PARALLEL IMPLEMENTATION OF
FUZZY ARTMAP
ON A HYPERCUBE (iPSC)

A Thesis Presented to
The Faculty of the College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
Anil Malkani
August, 1992
This thesis has been approved for the
Department of Electrical and Computer Engineering
and the College of
Engineering and Technology

Assistant Professor of Electrical
and Computer Engineering

Dean of the College of
Engineering and Technology
ACKNOWLEDGEMENTS

I wish to thank Dr. Constantinos A. Vassiliadis for his support, concern, understanding and advice. I would also like to thank all professors in my graduate studies who taught me the true meaning of "in depth knowledge applied to real world".

I also wish to thank the committee members namely Dr. L. C. Rabelo, Dr. Jeffery Dill, and Dr. Mehmet Celenk for their support and guidance. I am indebted to Dr. Md. Meybodi for his help and invaluable guidance.

Finally, I like to express my appreciation to Dr. S. Keyvan (University of Missouri Rolla), K. Hooks, and G. Bakshi for providing exemplars for different applications used to prove the capabilities of Parallel Fuzzy ARTMAP.
# TABLE OF CONTENTS

Table of contents ......................................................... i
List of Figures ............................................................. iii
List of Tables .............................................................. v

Chapter 1 Introduction ..................................................... 1
Chapter 1 Literature Review ................................................. 4

I Parallel Computing on a Hypercube ......................... 4
II Hypercube Architecture ............................................. 4
III Topological Properties of Hypercube ....................... 8
IV iPSC (Intel's Personnel Supercomputers) ................. 9
V Parallel Machines for Neural Networks ..................... 10

Chapter 3 Fuzzy Adaptive Resonance Theory (Fuzzy ART) .... 12

I Characteristics of Fuzzy networks ......................... 12
II Preprocessing of Input to Fuzzy Networks .......... 15
   A. Fast Learn Slow Recode ............................ 15
   B. Normalization .................................. 17
III Fuzzy ART Paradigms ........................................ 21
IV Analogy between Fuzzy ART and ART-1 ............. 28

Chapter 4 Fuzzy ARTMAP - A supervised ART Paradigm ...... 31

I Fuzzy ARTMAP Paradigm .................................. 34
II Simplification in Mapping ................................. 41
Chapter 5  Modified Fuzzy ARTMAP (M-FAM)  

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>M-FAM Neuro Architecture</td>
<td>43</td>
</tr>
<tr>
<td>II</td>
<td>M-FAM Paradigm</td>
<td>47</td>
</tr>
</tbody>
</table>

Chapter 6  Parallel Fuzzy ARTMAP  

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>P-FAM Architecture Distribution on N-CUBE</td>
<td>60</td>
</tr>
<tr>
<td>II</td>
<td>Neuro Dynamics of P-FAM</td>
<td>66</td>
</tr>
<tr>
<td>III</td>
<td>Communication in PFAM</td>
<td>70</td>
</tr>
<tr>
<td>IV</td>
<td>Algorithmic Flow of P-FAM</td>
<td>83</td>
</tr>
</tbody>
</table>

Chapter 7  Application using P-FAM  

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>P-FAM in trend Detection in quality Control</td>
<td>104</td>
</tr>
<tr>
<td>II</td>
<td>P-FAM in Manufacturing</td>
<td>118</td>
</tr>
<tr>
<td>III</td>
<td>P-FAM in Signal Processing</td>
<td>123</td>
</tr>
</tbody>
</table>

Chapter 8  Summary and Recommendations  

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>Conclusions</td>
<td>125</td>
</tr>
<tr>
<td>II</td>
<td>References</td>
<td>128</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 3.1  Relationships Between Complement Coding and Fuzzy Subset Theory .......................... 19
Figure 3.2  Fuzzy ART Architecture .......................................................... 22
Figure 4.1  Fuzzy ARTMAP Architecture ..................................................... 32
Figure 5.1  Modified Fuzzy ARTMAP Architecture ................................. 44
Figure 5.2  Map Field Network ................................................................. 52
Figure 6.1  4-dimensional Hypercube ......................................................... 58
Figure 6.2  Modified Fuzzy ARTMAP Architecture to be Implemented in Parallel ........................................ 61
Figure 6.3  Neurons Distributions on a Node in a Hypercube ....................... 64
Figure 6.4  Global Broadcast - Embedded Tree ........................................... 73
Figure 6.5  Tree Obtained for Root = 7, Exclusive or by 7 each Node of a Fan-Out Tree with Root = 0 ................. 74
Figure 6.6  Tree Obtained for Root = 7, Using Theorem 1 .......................... 75
Figure 6.7  Global Broadcast, with Node 0 of the 3-cube as the Root of the Tree (Network Diagram) .................... 77
Figure 6.8  Calculations and Communication in Nodes of Hypercube for Global Maximum .............................. 79
Figure 6.9  Communication Between Nodes of 3-cube during Global Maximum Calculations ................................ 82
Figure 6.10 Outline of P-FAM Algorithm .................................................... 84
Figure 6.11 Host Flow-chart During Training in P-FAM (Cont.) .................... 85
<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.12</td>
<td>Host Flow-chart During Training in P-FAM (Cont.)</td>
<td>86</td>
</tr>
<tr>
<td>6.13</td>
<td>Hypercube Flow-chart During Training in P-FAM (Cont.)</td>
<td>87</td>
</tr>
<tr>
<td>6.14</td>
<td>Hypercube Flow-chart During Training in P-FAM (Cont.)</td>
<td>88</td>
</tr>
<tr>
<td>6.15</td>
<td>Calculating Maximum Net in each Processor of Hypercube in P-FAM</td>
<td>90</td>
</tr>
<tr>
<td>6.16</td>
<td>Initiating Match Tracking or Learning in each Processor of Hypercube for P-FAM</td>
<td>91</td>
</tr>
<tr>
<td>6.17</td>
<td>Host Flow-chart During Retraining in P-FAM (Cont.)</td>
<td>92</td>
</tr>
<tr>
<td>6.18</td>
<td>Host Flow-chart During Retraining in P-FAM (Cont.)</td>
<td>93</td>
</tr>
<tr>
<td>6.19</td>
<td>Host Flow-chart During Retraining in P-FAM (Cont.)</td>
<td>94</td>
</tr>
<tr>
<td>6.20</td>
<td>Hypercube Flow-chart During Retraining in P-FAM</td>
<td>95</td>
</tr>
<tr>
<td>6.21</td>
<td>Host Flow-chart During Testing in P-FAM (Cont.)</td>
<td>96</td>
</tr>
<tr>
<td>6.22</td>
<td>Host Flow-chart During Testing in P-FAM (Cont.)</td>
<td>97</td>
</tr>
<tr>
<td>6.23</td>
<td>Hypercube Flow-chart During Testing in P-FAM (Cont.)</td>
<td>99</td>
</tr>
<tr>
<td>6.24</td>
<td>Hypercube Flow-chart During Testing in P-FAM (Cont.)</td>
<td>100</td>
</tr>
<tr>
<td>6.25</td>
<td>Hypercube Flow-chart During Testing in P-FAM (Cont.)</td>
<td>101</td>
</tr>
<tr>
<td>7.1</td>
<td>Exemplars for Downward Trends</td>
<td>110</td>
</tr>
<tr>
<td>7.2</td>
<td>Exemplars for Steady Trends</td>
<td>111</td>
</tr>
<tr>
<td>7.3</td>
<td>Exemplars for Upward Trends</td>
<td>112</td>
</tr>
<tr>
<td>7.4</td>
<td>L. T. M. Traces for Downward trends</td>
<td>113</td>
</tr>
<tr>
<td>7.5</td>
<td>L. T. M. Traces for steady trends</td>
<td>114</td>
</tr>
<tr>
<td>7.6</td>
<td>L. T. M. Traces for Upward trends</td>
<td>115</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 3.1</td>
<td>Comparison of Logical and Fuzzy AND operators</td>
<td>30</td>
</tr>
<tr>
<td>Table 6.1</td>
<td>The Two Processors in Different Plane’s but on the Same Vertex</td>
<td>81</td>
</tr>
<tr>
<td>Table 7.1</td>
<td>Training quality Control Data Using P-FAM (Cont.)</td>
<td>107</td>
</tr>
<tr>
<td>Table 7.2</td>
<td>Training quality Control Data Using P-FAM (Cont.)</td>
<td>108</td>
</tr>
<tr>
<td>Table 7.3</td>
<td>Testing quality Control Data Using P-FAM (Cont.)</td>
<td>109</td>
</tr>
<tr>
<td>Table 7.4</td>
<td>Message Traffic Summary for Training Samples for Quality Control exemplars</td>
<td>116</td>
</tr>
<tr>
<td>Table 7.5</td>
<td>Message Traffic Summary for Testing Samples for Quality Control exemplars</td>
<td>117</td>
</tr>
<tr>
<td>Table 7.6</td>
<td>Routing for Each Product Type</td>
<td>119</td>
</tr>
<tr>
<td>Table 7.7</td>
<td>Training Samples and Performance Parameters</td>
<td>121</td>
</tr>
<tr>
<td>Table 7.8</td>
<td>Results Obtained Using Parallel Fuzzy ARTMAP</td>
<td>122</td>
</tr>
<tr>
<td>Table 7.9</td>
<td>Training Nuclear Reactor Pump Signals Using P-FAM</td>
<td>124</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

Since the beginning of the computer age, scientists have dreamed of building a machine that mimics the human brain. However, limitations in both the theory and available hardware have had less ambitious results. The sequential machines have been extended with the introduction of pipelining, array processing and multiprocessing. The computations done on these extended concurrent architectures are less costly to perform than similar computations on sequential machines.

Scientists since long believe that something fundamental is missing in the understanding of the operations involved in a process within a biological system. The belief is substantiated by many examples where biological systems are capable of finding real time solutions to problems where even the largest supercomputer fails. These processes in the biological elements (such as in the brain) are one million times slower than the ones using semiconductor devices.

The recent surges of interest in connectionist models arose through the availability of high speed parallel supercomputers and the advent of new learning procedures. The name "connectionist model" is used to indicate the family of all artificial neural networks. These networks use distributed models for knowledge representation in forms of simple and neural-like computation elements. The capability of the networks to process information resides in the connections between these elements. In this sense the
connectionist models are inspired from the functioning of the human brain, where the information is distributed, and processed according to the activities of the neurons and their interconnections.

Artificial Neural Networks (ANN) use approaches radically different from today's conventional electronic computers. ANN offer the possibility of self organizing, fault-tolerance, real time vision capability, pattern matching and optimization. ANN not only improve performance with experience but also require little or no software engineering. However, many uncertainties make it difficult to predict when these technologies will mature to produce enough self explanatory reasoning and become commercial products.

It is intended to design and implement a parallel version of a supervised learning paradigm based on adaptive resonance theory, namely Fuzzy ARTMAP[2]. The algorithm developed is to be implemented on a hypercube architecture. The main emphasis is to identify the area of parallelism and use it to effectively distribute the paradigm on to a given set of processors. This will reduce the training time involved in learning a given example as the computational intensive task is well distributed among the processors in a hypercube. The implementation of the algorithm is to performed a (iPSC) hypercube simulator therefore, a timing analysis of the performance of the algorithm would to of no value. However the mapping of the paradigm on the hypercube where the computations are distributed among many processors strongly suggests the reduction of training time.

Fuzzy ARTMAP is a neural network paradigm build using Adaptive Resonance Theory-1 (ART-1)[3] and Fuzzy Adaptive Resonance Theory (Fuzzy ART)[4] as primary blocks. Fuzzy ARTMAP besides being a supervised ART-based network, encompasses
both neural and fuzzy logic. Incorporation of fuzzy logic into the ART paradigm has created a very powerful supervised learning paradigm (Fuzzy ARTMAP) preserving all the properties of ART. These properties such as the stability plasticity dilemma, the full memory capacity utilization, the adaptability to real time responses, the non-stationary worlds, the fast and slow learning, incremental learning, etc., make the paradigm closer to biological systems.

The above mentioned strengths of Fuzzy ARTMAP neural network paradigm make it a perfect candidate for parallel implementation. To realize the complexity involved in developing and parallelizing Fuzzy ARTMAP, the paradigm was implemented first on a Sparc workstation as a sequential one. Realizing its success[5-8] the second step was to design a supervised neural network paradigm by incorporating a simplification of match tracking or min-max rule concepts on unsupervised Fuzzy ART paradigms (M-FAM - modified Fuzzy ARTMAP). The third stage involved developing a parallel algorithm for M-FAM paradigm, i.e., P-FAM (parallel Fuzzy ARTMAP). As a fourth step, the paradigm is parallelized and mapped to fit a hypercube architecture of any dimension. The hypercube mapped architecture is implemented on a Intel Personal SuperComputer (iPSC) simulator.

The simulator build on the iPSC architecture is tested for numerous applications in areas of quality control, nuclear engineering, manufacturing systems, character recognition and protein analysis.
CHAPTER 2

LITERATURE REVIEW

I. PARALLEL COMPUTING ON A HYPERCUBE

Parallel processing provides a natural frame work for designing computing machines with ever-increasing computing power. Parallel computers have revolutionized the basic approach to scientific and engineering computing and simulation. There are various computer architectural classification schemes[10]. In general digital computers can be classified into four categories based on multiplicity of instruction and data streams as defined by Flynn’s classification.

1) Single instruction stream-single data stream (SISD), e.g., IBM 701, Cray 1.
3) Multiple instruction stream-single data stream (MISD).
4) Multiple instruction stream-multiple data stream (MIMD), e.g., iPSC, iPSC/2, IBM 3081/3084.

II. HYPERCUBE ARCHITECTURE

The hypercube is one of the most versatile and efficient network yet discovered for parallel processing. Hypercube or Cube was first proposed in 1977, and with the development of Cosmic cube in 1985 at Caltech, heavy interest in these architectures was revived.

A. Cube Topology

The number of connections for each processor grows logarithmically with the
size of the network. This problem is accomplished by using several bounded-degree derivatives of the hypercube. The derivative networks form an interesting and powerful class of networks commonly called hypercubic networks. Some of the popular derivatives are the butterfly, shuffle exchange graphs, de Bruijn graph, Beneš network and cube connected cycles (CCC). A r-dimensional hypercube has \( N = 2^r \) nodes and \( r2^{r-1} \) edges. Each node corresponds to an \( r \)-bit binary string, and two nodes are linked with an edge if and only if their arbitrary string differ in precisely one bit [12].

The name 'Cube' indicates that the topology can be thought of as a multidimensional cube, with a processor node at each vertex. In a cube of dimension \( d \), there are \( n = 2^d \) processor nodes, each connected to \( d \) neighboring nodes. One of the advantages of this configuration is that, the number of links per processor is equal to the dimension of the cube. This makes it feasible for very large number of nodes to be interconnected. For example, each processor in a 4-dimensional cube is connected to only four of its neighbors for a total of 16 processors. A processor has physical connections with its neighbors only. A message sent by a node to its neighbors, the time \( \tau \) may be short, but if the message is sent to more remote nodes, it could take a maximum of \( \tau \ast d \), in a cube of dimension \( d \). This is a clear indication that an algorithm cannot be fine grained since the resultant communication time may be very high.

In a cube topology the processor nodes have access only to local memory unlike share memory configuration of processing elements. The system has distributed memory and control (e.g., in iPSC hypercube configuration the communication processors are responsible for message passing.). Cubes are thus classified as distributed memory, message passing parallel processing systems. Lots of applications in artificial intelligence are suitable for such a topology as the program can be divided into number of independent programs with minimum interprocess communication. This has been one of the main reasons for choosing a hypercube architecture for implementing neural networks. Here the processors operate independently on a subsection of a larger problem according to instructions and data resident in the local memories. All the processors operate independently and simultaneously rather than lock-step or shared memory synchronization.
Cubes are an example of concurrency based on asynchronous operation of the processor nodes. Hypercube topology are an example of Multiple Instruction Multiple Data (MIMD) system as nodes operate with multiple instruction streams. Also 'p' different instruction streams are active at the same time on 'p' different data sets. Here it is a MIMD system with message passing used for communication between the nodes.

A task defined as a sequential program to send or receive messages, is executed by message passing as basic unit of computation. A processor node may have more than one task, hence nodes have a multi programmed operating system, the kernel. Kernel operates in parallel. The number of tasks in a node is limited by the amount of local memory available. With the increase in cube dimensions the communication capability increases more than the factor by which the computational capability increases. To take advantage of communication locality, communication should take place only between neighboring processors. Interprocess communication is performed through channels. These requests are system calls. Send and receive calls just enable message transmission, if not satisfied the request for it will be pending until activated. During the waiting time, the processor may continue executing programs.

B. Message Communication

Message Communication is important for simplicity, correctness, reliability and speed up of concurrent algorithms. There are two kinds of protocol style used in message communication:

1) Synchronized message-passing: A sender is blocked until a corresponding receive request has been issued, and a task initiating a receive request is waiting until a message has been received. This synchronizes the sender and receiver, therefore it makes indefinite blocking.

2) Asynchronous message-passing: The sender proceeds to compute while the message it has sent is waiting to be received. In the same way, the receiver proceeds with computations (if they are independent of the message) until the message arrives and even after the message arrives. At a pause of computation, the buffers are polled and the message is transferred. This minimizes waiting and facilitates easily understood algorithms, eliminating indefinite blocking.
For high efficiency and utilization, a cube should balance the load between the nodes with each node solving a part of the job. The manner of load balancing and message locality, influence the computer results but not the distribution of tasks.

A hypercube topology has a host or manager processor connected to all the nodes, or a subset of nodes by a global bus. The global bus is used for communication between the host to node and node to host only and never for node to node. The host serves as a link to the outside world. It is responsible for initiating communication and collecting results upon completion.

Cube can be used as subcubes. Where the subcubes are optimum number of processors needed for the problem to be efficient. Subcubes dimension vary as per problem definition. The variable granularity of the allocation ensures that program works near optimal points.

C. Advantages of cube configuration

The key to parallel processing in a cube is to have a balanced load among processor nodes and reduce the amount of interprocess communication. The cube topology has the following advantages[9,12] :

1) Its communication properties of high data bandwidth and low message latency increases the efficiency compared even to loosely coupled shared memory multiprocessors.

2) The distributed memory lessens contention, deadlock, etc.

3) Their topology can be changed to a ring[12], a tree[12] , and two or three dimensional meshes and thus adapt to a large class of problems.

4) Cube is homogenous (i.e, all nodes are identical), except for the manager. Because of reliability consideration, one of the nodes may be chosen to act as the manager with the additional advantage of faster communication between the manager and the nodes.

5) The Cube has very high inherent reliability due to the availability of multiple paths between the nodes.

6) Splitting the cube into subcubes increases the overall efficiency.
III. TOPOLOGICAL PROPERTIES OF HYPERCUBE

The hypercube is also called cosmic cube, n-cube, binary n-cube, boolean n-cube, etc. The n-dimensional hypercube is a highly concurrent loosely coupled multiprocessor based on the binary n-cube topology. Some properties that make its connectivity so appealing are discussed here[13]. The topological properties of Hypercube are:

1) An n-cube has $2^n$ nodes labelled 0 -- $2^n - 1$, such that there is an edge between any two vertices if and only if the binary representation of their labels differ by one and only one bit. This leads to an important property of the n-cube that it can be constructed recursively from lower dimensional cubes.

2) There are $n!$ $2^n$ different ways in which $2^n$ nodes of an n-cube can be numbered.

3) There are no cycles of odd length.

4) The diameter of a graph (defined as the maximum distance between any two nodes in a graph) for an n-cube is $n$.

5) A graph $G = (V, E)$ is an n-cube if and only if:
   a) $V$ has $2^n$ vertices.
   b) Every vertex is of degree $n$.
   c) $G$ is connected.
   d) Any two adjacent nodes $A$ and $B$ are such that the nodes adjacent to $A$ and those adjacent to $B$ are linked in a one-to-one fashion.

This leads to a 4*4 with wraparound at the edges (also called torus) mapped to a 4-cube and a 4*4*4 grid in 3-dimensional with wraparound edges is a 6-cube[13].

6) The minimum distance between the nodes $A$ and $B$ is equal to the number of bits that differ between $A$ and $B$, i.e., to the hamming distance $H(A, B)$.

7) Let $A$, $B$ be any two nodes of a n-dimensional hypercube and assume that $H(A, B) < n$. where $H(A,B) = i$ means that $A$ and $B$ differ by $i$ bits. Then there are $H(A,B)$ parallel paths of length $H(A,B)$ between nodes $A$ and $B$.

8) If $A$, $B$ are any 2 nodes of an n-cube and say $H(A,B) < n$. Then there are $n$ parallel paths between $A$, and $B$. Moreover, the length of each path is at most $H(A,B) + 2$. 
9) Binary sequence used to represent process-id in an n-dimensional hypercube is called gray code. Gray codes of arbitrary order can be generated using the following recursive definition

\[ G_{n+1} = \{ 0G_n, 1G^R_n \} \]  

where \( G^R_n \) is a sequence obtained by reversing \( G_n \).

### IV. iPSC (INTEL’s PERSONAL SUPERCOMPUTER)

The iPSC consists of two major functional units [14,15], the cube and the cube manager. The cube consists of an ensemble of one, two, or four computational units each consisting of 32 nodes. Nodes contain Intels 80286 CPUs coupled with 80287 numeric processing units and 512 Kbytes of local memory. The cube manager hardware consists of an Intel 286/310 microcomputer which is linked to each node over a global ethernet communication channel. A cube may also operate in stand-alone mode.

Cube software consists of a monitor and kernel residing on each board. The monitor is in PROM (Programmable Read Only Memory) and the kernel is loaded into node RAM (Random Access Memory) after initialization. Cube manager software consists of a Unix based programming and development environment with C, FORTRAN, cube control utilities and communication and system utilities. (In this thesis, the development of the Parallel Fuzzy ARTMAP simulator on hypercube was done using 'C'.)

Before any processing occurs, the CPU reads the program for node initialization. It loads the kernel into the node if initial diagnostics are successful. Memory management is maintained by the kernel. Processes are packed into high end memory and the messages are stored on the lower end of memory with movable partition between them, which is used by the kernel.

Interrupt processing is handled by two controllers. Each accepts interrupts from up to eight sources. They resolve parity and they interrupt the processor with a matching source.
The nodes have 80286 CPU, a 16 bit processor, operating at 8 MHz, at about 0.78 Mips. A five dimensional cube can execute at approximately 25 MIPS. The 80286 operates in virtual-address protected mode, isolating the kernel from the node processes. The CPU internally follows a pipelined fetch-decode-execute instruction cycle so as to provide maximum efficiency and throughput. The pipeline queue length is 6. The numeric floating point processors, 80287 handles 32, 64 and 80 bits floating point operations for the CPU.

The PROM is of 32K bytes, containing the software for node confidence test, utilities for down-loading kernel into the node. It also provides other diagnostic support. The Dual port RAM memory of 512 Kbytes contains the kernel of the node. The two ports are used to handle request both from the processors and the I/O bus.

Each node contains eight bidirectional communication channels managed by dedicated 82586 communication coprocessors. Seven out of eight channels link directly the nodes together and serve as dedicated point-to-point communication channels. The eight channels is a global ethernet channel provides direct access to and from the cube manager or host for program loading, data input/output and diagnostics.

Communication is done by using asynchronous message passing through communication channels. Each node is independent of the other and communicates with its neighbors using queued message passing. The messages are automatically routed from node to node, if necessary, to reach the destination node.

V. PARALLEL MACHINES FOR NEURAL NETWORKS

A lot of work is being done in developing parallel implementation of neural network paradigm especially for backpropagation networks. The main idea is to use the parallel nature of neural networks and map it on the actual machine. Neural Networks have been implemented on various computer configuration such as hypercubes, ring, torus, etc. The languages used include C, OCCAM, ADA, parallel C, etc. Companies like Adaptive Solutions Inc. are ready with a SIMD machine for neural network configuration of 128 or 256 processors per machine [11]. Intel has introduced 80170 [16] a neural

M Sette [17] has proposed an occam simulation of a general purpose neural network. Occam is available on Vax as well as on the transputers. In [17] there is a description of a simulator based on research on neural network paradigm.

Francois Robert and Shengrui Wang [18] discuss the implementation of neural network on a Hypercube F.P.S T20. In [18] there is a description of an efficient implementation of backpropagation network on the hypercube. With experimental analysis to back view point.

A. Petrowski, L. Personnaj, G. Drefus, C. Giravit [19] discuss various issues involved in the parallel implementation of multilayer feed forward neural network. In [19] there is a description of the results obtained using 3 loosely coupled architecture, i.e., toros, mesh, and ring.

J. Bourrley [20] discusses implementation of multilayer of back propagation network on a hypercube. Also discussed are results obtained using ring architecture.

The discussion in the literature concentrated in parallelizing backpropagation networks. This is mainly because of it being such a widely used network with very high training time and other problems[21]. None of the work concentrates in studying Adaptive Resonance Theory [2,3,4] based paradigms. ART networks are very efficient and closer to biological realities. In this thesis the parallel implementation of this network is discussed. Fuzzy ARTMAP[2] a supervised, analog input neural network paradigm with incremental learning feature has been implemented. Fuzzy ARTMAP has proved itself to generalize better on many of the real world problems than many of the other paradigms with far less number of epochs. Fuzzy ARTMAP is implemented in parallel with a hope that its efficiency in serial implementation can also be exploited to its limit with parallel implementation. This will help in learning very fast huge problems like analyzing protein structure to efficiently identify the secondary structures.
CHAPTER 3

FUZZY ADAPTIVE RESONANCE THEORY PARADIGM

I. CHARACTERISTICS OF FUZZY NETWORKS (ART based)

The Adaptive Resonance Theory architecture differs from other prevailing networks such as backpropagation, counterpropagation, Recurrent Networks, Boltzman machine, Hopfield Networks, etc., in a number of ways. Some of the main characteristics of ART-based Fuzzy Networks are highlighted below [22]:

A. Stability Plasticity Dilemma:

All intelligent systems capable of real time autonomous adaption to unexpected changes in their environment (also called world) face stability plasticity dilemma. A design issue for such a system is to preserve its previously learned knowledge while continuing to learn new facts without creating chaos. ART networks successfully adapt to new environments where rules may change without necessarily forgetting old knowledge or skills. The knowledge is incorporated globally in a self consistent manner.

B. Self Organization:

Humans can adapt to any new environment without any teacher or supervision. Learning in human is accomplished through experiences. Such adaption is called self organization in network modelling.

In ART-based Networks the top down weights keep track of the bottom up weights such that previously learned memories or rules are prevented from being
overwritten by any new learning. Knowledge is incorporated globally into the total system in a self consistent way.

C. Adaptability to Real-Time Responses:

The ART Network self organizes in real time, stable recognition codes to arbitrary sequence of input patterns. Within ART Network processing involves hypothesis testing, search, classification and learning.

D. Non Stationary Worlds:

The ART network self organizes to given input exemplars. In real time operations the network adapts to any arbitrary sequence. This sequence need not be preprocessed by any teacher before presentation. The network will accept both familiar and unfamiliar sequences.

E. Effective Usage of Memory Capacity:

ART-based network are not like other learning schemes (e.g, Hopfield networks) where the model is limited to use only partial memory capacity. The ART architecture is designed to learn with an unlimited number of input patterns or samples until it utilizes its full memory capacity. The networks does not get trapped in spurious memory states or local minima (as in backpropagation network and its variants). Once the learned exemplar is presented again, it will directly activate the output node in F₂ layer (output layer) that best represents it.

F. Fast and Slow Learning:

Speed of convergence depends on learning rate $\beta$ which lies within the range $[0,1]$. The search for finding the best match is fast and adaptive (not tree). It is accomplished
by direct access to codes of familiar events, i.e., similar events already learned by the model. In the approximate-match modes learning is rapid, stable and it occurs while buffering the system against noise. (In conservative limit, i.e, \( \alpha \rightarrow 0 \), the fuzzy networks do one shot learning, where \( \alpha \) is called the choice parameter).

G. Variable Vigilance Criteria:

Vigilance factor \((\rho)\) sets coarseness of the recognition code in response to environmental feedback. With higher vigilance, more self organizing categories are created and vice versa with lower vigilance, where less inputs features are encoded and hence less number of distinct categories are created. Category creation does not depend on the error criteria as is in many networks, e.g., backpropagation and its variant paradigms.

H. Properties:

Network Properties are preserved even for systems with large capacity and do not deteriorate like in other learning paradigms. The networks get scaled to arbitrarily large system capacities.

I. Competitive Learning Models:

ART models basically grew from Competitive Learning Models. Adaptive resonance theory embeds competitive learning models into self regulating control structures whose autonomous learning and recognition proceed stably and efficiently in response to any arbitrary sequence of input patterns. Input patterns in these networks are classified into mutually exclusive categories [23].
II. PREPROCESSING OF INPUTS TO FUZZY ART

Fuzzy ART [4] concepts are used in designing Fuzzy ARTMAP and Parallel Fuzzy ARTMAP. It is shown that concepts from fuzzy set theory when incorporated into Adaptive Resonance Theory 1 (ART-1), the new fuzzy paradigm generalize the network to handle both analog and binary input. Binary inputs are the limiting case where Fuzzy ART functions as ART-1.

The problem of pruning [24] is solved by normalizing the input in the preprocessing stage. Normalization of input could be done by either complementing the input vector or dividing each input element by the summation of all the input elements. Learning is stable as the network weights can only decrease in time. With fast-commit slow-recode, fast learning is combined with a forgetting rule that buffers system memory against noise, i.e., it takes care of statistically unreliable input fluctuation.

ART-1 and Fuzzy ART modules are used along with input to output mapping techniques to develop supervised paradigm namely ARTMAP and Fuzzy ARTMAP respectively. Incorporation of Fuzzy ART(s) in Fuzzy ARTMAP (or parallel Fuzzy ARTMAP) with a min-max rule or match tracking has resulted in a paradigm with rapid learning and stable categorical mapping between analog and binary input/output vectors.

The Fuzzy ART paradigm includes two optional features, i.e., learning and input preprocessing.

A. Fast-Learn Slow-recode

Many applications use fast learning. This results in a system to adapt quickly to inputs which may occur very rarely but which requires immediate and accurate
performance. The network is not related to the difference between the desired and actual output as backpropagation and thereby it is not necessarily limited to off line training.

In some applications it might be useful to initially combine fast learning and incorporate slower rate of forgetting. This is called "Fast-Commit Slow-Recode"[4]. The slow-recode part of the operation prevents features which have already been incorporated into a category's prototype from being mistakenly deleted in response to noisy inputs. Only persistent change in features can erase them from the prototype of the category. Fast learning on the other hand ensures adequate responses to input samples that occur only rarely.

In the algorithmic implementation of Fuzzy Art the following methods of fast-commit slow-recode have been provided:

1) Beta or learning rate for each epoch: Depending on the total number of epochs desired, learning rate for each epoch is specified.

2) Learning rate decays with an exponential factor for each epoch presented to the network: The exponential decay constant along with the present learning rate ($\beta$) decides the next iteration learning rate ($\beta_{\text{new}}$):

$$\beta_{\text{new}} = \beta_{\text{old}} - (\beta_{\text{old}} \times \text{decay})$$

3) The value of beta decreases by a constant factor for each iteration. The beta is constrained to be greater than or equal to zero. Therefore if $\beta_{\text{new}}$ results in negative value then the default value of $\beta_{\text{new}} = 0.0$ is set.
\[ \beta_{\text{new}} = \beta_{\text{old}} - \text{constant} \times 0.001 \] (3)

where

\[
\begin{align*}
0 & < \text{constant} \leq 1.0 \\
\beta_{\text{new}} & \geq 0.0
\end{align*}
\] (4)

Fast-commit slow-recode option in Fuzzy ART corresponds to ART 2 learning at intermediate learning rates.

B. Normalization

The preprocessing input stage is concerned with the normalization of input patterns. This prevents pruning or the problem of proliferation[24]. The input could either be normalized or be in a complement coding form.

1) Complement Coding: The input space of dimension M, is appended with its complement thereby increasing the dimension of input sample space to 2M dimension. Resulting in summation of inputs always equal to M.

Let the input of dimension M be represented as

\[ I = (a_1, a_2, a_3, \ldots, a_M) \] (5)

and the corresponding input in complement form of dimension 2M is

\[ I_{2M} = (a_1, a_2, \ldots, a_M, 1-a_1, 1-a_2, \ldots, 1-a_M) \] (6)
The concepts of complementing coding are interesting from three perspectives:

a) Neurobiological viewpoint: Both the inhibit cells and the excitor cells are used to represent an input pattern. This preserves amplitude while normalizing the total input vector comprising of inhibit/excitor cells.

b) Functional viewpoint: The inhibit cells of the input prototype encode features which are critically absent, while the excitor cells encode features that are critically present. Features occasionally present in the input samples or exemplars lead to lowering of weights in both inhibit and excitor cells.

c) Set theoretic viewpoint: Complement coding leads to a more symmetric theory, with both Fuzzy Max and Fuzzy Min from the fuzzy set theory. Complement coding establishes a relationship between excitor/inhibit cells of the input exemplar and fuzzy set theory operations (see Figure 3.1).

In Figure 3.1 two corners of a rectangular space are given:

For example, if \( X = (x_1, y_1) \) and 
\[
Y = (x_2, y_2)
\]

\[
X_{11} = (X \land Y)_1 = \min(x_1, x_2)
\]
\[
X_{12} = (X \lor Y)_1 = \max(x_1, x_2)
\]
\[
X_{21} = (X \land Y)_2 = \min(y_1, y_2)
\]
\[
X_{22} = (X \lor Y)_2 = \max(y_1, y_2)
\]
Fig 3.1 Relationship between Complement Coding and Fuzzy Subset Theory
Using the above equations the other two corners of the rectangle can be expressed in terms of fuzzy operators as

\[(X \land Y)_{(\text{bottom left corner})} = (X_{11}, X_{21})\]  \hspace{1cm} (11)

\[(X \lor Y)_{(\text{top right corner})} = (X_{12}, X_{22})\]  \hspace{1cm} (12)

2) Normalization or L1-normalization: The input vector is divided by the summation of the input features.

\[I_{\text{norm}} = \frac{I_i}{|I|}\]  \hspace{1cm} (13)

where

\[|I| = \sum_{i=1}^{\text{Dim of i/p}} I_i\]  \hspace{1cm} (14)

When the complement input is normalized, the summation of the input is M, where M is the dimension of the input sample space without complement.

\[|I| = \sum_{i=1}^{2M} I_i = M\]  \hspace{1cm} (15)
III. FUZZY ADAPTIVE RESONANCE THEORY PARADIGM

Fuzzy ART architecture (see Figure 3.2) comprises of two layers namely \( F_1 \) and \( F_2 \). The \( F_1 \) layer is a preprocessing stage for the input exemplars. The \( F_1 \) layer either normalizes the data or generates a one’s complement exemplar which is then fed as input to the network. The \( F_2 \) layer is the output layer consisting of categories corresponding to the input exemplar presented to the network and classified. The two layers are fully interconnected by feed forward and feed backward weights. The two layers are called Short Term Memory Traces (STM) while the weights which are adaptive are called Long Term Memory Traces (LTM). The \( F_2 \) layer is controlled by a reset circuit which inhibits the output neurons in this layer from further participation depending on the base line vigilance criteria for a given input sample.

Both feed-forward and feed-back weights in the paradigm are governed by the same learning rule. This results in both memories or weight matrices having the same memory contents. Since one set of weight matrices can serve the function of both feed-back and feed-forward weights the paradigm maintains only feed-forward weights, while feed-back weights are maintained virtually or hypothetically.

A. Input Vector

The input \( I \) is an \( M \)-dimensional vector,

\[
I = (I_1, I_2, \ldots, I_M)
\]  

(16)
Fig. 3.2  Fuzzy ART Architecture
If complement coding is active then

\[ I = (I_1, I_2, \ldots, I_i \ldots I_M, \ldots I_{2M}) \]  

(17)

where \(0 \leq I_i \leq 1\).

**B. Weight Vector**

Category \(j\) corresponds to a vector \(W_j\), defined as

\[ W_j = (W_{j1}, \ldots, W_{jM}) \]  

(18)

or if complement coding is active

\[ W_j = (W_{j1}, \ldots, W_{jM}, \ldots W_{j2M}) \]  

(19)

Where \(1 \leq j \leq N\), \(N\) being the number of potential categories and \(W_j\) is adaptive weights or Long Term Memory (LTM) traces.

The weights are initialized to:

\[ W_{j1} = W_{j2} = \ldots \ldots = W_{jM} = 1 \]  

or

\[ W_{j1} = \ldots \ldots W_{jM} = \ldots \ldots W_{j2M} = 1 \]  

\[ (20) \]

and the category \(j\) is said to be uncommitted. The weights could be initialized to a value greater than 1 but this biases the system against the selection of uncommitted nodes. In Fuzzy ART implementation in C the weights are initialized to 1.

After a category is committed, the weights or LTM traces \(W_{ji}\) decrease or remain constant, i.e., non-increasing through time. This eventually converges to a limit. As mentioned before the weight vector includes information for both bottom up and top
down weight vectors of ART-1 as the same learning rule governs both vectors.

C. Control parameters

The Fuzzy art paradigm has three main control parameters which determine the dynamics of the entire system:

1) Choice Parameter: Under conservative limit constrains, this parameter specifies for analog vectors, the degree to which the weight vector is a fuzzy subset of the input vector I. If $\alpha$ is the choice parameter then

$$\alpha > 0$$ \hspace{1cm} (21)

2) Learning rate ($\beta$): It determines the mode of learning to be either fast or slow and it must satisfy the following:

$$\beta \in [0, 1]$$ \hspace{1cm} (22)

where $\beta = 1$ implies fast learning mode.

3) Vigilance factor ($\rho$): It controls the total number of categories created due to the given set of input exemplars. The lower the vigilance factor (closer to zero) the less the importance of the input sample to the features. The higher the vigilance factor (closer to one) larger the number of the categories created at the output with distinct features of the input exemplars coded in the weight vectors. The vigilance factor must satisfy the following:

$$\rho \in [0, 1]$$ \hspace{1cm} (23)
D. Category Selection

For each input I presented in the paradigm, the net output for each category j is calculated as:

\[ net_j(I) = \frac{|I \land W_j|}{\alpha + |W_j|} \]  \hspace{1cm} (24)

where the fuzzy AND [25] is defined as

\[ (x \land y)_i = \min(x_i, y_i) \]  \hspace{1cm} (25)

and the norm of a vector X is defined as

\[ |X| = \sum_{i=1}^{M} |X_i| \]  \hspace{1cm} (26)

For a given input sample the max net is calculated and is assumed to be the winner, i.e.,

\[ Net_j = \max(net_j : j = 1 \ldots N) \]  \hspace{1cm} (27)

In the case of more than one winner, the probable winner is assumed to be the one with the lower category index j. This ensures that the nodes get committed in the order

\[ j = 1, 2, 3 \ldots . \]

E. Resonance

Resonance occurs if the match function of the chosen category meets the vigilance criterion. i.e.,
\[ \frac{|I \land W_j|}{|I|} \geq \rho \quad (28) \]

If the above equation is satisfied the network goes into learning mode and weight are updated to reflect the encoding of the training exemplars. 'J' is the winner neuron in the output layer of fuzzy ART.

If

\[ \frac{|I \land W_j|}{|I|} < \rho \quad (29) \]

then mismatch reset occurs, and the neuron J of the output layer $F_2$ is reset and restrained from further participation in the training of the present input exemplar. This prevents the neuron J from being persistently selected for the given input exemplar. The above process is repeated and a new index is chosen. The search continues until the chosen index satisfies the vigilance criteria or all the neurons in the output layer are exhausted, i.e., inhibited from further participation.

In case full memory capacity is utilized and the paradigm is unable to create any new category in the output layer, a modification is suggested. A neuron in the output layer with feed forward and feed back weights is created and initialized to one. Then the process to calculate $\text{net}_j$ is initiated which results in a new category (the new neuron in the output) being selected as the probable winner which also satisfies the vigilance criteria. The weights of the new category are updated, i.e., learning takes place to reflect the features of the input exemplar in the weight vector. This might change the final order
in which the nodes are committed, i.e., \( j = 1, 2, 3 \ldots \) can no more be guaranteed. The main advantage is that the training can continue with new initial conditions, and the number of neurons in the output layer updated to reflect the addition of new neurons. This ensures flexible neurons dynamics into the paradigm. This feature has been implemented in Fuzzy ART simulator using C.

F. Learning

The weight vector \( W_j \) of the winner neuron \( j \) is updated using the following learning equation suggested by [24]:

\[
W_j^{\text{new}} = \beta (I \wedge W_j^{\text{old}}) + (1 - \beta)W_j^{\text{old}}
\]  

(30)

Fast learning corresponds to \( \beta = 1 \), i.e.,

\[
W_j^{\text{new}} = (I \wedge W_j^{\text{old}})
\]

(31)
IV. ANALOGY BETWEEN FUZZY ART AND ART-1

The multiplication operation in mathematics can be mapped to an intersection operator \(( \cap \) in set theory which can be used to draw an analogy in fuzzy subset theory with fuzzy AND \(( \land \)). Similar analysis can be done for union operator \(( \cup \) in set theory and fuzzy OR \(( \lor \) in fuzzy subset theory.

A. Network Nature

The two paradigms (Fuzzy ART and ART-1) are unsupervised ART paradigms and are very similar in neural dynamics. Fuzzy ART is an analog network, i.e., it can accept both binary and analog inputs. On the other hand ART-1 is a binary network, i.e., only binary input exemplars are acceptable. Fuzzy ART network behaves as an ART-1 network if input exemplars are binary. Analogy is drawn between the dynamics of the ART-1 and Fuzzy ART paradigms to emphasize the one to one correspondence between set theory and fuzzy subset theory.

B. Category selection

The decision to select the acceptable category as the best output neuron representing the input exemplar is done as follows:

\[
Net_j \text{ ( ART-1 ) } = \frac{|I \cap W_j|}{\alpha + |W_j|}
\]

\[
Net_j \text{ ( Fuzzy ART ) } = \frac{|I \land W_j|}{\alpha + |W_j|}
\]
In Fuzzy ART the above relation represents [26] the degree to which the adaptive weights are a fuzzy subset of the input exemplars. Both the equations are similar, i.e., the intersection operator of the set theory corresponds to an "AND" operation in fuzzy subset theory. In the limiting case when binary inputs are presented to a fuzzy network, the fuzzy operators behave as binary operators, i.e.,

\[
(X \land Y) = (X \cap Y)
\]

and

\[
(X \lor Y) = (X \cup Y)
\]

The max Net\(_j\) is calculated and examined to see if the vigilance condition is satisfied or not.

\[
\rho \left( \text{ART-1} \right) \leq \frac{|I \cap W|}{|I|}
\]

\[
\rho \left( \text{Fuzzy ART} \right) \leq \frac{|I \land W|}{|I|}
\]

In fuzzy ART networks the equation represents the degree to which the input exemplar I, is fuzzy subset of the weight vector.

If mismatch does not occur, i.e., the winner neuron satisfies the vigilance criteria then learning takes place. Assuming fast learning, i.e., \(\beta = 1\), then the following rules are used to update the weights to reflect the features of the input exemplars.
\[
W_{j}^{(new)} \ (\text{ART-1}) = I \cap W_{j}^{(old)} \\
W_{j}^{(new)} \ (\text{Fuzzy ART}) = I \land W_{j}^{(old)}
\] (35)

Comparing equations for finding nets, mapping and learning, an important observation can be made about fuzzy operators and hence Fuzzy ART. In case the Fuzzy operators act upon a binary set of data rather than an analog one, the fuzzy operators are equivalent to their digital logic counterparts, see Table 3.1

**TABLE 3.1**

<table>
<thead>
<tr>
<th>Input 1</th>
<th>Input 2</th>
<th>AND</th>
<th>Fuzzy AND</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The above table clearly shows that the two operations Logical AND and Fuzzy AND are equivalent under binary state and so are ART-1 and Fuzzy ART paradigms under similar conditions.
CHAPTER 4

Fuzzy ARTMAP - A SUPERVISED ART PARADIGM

Fuzzy ARTMAP [2] is to ARTMAP as Fuzzy ART is to ART-1. The paradigm is an example of incremental supervised learning of recognition categories and multidimension maps in response to an arbitrary sequence of input vectors analog or digital.

Fuzzy ARTMAP architecture (see Figure 4.1) consists of two Fuzzy ART modules mapped via map field using Mini-max Learning Rule [2]. The Mini-max rule concurrently minimizes error and maximizes generalization. The basic idea behind the Mini-max rule is to increase the base line vigilance by a minimum amount so that the predictive error could be overruled. The network is prevented from pruning or category proliferation by normalizing the input to the network using a complement coding technique. A voting strategy can be used to assign probability estimates to competing predictions given a constrained training set.

Fuzzy ARTMAP is a supervised paradigm with a small number of control parameters, which require no problem specific adjustment or choice of initial weights. Fuzzy ARTMAP unlike backpropagation and its variants, does not get entangled in local minima. The Mini-max learning rule enables the network to learn quickly, accurately, and efficiently. The rule simultaneously minimizes error and maximizes generalization. This is done by increasing left modules’s (Fuzzy ART_a) vigilance factor $\rho_a$ by an amount that the network needs to correctly map a given input and output exemplar pair. The factor
Fig 4.1 Fuzzy ARTMAP Architecture
\( \rho \) is a measure of confidence that Fuzzy \( \text{ART}_a \) must have for the input exemplars to be acceptable, rather than go into search phase to prove the hypothesis. A low value of \( \rho \) leads to a broader generalization and to a high code compression since less categories are formed in the output layer. On the other hand a higher value of vigilance leads to a large number of categories being formed at the output layer for a given training samples. In the limiting case where the input exemplars are binary, the two Fuzzy ART modules behave as \( \text{ART}-1 \) and the whole network functions as \( \text{ARTMAP} \). Thus the analogy between \( \text{ART}-1 \) and Fuzzy ART can be extended further between \( \text{ARTMAP} \) and Fuzzy \( \text{ARTMAP} \).

The input to the network is preprocessed using the one's complement technique discussed earlier. A complement condition normalizes the input as well as preserves the amplitude of the original inputs. Without complement coding the Fuzzy ART module of Fuzzy \( \text{ARTMAP} \) during training encodes the degree of critical features consistently present in the input exemplars. With the presence of complement coding both the degree of absence and presence of features are represented by the category weight vector.
I. Fuzzy ARTMAP PARADIGM

The Fuzzy ARTMAP[2] paradigm (see Figure 4.1) system embodies two Fuzzy ART modules, i.e., Fuzzy ART\(_a\) and Fuzzy ART\(_b\) which are coupled via a map field \(F_{ab}\). The map field module's function is to form predictive associations between the categories of the two Fuzzy ART modules. It uses match tracking on Fuzzy ART\(_a\) to assert the association in case the two categories do not agree as desired. In match tracking the base line vigilance control( \(\rho_a\) ) of Fuzzy ART\(_a\) is increased by a minimum value necessary for the training exemplar to be correctly predicted. Match tracking reorganizes category structure such that the predictive error is not repeated on subsequent presentations of the training sample.

The network is unique because it does a forward mapping of many-to-one, i.e., large number of input sequence can get mapped to a single output. It also permits a backward mapping of one-to-many, i.e., inverse relation given single output, number of possible inputs to the network can be generated. The backward mapping is a very important feature and one can exploit it in calculation of inverse relations.

A. Dynamics of Fuzzy ART\(_a\) and Fuzzy ART\(_b\).

1) Input vector: The input to the network is in complement form of dimension \(2M\) representing both inhibit and excitor cells. Where \(M\) is the dimension of the given input exemplar.
In Fuzzy ART module

\[ A = (i, i^c) \]

\[ \equiv (i_1^a, \ldots, i_M^a, \ldots, i_{2M}^a) \]  

In Fuzzy ART module

\[ B = (O, O^c) \]

\[ \equiv (O_1^b, \ldots, O_M^b, \ldots, O_{2M}^b) \]  

2) **Weight Vector**: It encodes the training sample in its long term memory. Weights are adaptive in nature and are modified to encode the training exemplars. The \( j^{th} \) Fuzzy ART\(_a\) weight vector be denoted as:

\[ W_j^a \equiv (w_{j1}^a, \ldots, w_{jM}^a, \ldots, w_{j2M}^a) \]  

and \( k^{th} \) Fuzzy ART\(_b\) weight vectors be denoted as:

\[ W_k^b \equiv (w_{k1}^b, \ldots, w_{kM}^b, \ldots, w_{k2M}^b) \]  

3) **Output Vector**: The corresponding output vector in the output layer of individual fuzzy networks may be denoted as:

In Fuzzy ART\(_a\)

\[ F^a \equiv (f_1^a, f_2^a, \ldots, f_{N_a}^a) \]
In Fuzzy ART<sub>b</sub>

\[ F^b \equiv (f_1^b, f_2^b, \ldots, f_{N^b}^b) \]  \hspace{1cm} (41)

where \( N_a \) and \( N_b \) are the total number of categories in the output layer of the Fuzzy art networks.

4) **Dynamics of mapping field, i.e., auto-associative memory:** Fuzzy ART<sub>b</sub> has a one to one correspondence with the output of the mapping field \((X^{ab})\). Let map field output be denoted as:

\[ X^{ab} = (x_1^{ab}, x_2^{ab}, \ldots, x_{N^b}^{ab}) \]  \hspace{1cm} (42)

The output layer of Fuzzy ART<sub>a</sub> is mapped one-to-many to map field \( F^{ab} \) via weight vector \((W_1^{ab})\).

\[ W_j^{ab} \equiv (W_{j1}^{ab}, W_{j2}^{ab}, \ldots, W_{jN^b}^{ab}) \]  \hspace{1cm} (43)

The weight vector \((W_j^{ab})\) along with the output vector \((F^b)\) of Fuzzy ART<sub>b</sub> are two inputs to the map field giving an output vector \( X^{ab} \) based on probability estimate \( \rho_{ab} \).

5) **Initialization:** The following adaptive weights, i.e., long term memory traces are initialized to "1" before initiating any learning. In case of retraining the network starts learning from an initial set of already known weight:

a) Long term traces of Fuzzy ART<sub>a</sub> module \((W_j^a)\).

b) Long term traces of Fuzzy ART<sub>b</sub> module \((W_k^b)\).

c) Map field weights \((W_j^{ab})\).

The above three are the adaptive weights used to encode the given training exemplars.
It is assumed that during the presentation of each training input the following are initialized to zero:

a) Output layer of Fuzzy ART_a (F^a).

b) Output layer of Fuzzy ART_b (F^b).

c) Outputs of Map field (X^{ab}).

6) **Map field activation**: The map field F^{ab} is activated whenever one of the Fuzzy ART modules is active. In other words, at least one of the output neuron in any one of two Fuzzy ART module satisfies its vigilance. If node J from Fuzzy ART_a, F^a layer is chosen, then its weight W_{J}^{ab} activates F^{ab}. If node K from Fuzzy ART_b, F^b layer is chosen, then node K in F^{ab} is activated by one-to-one mapping between Fuzzy ART_b output layer and the map field. In case both Fuzzy ART modules are active then the map field becomes active if and only if Fuzzy ART_a using adaptive weights W_{J}^{ab} predicts the same category as Fuzzy ART_b. The F^{ab} output vector X^{ab} is governed by the following set of rules:

\[
X^{ab} = \begin{cases} 
F^{b} \land W_{J}^{ab} \text{ if both modules are active} \\
W_{J}^{ab} \text{ if Fuzzy ART}_a \text{ is active} \\
Y^{b} \text{ if Fuzzy ART}_b \text{ is active} \\
0 \text{ otherwise} 
\end{cases}
\]

(44)

Match tracking is initiated if X^{ab} = 0, assuming that probability estimate \( p_{ab} = 1.0 \) and that both modules are active. Match tracking leads to category selection that...
represents the given training sample.

7) **Match tracking**: The base line vigilance for Fuzzy ART\(_a\) is set to the vigilance factor \(\rho_a\) for each new input presentation. If

\[
|X^{ab}| < \rho_{ab} |Y^b|
\]  

(45)

then \(\rho_a\) is increased to become slightly larger than factor \(I_a\). Where \(I_a\) for the analog vector \(I\) is defined as the degree to which the input is the fuzzy subset of the weight vector \(J\) (the winner neuron in the output layer of Fuzzy ART\(_a\), i.e., \(F^a\)). In other words

\[
I_a = \frac{|A \land \tilde{w}_{Ja}|}{|A|}
\]  

(46)

where \(A\) is the input to Fuzzy ART\(_a\) in complement form denoted as:

\[
A = (i, i^c)
\]  

(47)

This leads to either activation of another \(F^a\) node \((J)\) satisfying

\[
\frac{|A \land \tilde{w}_{Ja}|}{|A|} \geq \rho_a
\]  

(48)

and

\[
\frac{|X^{ab}|}{|Y^b|} \geq \rho_{ab}
\]  

(49)

or, the network has exhausted all possible categories and no more node exists, i.e., full memory capacity utilization. If the network has reached its full capacity, then it is turned off for the remainder of input presentation. Otherwise the network changes modes to the learning phase.
8. Map field learning: For mapping field, initially the weights are initialized to all "1", i.e.,

\[
\begin{align*}
\mathcal{W}_{jk}^{ab} &= 1 \\
  j &= 1, 2, 3, \ldots, N_a \\
  k &= 1, 2, 3, \ldots, N_b
\end{align*}
\]  

(50)

and on resonance Fuzzy ART\textsubscript{a} category J maps to category K of Fuzzy ART\textsubscript{b} and the corresponding field weights are updated as follows:

\[
\mathcal{W}_{jL}^{ab} = \begin{cases} 
  1 & L = K \\
  0 & L = 1, 2, \ldots, N_b ; \quad L \neq K
\end{cases}
\]

(51)

where \(N_a\) and \(N_b\) are the total number of categories in the output layer of Fuzzy ART\textsubscript{a} and Fuzzy ART\textsubscript{b} respectively.

These weights are fixed hence forth. This means that category J has been assigned and is not available for reclassification. Many neurons from Fuzzy ART\textsubscript{a} can be mapped to one neuron in the map field, but under no circumstances there can be more than one neuron from Fuzzy ART\textsubscript{a} module allowed to map to more than one neuron in the mapping field.

The corresponding Long Term Memory traces in the Fuzzy ART modules are also updated to reflect a perfect mapping. Fast learning (\(\beta = 1\)) occurs in both modules simultaneously.
Learning in Fuzzy ART$_{a}$:

\[(W^a_j)^{new} = (I \land (W^a_j)^{old})\]  \hspace{1cm} (52)

Learning in Fuzzy ART$_{b}$:

\[(W^b_k)^{new} = (I \land (W^b_k)^{old})\]  \hspace{1cm} (53)
II. SIMPLIFICATION IN MAPPING.

The mapping field is well equipped to decide whether match tracking or learning takes place and how. The map field can be simplified with certain assumptions such as:

1) The probability estimate of mapping field is unity:

\[ \rho_{ab} = 1.0 \quad (54) \]

2) During training both Fuzzy ART modules get inputs simultaneously.

3) In mapping analysis phase both modules are active simultaneously.

Since only one category is selected as a winner in each Fuzzy ART module, a binary representation would suffice. The winner neuron, J in Fuzzy ART\textsubscript{a} and K in Fuzzy ART\textsubscript{b} can be represented as binary ones while the remaining neurons can be represented as binary zeros.

The weight vector \( W_{J}^{ab} \) corresponding to active output neuron J from fuzzy ART\textsubscript{a} is mapped to the winner neuron k of Fuzzy ART\textsubscript{b} giving \( X^{ab} \) as:

\[
X_{k}^{ab} = F_{k}^{b} \cap W_{jk}^{ab} \quad (55)
\]

\[
k = 1, 2, 3, \ldots, N_b
\]

since only category K in the output layer is active and all other output are zero, i.e., inhibited, the above equation can be simplified to

\[
X_{K}^{ab} = F_{K}^{b} \cap W_{JK}^{ab} \quad (56)
\]
If

\[ X_K^{ab} = |X^{ab}| = 1 \]  \hspace{1cm} (57)

then the Fuzzy ART modules can undergo learning and update their corresponding weights to encode the training exemplars in the long term memory traces. Else if

\[ X_K^{ab} = |X^{ab}| = 0 \]  \hspace{1cm} (58)

then match tracking takes place. The vigilance factor \( \rho_a \) is changed to a new higher value as follows:

\[
(\rho_a)^{new} = \frac{|A \wedge W_J|}{|A|} + 0.0001 \]  \hspace{1cm} (59)

The new vigilance factor is used by Fuzzy ART to search for the next best possible category in the output layer, that represents the given training sample.

All the above modifications are valid as the map field can be treated as a binary network. This simplified mapping field is implemented in the simulator built using dynamic storage allocation in C programming language on Sparc workstation. The simulator is built using concepts of dynamic memory allocation as the network configuration is problem dependent.
CHAPTER 5

MODIFIED FUZZY ARTMAP (M-FAM)

Modified Fuzzy ARTMAP is a supervised paradigm designed on the principles of ART based networks. It includes all the modifications suggested in designing Fuzzy ART and Fuzzy ARTMAP. Modified Fuzzy ARTMAP (M-FAM) is designed by incorporating Mini-Max Rule from Fuzzy ARTMAP[2] on unsupervised Fuzzy ART paradigm. M-FAM is quite similar in operation to Fuzzy ARTMAP. But redefining the neuro architecture and neuro dynamics of the paradigm has made its distribution on any parallel architecture very simple. The distribution of neurons in hypercube is characterized by balanced load distribution.

I. M-FAM NEURO ARCHITECTURE

M-FAM can be broadly classified into four main sections (see Figure 5.1):

A. Input Network.
B. Fuzzy ART.
C. Map Fields.
D. Output Network.

Each of these sections perform's specific function in a coordinated manner. The coordination among these sections ensures supervised learning. The operation of the four sections can be summarized as follows:
Fig 5.1 Modified Fuzzy ARTMAP Architecture
A. Input Network

The raw input exemplar is preprocessed before the network performs any analysis on it. The given input is always complemented. This is a preferred method due to various reason ranging from neurobiological to symmetric set-theory as discussed in the chapter-3 on Fuzzy ART.

B. Fuzzy ART[4]

Fuzzy ART is an unsupervised ART-based paradigm. A teacher or critic is not necessary for the network to perform any learning. A simple modification is done to ensure the paradigm always operate under fast learning mode ( $\beta = 1$). The training is initiated with a base line vigilance. This vigilance may be modified by the map field during training if necessary. The learning rule[24] is

$$W_J^{new} = \beta \left( I \land W_J^{old} \right) + (1 - \beta) W_J^{old} \quad (60)$$

is simplified to

$$W_J^{new} = I \land W_J^{old} \quad (61)$$

Learning is initiated by the map field when it deems necessary. The output category being selected as winner in unsupervised training need not necessary mean the category is overall acceptable. The category declared as winner by the Fuzzy ART has to map to the corresponding output via a Map Field. Only when the map field agrees to the mapping, the entire network goes in learning phase.
C. Map Field

It maps the corresponding input-output pair. It is similar in operation to the map field in Fuzzy ARTMAP. The output of the map field must be the same as the desired output for letting the network go in learning phase. Else the network does match tracking and modifies the vigilance factor of the Fuzzy ART to a value such that the predictive error is not repeated.

D. Output Network

Since M-FAM is a supervised paradigm, the output of the network plays the role of directing the network on what to learn and how to learn it. The output presented to the network is in binary form. The output can have no more than one excited neuron at a given instant. Nor can all the output neurons be zero simultaneously.

During training the input/output exemplars are presented together. The network learns to predict training exemplars in fast learning mode, i.e., one shot learning till all samples can be accurately predicated.
II. M-FAM PARADIGM

The architecture associates input and output network using Map field. The map field is a binary network. Its main function is to emphasize the association between the predicted category by Fuzzy ART and the desired output in the output network.

The map field is governed by Mini-Max rule [2] to assert the input/output association. Mini-max rule does match tracking in case of discrepancies. Using match tracking the vigilance factor of fuzzy ART is increased such that the given training exemplar is correctly mapped. The categories are reorganized in a fashion such that the predictive error is not repeated. The Mini-max rule conjointly minimizes predictive error and maximizes code compression.

The network designed preserves all the properties of Fuzzy ARTMAP, e.g., many to one and one to many mapping. Many to one association is forward mapping, i.e., predicting output given input(s). While one to many association is backward mapping, i.e., predicting input(s) for a given output.

A. M-FAM Network Dynamics

Each M-FAM module has unique properties. These properties when coordinated together make the network a supervised paradigm. The paradigm is neither iterative nor root mean square error bound like backpropagation and its variants.

1) Input Network: The raw input exemplar (i) is preprocessed. Preprocessing generates a new input exemplar (I) of dimension 2M, where M was the dimension of the given input sample (i). The given input is in one's complement, i.e.,
This is a preferred method for the reasons discussed before (neurobiological, symmetric set theory, etc.).

2) **Fuzzy ART**: The unsupervised ART paradigm is capable of learning both analog and binary training samples. The Paradigm dynamics have been modified to incorporate it as a module of Modified - Fuzzy ARTMAP (M - FAM). The modification discussed ensures the network operate under supervised mode when desired.

a) **Input**: The input to the network is of dimension 2M in complement form, i.e,

\[
I = (i_1, i_2, \ldots, i_M, \ldots, i_{2M})
\]

where

\[
I_i \in [0, 1]
\]

b) **Weight**: The weight vector \((W_j)\) also called long term memory trace has the same dimension as the input network \((2M)\). The weights represent the encoded features of the inputs and are adaptive. They are updated when the map field triggers the learning phase for the entire network. The weight vector ensures fully interconnected configuration and is represented as

\[
W_j = (W_{j1}, W_{j2}, \ldots, W_{jM}, \ldots, W_{j2M})
\]

where \(1 \leq j \leq N\), \(N\) being the total number of categories in the output layer of Fuzzy ART.
The weight vector(s) are initialized as

\[ W_{j1} = W_{j2} = \ldots = W_{jM} = 1.0 \] (66)

c) **Control Parameters**: The following control parameters are used for tuning the network to learn and predict the categories accurately. The vigilance factor is manipulated by the map field if necessary for each training sample to ensure correct prediction. Three control parameters for tuning the network are defined as:

**c.1) Choice Parameter** \( (\alpha) \): M-FAM operates in the conservative limit, where the choice parameter tends to zero, i.e.,

\[ \alpha \rightarrow 0.0 \] (67)

In the implementation

\[ \alpha = 0.0001 \] (68)

The choice parameter in the conservative limit represents the degree to which the weight vector is a fuzzy subset of the given analog input vector \( I \).

**c.2) Learning Rate** \( (\beta) \): The paradigm operates in fast learning mode, i.e.,

\[ \beta = 1.0 \] (69)

This ensures one shot learning for a given training sample in the Fuzzy ART module. Which necessary is not the case for Modified Fuzzy ARTMAP.

**c.3) Vigilance factor** \( (\rho) \): The vigilance factor must satisfy the following boundary condition:

\[ \rho \in [0, 1] \] (70)
The initial value of the vigilance factor forms the base line vigilance for each training sample. The vigilance factor is increased by the map field using match tracking until the input/output pair are correctly mapped.

d) Category Selection: Fuzzy ART predicts a category which the map field must map to the given output for the training exemplar. The predicted category must satisfy the vigilance criteria of Fuzzy ART (resonance but no learning). Fuzzy ART calculates the Net\(_j\) for each category and assumes the maximum of them as the probable winner.

\[
Net_J(I) = \frac{|I \land W_j|}{0.0001 + |W_j|}
\]  

where probable winner is

\[
Net_J = \max (Net_j : j : 1, 2, 3, \ldots \ldots \ldots N)
\]

and N is the total number of category in the output layer of Fuzzy ART.

e) Resonance: Occurs in Fuzzy ART if the chosen category J satisfies

\[
\frac{|I \land W_j|}{|I|} \geq \rho
\]

It results in category J being selected which is mapped with the desired output vector via the map field.

If the category J does not satisfy equation (73), i.e.,

\[
\frac{|I \land W_j|}{|I|} < \rho
\]

then mismatch occurs. The probable winner J is inhibited from further participation as a
probable winner for the given training sample. The above process is repeated to search for another probable winner that best represents the given input exemplars. The search terminates either when a winner is found or when the network has reached its full memory capacity.

3) Map Field Activations: Figure 5.2 gives details of the various inputs, outputs, and control parameters of the map field. All the neurons in the output layer of Fuzzy ART are mapped by $W_j^M$ in many to one fashion. Many to one mapping implies more than one neuron from Fuzzy ART may be mapped to the same neuron in the Map field.

Analysis of Map Field can be broken into testing map field analysis and training map field analysis

a) Training map field analysis: The output network has one to one mapping with the neurons in the map field. The two input vectors $W_j^M$ and $O_K$ (see Figure 5.2) to the map field are binary. The output vector $X^M$, is defined as function of the two input vectors according to the equation:

$$X^M = O_K \cap W_j^M$$

(75)

Since probability estimate is always one ($\rho_m = 1$) and only one neuron in the output network is active, then

$$|X^M| \geq \rho_{ab}|O_K|$$

or

$$|X^M| \geq |O_K|$$

or

$$|X^M| = |O_K| = 1$$

(76)
Fig 5.2  Map Field Network
Equation (76) leads to an important rule for mapping during training, i.e.,

If

\[ |X^M| = 1 \]  

(77)

then

\[ \frac{|X^M|}{|O|} = \rho_{ab} = 1.0 \]  

(78)

Since no match tracking is needed, the network can go in learning phase to encode the present input in the Fuzzy ART long term memory traces and the map field adaptive weights.

Else if

\[ |X^M| = 0 \]  

(79)

then

\[ \frac{|X^M|}{|O|} < (\rho_{ab} = 1.0) \]  

(80)

Equation (80) implies the application of Match Tracking to change the Fuzzy ART vigilance factor. Mini-Max rule results in the minimizing of the predictive error.

**a.1) Mini max Rule:** When the two outputs, i.e., the predicted and the desired don’t match, mini-max rule\([2]\) is applied to generate a new vigilance factor for Fuzzy ART. The new vigilance factor is higher than the present base line vigilance. This ensures that the same category is not predicted again as the winner for the given input exemplars.
Using match tracking the vigilance factor is set to a value slightly higher than a factor $\Omega$. Which represents the degree to which the input is a fuzzy subset of the weight vector of category $j$ of Fuzzy ART, i.e.,

$$\Omega = \frac{|A \wedge W_j|}{|A|} \quad (81)$$

The new vigilance factor for the training sample is defined as a function of $\Omega$ by the following equation:

$$\rho_{\text{new}} = \Omega + 0.0001 \quad (82)$$

\textbf{a.2) Learning in M-FAM:} During learning the map field adaptive weights as well as the long term memory traces of Fuzzy ART are modified to encode the training exemplar. The map field weight vector $W_j^M$ is updated whenever resonance occurs between Fuzzy ART category $J$ and category $K$ of the output network. The map field update is defined using the following equation:

$$W_{jLM} = \begin{cases} 1 & L = K \\ 0 & \text{otherwise} \end{cases} \quad (83)$$

Where $1 \leq L \leq$ output neurons in the output or map field modules.

Node $J$ of Fuzzy ART becomes committed to node $K$ of the output network and is no more available for reclassification. The mapping is many to one from output layer of Fuzzy ART to map field. Therefore there can be more than one output neuron mapped to the same neuron in the map filed but under no circumstances can there be more than one output neuron of mapping field mapped to category $J$ (a committed node) of Fuzzy ART.
Similarly the weights in the Fuzzy ART module undergo learning to encode the input pattern in the weight vector. The weights, $W_i$ of Fuzzy ART are updated under fast learning mode ($\beta = 1$) as follows

$$W_J^{new} = \bigwedge I \land W_J^{old} \quad (84)$$

b) Testing map field analysis: When in testing mode either the input network or the output network is not zero but not both. Here no adaptive learning is initiated.

b.1) Input is active: The paradigm has to predict the output given input exemplar in complement code. The map field output ($X_M$) is defined as

$$X_M = W_J^M \quad (85)$$

where J is the predicted winner neuron in the output layer of Fuzzy ART.

b.2) Output is active: The paradigm has to predict input(s) for the given non zero output exemplar. The map field output ($X_M$) for a given active neuron (K) in the output network as:

$$X_M = O \quad (86)$$

4) Output Network: The output of the network is presented in binary form with no more than one neuron active for a given training exemplars. The output neurons are directly mapped to the map field.

For example: Given the network has to learn to predict three different categories

a) Downward trend.
b) Steady trend.
c) Upward trend.
The following representation or something similar would suffice to represent the above categories.

a) Downward trend  -------  0 0 1
b) Steady trend     -------  0 1 0
c) Upward trend     -------  1 0 0

The above representation has 3 neurons to represent 3 categories and each category has an excitor cell at a unique position. The summation of each representation always sums to one, i.e.,

\[ |O| = 1 \quad (87) \]

5) Initialization: The network is assumed to have started from some initial condition. These initial conditions are assumed to be true irrespective of the environment.

a) Following memory traces are initialized to all ones before any training is initiated.

a.1) A Fuzzy ART long term memory traces \((W_j)\).

a.2) Map field weights, mapping output network \((O)\) to Fuzzy ART output \((W^M_j)\)

b) Following assumptions are made during training, on presentation of new input to the networks as initialized to zero:

b.1) Output Vector of Fuzzy ART \((Y)\).

b.2) Output vector of Output network \((O)\)

b.3) Output of map field \((X^M)\).
CHAPTER 6

PARALLEL FUZZY ARTMAP (P-FAM)

P-FAM is a generalized ART based neural network paradigm implemented on a n-dimensional hypercube configuration of processors. The parallel implementation can be adapted to any parallel architecture such as Butterfly networks, Omega networks, Shuffle exchange networks, Ring networks, Torus networks, etc. Message communication among processors in the above mentioned architecture are implementation dependent.

The n-dimensional hypercube is a highly concurrent loosely coupled multi processor-based architecture on the binary n-cube topology. The development of P-FAM was done on Intel's Personal Supercomputer (iPSC) simulator. The algorithm discussed is implemented on a hypercube topology of dimension four (see Figure 6.1). Hypercube of dimension four constitutes of 16 processors interconnected as a cube. iPSC machines exits usually in multiple of 32 processor modules, but the iPSC simulator used in this work is limited to 16 processors maximum. The algorithm designed is developed for an n-dimensional hypercube and is not limited to the available simulator configuration. The algorithmic implementation of the paradigm makes a system call to identify the available configuration of the hypercube used. The dimension of the hypercube is used for the calculation of the total number of processors available in parallel. The processor configuration is used as a criterium to distribute the P-FAM configuration among processors with a balanced load distribution.
FIG 6.1 4-dimensional Hypercube
An n-cube parallel processor consists of $2^n$ identical processors, each with its own local memory and interconnected with $n$ neighbors. The processors in the hypercube are numbered using gray-code representation from 0 to $(2^n - 1)$ such that two processors are connected if and only if the binary representation differ by no more than one bit. The number of different ways in which a $2^n$ processors or nodes of a n-cube can be numbered is given by equation (86).

$$n! \ 2^n$$

For our discussion the representation shown in Figure 6.1, is selected. Note that the implementation of P-FAM is transparent as to how the processors are numbered.
I. P-FAM ARCHITECTURE DISTRIBUTION ON N-CUBE

Modified Fuzzy ARTMAP architecture (see Figure 6.2) is to be implemented on an n-dimensional hypercube. The four sections of the paradigm highlighted in Figure 6.2 are:

a) Input Network,
b) Fuzzy ART,
c) Map Field,
d) Output Network.

The input to the network is always assumed to be in one’s complement form. The training file will have the following format:

a) Total number of training samples,
b) Probable dimension of the input sample space. (Number of input neurons I),
c) Probable number of neurons in the output layer of Fuzzy ART (N),
d) Dimension of the map field ($N_o$) or the dimension of the output sample space (O),
e) Input sample,
f) Corresponding output samples.

Input/output pairs should correspond to the total number of training samples the network must be trained for. Options (b) through (d) determine the network configuration which is globally broadcasted to all the nodes of the hypercube. The network configuration is used to create P-FAM dynamically, with balanced load distribution on all the processors of the hypercube. The Load balancing feature is dependent on the total number of neurons in the output layer (Y) of Fuzzy ART.
Fig 6.2 Modified Fuzzy ARTMAP Architecture to be Implemented in Parallel
Fuzzy ART, Map field and Output modules are synchronized so that the network learns to predict the input/output pattern. The weights are updated to reflect the training. The resultant winner category number along with the processor id and neuron id in that processor is transferred to the host. The host records the category and transfers the next sample in queue to the n-cube/hypercube. The process is repeated until all the samples are correctly predicted by the paradigm during training or the starting configuration specified is inadequate.

The testing file will have the following format:

a) Total number of testing samples.
b) Dimension of the input sample space. (Number of input neurons I).
c) 0 , any other value is ignored.
d) 0 , any other value is ignored. During testing if the value is non zero, then it is assumed that the outputs are specified but may be ignored.

This helps in testing a training file without any changes in data format.
e) Input testing sample.
f) Corresponding output testing samples. Ignored even if present.

Input/output pairs should correspond to the total number of testing samples, the network must be tested for. Network configuration which is globally broadcasted to all the nodes of the hypercube is taken from the weight file (generated during training). The testing phase predicts the output(s) for a given input(s) along with the processor id and the neuron id in that processor.

The network file generated at the end of training or retraining constitutes information calculation using the neuro dynamics of P-FAM. The network file has all the adaptive weights with relevant network configuration.
The Network configuration decides the total number of neurons per processor which are dynamically created before initiating any training. Figure 6.3 shows the neuron distribution in a processor of a n-dimensional hypercube. The algorithm in a processor \( p \) decides dynamically the number of neurons in processor \( p \) of a hypercube. The total number of neurons created in processor \( p \) are limited by the total amount of memory available per processor \( p \).

Each processor contains a number of Fuzzy ART output neurons. Each of these neurons has a set of input neurons, an associated map field and an output network. Each neuron created in the processor has the following configuration:

- a) Output neuron of Fuzzy ART \( (Y_p) \).
- b) Input neuron of dimension \( 2M \).
- c) Interconnection weight \( (W_j) \) to the output neuron of Fuzzy ART.
- d) Interconnection weights \( (W^M_j) \) between output neuron of Fuzzy ART to Map Field neurons.
- e) The Map field has the same dimension as the output network \( (N_o) \). There is a one-to-one mapping between the neurons of the map field and the neurons of the output layer. Hence the output of the training sample is connected directly as an input to the map field.
- f) Reset/Match tracking: Depending on the map field output, the network increases Fuzzy ART base line vigilance or goes in learning phase to update the various adaptive weights in the network.

Each neuron in a processor is an independent network by itself and it is in a position to represent only one category. Each processor in the n-cube will have a number of similar neuro architectures. All the neurons individually calculate the net output and find a local maximum among themselves. The local maxima (as it is not the maxima of
FIG 6.3 Neurons Distribution on a Node in a Hypercube
the entire network) of the neuron’s adaptive weights are used to calculate the probability of representing the given input exemplar. This probability must satisfy the vigilance criteria for the neuron to be selected as probable winner or else the next highest local maximum in the same processor is evaluated.

All processors calculate their local maximum in parallel. The processors then exchange their maximum to calculate the global maximum of the network. The global maximum neuron on satisfying the map field probability estimate undergoes learning or else a new vigilance is calculated and the network repeats the process again. In case no neuron in a processor satisfies the vigilance, the local maximum in that processor is set to -1 (impossible maximum as Net can never be negative). In case that the network evaluates the global maximum as -1, the training/retraining is terminated. In such cases the host is informed that the initial network configuration was not adequate.
II. NEURO DYNAMICS OF P-FAM

Neuro dynamics describe the flow of control in P-FAM. Each processor has 'NP' neurons per processor. The Architecture of these neurons is as described before (Figure 6.3). Following equations governs the neuro dynamics in P-FAM :

A. Category Selection in Processor 'p' of Hypercube

The first objective is to find the local Net maximum in each processor. If neuron 'J' is selected by processor 'p' as its local maximum then neuron 'J' must satisfy the Fuzzy ART vigilance factor (\(\rho\)) to qualify for local maximum competition in processor 'p', i.e.,

\[
\frac{|I \land W^p_{J}|}{|I|} \geq \rho \quad (87)
\]

where \(I\) is the input vector and \(W^p_{J}\) are the interconnection weights from the input neurons to the 'J' neuron in the output layer (Y), in processor 'p'.

The local winner neuron 'J' is the neuron with \(\max \text{Net}(\text{Net}^p_J)\), i.e,

\[
\text{Net}^p_J(I) = \frac{|I \land W^p_{J}|}{0.0001 + |W^p_{J}|} \quad (88)
\]

where the max net is

\[
\text{Net}^p_J = \max (\text{Net}^p_J : j:1, 2, \ldots, NP) \quad (89)
\]

and where NP is a constant specifying number of neurons per processor in a d-dimensional hypercube and where \(p\) satisfies \(0 \leq p \leq 2^d - 1\).
The summation of weights associated with neuron 'J' in processor 'p' is defined as:

$$| \mathcal{W}_J^p | = \sum_{i=0}^{2M} \mathcal{W}_{J_i}^p $$ (90)

Where $2M$ is the dimension of the input sample space.

**B. Map Field Activation of Neuron 'J' in Processor 'p'**

The input/output pair is given to each neuron 'j' in processor 'p' of a n-dimensional hypercube. A look ahead is performed to know whether match tracking or learning will take place in case neuron 'J' in processor 'p' is the global winner.

The mapping output for local winner neuron 'J' in processor 'p' is evaluated as

$$X_J^{MP} = O \cap \mathcal{W}_J^{MP} $$ (91)

where $O$ is the constant output vector for a given exemplar. $\mathcal{W}_J^{MP}$ and $O$ are the input vector to the map field for neuron 'J' corresponding to processor 'p', giving output vector $X_J^{MP}$ in processor 'p' for neuron 'J'.

If

$$|X_J^{MP}| = 1 $$ (92)

then the network will go in learning mode in case neuron 'J' in processor 'p' is the global winner.

Else if

$$|X_J^{MP}| = 0 $$ (93)

then the network will go in match tracking mode in case neuron 'J' in processor 'p' is the global winner.
C. Mini Max Rule

In case the network goes into match tracking mode a new vigilance factor is calculated assuming neuron 'J' in processor 'p' is the winner.

\[
\rho_{new,\text{J}}^P = \frac{Net^P_J \cdot (0.0001 + |W^P_J|)}{|I|} + 0.0001 \tag{94}
\]

Where I is a constant input vector for all the neuron for a given exemplar. When the network actually does match tracking the new vigilance is set to

\[
\rho_{new} = \rho_{new,\text{J}}^P \tag{95}
\]

D. Learning in P-FAM

If the network goes into learning mode then neuron 'J' in processor 'p' on being declared as the winner will go under the following learning to encode the given input/output exemplar.

1) Map Field Updates: The map weights \( W_{JL}^M \) associated with neuron 'J' in processor 'p' will go through the following map field learning equation:

\[
W_{JL}^{MP} = \begin{cases} 
1 & L = K \\
0 & \text{else}
\end{cases} \tag{96}
\]

where K is the active output neuron in the given output samples and L satisfies the relation \( 1 \leq L \leq N_o \), where \( N_o \) is the output sample space.

2) Interconnection Weights in Fuzzy ART: The adaptive weights interconnecting the input sample space to the output layer of Fuzzy ART (Y) associated with neuron 'J' in processor 'p' undergoes learning using the following equation:
\[ W^\text{new}_J = I \wedge W^\text{old}_J \]  

(97)

This results in previous weights being updated to encode the features of the present input exemplars.

Initialization of weights and selection of control parameters remain the same as described in Modified Fuzzy ARTMAP.
III. COMMUNICATION IN P-FAM

The hypercube connectivity is exploited during communication among different processors of the topology. All communications are designed to ensure dilation of one, i.e., two nodes involved in communication have to travel no more than a unit distance. This is one of the major goals in the design.

Communication in iPSC hypercube is performed by means of asynchronous message transfer through communication channels. Each node is completely independent and communicates with its neighbors by queued message passing.

In iPSC n-cube, the process of sending or receiving by a processor can be of non-blocked or blocked nature. If it is a blocked receive or blocked send, the processor sending the message waits for an acknowledgement from the receiver. Meanwhile the receiver waits for a message from the sender. If the communication is unblocked, the sender processor ensures that the message is put in the proper queue and continues with the next processing in line.

When a processor has a message to send, it follows the following sequence:

a) The processor writes the message in memory (RAM) and initiates the communication coprocessor.

b) The communication coprocessor reads the message from memory and sends it over to the serial link.

c) An interrupt is issued by the communication coprocessor to report to the processor that sent the message.

When a processor has a message to receive, it follows the following sequence:

a) The communication processor receives a message and writes into memory.
b) It interrupts the processor to give the address of the message.
c) The processor read the message.

All processors are connected to the host processors. This permits any processor to directly communicate with the host if needed, e.g., end of training when all processors have to transmit the network weights to the host processor. The communication between the host and the processors is the main bottleneck due to limited channel capacity.

The two main types of communication in P-FAM are:

A. Global Broadcast: For sending training samples to all processors of an n-cube/hypercube.

B. Global Evaluation: Calculation and communication of Maxima e.g., to calculate global maxima and transmit it to all the processor of hypercube.

A. Global Broadcast

At times it is necessary for one node in a hypercube machine to pass the same message to all the other nodes. One way would be to do $2^d - 1$ sequential sends, where $d$ is the dimension of hypercube. A more balanced approach would be to treat the n-cube/hypercube as a ring and send the message sequentially around the ring. A still better approach is to use a fan-out tree in which each node sends the message to all its neighbors that have not already received it. It is the fan-out tree approach that is implemented in this thesis so that each processor in the hypercube receives a copy of training input/output sample pair from node zero. The global broadcast is initiated with node zero as the root of fan out tree. Each node need not know which processor is the root. The only information it needs to know is who sent the message.
Assume that each node sends the message to its neighbors by increasing the node number. The tree starting at 0 is shown in Figure 6.4, for a 3 dimensional hypercube (d=3).

The neighbors of any processor P in a hypercube are of the form

\[ P \quad \oplus \quad 2^i \quad \text{for} \quad i = 0, 1, \ldots, d-1 \]  

(98)

Each node in the send algorithm needs to send message only to those neighbors satisfying

\[ P \quad \oplus \quad 2^j \quad \text{for} \quad 2^j > P \]  

Such that \( 2^j > P \)  

(99)

The tree rooted at any other processor R can be computed by taking the exclusive or (\( \oplus \)) of every node in tree with root zero, with new root R. The same tree can be obtained using Theorem 1[13] also.

**Theorem 1:** If the root R sends a message to every processor and each node P, which receives a message from Z, sends it to all neighbors, \( Y = P \quad \oplus \quad 2^i \) such that \( 2^i > P \quad \oplus \quad Z \). The resulting communication tree is the same as that obtained from the exclusive-or operation of each node with R of the tree rooted at zero. (See Figures 6.5 and 6.6).

In Figure 6.4. a fan-out tree of a 3-dimensional hypercube with zero as root is shown. The tree numbering corresponds to processor-id. The processors communicate only with their neighbors (actual physical serial link between nodes) and in \( O(\log N) = O(\text{dimension of cube}) \) all the nodes in the hypercube receive the message.
FIG 6.4  Global Broadcast - Embedded Tree.
Fig 6.5 Tree Obtained for Root = 7, Exclusive Oring by 7 each Node of a Fan-out Tree with Root = 0
Fig 6.6 Tree Obtained for Root = 7 (Theorem 1)
where $N$ is the total number of processors in the d-cube.

The processors participating in global broadcast in a fan out tree fashion are of three different types:

- Send data only (root node, e.g., 0).
- Send and receive data (internal nodes, e.g., 1, 2, 3).
- Receive data only (leaf nodes, e.g., 4, 5, 6, 7).

The node computes its neighbors and sends data only to them. This is in staying with one of the major design principle of keeping node-node distance as one, i.e., $\text{hop} = 0$. Leaf nodes are the ones with no neighbors to send data to. These nodes just receive the message from their parents.

Synchronization between various nodes communicating is ensured using block receive and block send. Figure 6.7 gives the network diagram of the communication involved in a global broadcast with node zero as the root node. The network diagram shows the communication among nodes at each level i (1, 2, 3) for a d (=3) level fan out tree.

**B. Global Evaluation**: Here the main concept is to evaluate a function globally and communicate the results to all the processors in the hypercube. The steps involved in evaluation and communication among the processors of the hypercube are shown in Figure 6.8.

The communication among nodes takes place between neighbors lying in different planes of the d-dimensional hypercube. The neighbors of any node X in a hypercube are of the form
**Fig 6.7** Global Broadcast, with Node 0 of the Cube as the Root of the Tree. (Network Diagram)
\[ x \oplus 2^i \quad \text{for } i = 0, 1, \ldots, d - 1 \quad (100) \]

where \( d \) is the dimension of the hypercube, for example

given \( X = 000 \) and \( d = 3 \) then

\[
\begin{align*}
X \oplus 2^0 &= 001 \\
X \oplus 2^1 &= 010 \\
X \oplus 2^2 &= 100
\end{align*}
\]

There are \( d \) steps in a \( d \)-dimensional hypercube for ensuring global evaluation and communication to all the nodes of the hypercube. The algorithm developed is explained in Figure 6.8 using 3 steps in a 3-dimensional hypercube. The evaluation function is used to calculate the maximum data in a cube. In P-FAM implementation, the local max Net from each processors are sent to its neighbors to calculate the global Net maximum. The algorithmic can be generalized for step \( i \) as follows:

The processors can be imagined to be lying in two hypothetical hyperplanes \((P_1^i \text{ and } P_2^i)\). All \( 2^d \) processors are involved in parallel communication between the two hypothetical hyperplanes. The two hyperplanes involved in processing have the following processors:

**Plane 1 \((P_1^i)\):** Processors with gray code representation having a zero in \( 2^i \) bit position, i.e,

\[
\begin{align*}
P_{1j}^i &= X \cap 2^i \\
P_{1j}^i \oplus 2^i &= 1
\end{align*}
\quad (101)
\]

where \( 1 \leq j \leq 2^{d-1} \) is an index of processor. For a 3-dimensional cube and step 0, the following processors qualify with process-id equal to 0,2,4,6 to be lying in plane 1.
FIG 6.8  Calculations and Comunications in Nodes of Hypercube for Global Maximum
Plane 2 \((P_2^i)\): Processors with gray code representation having a one in 
2\(^i\) bit position, i.e.,

\[
\begin{align*}
P_{2j}^i &= X \cap 2^i \\
\bigoplus P_{2j}^i 2^i &= 0
\end{align*}
\]

where \(1 \leq j \leq 2^{d-1}\) is an index of processor. For a 3-dimensional cube and step 0, the 
following processors qualify with process-id equal to 1,3,5,7.

The two hyperplanes each with \(2^{d-1}\) processors communicate to evaluate the given function (in our case calculating maxima). The two processors located in different 
hyperplanes but on the same vertex communicate. The two processors in different 
hyperplanes differing in only \(2^i\) bit position (i.e, same vertex on different hyperplanes) 
communicate and update their local maxima to the actual maxima between them, i.e.,

\[
\begin{align*}
P_{1j}^i &= Me \oplus 0 \\
P_{2k}^i &= Me \oplus 1
\end{align*}
\]

where \(1 \leq j \leq 2^{d-1}\) and \(1 \leq k \leq 2^{d-1}\).

For example, for step \(i = 0\), Table 6.1 lists the processors that communicate with each 
other to establish a maxima. The two processor communicate and get the resultant 
maximum among them. The two processors involved update their maximum to this new 
maximum. All the processors in one hyperplane get the maxima with respect to the other 
hyperplane in one step as all the \(2^d - 1\) processors find maximum in parallel.
TABLE 6.1
The Two Processors in Different Plane's but on the Same Vertex

<table>
<thead>
<tr>
<th>S. No</th>
<th>Processors in Plane 1</th>
<th>Processors in plane 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

In d steps in a d-dimensional hypercube the global maximum is evaluated and all the processors have a copy of the maximum with a corresponding processor-id and neuron-id. In the algorithmic implementation the processes are synchronized using blocked message communication. That is receive and send are applied in blocked mode. The processors involved in sending operations waits for an acknowledgement from the receiver node.

The network diagram shown in Figure 6.9 depicts the bi-directional communication among the processors in each step for a 3 dimensional hypercube. The gray code representation of process id is as shown in Figure 6.8. This ensures that all processes know the maximum Net simultaneously in d steps, where d is the dimensional of the cube. In P-FAM algorithm the global evaluation is used to find the global maximum Net among all the processors of the hypercube and hence letting each processor in turn know who is the probable winner. The information attached with each probable winner helps losers to know whether the process will be repeated with a new vigilance factor (match tracking) or the network will switch to the learning phase.
Fig 6.9 Communication Between Nodes of 3-cube during Global Maximum Calculations.
IV. ALGORITHMIC FLOW OF P-FAM

The P-FAM development is divided into three stages. Each module is specialized to do a specific function. The three stages (see Figure 6.10) are:

Testing : The paradigm could be used only for testing a pre-trained network configuration.

Training : A new network configuration is generated and the paradigm learns to predict the set of training exemplars.

Retraining : The network starts from a pre-trained configuration and learns to predict a new set of training exemplars. The weights from previous training are loaded as the initial adaptive weights.

Each of the above mentioned modules have a program that is executed in the host processor (elementary program) and each processor of the hypercube (main algorithm).

During training (see Figures 6.11 and 6.12) the host processor loads initial network configuration in all the processors of the n-cube. Each sample is read by the host processor and send to processor/node zero of the hypercube. The processor receives the network classified category from node zero along with processor-id 'p' and the neuron-id 'J' in processor 'p'. The network’s adaptive weights are received from each processor and stored by the host when the networks has concluded training.

The corresponding flow control in the cube explains the processing involved in receiving the training exemplar, predicting the output and performing learning if it is correctly predicted (see Figure 6.13 and 6.14). On correct prediction the network
FIG 6.10 Outline of P-FAM Algorithm
Training (HOST)

Load all Nodes of the Hypercube with Source Code

Load Network Configuration in all Nodes of Cube

Read Vigilance Factor

Input Choice Parameter

Input Learning Rate

HOST1

FIG 6.11 Host Flow-chart During Training in P-FAM (Continued)
FIG 6.12 Host Flow-chart During Training in P-FAM (Concluded)
Training (CUBE)

Allocate Memory to Generate network Dynamically

Receive Training Sample

Distribute Sample to all the Processors of Hypercube

Proc. 0
Call Process Net

Proc. 1
Call Process Net

Proc. 2
Call Process Net

Proc. N-1
Call Process Net

Find Global Net Max

Cube 0

Fig 6.13 Hypercube Flow-chart During Training in P-FAM (Continued)
Fig 6.14 Hypercube Flow-chart During Training in P-FAM (Concluded)
synchronizes with the host to receive the next sample, or sends back the network weights for each neuron in processor 'p' and terminates training.

The two main processing steps involved in training or retraining stages are the calculation of max net (see Figure 6.15) and the decision to do match tracking or learning (see Figure 6.16). In Figure 6.16 the points of synchronization are shown to highlight whether the networks will do learning or match tracking. During learning the losers initialize themselves and prepare for the next training exemplar. If match tracking is applied then the losers as well as the winner calculate the max net again with the new vigilance factor. During learning the winner modifies its long term memory traces, i.e., map weights \( W_j^M \) and Fuzzy ART interconnection weights \( W_j \).

The retraining mode (see Figure 6.17, 6.18, 6.19 and 6.20) is similar to training phase as described above with two major differences, i.e., the network is initialized with pre-trained adaptive weights and the network configuration is fixed. Retraining permits incremental learning. The initial weights are loaded in each neuron of processor 'p' in an \( d \)-dimensional hypercube in a sequence. Figure 6.20 details out the retraining process in all the 'n' neurons for \( 2^d \) processors. Where \( d \) is the dimension of the hypercube. The processes synchronize themselves after receiving the weights and before initiating any training.

The testing phase of P-FAM predicts an output for a given set of testing exemplars. The network configuration is fixed and is initially loaded by the host into each processor of the hypercube along with network weights (see Figure 6.21). Figure 6.22 shows the flow control involved in sending the testing exemplar to node zero of the
Process Net

Calculate Net for Each Neuron in a Processor

Find Maximum Net in a Processor

Inhibit Present Max Neuron

Satisfies Vigilance

Assign Local Maximum

Fig 6.15 Calculating Maximum Net in each Processor of Hypercube for P-FAM
Fig 6.16 Initiating Match Tracking or Learning in each Processor of Hypercube for P-FAM
Retraining
(HOST)

Load all Nodes of the Hyper cube with Source

Load Network Configuration in all Nodes of Cube

Read Vigilance Factor

Input Choice Parameter

Input Learning Rate

HOST-RT

FIG 6.17 Host Flow-chart During Retraining in P-FAM. (Continued)
FIG 6.18 Host Flow-chart During Retraining in P-FAM (Concluded)
FIG 6.19 Host Flow-chart During Retraining in P-FAM (Concluded)
Fig 6.20 Hypercube Flow-chart During Retraining in P-FAM
Process count = 0

Load all Nodes of the Hypercube with TEST Source Code

Load Network Configuration and Control Parameters in all Nodes of Cube

Process_count = 0

Read Neuron Weights for Processor = Process_count

Send Data to Processor/node of Hypercube

Increment Process_count

Yes

No

no. of processors > Process_count

TEST

FIG 6.21 Host Flow During Testing in P-FAM
FIG 6.22 Host Flow-chart During Testing in P-FAM (Concluded)
hypercube and receiving the output configuration along with processor-id 'p' and the neuron-id 'J' in processor 'p' for the winner.

Figures 6.23, 6.24, and 6.25 explain the steps involved during testing in predicting the winner for the given training exemplar. Testing is a portion of the training program. It is a simplified version of training as the network neither does learning or match tracking. The network calculates the maximum Net_J for a given probability estimate or vigilance factor (ρ). The associated map weights of the global winner neuron J form the predicted output. At the end of testing the network weights are not transmitted back to the host as in the case of training.

The three stages explained using flow chart are implemented on an n-cube boolean architecture. The network speed is a factor of its dimension with a constant overhead of communication. The communication overhead is dependent on parameters such as number of training samples or testing samples, the input and output dimensions of the network, and the number of processors available for the network to be built. In the limiting case where only one processor is available (i.e, one-dimensional hypercube) the network implemented becomes sequential in nature and is similar in operation as Fuzzy ARTMAP.

To reduce the communication overhead as much as possible, a look ahead scheme has been implemented in the paradigm. This ensures that neuron 'J' if selected will lead to match tracking/learning as the case may be. This look ahead is not an overhead in terms of processing. It is a boon in reducing communication and synchronizing overhead among processors of the n-cube. This ensures that along with the winner neuron, each processor also receive's information about match tracking/learning. And if match
Fig 6.23 Hypercube Flow-chart During Testing in P-FAM (Continued)
Distribute Sample to all the Processors of the Hypercube

Find Max ••• Net

Fig 6.24 Hypercube Flow-chart During Testing in P-FAM (Continue)
Fig 6.25 Hypercube Flow-chart During Testing in P-FAM (Concluded)
tracking is effective then the new vigilance factor can be easily calculated without any communication with the winner neuron 'J' in processor 'p'.

The data structure described in the header file to emphasize the implementation of the above mentioned look ahead is given as:

```c
struct used_for_communication
{
    float     max_data;
    float     weights_summation;
    int        mapping_flag;
    float     new-vigilance;
    int        node;
    int        neuron_in_node;
} MAX_NEURON;
```

The fields in the above structure describe the information for the neurons 'J' selected by processor 'p' as its maximum. Neuron 'J' must satisfy the Fuzzy ART vigilance factor \( \rho \) to qualify for competition for local maxima in processor 'p'. The fields contain the following information pertaining to the local winner neuron 'J' in processor 'p':

1. The value of max net \( \text{Net}_J \).
2. The summation of weights \( \text{W}_J \).
3. Since the input/output pair is given to each of the neurons in processor 'p' of an n-dimensional hypercube. A look ahead as mentioned in the neuro dynamics is performed to know whether match tracking or learning will take place in case neuron j is the winner. The mapping_flag is set in case the winner j satisfies the mapping rule for learning else the mapping_flag is reset to assert match tracking.
4. In case the mapping_flag is reset the new vigilance factor is calculated for a given input/output pair.

5. The last two entries store the neuron id 'J' and the processor id 'p'. The two identifications together form a unique name for the neuron. Incidentally this naming corresponds to the id had this network being implemented in a sequential fashion.
CHAPTER 7

APPLICATIONS USING P-FAM

I. P-FAM IN DETECTION OF TRENDS IN QUALITY CONTROL.

Statistical quality is a tool used in decision making related to specification, production, and inspection within a manufacturing environment. Quality control charts are means of representing the statistical variation in a manufacturing process. These control charts are a measure for reducing loss due to rework and scrap caused by an out of control process. Control charts are generally charts for attributes or variables. These are often used to find out the average values in the process, the variation in the process and the presence or absence of assignable causes of variation.

A general model of a control chart consists of an upper limit, lower limit and mean. The process is in control within the limits for a general case. The control rules learned by P-FAM broadly fall into the following models:

1) Downward Tending Models.
2) Steady Tending Models, i.e., models in control.
3) Upward Tending Models.

The idea behind using P-FAM is to eliminate human errors during plotting and analysis of control data.

The data samples used represent the process control charts used by quality control specialists. Here the data is a series of 10 points representing the random variation in a
manufacturing process[8]. Random points were generated and controlled to a degree in order to assure a trend in the proper direction (downward, upward, steady). The points were generated using Monte Carlo simulation techniques. The Monte Carlo simulation for both upward and downward trends was initialized using a uniformly distributed random process between the control limits. Initial upper trend points were less than 60% of the distance between the mean point to the upper limit. This helps the simulation from not generating trends which start very near the upward trend limit and generate samples rounded to the upper limit giving steady instead of upward trend. The remaining points were generated with a normal distribution with a mean of 0.05. The normal random variable generated is added to the previous sum to get the upward trend [8].

Down wards trends followed the steps in the opposite direction as compared to upward trend.

For steady trend the initial point generated using uniform distribution could start anywhere. The remaining point for steady trend were generated using a normal distribution with mean as zero. Here the previous values were not accumulated as was the case in the other two trends. However they were shifted on the initial point.

The points thus generated were between the limits [-0.5, 0.5] with a mean of 0.05. This data was scaled to satisfy the input condition of P-FAM, i.e, [0.0, 1.0]. These inputs were then complemented giving the input dimension of 20. There were 20 samples for each trend used to train the network. The network was then tested for 10 samples for each trend. The total number of samples used in training and testing was 90.
Tables 7.1, 7.2 and 7.3 show the training and testing output obtained using Parallel Fuzzy ARTMAP with vigilance factor of 0.6. Figure 7.1, 7.2, and 7.3 show some examples of the three trends used for testing or training the network. Figures 7.4, 7.5, 7.6 highlight the encoding of the data samples used for training the network.

Figure 7.4 shows encoding of 20 upward tending training exemplars into three categories. The weights also encode the complement portion of the training exemplars, i.e., both inhibitory and exciter cells are encoded. The upward trend effect is clear looking at the plot of the first 10 adaptive weights. The remaining adaptive weights are approximately complement. Similar observations can also be made for steady trend (see Figure 7.5) and downward trend (see Figure 7.6). The weights being true reflection of the training exemplar emphasize encoding of the input features and hence learn by experience.

Table 7.4 gives the summary of message traffic while training the network for 60 data samples, 20 for each trend. Table 7.5 summarizes the message traffic for 30 samples used during testing. The observation to note is that all the processors have more or less the same order of communication among neighboring processors. Another observation is that the average distance for node to node message is 1.0. These two observation were the main objectives in implementing the paradigm on a n-dimensional hypercube.
TABLE 7.1
Training Quality Control Data Using P-FAM (Continued)

<table>
<thead>
<tr>
<th>Serial Number</th>
<th>Node in Hypercube</th>
<th>Neuron in Node</th>
<th>Winner Neuron</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>22</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>24</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>27</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>29</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>30</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>
### TABLE 7.2
Training Quality Control Data Using P-FAM (Concluded)

<table>
<thead>
<tr>
<th>Serial Number</th>
<th>Node in Hypercube</th>
<th>Neuron in Node</th>
<th>Winner Neuron</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>32</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>33</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>34</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>35</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>36</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>37</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>38</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>39</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>40</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>41</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>42</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>43</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>44</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>45</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>46</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>47</td>
<td>3</td>
<td>0</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>48</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>49</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>50</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>51</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>52</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>53</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>54</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>55</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>56</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>57</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>58</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>59</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>60</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>Serial Number</td>
<td>Node in Hypercube</td>
<td>Neuron in Node</td>
<td>Winner Neuron</td>
<td>Category</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------</td>
<td>----------------</td>
<td>--------------</td>
<td>----------</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
<td>0</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>21</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>22</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>23</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>25</td>
<td>3</td>
<td>1</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>26</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>27</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>28</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>29</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Figure 7.1 Exemplars for Downward Trends
Figure 7.2 Exemplars of Steady Trends
FIG 7.4  L. T. M. Traces for Downward Trends
FIG 7.5 L.T.M. Traces for Steady Trends
FIG 7.6 L.T.M. Traces for Upward Trends
Table 7.4
Message Traffic Summary for Training 60 Samples for Quality Control Exemplars.

<table>
<thead>
<tr>
<th>Node</th>
<th># Msgs out</th>
<th># Msgs in</th>
<th># Msgs recd</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>137</td>
<td>152</td>
<td>152</td>
</tr>
<tr>
<td>0</td>
<td>1702</td>
<td>1218</td>
<td>1218</td>
</tr>
<tr>
<td>1</td>
<td>1461</td>
<td>1218</td>
<td>1218</td>
</tr>
<tr>
<td>2</td>
<td>1203</td>
<td>1081</td>
<td>1081</td>
</tr>
<tr>
<td>3</td>
<td>1203</td>
<td>1081</td>
<td>1081</td>
</tr>
<tr>
<td>4</td>
<td>945</td>
<td>944</td>
<td>944</td>
</tr>
<tr>
<td>5</td>
<td>945</td>
<td>944</td>
<td>944</td>
</tr>
<tr>
<td>6</td>
<td>945</td>
<td>944</td>
<td>944</td>
</tr>
<tr>
<td>7</td>
<td>945</td>
<td>944</td>
<td>944</td>
</tr>
<tr>
<td>8</td>
<td>687</td>
<td>807</td>
<td>807</td>
</tr>
<tr>
<td>9</td>
<td>687</td>
<td>807</td>
<td>807</td>
</tr>
<tr>
<td>10</td>
<td>687</td>
<td>807</td>
<td>807</td>
</tr>
<tr>
<td>11</td>
<td>687</td>
<td>807</td>
<td>807</td>
</tr>
<tr>
<td>12</td>
<td>687</td>
<td>807</td>
<td>807</td>
</tr>
<tr>
<td>13</td>
<td>687</td>
<td>807</td>
<td>807</td>
</tr>
<tr>
<td>14</td>
<td>687</td>
<td>807</td>
<td>807</td>
</tr>
<tr>
<td>15</td>
<td>687</td>
<td>807</td>
<td>807</td>
</tr>
</tbody>
</table>

Total # message sent: 14982

Average distance for node to node message: 1
Table 7.5
Message Traffic Summary for Testing 30 Samples
for Quality Control Exemplars.

<table>
<thead>
<tr>
<th>Node</th>
<th># Msgs out</th>
<th># Msgs in</th>
<th># Msgs recd</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>63</td>
<td>62</td>
<td>62</td>
</tr>
<tr>
<td>0</td>
<td>396</td>
<td>277</td>
<td>277</td>
</tr>
<tr>
<td>1</td>
<td>336</td>
<td>276</td>
<td>276</td>
</tr>
<tr>
<td>2</td>
<td>275</td>
<td>245</td>
<td>245</td>
</tr>
<tr>
<td>3</td>
<td>275</td>
<td>245</td>
<td>245</td>
</tr>
<tr>
<td>4</td>
<td>214</td>
<td>214</td>
<td>214</td>
</tr>
<tr>
<td>5</td>
<td>214</td>
<td>214</td>
<td>214</td>
</tr>
<tr>
<td>6</td>
<td>214</td>
<td>214</td>
<td>214</td>
</tr>
<tr>
<td>7</td>
<td>214</td>
<td>214</td>
<td>214</td>
</tr>
<tr>
<td>8</td>
<td>153</td>
<td>183</td>
<td>183</td>
</tr>
<tr>
<td>9</td>
<td>153</td>
<td>183</td>
<td>183</td>
</tr>
<tr>
<td>10</td>
<td>153</td>
<td>183</td>
<td>183</td>
</tr>
<tr>
<td>11</td>
<td>153</td>
<td>183</td>
<td>183</td>
</tr>
<tr>
<td>12</td>
<td>153</td>
<td>183</td>
<td>183</td>
</tr>
<tr>
<td>13</td>
<td>153</td>
<td>183</td>
<td>183</td>
</tr>
<tr>
<td>14</td>
<td>153</td>
<td>183</td>
<td>183</td>
</tr>
<tr>
<td>15</td>
<td>153</td>
<td>183</td>
<td>183</td>
</tr>
</tbody>
</table>

Total # message sent: 3425

Average distance for node to node message: 1
II. P-FAM IN MANUFACTURING.

Simulation modeling is a powerful experimental technique available for analysis in study of Flexible Manufacturing Systems (FMS). The complexity involved in FMS has precluded the use of an effective analytical solving technique. Simulation modelling has been utilized as a method to determine desirable variable values, and inter-relationships between the variable of the Flexible Manufacturing Systems under analysis. Simulations can be integrated with artificial neural network (e.g. P-FAM) in order to accelerate and improve the design and analysis process of FMSs [6].

Artificial Neural Networks are used to help step-wise refinement of an abstract conceptual model of a real-world system. The analysis is used to obtain a detailed functional model by supporting the structural and behavioral properties of the manufacturing systems under consideration. Traditional mathematical and formal models have the tendency to simplify complex models and therefore omitting important concepts.

Parallel Fuzzy ARTMAP (P-FAM) can handle categorization of both qualitative and quantitative data. It can process experimental data for finding functional and more realistic relationship beyond the common approach of a finding function. These in many cases do not represent at all times the concrete facts of the modelled systems.

An application of P-FAM in simulating a flexible manufacturing system is discussed. Specifically the use of P-FAM to speed up iterative design process and the responsiveness of on-line simulation are presented.
Simulation for design tasks involves the utilization of a long and iterative process. The example taken is concerned with 3 types of product manufactured at a Flexible Manufacturing System consisting of three cells. The example illustrates that P-FAM when used for large problems will be more effective during training than the sequential counterpart. It results in substantial reduction in processing time. The reduction in processing factor influences the processing time involved in training.

The example [6] used for training the P-FAM paradigm consists of three types of products manufactured at an FMS which consists of three cells, the possible configurations are:

CELL 1  Milling (1 to 3 machines);
CELL 2  Drilling (1 to 3 machines);
CELL 3  Turning (1 to 3 machines);

The routing for each product type is given in Table 7.6:

<table>
<thead>
<tr>
<th>Product Type</th>
<th>Process Plan</th>
<th>Processing Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Milling</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Drilling</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Turning</td>
<td>15</td>
</tr>
<tr>
<td>2</td>
<td>Drilling</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Milling</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>Turning</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>Milling</td>
<td>3</td>
</tr>
</tbody>
</table>
The above setup was simulated using SIMAN language. Statistics on performance measures such as Flowtime, Tardiness, Completion-time, Machine Utilization and Work-in-process were collected. There are $3^3$ possible configurations (i.e. 1-3 milling 1-3 drilling, 1-3 turning in each work center). Furthermore each dispatching rule adds an independent dimension and all the configurations have to be simulated for the respective rule. Two dispatching rules, EDD (Earliest Due Date) and SPT (Shortest processing Time) are used in the study.

A. Training Phase

Eighteen training exemplars or input-output vectors are generated using simulation. The training data is given in Table 7.7. All the input-output elements are normalized between 0 and 1 to train using Parallel Fuzzy ARTMAP and complemented. The training is Supervised i.e the training sample has both input and output pairs. Training is performed with the following adjustable control parameters till optimal results are obtained:

i) Fuzzy ART vigilance factor ($\rho$) [0,1].

ii) Map Field vigilance factor ($\rho_M$) [0,1], in the implementation it was set to 1.

iii) Choice factor ($\alpha$), in the implementation it was set to 0.0001.

iv) Beta ($\beta$) the learning rate is set to 1.

The network was able to generalize the training data using only one epoch (i.e presenting data samples only once).
Table 7.7

Training Samples and Performance Parameters

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>M D T R</td>
<td>F TT MT C</td>
</tr>
<tr>
<td>1 1 3 SPT</td>
<td>41.500 105.000 35.000 88.00</td>
</tr>
<tr>
<td>1 2 1 SPT</td>
<td>45.700 120.000 40.000 98.00</td>
</tr>
<tr>
<td>1 3 2 SPT</td>
<td>42.100 105.000 35.000 88.00</td>
</tr>
<tr>
<td>2 1 1 SPT</td>
<td>42.300 96.000 32.000 90.00</td>
</tr>
<tr>
<td>2 2 2 SPT</td>
<td>29.200 22.000 22.000 58.00</td>
</tr>
<tr>
<td>2 3 3 SPT</td>
<td>27.700 18.000 18.000 54.00</td>
</tr>
<tr>
<td>3 1 2 SPT</td>
<td>30.400 16.000 16.000 52.00</td>
</tr>
<tr>
<td>3 2 3 SPT</td>
<td>24.200 7.000 7.000 43.00</td>
</tr>
<tr>
<td>3 3 1 SPT</td>
<td>39.700 66.000 32.000 90.00</td>
</tr>
<tr>
<td>1 1 3 EDD</td>
<td>47.100 91.000 13.000 88.00</td>
</tr>
<tr>
<td>1 2 1 EDD</td>
<td>52.300 122.003 17.429 90.00</td>
</tr>
<tr>
<td>1 3 2 EDD</td>
<td>46.500 91.000 13.000 88.00</td>
</tr>
<tr>
<td>2 1 1 EDD</td>
<td>44.300 99.000 24.750 90.00</td>
</tr>
<tr>
<td>2 2 2 EDD</td>
<td>30.100 8.000 4.000 55.00</td>
</tr>
<tr>
<td>2 3 3 EDD</td>
<td>28.800 6.000 3.000 55.00</td>
</tr>
<tr>
<td>3 1 2 EDD</td>
<td>30.400 13.000 13.000 52.00</td>
</tr>
<tr>
<td>3 2 3 EDD</td>
<td>25.200 7.000 7.000 43.00</td>
</tr>
<tr>
<td>3 3 1 EDD</td>
<td>41.400 99.000 24.750 90.00</td>
</tr>
</tbody>
</table>

M = milling,
D = drilling,
R = dispatching rule,
F = flowtime,
TT = total tardiness,
MT = mean tardiness,
C = completion time.

B. Testing Phase

In the testing phase the trained neural network is fed with the input pattern, the output of the network is compared with the simulation outputs to determine the generalization capability of the P-FAM.

The testing mode of Parallel Fuzzy ARTMAP simulator was used to generate the networks response to the testing samples presented. The output generated by the network was compared to the actual output generated using simulation, for statistical purposes. (Table 7.8).
Table 7.8
Results Obtained Using Parallel Fuzzy ARTMAP ($\rho_* = [0.5,0.6]$)

<table>
<thead>
<tr>
<th>Performance Measures</th>
<th>Total Number of Samples</th>
<th># of Correct Responses</th>
<th>% Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLOW TIME</td>
<td>36</td>
<td>36</td>
<td>100</td>
</tr>
<tr>
<td>TOTAL TARDINESS</td>
<td>36</td>
<td>33</td>
<td>91.67</td>
</tr>
<tr>
<td>MEAN TARDINESS</td>
<td>36</td>
<td>29</td>
<td>80.55</td>
</tr>
<tr>
<td>COMPLETION TIME</td>
<td>36</td>
<td>36</td>
<td>100</td>
</tr>
</tbody>
</table>
III. P-FAM IN SIGNAL PROCESSING.

Parallel Fuzzy ARTMAP can also be used in signal processing applications such as monitoring nuclear reactor signals for diagnostic purposes[5]. The present technique for monitoring these conditions is very complex and takes a lot of time. With the introduction of neural network paradigms like Parallel Fuzzy ARTMAP, problems like curve fitting, system diagnosis using signal analysis, mine detection, monitoring enemy military weaponry, etc., can be used in real time. The signals will be collected from site in real time and after preprocessing fed into the neural networks for categorization.

In Nuclear power plants in order to ensure safe operation, power plants are designed with a large number of sensors of various kinds to monitor reactor condition/parameters at all times. Signals could be collected for early detection of degradation of any critical device, e.g., pumps shafts. In this study the objective is to find the effectiveness of utilizing P-FAM for monitoring applications taking real time signals as inputs. As an example P-FAM is tested for its capability of detecting and recognizing different levels of degradation apart. Due to the large number of pattern variations, incremental learning is desired. P-FAM is selected due to its capability of handling non-stationary stochastic signals, and ability to learn in real time [5].

The signals utilized here consist of 4 samples of 250 data points for each level of degradation, i.e., normal, first level, and second level of degradation. The network was
trained with 12 data samples, 4 for each level of degradation. For testing 20 - 40 % of the signals were distorted by adding noise or by grounding the signals. 60 data samples were generated with noise in the original signals. The network tested had recall performance of 100 %. The learning time was in order of milliseconds. Table 7.9 gives the categorization of the twelve signals using Parallel Fuzzy ARTMAP.

**TABLE 7.9**

Training Nuclear Reactor Pump Signals Using P-FAM

<table>
<thead>
<tr>
<th>Serial Number</th>
<th>Node in Hypercube</th>
<th>Neuron in Node</th>
<th>Winner Neuron</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Since only one neuron exits per processor the winner neuron and the node in the processor have the same identification.
CHAPTER 8

CONCLUSION

The properties of Fuzzy ARTMAP such as stability plasticity dilemma, full memory capacity utilization, incremental learning, analog inputs, etc., are all preserved in its parallel implementation, i.e., P-FAM (Parallel Fuzzy ARTMAP).

PFAM was divided in a manner to ensure uniform distribution of neurons in each processor of the hypercube. The processors communicate to transmit the global maximum and do weight update in the algorithm. At the end of training, the processors communicate with the host to transmit the networks's configuration and results. Various routines are built to ensure synchronization among processors of the hypercube. The communication can be a bottleneck if the implementation of any algorithm does not take it into consideration in design stages. In PFAM design, communication takes place only with its neighbors, i.e., nodes that are physically connected.

The implementation of the algorithm has two phases, i.e., training and testing. The purpose of the training phase is to do learning in parallel. The testing phase is responsible for building the network and testing the paradigm for a given set of examples.

The generality of PFAM algorithm, makes it very easy for one to implement the architecture on any type of parallel machine e.g., Torus, Butterfly, Ring, Tree, etc. The major difference lies in implementing the communication among various processors. The approach used in distribution of fuzzy ARTMAP on a hypercube, i.e., PFAM can easily
be adapted to implement other Adaptive Resonance Theory based paradigms, i.e., ART-1, ARTMAP, Fuzzy-ART, etc.

PFAM was trained and tested for numerous applications, i.e., signal processing, manufacturing and scheduling, quality control, protein analysis, and robotics (path planning). Some of the applications are discussed in chapter 7.

If the number of processors are less than the total number of neurons, then the algorithm behaves as sequential implementation of Fuzzy ARTMAP. (The process id is the category id)

The present implementation is not limited to 16 processors. Systems call are used to get the hypercube topology. The configuration is used to divide the total number of neurons equally among all the processors of the hypercube.

The detailed discussion of quality control charts application explains very clearly how to interpret different aspects of the paradigm. The weights, (long term memory traces) records the examples presented during training. The different traces of a particular trend basically show that the network has learned from a given set of training exemplars.

The vigilance factor or the probability estimate factor decides the total number of categories to be created. Fuzzy operators use help in approximate reasoning. The results obtained were the same as the sequential implementation of Fuzzy ARTMAP.

The thesis is a step towards computers that learn like humans. In many neural network simulators training time is a bottleneck, but in PFAM it becomes a trivial issue.

PFAM can be easily adapted to any application with the additional advantage of reduced training time. Secondly PFAM has a potential in real time application where
testing and training has to be done in real time. Such networks will learn unknown exemplars in the given environment. Examples of real time application include controlling chemical plants, nuclear reactors, underwater sonar detection, etc. To summarize PFAM has its application both in civil and defense areas.

The results of the research conclude that future work should concentrate more on implementing other ART based paradigms such as Fuzzy ART, ART-2, ART-2A, and ART-3 on numerous parallel architectures. To emphasize, it is very important to recognize area of parallelism in these paradigms. These regions of parallelism directly influence the overall algorithm implementation on any parallel machine.
REFERENCES


a Robotic Manipulator Using Neural Networks," to appear in proc. of First IEEE 

Statistical Control Charts," to appear in proc. of Fifth Conference on 
Neural Networks and Parallel Distributed Processing, 1992.


Presented at IJCNN 92, 1992.

[12] F. T. Leighton, "Hypercubes and Related Networks," in Introduction to Parallel 
Kaufmann, 1992, pp 389-784


[16] 80170x experimental data sheets from Intel Corporation.

Italian Workshop on Parallel Architectures and Neural Networks, E. R. Cainiello, 


