_all()
{
int i;
for(i=0; i<p_num; i++)
  free(I[i]);
free(I);
free(IL);
free(p_mag);
if(c_p != NULL)
 free(c_p);
 }

 void p_display(p_n)
 int p_n;
 {
 int i,j;
 fprintf(stderr, "pattern %d (%d-D): ",p_n,p_dim);
 for (i=0; i<p_dim; i++)
 {
  fprintf(stderr, "%4.31f ",I[p_n][i]);
  }
  fprintf(stderr,"\n");
  }
  double RungeKutta(func,i,x,h)
   double (*func) (),x,h;
   int i;
   {
    double delta1, delta2, delta3, delta4;
    delta1 = h * (*func) (i,x);
    delta2 = h * (*func) (i,x + delta1/2.0);
    delta3 = h * (*func) (i,x + delta2/2.0);
```

```
delta4 = h * (*func) (i,x + delta3);
return (x+(delta1+2.0*delta2+2.0*delta3+delta4)/6.0);
}
double f(x,sigma)
 double x, sigma;
{
 return((x > sigma) ? x : 0.0);
 }
 double fab(x,a,b)

  double x,a,b;
  {
      x=(x > b) ? b : x;
      return ((x > a) ? x-a : 0.0);
  }
  double L2_norm(vec,dim)
       double *vec;
       short dim;
      {
        int i;
        double t_sum=0.0;
        for(i=0; i<dim; i++)
        t_sum += vec[i]*vec[i];
        return(sqrt(t_sum));
       }
       double vec_diff(v1,v2,dim)
          double *v1,*v2;
          int dim;
         {
            int i;
            double d_mag=0.0;
```

```
 for(i=0;i<dim;i++)
 d_mag +=fabs(v1[i]-v2[i]);
 return(d_mag/(double)dim);
}
void main(argc,argv)
      int argc;
      char **argv;
 {
   int i,choice;
   void do_cycle(),alloc_pops(),free_pops();
   void par_modify(),show_setup(),init_weights();
   char tname[20];
   char tmp[10];
   if (argc > 1)
       strcpy(fname,*++argv);
    else
       strcpy(fname,"");
    get_inputs();
    alloc_pops();
    if(argc<3)
    {
        fprintf(stderr,"Enter output filename(no extension) -> ");
        scanf("%s",tname);
        }
        else
         strcpy(tname,*++argv);
         sprintf(out_name,"%s.dat",tname);
         sprintf(log_name,"%s.log",tname);
         fout = fopen(out_name,"w");
         flog = fopen(log_name,"w");

BU_reset();
```

```
D_reset();
printf(flog,"******> %s\n",log_name);
printf(flog,"Input file is %s.inp\n",fname);
do
 {
  fprintf(stderr, "\n*\nSelect one option:\n");
  fprintf(stderr,"1) Select input pattern\n");
  fprintf(stderr,"2) Select a series of patterns\n");
  fprintf(stderr,"3) Cycle input pattern ( fast learning)\n");
  fprintf(stderr,"4) Cycle input pattern ( slow learning)\n");
  fprintf(stderr,"5) Modify a parameter\n");
  fprintf(stderr,"6) Initialize weights\n");
  fprintf(stderr,"7) Reset Learning flag\n");
  fprintf(stderr,"8) Show setup\n");
  fprintf(stderr,"9) Reset everything\n");
  fprintf(stderr,"Other) Quit\n\n");
  fprintf(stderr,"Enter number of your choice -->");
  scanf("%d", &choice);
  switch (choice) {
   case 1 : select_cycle(1);
    break;
   case 2 : select_cycle(0);
    break;
   case 3 : l_flag=0;
    do_cycle();
    break;
   case 4 :
    l_flag=1;
    do_cycle();
    break;
   case 5 : par_modify();
```

```
break;
case 6 : init_weights();
 break;
 case 7 : new_learning();
  break;
  case 8 : show_setup();
   break;
   case 9 : init_all();
    break;
    default :
     fprintf(stderr, "Do you really want to exit (y/n)?");
     scanf("%s", tmp);
     if (tmp[0] == 'y' || tmp[0] == 'Y')
      choice = 0;
     else
      choice = -1;
     break;
     }
     } while (choice != 0);
     free_all();
     fclose(fout);
     fclose(flog);
     }
     void do_cycle()
      {
       int i,j;
       void update_F0(), update_F1(), update_F2(), print_weights();
       double update_R();
       if (l_flag)
        fprintf(flog, "\nStarting a learning cycle.\n");
        else
```

```
fprintf(flog, "\nStarting a cycle with no learning.\n");
for (j=0; j<N-M; j++)
        mis[j]=0;
 n_mis=0;
 for (i=0; i<ppc; i++)
 {
    for (j=0; j<M; j++) {
     Wp[j]=Xp[j]=Up[j]=Vp[j]=Pp[j]=Qp[j]=0.0;
     W[j]=P[j]=X[j]=Q[j]=V[j]=U[j]=0.0;
     }
     F2_on=-1;

 fprintf(stderr,"***\nNo.%d (of %d) in cycle is input pattern %d\n", i,ppc,c_p[i]);
 fprintf(flog,"***\nNo.%d (of %d)in cycle is input pattern %d\n", i,ppc,c_p[i]);
 update_F0(c_p[i]);
 update_F1(c_p[i]);
 }
 for (i=0; i<ppc; i++) {
 fprintf(stderr,"pattern %d is coded at F2 node %d\n",c_p[i], IL[c_p[i]]);
 fprintf(flog, "pattern %d is coded at F2 node %d\n",c_p[i], IL[c_p[i]]);
 sleep(2);
 }
 fprintf(stderr, "\n");
 fprintf(flog, "\n");
 if (l_flag)
  print_weights();
  }

  void update_F0(p)
   int p;
   {
```

```
int i,cntr=0;
double *old_W = old1M;
double ptmp,wtmp,vtmp;
double v_d=1.0;
do
 {
  for (i=0; i<M; i++)
   {
    old_W[i]=Wp[i];
    Wp[i]=I[p][i]+a*Up[i];
    Pp[i]=Up[i];
    }
    ptmp=L2_norm(Pp,M);
    wtmp=L2_norm(Wp,M);
    ptmp = (ptmp > 0.1) ? ptmp : 1.0;
    wtmp = (wtmp > 0.1) ? wtmp : 1.0;
    for (i=0; i<M; i++)

     {
        Qp[i]=Pp[i]/ptmp;
        Xp[i]=Wp[i]/wtmp;
        Vp[i]=f(Xp[i],theta)+b*f(Qp[i],theta);
        }
      vtmp=L2_norm(Vp,M);
      vtmp = (vtmp > 0.1) ? vtmp : 1.0;
      for (i=0; i<M; i++) {
          Up[i]=Vp[i]/vtmp;
          }
       cntr++;
       v_d = (vec_diff(old_W,Wp,M));
       } while ((v_d > ac_dif && cntr < nsteps) || cntr < 2);
```

```
  clrscr();
  fprintf(flog,"F0 activity (W') has stabilized within %5.41f after %d cycles.\n\n",
_d,cntr);
  fprintf(flog," Wp   Pp    Xp    Qp    Vp    UP\n");
  for(i=0; i<M; i++)
     fprintf(flog,"%4.31f    %4.31f    %4.31f    %4.31f       %4.31f       %4.31f\n",
Vp[i],Pp[i],Xp[i],Qp[i],Vp[i],Up[i]);
      fprintf(flog, "\n");
      }
    void update_F1(p)
      int p;
      {
       int i,cntr=0;
       int pf;
       double ptmp,wtmp,vtmp;
       double *old_W=old1M, *old_P=old2M;
       double v_d = 1.0;
       double z_d = 0.0;
       double update_R(), update_Zs();
       void update_F2(), reset_F2();
       do
        {
         pf = (cntr%10==0);
         for (i=0; i<M; i++)
          {
           old_W[i]=W[i];
           old_P[i]=P[i];
           W[i]=Qp[i]+a*U[i];
           ptmp=F2_on < 0 ? 0.0 : d*zji[F2_on][i];

           P[i]=U[i]+ptmp;
```

```
        }
ptmp=L2_norm(P,M);
wtmp=L2_norm(W,M);
ptmp = (ptmp > 0.1) ? ptmp : 1.0;
wtmp = (wtmp > 0.1) ? wtmp : 1.0;
for (i=0; i<M; i++)
 {
 Q[i]=P[i]/ptmp;
 X[i]=W[i]/wtmp;
 V[i]=f(X[i],theta)+b*f(Q[i],theta);
 }
 vtmp=L2_norm(V,M);
 vtmp = (vtmp > 0.1) ? vtmp : 1.0;
 for (i=0; i<M; i++)
 {
  U[i]=V[i]/vtmp;
  }
  v_d=vec_diff(old_W,W,M)+vec_diff(old_P,P,M);
  z_d=0.0;
  if (!rsnc)
   update_F2(cntr);
   else {
         if (cntr < 3) {
          sleep(2);
          fprintf(stderr,"%d) %d is resonating on F2 node %d\n",cntr,P,F2_on);
          fprintf(flog,"%d) %d is resonating on F2 node %d\n",cntr,P,F2_on);
          }
          IL[p]=F2_on;
          }
        if(update_R(cntr) < rho) {
        reset_F2(P,cntr);
```

```
cntr=0;
rsnc=0;
}
else {
      if (l_flag > 0 && F2_on > -1)
      z_d=update_Zs(cntr);
      }
      if (n_mis == N-M) {
      fprintf(stderr,"%d)All F2 nodes have been reset by input %d!\n", cntr,P);
      fprintf(flog,"%d) All F2 nodes have been reset by input %d!\n", cntr,P);

       break; }
       cntr++;
       } while ((v_d > ac_dif && cntr < nsteps) || cntr < 2 || z_d > z_dif);
       rsnc=0;
       sleep(2);
       fprintf(flog,"F1 activity has stabilized within %1f after %d cycles.\n",
ac_dif,cntr);
       fprintf(stderr,"F1 activity has stabilized within %1f after %d cycles.\n",
ac_dif,cntr);
       sleep(2);
       fprintf(stderr," W   P   X   Q   V   U   R\n");
       fprintf(flog," W   P   X   Q   V   U   R\n");
       for (i=0; i<M; i++) {
       fprintf(stderr,"%4.31f    %4.31f    %4.31f    %4.31f    %4.31f    %4.31f
 %4.31f\n",W[i],P[i],X[i],Q[i],V[i],U[i]);
       fprintf(flog,"%4.31f    %4.31f    %4.31f    %4.31f    %4.31f    %4.31f
 %4.31f\n",W[i],P[i],X[i],Q[i],V[i],U[i]);
       }
       fprintf(stderr,"*** F1 diff = %1f\n\n",v_d);
       fprintf(flog,"*** F1 diff = %1f\n\n",v_d);
```

```
}
void update_F2(cntr)
 int cntr;
{
 int i,j;
 double ymax;
 int pf = (cntr%10==0);
 ymax=0.0;
 for (j=0; j<N-M; j++) {
  Y[j]=0.0;
  if (mis[j] > -1)
   for (i=0; i<M; i++)
    Y[j] += zij[i][j]*P[i];
    ymax = (ymax < Y[j] ? Y[j] : ymax);
     }
     if (ymax == 0.0) {
      fprintf(stderr, "%d) All F2 nodes are inactive! No category chosen.\n", cntr);
      fprintf(flog,"\n%d) All F2 nodes are inactive! No category chosen.\n",  cntr);
      F2_on = -1;
      return;
      }

for (i=0; i<N-M && Y[i]<ymax; i++);
F2_on = i;
rsnc = 1;
}
 double update_R(cntr)
  int cntr;
 {
  int i;
  double ptmp=L2_norm(P,M), qtmp=L2_norm(Qp,M),utmp=L2_norm(U,M);
```

```
double L2_R;
int pf = (cntr%10==0 ? 1 : 0);
if (qtmp < 0.1 || ptmp < 0.1 || utmp < 0.1) {
 fprintf(stderr,"%d     Warning:     (Qp)     =     %5.41f          (p)     =     %5.41f
U)=%5.41f\n",cntr,qtmp,ptmp,utmp);
  fprintf(flog,"%d     Warning:     (Qp)     =     %5.41f     (p)     =     %5.41f
u)=%5.41f\n",cntr,qtmp,ptmp,utmp);
  }
   for(i=0; i<M; i++)
    {
      R[i]=(Qp[i]+c*P[i])/(c*ptmp+qtmp);
      }
      L2_R=L2_norm(R,M);
      if(L2_R < rho) {
      sleep(1);
       fprintf(stderr,"\n%d) L2_norm(R)=%4.31f\n",cntr,L2_R);
       fprintf(flog,"\n%d) L2_norm(R)=%4.31f\n",cntr,L2_R);
       }
       else
        if(cntr < 10) {
         sleep(1);
         fprintf(stderr, "\n%d) L2_norm(R)=%4.31f\n",cntr,L2_R);
         fprintf(flog,"\n%d) L2_norm(R)=%4.31f\n",cntr,L2_R);
         }
         return(L2_R);
         }
        void reset_F2(p,cntr)
           int p,cntr;
           {
            int j;
```

```
fprintf(stderr,"%d) input %d caused a reset of node %d\n",cntr,p,F2_on);
        fprintf(flog,"%d) input %d caused a reset of node %d\n",cntr,p,F2_on);
        Y[F2_on]=0.0;
        mis[F2_on]=-1;
        n_mis++;
        rsnc = 0;
        F2_on=-1;
        IL[p]=-1;
         for(j=0; j<M; j++)
         W[j]=P[j]=X[j]=Q[j]=V[j]=U[j]=0.0;
        }
        void resonate(p,i)
         int p,i;
         {
          fprintf(stderr,"*** Pattern %d resonates with node %d\n",p,i);
          fprintf(flog,"*** Pattern %d resonates with node %d\n",p,i);
          }
          void par_modify()
          {
           int which;
           fprintf(flog,"Just called par_modify: ");
           do {
               fprintf(stderr,"1) vigilance (rho)\n");
               fprintf(stderr,"2) threshold (theta)\n");
               fprintf(stderr,"3) RK4 step size (h)\n");
               fprintf(stderr,"4) Number of integration steps(nsteps)\n");
               fprintf(stderr,"5) stability criterion(ac_dif)\n");
               fprintf(stderr,"6) LTM stability criterion(z_dif)\n");
               fprintf(stderr,"Which parameter ?");
               scanf("%d", &which);
               switch (which) {
```

```
            case 1 : fprintf(stderr,"Present value for rho is %6.31f\nEnter new value --
,rho);
                     fprintf(flog,"changed rho from %5.41f",rho);
                     scanf("%1f",&rho);
                     fprintf(flog," to %5.41f\n",rho);
                     break;
            case 2 : fprintf(stderr,"Present value for theta is %6.31f\nEnter new value
->",theta);
                     fprintf(flog,"changed theta from %5.41f",theta);
                     scanf("%1f",&theta);
                     fprintf(flog," to %5.41f\n",theta);
                     break;
            case 3 : fprintf(stderr,"Present value for h is %6.31f\nEnter new value --
>",h);
                     fprintf(flog,"changed h from %5.41f",h);
                     scanf("%1f",&h);
                     fprintf(flog,"to %5.41f\n",h);
                     break;
            case 4 : fprintf(stderr,"Present value for nsteps is %d\nEnter new value --
 >", nsteps);
                     fprintf(flog,"changed nsteps from %d",nsteps);
                     scanf("%d",&nsteps);
                     fprintf(flog,"to %d\n",nsteps);
                     break;
            case 5 : fprintf(stderr,"Present value for ac_dif is %6.31f\nEnter new
  value -->",ac_dif);
                     fprintf(flog,"changed ac_dif from %5.41f",ac_dif);
                     scanf("%1f",&ac_dif);
                     fprintf(flog,"to %5.41f\n",ac_dif);
                     break;
```

```
        case 6 : fprintf(stderr,"Present value for z_dif is %6.31f\nEnter new value
>",z_dif);
                 fprintf(flog,"changed z_dif from %5.41f",z_dif);
                 scanf("%1f",&z_dif);
                 fprintf(flog,"to %5.41f\n",z_dif);
                 break;
        default : which=0;
                  break;
        }
        fprintf(flog," to change case %d\n",which);
        } while (which != 0);
       }
       void show_setup()
       {
        int i;
        fprintf(stderr,"\nINPUT PATTERNS:\n");
        for(i=0; i<p_num; i++)
         p_display(i);
        if(ppc > 0) {
         fprintf(stderr,"\nSELECTED PATTERNS:\n");
         fprintf(flog,"\nSELECTED PATTERNS:\n");
         for(i=0; i<ppc; i++) {
         fprintf(stderr,"No.%d in cycle is input pattern %d, coded at F2 node
%d\n",i,c_p[i],IL[c_p[i]]);
          fprintf(flog,"No.%d in cycle is input pattern %d coded at F2 node
%d\n",i,c_p[i],IL[c_p[i]]);
         }
         }
         fprintf(stderr,"\nPARAMETERS:\n");
         fprintf(stderr,"rho=%5.41f      theta=%5.41f      h=%4.31f\nnsteps=%d
ac_dif=%6.51f z_dif=%5.41f\n",rho,theta,h,nsteps,ac_dif,z_dif);
```

```
fprintf(flog,"rho=%5.41f        theta=%5.41f        h-%4.31f\nnsteps=%d
_dif=%6.51f z_dif=%5.41f\n",rho,theta,h,nsteps,ac_dif,z_dif);
}
void init_weights()
{
 int i,j,choice=1;
 void BU_set1(), TD_set1(), show_weights(), print_weights(), zero_all();
 fprintf(stderr,"Initializing weights.\n");
 do {
  fprintf(stderr,"1) Initialize BU weights\n");
  fprintf(stderr,"2) Initialize TD weights\n");
  fprintf(stderr,"3) Initialize all weights\n");
  fprintf(stderr,"4) Manually set one BU weight\n");
  fprintf(stderr,"5) Manually set one TD weight\n");
  fprintf(stderr,"6) Show all weights\n");
  fprintf(stderr,"7) Print weights\n");
  fprintf(stderr,"8) Zero all weights\n");
  fprintf(stderr,"Other) Return to main menu\n\n");
  fprintf(stderr,"--- Select a number --->");
  scanf("%d",&choice);
  switch (choice) {
   case 1 : BU_reset();
    break;
   case 2 : TD_reset();
    break;
   case 3 : BU_reset();
    TD_reset();
    break;
   case 4 : BU_set1();
    break;
   case 5 : TD_set1();
```

```
break;
case 6 : show_weights();
 break;
case 7 : print_weights();
 break;
case 8 : zero_all();
 break;
default : choice = 0;
}
} while (choice != 0);
}
void BU_reset()
{
 int i,j;
 fprintf(flog,"Just reset all TD weights.\n");
 for(i=0; i<M; i++)
  for(j=0; j<N-M; j++)
  zij[i][j]=1.0/((1.0-d)*sqrt((double)M));
  }
  void BU_set1()
  {
   int i,j;
   fprintf(stderr,"Enter i ---->");
   scanf("%d",&i);
   if (i < 0 || i >= M)
   {
    fprintf(stderr,"Invalid weight index %d\n",i);
    return;
    }
    fprintf(flog,"Resetting zij[%d] to : ",i);
    for(j=0; j < N-M; j++)
```

```
{
 fprintf(stderr,"Enter zij[%d][%d] (%4.31f)-->",i,j,zij[i][j]);
 scanf("%1f",&zij[i][j]);
 fprintf(flog,"%4.31f",zij[i][j]);
 }
 fprintf(flog,"\n");
}
void TD_reset()
{
 int i,j;
 fprintf(flog,"Just reset all TD weights.\n");
 for(j=0; j<N-M; j++)
  for(i=0; i<M; i++)
  zji[j][i]=0.0;
  }
  void TD_set1()
  {
   int i,j;
   fprintf(stderr,"Enter j -->");
   scanf("%d",&j);
   if(j < 0 || j >= N-M)
    {
     fprintf(stderr,"Invalid weight index %d\n",j);
     return;
     }
     fprintf(flog,"Setting zji[%d] to: ",j);
     for (i=0; i<M; i++)
      {
          fprintf(stderr,"Enter zji[%d][%d] (%4.31f)-->",j,i,zji[j][i]);
          scanf("%1f",&zji[j][i]);
          fprintf(flog,"%4.31f",zji[j][i]);
```

```
    }
    fprintf(flog,"\n");
    }
void show_weights()
{
 int i,j;
 fprintf(stderr,"\nBU WEIGHTS:\n");
 fprintf(flog,"\nBU WEIGHTS:\n");
 for(i=0; i<M; i++)
 {
     for(j=0; j<N-M; j++)
     {
      fprintf(stderr,"%6.41f",zij[i][j]);
      fprintf(flog,"6.41f",zij[i][j]);
      }
      fprintf(stderr,"\n");
      fprintf(flog,"\n");
      }
      fprintf(stderr,"\nTD WEIGHTS:\n");
      fprintf(flog,"\nTD WEIGHTS:\n");
      for(j=0; j<N-M; j++)
      {
       for(i=0; i<M; i++)
       {
        fprintf(stderr,"%6.41f",zji[j][i]);
        fprintf(flog,"6.41f",zji[j][i]);
        }
        fprintf(stderr,"\n");
        fprintf(flog,"\n");
        }
        }
```

```
void zero_all()
{
 int i,j;
 for(i=0; i<M; i++)
 for(j=0; j<N-M; j++)
 zij[i][j]=zji[j][i]=0.0;
 fprintf(stderr,"Warning: ART2 will not work with all weights at

zero.\n");

 fprintf(flog,"Reset all weights to zero!\n");
 }
 double zt1,zt2;
 double dzij(i,z)
 int i;
 double z;
 {
  return(zt1*(zt2-z));
  }
  double dzji(i,z)
 int i;
 double z;
 {
  return(zt1*(zt2-z));
  }
  double update_Zs(cntr)
   int cntr;
   {
    int i;
    int pf = (cntr%10==0);
    double *ozij=old1M, *ozji=old2M;
    double BUdiff=0.0;
    double TDdiff;
```

```
for(i=0; i<M; i++)
{
 ozij[i]=zij[i][F2_on];
 ozji[i]=zji[F2_on][i];
 zt1=d*(1.0-d);
 zt2=U[i]/(1.0-d);
 zt1=d;
 zt2=P[i];
 zij[i][F2_on]=RungeKutta(dzij,i,zij[i][F2_on],h);
 zji[F2_on][i]=RungeKutta(dzji,i,zji[F2_on][i],h);
  BUdiff += fabs(ozij[i]-zij[i][F2_on]);
  }
  TDdiff=vec_diff(zji[F2_on],ozji,M);
  if(pf
  ) {
   fprintf(stderr,"%d)  (BU,TD)  vector  LTM  changes  are
(%5.41f,%5.41f)\n",cntr,BUdiff,TDdiff);
   fprintf(flog,"%d)  (BU,TD)  vector  LTM  changes  are
(%5.41f,%5.41f)\n",cntr,BUdiff,TDdiff);
   }
   return(BUdiff+TDdiff);
  }
  void print_weights()
  {
   int i,j;
   fprintf(fout,"\t\t Bottom-up weights:\n\n");
   for (i=0; i<M; i++) {
    for (j=0; j<N-M; j++)
     fprintf(fout,"%5.41f",zij[i][j]);
     fprintf(fout,"\n");
     }
```

```
fprintf(fout,"\n\t\t Top-down weights:\n\n");
for(j=0; j<N-M; j++){
 for(i=0; i<M; i++)
  fprintf(fout,"%5.41f",zji[j][i]);
 fprintf(fout,"\n");
 }
 fprintf(fout,"\n");
 }
 void alloc_pops() {
  int i;
  X=(double *)calloc(M,sizeof(double));
  Xp=(double *)calloc(M,sizeof(double));
  W=(double *)calloc(M,sizeof(double));
  Wp=(double *)calloc(M,sizeof(double));
  U=(double *)calloc(M,sizeof(double));
  Up=(double *)calloc(M,sizeof(double));
  P=(double *)calloc(M,sizeof(double));
  Pp=(double *)calloc(M,sizeof(double));
  V=(double *)calloc(M,sizeof(double));
  Vp=(double *)calloc(M,sizeof(double));
  Q=(double *)calloc(M,sizeof(double));
  Qp=(double *)calloc(M,sizeof(double));
  R=(double *)calloc(M,sizeof(double));
  Y=(double *)calloc(N-M,sizeof(double));
  mis=(double *)calloc(N-M,sizeof(double));
  zij=(double **) calloc(M,sizeof(double *));
  for(i=0; i<M; i++)
   zij[i]=(double *) calloc(N-M,sizeof(double));
  zji=(double **) calloc(N-M,sizeof(double *));
  for(i=0; i<N-M; i++)
   zji[i]=(double *) calloc(M,sizeof(double));
```

```
old1M=(double *) calloc(M,sizeof(double));
old2M=(double *) calloc(M,sizeof(double));
}
void free_pops() {
 int i;
 free (X);
 free (Xp);
 free (W);
 free (Wp);
 free (U);
 free (Up);
 free (P);
 free (Pp);
 free (V);
 free (Vp);
 free (Q);
 free (Qp);
 free (R);
 free (Y);
 free (mis);
 for (i=0; i<M; i++)
  free (zij[i]);
 free (zij);
 for(i=0; i<N-M; i++)
  free (zji[i]);
 free (zji);
 free (old1M);
 free (old2M);
} // End of ART-2
```

```
MPLEMENTATION OF FUZZY ART*/
nclude<stdio.h>
nclude<math.h>
nclude<conio.h>
lefine N 5
lefine PATT 10
oat rho,norm1,alpha=0.001,norm3,eta;
ouble norm2,minimum[2*N],norm4;
nt p,j,cluster[PATT],noc=1,i,index[PATT],windex,flag;
loat a[PATT][2*N],net[10];
loat b[2*N],w[10][2*N],lnorm[2*N],t[2*N],s[2*N];
char fname[50];
void initwts();
void writewts();
void normalize();
void computation();
void winner();
void vigilance();
void update();
void swap(int *x,int *y);
float min(float *u,float *v);
float max(float *u,float *v);
FILE *fp,*fp1;
void main(int argc, char *argv[])
{
clrscr();
printf("Enter the vigilance value");
scanf("%f",&rho);
initwts();
fp=fopen(argv[1],"r");
if(!fp)
```

```
intf("\n Error!! Could not open the file");
xit(1);

or(p=0;p<PATT;p++)

for(j=0;j<N;j++)
{
fscanf(fp,"%f",&a[p][j]);
b[j]=a[p][j];
}
normalize();
computation();
winner();
vigilance();
}
fclose(fp);
printf("\nVigilance threshold:%f",rho);
printf("\nTraining pattern\t Cluster");
for(j=0;j<PATT;j++)
{
printf("\n%d\t\t\t %d",j+1,cluster[j]+1);
}
}

void initwts()
{
for(j=0;j<2*N;j++)
{
w[noc-1][j]=1.0;
```

```
oid normalize()
{
norm1=0.0;
for(j=0;j<N;j++)
{
norm1+=b[j];
}
for(j=0;j<N;j++)
{
lnorm[j]=b[j]/=norm1;
b[j+5]=1.0-b[j];
}
}
void computation()
{
for(i=0;i<noc;i++)
{
norm2=0.0;
for(j=0;j<2*N;j++)
{
if(j<5)
t[j]=min(&lnorm[j],&w[i][j]);
else
t[j]=1-max(&lnorm[j],&w[i][j]);
norm2+=t[j];
minimum[j]=min(&b[j],&w[i][j]);
}
norm3=0.0;
for(j=0;j<2*N;j++)
```

```
orm3+=minimum[j];

et[i]=norm3/(alpha+norm2);
ndex[i]=i;


loat min(float *u,float *v)
{
f(*u<=*v)
return(*u);
else
return(*v);
}
float max(float *u,float *v)
{
if(*u>=*v)
return(*u);
else
return(*v);
}
void winner()
{
for(i=0;i<noc;i++)
{
for(j=0;j<noc-i;j++)
{
if(net[index[j]]<net[index[j+1]])
swap(&index[j],&index[j+1]);
}}
}
```

```
id swap(int *x,int *y)

t temp=*x;
x=*y;
y=temp;

oid vigilance()
{
for(i=0;i<noc;i++)
 {
 windex=index[i];
 flag=0;
 norm4=0.0;
 for(j=0;j<2*N;j++)
  {
  s[j]=min(&b[j],&w[windex][j]);
  norm4+=s[j];
  }
  if((norm4/N)>=rho)
   {
   flag=1;
   cluster[p]=windex;
   eta=1.0;
   update();
   writewts();
   break;
    }
    }
    if(!flag)
     {
     noc++;
```

```
initwts();
windex=noc-1;
cluster[p]=noc-1;
eta=1.0;
update();
writewts();
}}
void update()
{
for(j=0;j<2*N;j++)
w[windex][j]=eta*(min(&b[j],&w[windex][j]))+(1-eta)*w[windex][j];
}
void writewts()
{
fp1=fopen("weight.dat","w");
if(!fp1)
{
printf("Error, Could not open the file");
exit(1);
}
for(i=0;i<noc;i++)
{
for(j=0;j<2*N;j++)
fprintf(fp1,"%.2f",w[i][j]);
}
fclose(fp1);  }  /* End of Fuzzy ART*/
```

REFERENCES

Abhijit .S. Pandya, Robert .B. Macy, "*Pattern Recognition with Neural Networks in C++*" ,CRC Press in cooperation with IEEE press, N.W. Boca Raton, Florida,1995.

T.Frank, K.F. and T.Kuhlen, " *Comparative Analysis of Fuzzy ART and ART-2A Network Clustering Performance* ", IEEE Transactions on Neural Networks,Vol.9,No:3,pp.544-559,May 1998.

Gail .A. Carpenter, Stephen Grossberg, Natalya Markuzon, John .H. Reynolds, "*Fuzzy ARTMAP: A Neural Network Architect for International Supervised Learning of Analog Multidimenional Maps*",IEEE Transaction on Neural Networks, Vol.3, No:5, pp.698-713, September 1992.

Gail .A. Carpenter, Stephen Grossberg, "*ART 2 : Self-organization of stable category recognition codes for analog input patterns*", Applied optics, Vol.26,No:23,December 1987.

Gail .A. Carpenter, Stephen Grossberg, "*The ART of Adaptive Patterns Recognition" by a Self-Organizing Neural Networks*", IEEE Computer, pp.77-87,March 1998.

Jacek .M. Zurada, "*Introduction to Artificial Neural Systems*", Jaico Publishing House, 1997.

James A. Freeman, David M. Skapura, "Neural Networks- Algorithms, Applications, and Programming Techniques", Addison Wesley Publishing Company 1999.

Dr. Vallura Rao and Hayagriva Rao, "C++ Neural Networks and Fuzzy Logic", BPB Publications ,New Delhi.