A Partitioned Fuzzy ARTMAP Implementation for Fast Processing of Large Databases on Sequential Machines *

José Castro and **Michael Georgiopoulos** and **Ronald Demara** and **Avelino Gonzalez**
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, Florida
{jcastro, demara, gonzalez}@pegasus.cc.ucf.edu
michaelg@mail.ucf.edu

Abstract

Fuzzy ARTMAP (FAM) is a neural network architecture that can establish the correct mapping between real valued input patterns and their correct labels. FAM can learn quickly compared to other neural network paradigms and has the advantage of incremental/online learning capabilities. Nevertheless FAM tends to slow down as the size of the data set grows. This problem is analyzed and a solution is proposed that can speed up the algorithm in sequential as well as parallel settings. Experimental results are presented that show a considerable improvement in speed of the algorithm at the cost of creating larger size FAM architectures. Directions for future work are also discussed.

Introduction

Neural Networks have been used extensively and successfully for a variety of problems. As computing capacity and electronic databases grow, there is an increasing need to process considerably larger databases. Most learning algorithms are not up to the challenge because their computational complexity is high and their convergence time becomes prohibitive as the size of the database grows. An exception to this rule are tree learning algorithms such as CART and C4.5. Variations of these tree learning algorithms, such as SPRINT (Shafer, Agrawal, & Mehta 1996) and SLIQ (Mehta, Agrawal, & Rissanen 1996) have been successfully adapted to handle very large data sets.

One neural network architecture that holds promise in handling large databases is the family of ART neural networks. This family of neural networks is considerably faster than the backpropagation neural network architecture, one of the most popular neural network models. Furthermore, ART neural networks have the added advantage over the backpropagation neural network and decision trees that they can learn online. We will see though, that ART Neural Networks suffer from scalability issues, which amounts to stating that their computational complexity does not scale well as the size of the problem increases. This paper addresses this topic and recommends a solution for it.

The paper is organized as follows: First we review the Fuzzy ARTMAP architecture introduced by (Carpenter *et al.* 1992). Then we examine the computational complexity of FAM and a partitioning solution is proposed to speed up the training process in FAM, when FAM deals with large databases. Furthermore, experimental results are presented that illustrate the merit of our approach. We close the paper with a summary of the findings and suggestions for further research.

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

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

where:

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

The assumption here is that the input vector **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 $w_{ji}^a, (1 \leq j \leq N_a, 1 \leq i \leq 2M_a)$, and referred to as top-down weights. The vector of weights $\mathbf{w}_j^a = \left(w_{j1}^a, w_{j2}^a, \ldots, w_{j,2M_a}^a\right)$ 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

*This work was partially supported by the NSF grant CRCD:0203446, and by the Link Foundation Fellowship on simulation and training.

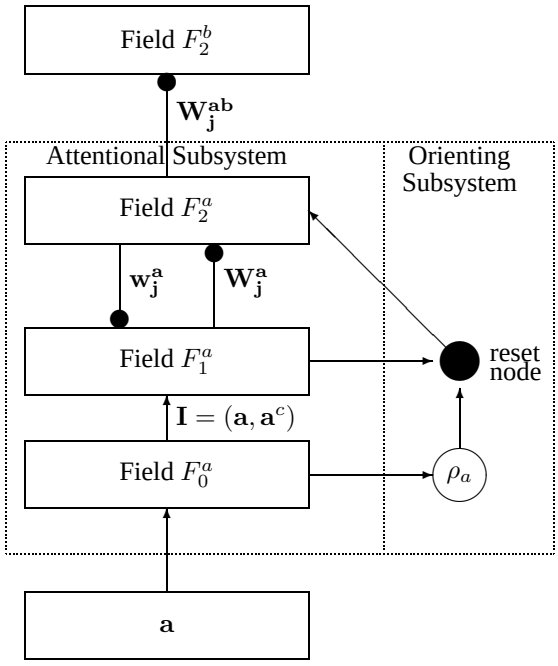

Figure 1: Fuzzy ARTMAP Diagram

W_{jk}^{ab} (called inter–ART weights). The vector of inter–ART weights emanating from every node j in Fuzzy ARTMAP $\left(i.e., \mathbf{W}_j^{ab} = \left[W_{j1}^{ab}, W_{j2}^{ab}, \ldots, W_{j,N_b}^{ab}\right]\right)$ 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, $\left\{(\mathbf{I}^1, \mathbf{O}^1), \ldots, (\mathbf{I}^r, \mathbf{O}^r), \ldots, (\mathbf{I}^{PT}, \mathbf{O}^{PT})\right\}$, 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 $\mathbf{I}^{PT}$ to F_1^a, and $\mathbf{O}^{PT}$ 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 $\tilde{\mathbf{I}}^1, \tilde{\mathbf{I}}^2, \ldots, \tilde{\mathbf{I}}^{PS}$, 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 described below.

The Fuzzy ARTMAP Learning Algorithm

Prior to initiating the training phase of Fuzzy ARTMAP the top–down weights (the w_{ji}^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 x, and y, designated as $x \wedge y$, is a vector whose components are equal to the minimum of components of x and y. Another specific operand involved in these equations is designated by the symbol $|\cdot|$. In particular, $|x|$ is the size of a vector 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:

$$T_j^a = \frac{|\mathbf{I}^r \wedge \mathbf{w}_j^a|}{\beta_a + |\mathbf{w}_j^a|} \tag{3}$$

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:

$$\frac{|\mathbf{I}^r \wedge \mathbf{w}_j^a|}{|\mathbf{I}^r|} \geq \rho_a \tag{4}$$

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} $\left(i.e., \mathbf{W}_{j_{max}}^{ab} = \left[W_{j_{max}1}^{ab}, W_{j_{max}2}^{ab}, \ldots, W_{j_{max},N_b}^{ab}\right]\right)$

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 $\frac{|\mathbf{I}^r \wedge \mathbf{w}_j^a|}{|\mathbf{I}^r|} + \varepsilon$ 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:

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

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).

Fuzzy ARTMAP learning has an interesting geometrical interpretation. The templates ($\mathbf{w}_j^a$) of nodes in the category representation layer of Fuzzy ARTMAP can be represented as hyper rectangles (rectangles in 2-D). The meaning of this hyper rectangle is that it encloses within each boundaries all the input patterns that chose this node (template) as their representative node (template) and were encoded by it (see Figure 2). This hyper rectangle starts from its trivial size of 0, corresponding to the case where it has encoded a single pattern and it grows as more patterns are encoded by it. The size of the hype rectangle is limited by the value of the vigilance parameter (ρ_a). The maximum size hyper rectangle is equal to $M_a(1 - \rho_a)$.

Partitioned FAM

In this section we start with the presentation of the FAM pseudo-code (training portion of the algorithm) that allows us to better understand what really contributes to FAM's complexity. Based on this understanding we propose the partitioned FAM approach to reduce the complexity of FAM training when it deals with large databases.

```
FAM-LEARNING-PHASE(Patterns, ρ̄_a, β_a, maxEpochs)
 1  templates ← {}
 2  iter ← 0
 3  repeat
 4        modified ← false
 5        for each I in Patterns
 6          do ρ_a ← ρ̄_a
 7             LEARN-PATTERN(I, templates, ρ_a, β_a)
 8             iterations ← iterations + 1
 9
10  until (iter = maxEpochs)or(modified = false)
11  return templates
```

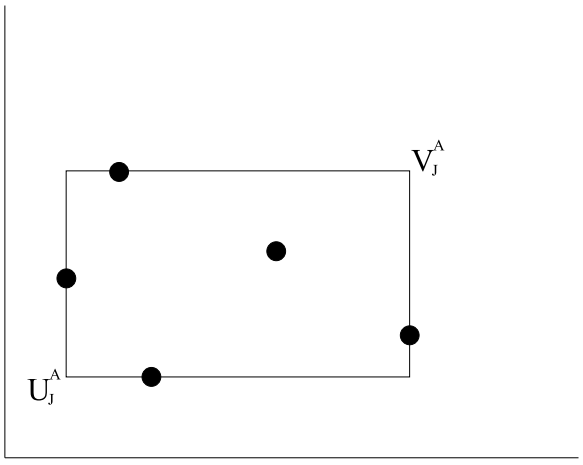

Figure 2: Hyper rectangle representation of a template

Where the procedure LEARN-PATTERN is:

```
LEARN-PATTERN(I, templates, ρ_a, β_a)
 1  repeat
 2        status ← FoundNone
 3        j_max ← GET-MAX-ARG(I, templates, ρ_a, β_a)
 4        if status = FoundOne
 5          then if class(I) = class(w^a_{j_max})
 6                  then
 7                       status ← ThisIsIT
 8                  else
 9                       status ← Matchtracking
10                       ρ ← ρ(I, w^a_{j_max}) + ε
11
12  until status ≠ Matchtracking
```

From the pseudocode we can see that the FAM-LEARNING-PHASE computational complexity is proportional to the number of epochs, number of patterns presented per epoch, and number of templates in the category representation layer of FAM. The LEARN-PATTERN procedure's computational complexity is proportional to the amount of matchtracking loops, The GET-MAX-ARG function's computational complexity is proportional to the total number of templates in the FAM neural network. It is therefore reasonable to consider that the computational complexity of FAM algorithm can be improved by reducing any one of these parameters: the number of epochs, the amount of matchtracking or the number of templates in the data set.

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 $\frac{n}{p}$ 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.

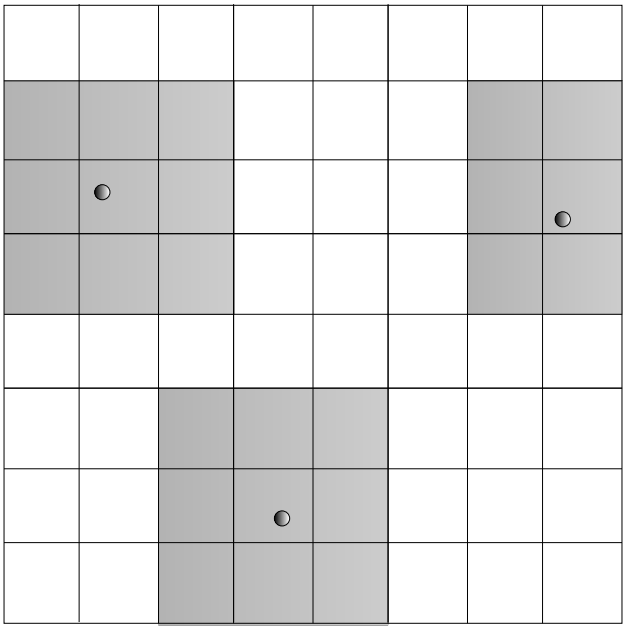

Figure 3: Boxing with neighbors, each patterns can *see* templates in the contiguous boxes, this creates overlap and reduces the number of templates created.

Many different approaches can be proposed to partition the data set between the processors. The key to success of any of these techniques is being able to divide the data set into equally sized data partitions that do not interfere with each other. The difficulty is that there is no fast and easy way to divide the data set into equally sized partitions without analyzing the data set, and this can be a time consuming task (usually $O(n^2)$). Our approach is inspired by the projective clustering approach proposed and implemented in (Procopiuc *et al.* 2002), 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 $\hat{M}_a \ll M_a$, 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 ρ_a and by the dimensionality M_a of the input patterns. In particular,

$$size\left(\mathbf{w}_j^a\right) \leq M_a(1-\rho_a) \tag{6}$$

If we assume that templates grow more or less evenly across every dimension, then by choosing boxes in the projected space of size $(1-\rho_a)$ across every dimension, we are actually restricting the template size to the size that the FAM algorithm 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.

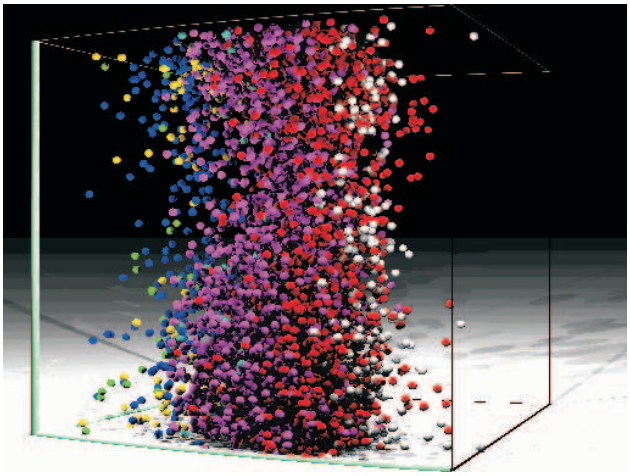

Figure 4: Forest covertype data projected to first 3 dimensions.

Since the partitioning of the data into boxes is implemented after the projection of the data on the fewer ($\hat{M}_a$) than the available dimensions (M_a) dimensions, choosing the right set of dimensions to project is an issue of importance. One way of choosing these fewer dimensions on which to project the data is based on an entropy measure criterion. This criterion is used extensively with decision trees (Quinlan 1993) when a determination needs to be made regarding the next to be chosen data attribute (dimension) with respect to which to split the tree into smaller subtrees. This criterion is defined below:

$$Gain(Dimesion) = E(\text{PART}) - E(\neg\text{PART}) \tag{7}$$

where $E(\neg\text{PART})$ is defined to be the entropy of the data without partitioning, and $E(\text{PART})$ is defined to be the average entropy of the datasets that the data are split into after

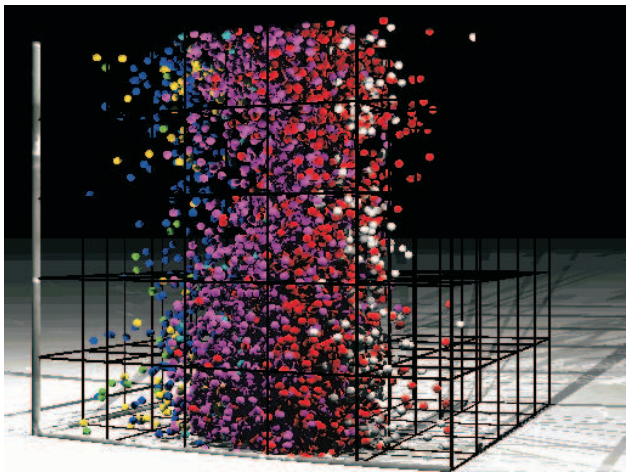

Figure 5: Forest covertype data projected to first 3 dimensions and partitioned using boxing scheme.

the partitioning with respect to a certain dimension takes effect. The entropy of a labeled dataset is defined as

$$\sum_k -p_k \log(p_k) \tag{8}$$

where p_k represents the probability that the data will have label k.

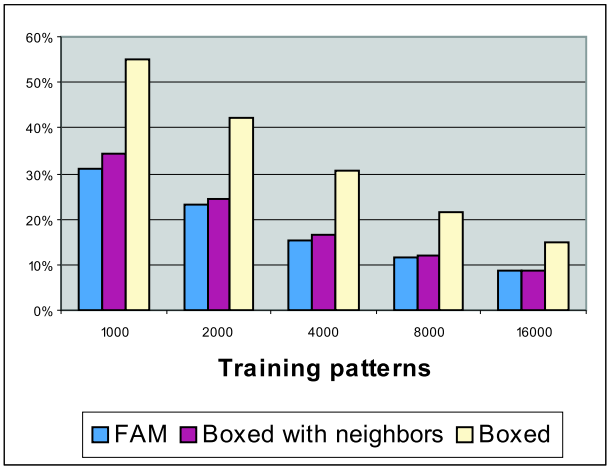

Figure 6: Letters recognition database error rate by training set size.

Experimental Results

To prove the feasibility of our approach a number of tests were conducted using 2 databases from (University of California): the letter recognition database from David Slate and the Covertype database by Jock A. Blackard (University of California). On both databases the size of the data used for training was increased by a factor of 2 starting from 1000 data points and ending with 16000 data points for the letters

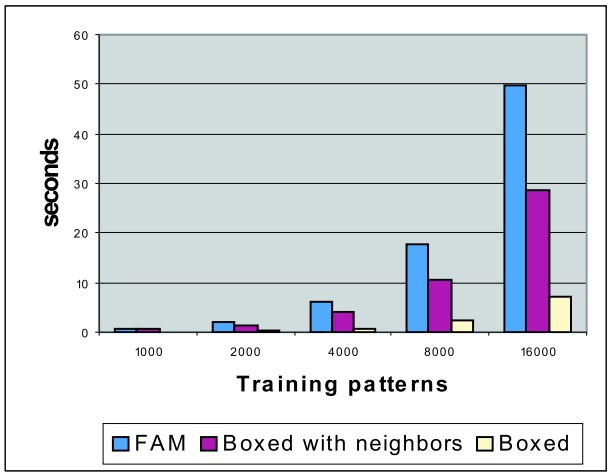

Figure 7: Letters recognition duration in seconds against training set size

database and starting with 1000 data points and ending with 512,000 for the covertype data. Classification performance was evaluated with a fixed set of 4,000 patterns for the letters database and with fixed set of 20,000 patterns for the covertype data.

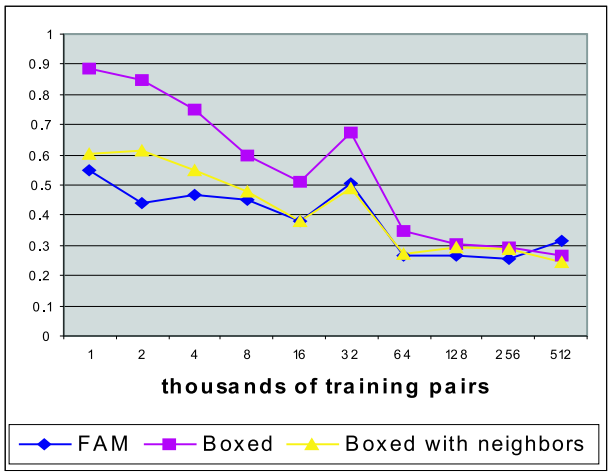

Figure 8: Forest covertype database error rate by training set size (in 1000^{nd}'s of patterns).

Comparisons of the training performance of the partitioned FAM approach and the non-partitioned FAM approach on the aforementioned databases were conducted. The training performance was based on two measures. 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 Figures 6 to 9). 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. Automatically choosing the dimensions on which to project

(using the entropy measure) was also investigated but the results were not that different from the hand-picked approach. Due to lack of space we are only reporting the results of the hand-picked approach. For the letters database a baseline vigilance parameter value of 0.85 for was used. This creates a partitioning scheme where the reduced input space of 3 dimensions was partitioned into boxes of side length equal to 0.15; this results in a total of 343 = 73 boxes. For the covertype database 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.

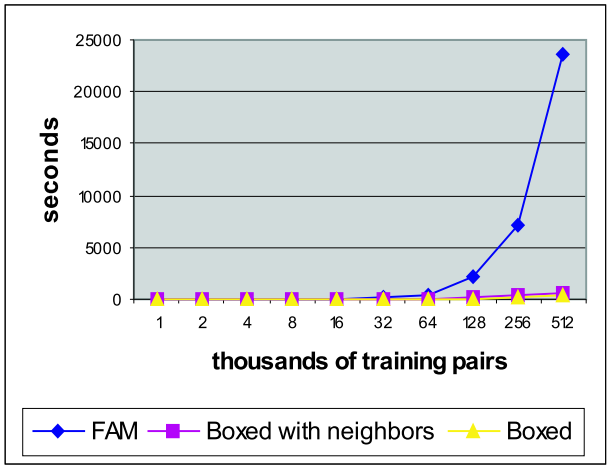

Figure 9: Forest covertype database duration in seconds against training set size (in 1000^{nd}'s of patterns).

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., covertype database where 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 inferior to FAM especially for training sets of smaller size. This effect can be countered by allowing each FAM in the partitioning approach to be influenced not only by data within its associated boxed region but also by the boxed regions that are neighbors to its associated region (referred to in the figures as the boxed with neighbors approach). This creates an overlapping FAM approach that is feasible in the sequential machine but creates communication issues in the parallel approach. Figure 3 shows a representation of boxing with neighbors in 2 dimensions. The improvement obtained in generalization of the partitioned FAM with neighbors is made at the expense of increasing its computational complexity.

Conclusions

The Fuzzy ARTMAP algorithm is a good, general-purpose classifier that can learn online and achieve good generalization capabilities on most databases. When the data set grows though, its required training time increases proportionally with the square of the datapoints used for training. A mechanism for accelerating the convergence of the algorithm when it deals with large databases was proposed. This mechanism appears to work well on the 2 databases tested, especially for the largest of the 2 databases. More experimentation is needed to increase confidence in the results reported here. This is the topic of our on-going investigation. The speed up observed was impressive, especially for the larger size database. We believe that the proposed partitioning technique can facilitate the usage of the FAM algorithm for very large databases. It is worth noting that the proposed partitioning scheme does not affect the desirable ART feature of training in an on-line fashion. The partitioning scheme proposed is also amenable to parallel implementation but does not require parallelization to speed up the algorithm.

References

Carpenter, G. A.; Grossberg, S.; Markuzon, N.; Reynolds, J. H.; and Rosen, D. B. 1992. Fuzzy artmap: A neural network architecture for incremental learning of analog multidimensional maps. *IEEE Transactions on Neural Networks* 3(5).

Mehta, M.; Agrawal, R.; and Rissanen, J. 1996. SLIQ: A fast scalable classifier for data mining. In *Extending Database Technology*, 18–32.

Procopiuc, C. M.; Jones, M.; Agarwal, P. K.; and Murali, T. 2002. A monte carlo algorithm for fast projective clustering. ACM SIGMOD, 418–427.

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

Shafer, J. C.; Agrawal, R.; and Mehta, M. 1996. SPRINT: A scalable parallel classifier for data mining. In Vijayaraman, T. M.; Buchmann, A. P.; Mohan, C.; and Sarda, N. L., eds., *Proc. 22nd Int. Conf. Very Large Databases, VLDB*, 544–555. Morgan Kaufmann.

University of California, I. Uci machine learning repository. http://www.icf.uci.edu/m̃learn/MLRepository.html.