This document is an author-formatted work. The definitive version for citation appears as:

J. Castro, M. Georgiopoulos, J. Secretan, R. F. DeMara, G. Anagnostopoulos, and A. J. Gonzalez, "Parallelization of Fuzzy ARTMAP to Improve its Convergence Speed: The Network Partitioning Approach and the Data Partitioning Approach," accepted to *Nonlinear Analysis: Theory, Methods, and Applications*, Vol. 60, No. 8, June, 2005, to appear.

Parallelization of Fuzzy ARTMAP to Improve its Convergence Speed: The Network Partitioning Approach and the Data Partitioning Approach

José Castro, Michael Georgiopoulos, Jimmy Secretan, Ronald F. DeMara, Georgios Anagnostopoulos, Avelino Gonzalez

***Abstract*—One of the properties of FAM, which can be both an asset and a liability, is its capacity to produce new neurons (templates) on demand to represent classification categories. This property allows FAM to automatically adapt to the database without having to arbitrarily specify network structure. We provide two methods for speeding up the FAM algorithm, the first one referred to as the *Data Partitioning* partitions the data into subsets for independent processing. The second one refered to as the *Network partitioning* approach uses a pipeline to distribute the work between processes during training. We provide experimental results on a Beowulf cluster of workstations for both approaches that confirm the merit of the modifications.**

***Index Terms*—Fuzzy ARTMAP, BEOWULF parallel processing, Data Partitioning, Network Partitioning.**

J. C. Author is in the University of Central Florida, Department of Electrical and Computer Engineering, Orlando, FL 32816–2786 USA.

M. G. Author, is in the University of Central Florida, Department of Electrical and Computer Engineering, Orlando, FL 32816–2786 USA.

J. S. Author, is in the University of Central Florida, Department of Electrical and Computer Engineering, Orlando, FL 32816–2786 USA.

R. F. D. Author, is in the University of Central Florida, Department of Electrical and Computer Engineering, Orlando, FL 32816–2786 USA.

G. A. Author is in the Florida Institute of Technology in the Department of Electrical and Computer Engineering.

A. G. Author is in the University of Central Florida, Department of Electrical and Computer Engineering, Orlando, FL 32816–2786 USA.

The authors would like to thank the Institute of Simulation and Training and the Link Foundation Fellowship program for partially funding this project. This work was also supported in part by the National Science Foundation under grant no. 0203446.

I. Introduction

NEURAL Networks have been used extensively and successfully to address a wide variety of problems. As computing capacity and electronic databases grow, there is an increasing need to process considerably larger datasets. In this context, the algorithms of choice tend to be ad--hoc algorithms or tree based algorithms such as CART and C4.5 [6]. Variations of these tree learning algorithms, such as SPRINT (Shafer, et al., [7]) and SLIQ (Mehta, et al., [4]) have been successfully adapted to handle very large data sets.

Neural network algorithms have, for some applications, prohibitively high convergence times. Even one of the fastest neural network algorithms, the Fuzzy ARTMAP (FAM) algorithm, tends to lag in convergence time as the size of the network grows. The FAM algorithm corresponds to a family of neural network architectures introduced by Carpenter, et al., 1991-1992 [2][3] and has proven to be one of the premier neural network architectures for classification problems.

Some of the advantages that FAM has, compared to other neural network classifiers are that it learns the required task fast, it has the capability to do on-line learning, and its learning structure allows the explanation of the answers that the neural network produces.

One of FAM's properties which is a mixed blessing, is its capacity to produce new neurons (*templates*) on demand to represent classification categories. This property allows FAM to automatically adapt to the database without having to arbitrarily and a-priori specify its network structure, but it also has the undesirable side effect that on large databases it can produce a large network size that can dramatically slow down the algorithm's training time. It would be desirable to have a method capable of keeping FAM's convergence time manageable, without affecting the generalization performance of the network or its resulting size when the training is completed.

In this paper we propose two partitioning approaches for the FAM algorithm to be used in a parallel setting. The *Network partitioning* approach and the *Data partitioning* approach. Our research on Network and Data partitioning for FAM has shown that they it dramatically reduce the training time of FAM by partitioning the training set without a significant effect on the classification (generalization) performance of the network. We also propose a pipelined FAM implementation that speeds up the algorithm linearly with respect to the number of processors used in the pipeline.

This paper is organized as follows: First we review the Fuzzy ARTMAP architecture and functionality, then we examine the computational complexity of FAM and analyze how and why a data partitioning and network partitioning approach can considerably reduce the algorithm's training time. After that we discuss our implementation of data partitioning FAM and provide a network partitioning parallel pipelined algorithm for FAM. we also provide some proofs of correctness on the network partitioning pipelined FAM approach. Furthermore, experimental results are presented on a Beowulf cluster of workstations that illustrate the merit of our approach. We close the paper with a summary of the findings and suggestions for further research.

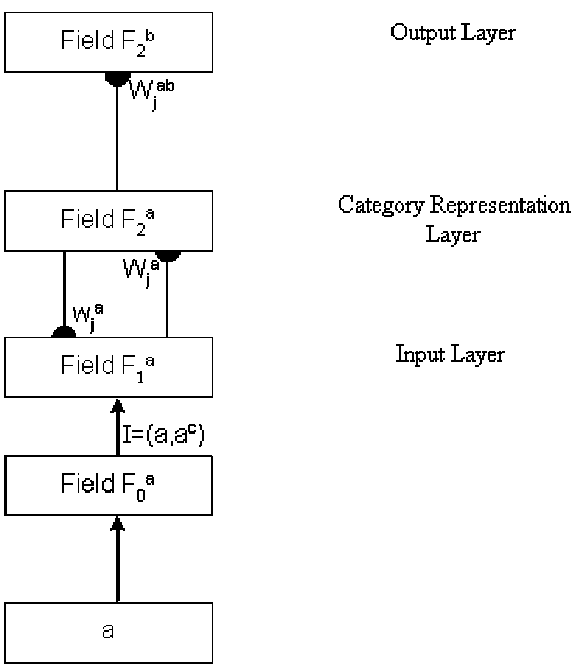

Fig. 1. Fuzzy ARTMAP diagram

II. Fuzzy ARTMAP

A. The Fuzzy ARTMAP Architecture

The Fuzzy ARTMAP architecture consists of four layers or fields of nodes (see Figure 1). The layers that are worth describing are the *input layer* F_1^a, the *category representation* layer F_2^a, and the *output layer* F_2^b. The input layer of Fuzzy ARTMAP is the layer where an input vector of dimensionality $2M_a$ of the following form is applied

$$\mathbf{I} = (\mathbf{a}, \mathbf{a}^c) = (a_1, a_2, \ldots, a_{M_a}, a_a^c, a_2^c, \ldots, a_{M_a}^c) \quad (1)$$

$$a_i^c = 1 - a_i; \forall i \in \{1, 2, \ldots, M_a\} \quad (2)$$

The assumption here is that the input vector $\boldsymbol{a}$ is such that each one of its components lies in the interval [0,1].

The layer F_2^a of Fuzzy ARTMAP is referred to as the *category representation layer*, because this is where categories (or groups) of input patterns are formed. Finally, the output layer is the layer that produces the outputs of the network. An output of the network represents the output to which the input applied at the input layer of FAM is supposed to be mapped to.

There are two sets of weights worth mentioning in FAM. The first set of weights are weights from F_2^a to F_1^a, designated as $\mathbf{w}_{ij}^a$, $1 \le j \le N_a, 1 \le i \le M_a$, and referred to as top-down weights. The vector of weights:

$$\mathbf{w}_j^a = (w_{j1}^a, w_{j2}^a, \cdots, w_{j,2M_a}^a)$$

is called a *template*. Its functionality is to represent the group of input patterns that chose node *j* in the category representation layer of Fuzzy ARTMAP as their representative node. The second set of weights, worth mentioning, are weights that emanate from every node *j* in the category representation layer to every node *k* in the output layer. These weights are designated as W_{jk}^{ab} (called inter--ART weights). The vector of inter--ART weights emanating from every node *j* in Fuzzy ARTMAP:

$$\mathbf{W}_j^{ab} = \left(W_{j1}^{ab}, W_{j2}^{ab}, \cdots, W_{j,N_b}^{ab}\right)$$

corresponds to the output pattern that this node j is mapped to.

Fuzzy ARTMAP can operate in two distinct phases: the *training phase* and the *performance phase*. The training phase of Fuzzy ARTMAP can be described as follows: Given a list of input/output pairs,

$$\left\{\left(\mathbf{I}^1, \mathbf{O}^1\right), \left(\mathbf{I}^2, \mathbf{O}^2\right), \ldots, \left(\mathbf{I}^P, \mathbf{O}^P\right)\right\}$$

we want to train Fuzzy ARTMAP to map every input pattern of the training list to its corresponding output pattern. To achieve the aforementioned goal we present the training list to Fuzzy ARTMAP architecture repeatedly. That is, we present $\mathbf{I}^1$ to F_1^a, $\mathbf{O}^1$ to F_2^b, $\mathbf{I}^2$ to F_1^a, $\mathbf{O}^2$ to F_2^b, and finally $\mathbf{I}^P$ to F_1^a, and $\mathbf{O}^P$ to F_2^b. We present the training list to Fuzzy ARTMAP as many times as it is necessary for Fuzzy ARTMAP to correctly classify all these input patterns. The task is considered accomplished (i.e., the learning is complete) when the weights do not change during a list presentation. The aforementioned training scenario is called *off–line* learning. The performance phase of Fuzzy ARTMAP works as follows: Given a list of input patterns, such as $\widetilde{\mathbf{I}}^1, \widetilde{\mathbf{I}}^2, \cdots, \widetilde{\mathbf{I}}^S$, we want to find the Fuzzy ARTMAP output produced when each one of the aforementioned test patterns is presented at its F_1^a layer. In order to achieve the aforementioned goal we present the test list to the trained Fuzzy ARTMAP architecture and we observe the network's output.

The operation of Fuzzy ARTMAP is affected by two network parameters, the choice parameter β_a, and the baseline vigilance parameter $\bar{\rho}_a$. The choice parameter takes values in the interval $(0,\infty)$, while the baseline vigilance parameter assumes values in the interval [0,1]. Both of these parameters affect the number of nodes created in the category representation layer of Fuzzy ARTMAP. Higher values of β_a and $\bar{\rho}_a$ create more nodes in the category representation layer of Fuzzy ARTMAP, and consequently produce less compression of the input patterns. There are two other network parameter values in Fuzzy ARTMAP that are worth mentioning. The vigilance parameter ρ_a, and the number of nodes N_a in the category representation layer of Fuzzy ARTMAP. The vigilance parameter ρ_a takes value in the interval $[\bar{\rho}_a, 1]$ and its initial value is set to be equal to $\bar{\rho}_a$. The number of nodes N_a in the category representation layer of Fuzzy ARTMAP increases while training the network and corresponds to the number of committed nodes in Fuzzy ARTMAP plus one uncommitted node..

B. The Fuzzy ARTMAP Learning Algorithm

Prior to initiating the training phase of Fuzzy ARTMAP the top–down weights (the w_{ij}^a's) are chosen equal to 1, and the inter–ART weights (the W_{jk}^{ab}'s) are chosen equal to 0. There are three major operations that take place during the presentation of a training input/output pair (e.g., $(\mathbf{I}^r, \mathbf{O}^r)$) to Fuzzy ARTMAP. One of the specific operands involved in all of these operations is the *fuzzy min operand*, designated by the symbol $\wedge$. Actually, the fuzzy min operation of two vectors $\mathbf{x}$, and $\mathbf{y}$, designated as $\mathbf{x} \wedge \mathbf{y}$, is a vector whose components are equal to the minimum of components of $\mathbf{x}$ and $\mathbf{y}$. Another specific operand involved in these equations is designated by the symbol $|\bullet|$. In particular, $|\mathbf{x}|$ is the size of a vector $\mathbf{x}$ and is defined to be the sum of its components.

Operation 1: Calculation of bottom up inputs to every node j in F_2^a, as follows:

$$T_j^a = \frac{\left|\mathbf{I}^r \wedge \mathbf{w}_j^a\right|}{\left|\mathbf{w}_j^a\right| + \beta_a} \qquad (3)$$

after calculation of the bottom up inputs the node $j_{\max}$ with the maximum bottom up input is chosen.

Operation 2: The node $j_{\max}$ with the maximum bottom up input is examined to determine whether it passes the vigilance criterion. A node passes the vigilance criterion if the following condition is met:

$$\frac{\left|\mathbf{I}^r \wedge \mathbf{w}_j^a\right|}{\left|\mathbf{I}^r\right|} \geq \rho_a \qquad (4)$$

if the vigilance criterion is satisfied we proceed with operation 3 otherwise node j_{max} is disqualified and we find the next node in sequence in F_2^a that maximizes the bottom up input. Eventually we will end up with a node $j_{\max}$ that maximizes the bottom up input and passes the vigilance criterion.

Operation 3: This operation is implemented only after we have found a node $j_{\max}$ that maximizes the bottom-up input of the remaining nodes in competition and that passes the vigilance criterion. Operation 3 determines whether this node $j_{\max}$ passes the prediction test. The prediction test checks if the inter--ART weight vector emanating from node $j_{\max}$ ie. $\mathbf{W}_{j_{\max}}^{ab} = \left(W_{j_{\max}1}^{ab}, W_{j_{\max}2}^{ab}, \cdots, W_{j_{\max},N_b}^{ab}\right)$ matches exactly the desired output vector $\mathbf{O}^r$ (if it does this is referred to as *passing the prediction test*). If the node does *not* pass the prediction test, the vigilance parameter ρ_a is increased to the level of

$$\rho_a \leftarrow \frac{\left|\mathbf{I}^r \wedge \mathbf{w}_{j\max}^a\right|}{\left|\mathbf{I}^r\right|} + \varepsilon$$

where ε is a very small number, node $j_{\max}$ is disqualified, and the next in sequence node that maximizes the bottom-up input and passes the vigilance is chosen. If, on the other hand, node $j_{\max}$ passes the predictability test, the weights in Fuzzy ARTMAP are modified as follows:

$$\mathbf{w}_{j_{\max}}^a \leftarrow \mathbf{w}_{j_{\max}}^a \wedge \mathbf{I}^r, \mathbf{W}_{j_{\max}}^{ab} \leftarrow \mathbf{O}^r \qquad (5)$$

Fuzzy ARTMAP training is considered complete if and only if after repeated presentations of all training input/output pairs to the network, where Operations 1-3 are recursively applied for every input/output pair, we find ourselves in a situation where a complete cycle through all the input/output pairs produced no weight changes. In some databases noise in the data may create over-fitting when we repeat the presentations of the input/output pairs, so a single pass over the training set may be preferable, this situation also happened when we do *online* training of the network with an unlimited data source.

In the performance phase of Fuzzy ARTMAP only Operations 1 and 2 are implemented for every input pattern presented to Fuzzy ARTMAP. By registering the network output to every test input presented to Fuzzy ARTMAP, and by comparing it to the desired output we can calculate the network's performance (i.e., network's misclassification error).

III. Fuzzy ARTMAP Time Complexity Analysis

A. *Online Fuzzy ARTMAP Time Complexity*

The pseudocode for the FAM algorithm can be presented is shown in figure 2. The FAM algorithm tests every input pattern **I** in the training set against each template $\mathbf{w}_j^a$ at least once. Let us call Γ the average number of times that the inner searching loop is executed for each input pattern, and christen it the *matchtracking factor*. Then the number of times that a given input pattern **I** passes through the code will be:

$$Time(\mathbf{I}) = O(\Gamma \times |templates|) \tag{6}$$

If we assume that the number of templates does not change during training it is easy to see that the time complexity of the algorithm is:

$$Time(FAM) = O(\Gamma \times P \times |templates|) \tag{7}$$

For a fixed type of database the FAM algorithm achieves a certain *compression ratio*. This means that the number of templates created is actually a fraction of the number of patterns P in the training set. Let us call this compression ratio κ so that:

$$|templates| = \kappa P \tag{8}$$

and its time complexity is given by the formula

$$O(FAM) = O(\Gamma P \kappa P) = O(\kappa \Gamma P^2) \tag{9}$$

```
FAM-LEARNING-PHASE(Patterns, ρ̄a, βa, maxEpochs)
  templates ← {}
  iter ← 0
  repeat
        modified ← false
        for each I in Patterns
        do ρa ← ρ̄a
            LEARN-PATTERN(I, templates, ρa, βa)
            iterations ← iterations + 1

  until (iter = maxEpochs) or (modified = false)
  return templates
Where the procedure LEARN-PATTERN is:
LEARN-PATTERN(I, templates, ρa, βa)
  repeat
        status ← FoundNone
        jmax ← GET-MAX-ARG(I, templates, ρa, βa)
        if status = FoundOne
          then if class(I) = class(w^a_jmax)
                then
                      status ← ThisIsIT
                else
                      status ← Matchtracking
                      ρ ← ρ(I, w^a_jmax) + ε
  until status ≠ Matchtracking
```

Fig.2. Fuzzy ARTMAP algorithm

B. *Off–line Fuzzy ARTMAP Time Complexity*

Without any previous assumptions the complexity of the Off–line FAM algorithm would be:

$$O\left(\underbrace{\sum_{i=1}^{epochs} \kappa_i \Gamma_i P^2}_{A}\right) \tag{10}$$

where Γ_i is the matchtracking factor for the i^{th} epoch, and κ_i is the compression ratio of the i^{th} epoch.

1) *Off–line Fuzzy ARTMAP Time Complexity Simplification*

We can simplify A by making the reasonable assumptions than a) the number of epochs is dependent on the number of training patterns, and b) Γ_i does not vary from iteration to iteration but is a linear function of the compression ratio κ_i and c) $\kappa_i = \kappa^i$, which basically means that the amount of templates created decreases geometrically from iteration to iteration. Using these assumptions we can state the formula as:

$$A = \sum_{i=1}^{epochs} \kappa^i \Gamma P^2 = \Gamma P^2 \sum_{i=1}^{epochs} \kappa^i = \Gamma P^2 \kappa \left(\frac{1-\kappa^{epocchs(P)}}{1-\kappa} \right) \tag{11}$$

We can develop a stopping criteria for the algorithm based on the previous formula, which this leads us to the equation

$$A = \kappa\Gamma P^2 \left(\frac{1-\kappa^{\left\lceil -\frac{\ln(P)}{\ln(\kappa)} \right\rceil}}{1-\kappa} \right) \quad (12)$$

and a number of templates given by the equation

$$\kappa P \left(\frac{1-\kappa^{\left\lceil -\frac{\ln(P)}{\ln(\kappa)} \right\rceil}}{1-\kappa} \right) \quad (13)$$

We can see from this formula that the convergence of the FAM algorithm is quadratic with respect to \$P\$, the number of patterns in the training set. This relationship suggests that it may be beneficial to use methods that reduce the size of the training set in FAM to speed up the convergence of the algorithm, and that by doing so we may get a quadratic speedup on the time complexity of the algorithm. The following sections are motivated by this partitioning, one reduces the parameter P by partitioning the input set of FAM, the other reduces the number of templates κP by doing network partitioning.

IV. Partitioned FAM

A. Data Partitioned Boxing FAM

We chose to follow a data partitioning approach that allows to reduce the number of patterns presented to FAM, and consequently the number of templates created in FAM. Through the data-partitioning scheme that we are proposing the dataset is split into smaller datasets, each one of which trains a different FAM network. This approach has the advantage that it can reduce the computational complexity of the algorithm and lends itself well to parallel implementation.

Under the fairly reasonable assumption that the number of templates in the ART neural network is proportional to the number of training patterns we can state that the ART convergence time is quadratic $O(n^2)$ where n is the number of patterns in the data set, partitioning the data set will result in a significant improvement. For instance, if we divide the data set into p equal partitions of size n/p, the partitioning will theoretically produce a speedup in the order of p^2. So we should theoretically expect a quadratic improvement in speed as we increase the number of partitions in the data set.

Our approach is inspired by the projective clustering approach proposed and implemented in [5], and by the natural properties of the FAM algorithm. We project the data contained in the M_a–dimensional hypercube to $\hat{M}_a$ dimensions, where $\hat{M}_a << M_a$, and we partition the data set by a grid in this projected space. If, for example, we use $\hat{M}_a = 3$ dimensions and a grid size of 10 divisions per dimension, we would divide the 3-dimensional space, on which we projected the original data, into 1000 boxes of side length equal to 0.1. If each set of data within a box trains a different FAM architecture (partitioned-FAM) we are guaranteed that the number of templates created by each such FAM is not going to be large (especially if the number of boxes is large). It is likely though that total number of templates created by the partitioned FAM is larger than the number of templates created by the non-partitioned FAM. Classification performance may be negatively affected by this partitioning approach. To avoid this unwelcome side effect we take advantage of the natural partitioning imposed by the FAM algorithm. We know that templates in FAM are geometrically represented by hyper-rectangles. Furthermore each hyper-rectangle size is restricted by the vigilance parameter $\bar{\rho}_a$ and by the dimensionality M_a of the input patterns. In particular:

$$size(\mathbf{w}_j^a) \le M_a(1-\bar{\rho}_a) \quad (14)$$

If we assume that templates grow more or less evenly across every dimension, then by choosing boxes in the projected space of size $(1-\bar{\rho}_a)$ across every dimension, we are actually restricting the template size to the size that the FAM algorithm already enforces. This partitioning approach is most effective when the value of the vigilance parameter is large, and this is the case when the number of templates grows the most, and tends to slow down the training of the algorithm.

B. Network Partitioned Pipelined FAM

In this second approach we chose to do network partitioning to bring down the complexity of the FAM algorithm, this requires the use of a pipeline of processors which exchange input patterns and templates through the pipeline. This is depicted in figure 3.

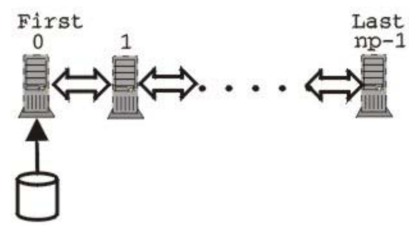

Fig. 3. Pipeline Structure

One of the problems presented in pipelining the FAM algorithm is it's use of matchtracking: a condition that if met induces the network to retrain a pattern and compare it again to al the templates in the network. This situation poses serious issues to a pipeline design, since when a pattern incurs in matchtracking we must flush all the elements of the pipeline to guarantee equivalence to the sequential algorithm. Obviously this approach is inefficient, so we opted by implementing a variant of FAM called no-matchtracking FAM developed by Anagnostopoulos [1]. The modified algorithm is depicted in figure [4].

When dividing the network we must guarantee that certain consistency properties hold. These properties have been proven formally and we will state them here as theorems.

Non--Duplication: *A template* **w** *will either be owned by a* *single processor, or it will be in transit on a single processor (i.e. only one copy of the template exists in the system).*

Bundel Size: *The excess of templates for a process* $k \neq 0$, *at any given time, always fits in the packet size 2p to be sent back.*

No overflow: *The first processor in the pipeline can always absorb all templates that are sent back to it.*

Bounded Delay: *The templates that on the current iteration a process k has were created at least n-k-1 iterations ago where n is the total number of processors in the pipeline.*

Pipeline Depth Invariance: *The difference in the number of templates that 2 arbitrary processes in the pipeline have cannot exceed p+1 where p is the packet size.*

FAM-NO-MATCHTRACKING-LEARNING($\{\mathbf{I}^1, \ldots, \mathbf{I}^P\}, \rho_a, \beta_a$)

$\mathbf{w}_0 \leftarrow \underbrace{(1, 1, \ldots, 1)}_{2M_a}$

templates $\leftarrow \{\mathbf{w}_0\}$

for each $\mathbf{I}^r$ **in** $\{\mathbf{I}^1, \mathbf{I}^2, \ldots, \mathbf{I}^P\}$

do $T_{max} \leftarrow 0$

  $\mathbf{w}_{max} \leftarrow$ none

  for each $\mathbf{w}_j^a$ **in** *templates*

  do if $[\rho(\mathbf{I}^r, \mathbf{w}_j^a) \geq \rho_a]$ & $[T(\mathbf{I}^r, \mathbf{w}_j^a, \beta_a) > T_{max}]$

    then

      $T_{max} \leftarrow T(\mathbf{I}^r, \mathbf{w}_j^a, \beta_a)$

      $\mathbf{w}_{max}^a \leftarrow \mathbf{w}_j^a$

      if $\mathbf{w}_{max}^a \neq \mathbf{w}_0$ & $class(\mathbf{I}^r) = class(\mathbf{w}_{max}^a)$

        then $\mathbf{w}_{j_{max}} \leftarrow \mathbf{w}_{j_{max}} \wedge \mathbf{I}^r$

        else *templates* $\leftarrow$ *templates* $\cup \{\mathbf{I}^r\}$

return *templates*

Fig. 4. Anagnostopoulos *No–Matchtracking* FAM

V. Experimental Results

To prove the feasibility of our approach we conducted a series of tests on the `Covertype` database by Jock A. Blackard [8]. The size of the data used for training was increased by a factor of 2 starting from 1000 data points and ending with 512,000. Classification performance was evaluated with a fixed set of 20,000 patterns on al test set sizes.

A. Partitioned FAM

Comparisons of the training performance of the partitioned FAM approach and the non-partitioned FAM approach on the aforementioned database were conducted. The training performance was based on two measures: the time that it took for the networks to undergo one epoch of training and generalization performance of the trained networks on the chosen test sets (see Figure [6]). The dimensions to project the data in the partitioned-FAM approach were chosen manually by simply observing the range, variation of the values of the datasets across the chosen dimensions. A vigilance value of 0.96 was used in the training. This gives a partitioning scheme with boxes of side size of 0.04 or 253 = 15625 boxes in the three dimensions that we used to project the data.

Some of the observations from the results (error rate and training time) of the FAM and partitioned FAM with these databases are: (a) the partitioned FAM reduces the training time compared to FAM, and at times significantly, especially when the size of the training set is large (e.g., in the `Covertype` database the training time is reduced by a factor of 78 for the largest training set size) (b) the generalization performance of the partitioned FAM is slightly inferior to FAM especially for training sets of smaller size.

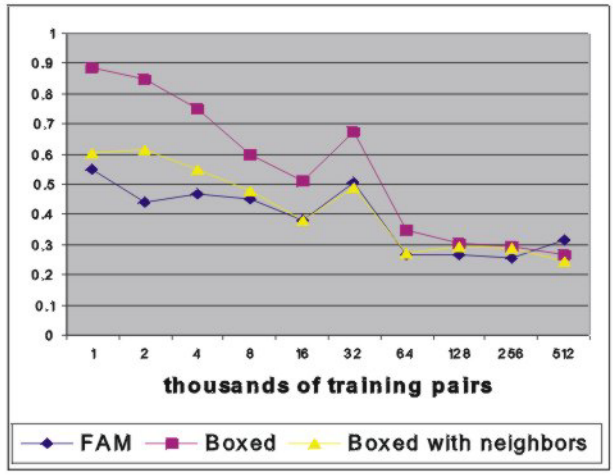

Fig. 5. Forest Covertype database error rate by training set size (in 1000^{ds} of patterns).

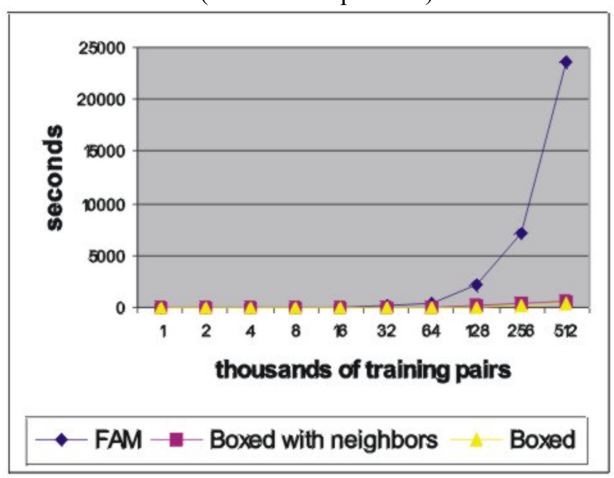

Fig. 6. Forest Covertype database duration in seconds against training set size (in 1000nd's of patterns).

B. Pipelined FAM

Classification results where comparable to those of the unmodified FAM algorithm when wsing the pipelined FAM run with the same database. Nevertheless speedup was substantial, and as expected, it's training time is linear with respect to the number of processors used in the pipeline.

The table of times clearly shows a linear speedup on the cluster of workstations for the pipelined FAM.

VI. Conclusions

Two partitioning schemes where proposed for the FAM algorithm to work on a parallel setting, both of them provided good speedup and classification results. These results show promise and encourage us to continue with further research.

Among our current interests lie a) The development of a pipelined \emph{matchtracking} FAM parallel algorithm b) The addition of a pruning scheme for the Boxing approachto better it's compression ratio and c) Use of other parallel designs like master slave arrangements that require broadcast send and broadcast receive operations.

Procs	DB sz				
	32	64	128	256	512
1	9.3	28.8	96.0	357.0	1367.9
2	4.8	14.4	49.9	155.4	552.7
4	2.5	7.9	22.1	81.6	273.2
8	1.4	4.2	11.0	51.5	140.2
16	0.9	2.0	7.6	23.4	76.5
32	0.9	1.6	4.5	14.6	38.3

TABLE I

DURATION IN SECONDS IN BEOWULF CLUSTER OF WORKSTATIONS FOR PIPELINED FAM

REFERENCES

[1] G. C. Anagnostopoulos, "Putting the utility of match tracking in Fuzzy ARTMAP to the test," in *Proceedings of the Seventh International Conference on Knowledge–Based Intelligent Information Engineering,* vol. 2, University of Oxford, UK. KES'03, 2003, pp. 1–6.

[2] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen, "Fuzzy ARTMAP: A neural network architecture for incremental learning of analog multidimensional maps," *IEEE Transactions on Neural Networks,* vol. 3, no. 5, pp. 698–713, September 1992.

[3] G. A. Carpenter, S. Grossberg, and J. H. Reynolds, "Fuzzy ART: An adaptive resonance algorithm for rapid, stable classification of analog patterns," in *International Joint Conference on Neural Networks, IJCNN'91,* vol. II, IEEE/INNS/ENNS, Seattle, Washington, 1991, pp. 411–416.

[4] M. Mehta, R. Agrawal, and J. Riassnen, "SLIQ: A fast scalable classifier for data mining," in *Extending Database Technology,* Avignon, France: Springer, 1996, pp18–32. [Online]. Available: http://citeseer.mj.nec.com/mehta96sliq.html

[5] C. M. Procopiuc, M. Jones, P. K. Agarwal, and T. Murali, "A monte carlo algorithm for fast projective clustering," ser. ACM SIGMOD, ACM, Ed., Madison, Wisconsin, USA, June 2002, pp. 418–427.

[6] J. R. Quinlan, *C4.5: Programs for Machine Learning.* San Mateo, California: Morgan Kaufmann, 1993.

[7] J. C. Shafer, R. Agrawal, and M. Mehta, "SPRINT: A scalable parallel classifier for data mining," in *Proc. 22nd Int. Conf. Very Large Databases, VLDB,* T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, Eds. Bombay, India: Morgan Kaufmann, September 1996, pp. 544–555.

[8] University of California, Irvine, "UCI machine learning repository," http://www.icf.ucf.edu/mlearn/MLRepository.html, 2003.