Available online at www.sciencedirect.com
SCIENCE DIRECT
Neural Networks 18 (2005) 967–984
Neural Networks
www.elsevier.com/locate/neunet

Data-partitioning using the Hilbert space filling curves: Effect on the speed of convergence of Fuzzy ARTMAP for large database problems

José Castro, Michael Georgiopoulos*, Ronald Demara, Avelino Gonzalez

Department of Electrical and Computer Engineering, University of Central Florida, 4000 Central Florida Blvd. Engineering Building 1, Suite 407, Orlando, FL 32816-2786, USA

Received 5 February 2004; revised 27 January 2005; accepted 27 January 2005

Abstract

The Fuzzy ARTMAP algorithm has been proven to be one of the premier neural network architectures for classification problems. One of the properties of Fuzzy ARTMAP, which can be both an asset and a liability, is its capacity to produce new nodes (templates) on demand to represent classification categories. This property allows Fuzzy ARTMAP to automatically adapt to the database without having to a priori specify its network size. On the other hand, it has the undesirable side effect that large databases might produce a large network size (node proliferation) that can dramatically slow down the training speed of the algorithm. To address the slow convergence speed of Fuzzy ARTMAP for large database problems, we propose the use of space-filling curves, specifically the Hilbert space-filling curves (HSFC). Hilbert space-filling curves allow us to divide the problem into smaller sub-problems, each focusing on a smaller than the original dataset. For learning each partition of data, a different Fuzzy ARTMAP network is used. Through this divide-and-conquer approach we are avoiding the node proliferation problem, and consequently we speedup Fuzzy ARTMAP's training. Results have been produced for a two-class, 16-dimensional Gaussian data, and on the Forest database, available at the UCI repository. Our results indicate that the Hilbert space-filling curve approach reduces the time that it takes to train Fuzzy ARTMAP without affecting the generalization performance attained by Fuzzy ARTMAP trained on the original large dataset. Given that the resulting smaller datasets that the HSFC approach produces can independently be learned by different Fuzzy ARTMAP networks, we have also implemented and tested a parallel implementation of this approach on a Beowulf cluster of workstations that further speeds up Fuzzy ARTMAP's convergence to a solution for large database problems.

Keywords: Fuzzy-ARTMAP; Hilbert space-filling curve; Data mining; Data-partitioning

1. Introduction

Neural Networks have been used extensively and successfully to tackle a wide variety of problems. As computing capacity and electronic databases grow, there is an increasing need to process considerably larger databases. In this context, the algorithms of choice tend to be ad hoc algorithms (Agrawal & Srikant, 1994) or tree based algorithms such as CART (King, Feng, & Shutherland, 1995) and C4.5 (Quinlan, 1993). 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 datasets.

Neural network algorithms can have a prohibitively slow convergence to a solution, especially when they are trained on large databases. Even one of the fastest (in terms of training speed) neural network algorithms, the Fuzzy ARTMAP algorithm (Carpenter, Grossberg, Markuzon, Reynolds, & Rosen, 1992; Carpenter, Grossberg, & Reynolds, 1991), and its faster variations (Kasuba, 1993; Taghi, Baghmisheh, & Pavesic, 2003) tend to converge slowly to a solution as the size of the network increases. The performance of Fuzzy ARTMAP and its variants has been documented extensively in the literature. Some of these references favor Fuzzy ARTMAP some of them do not, as compared to other neural network classifiers or other classifiers in general. For example, Joshi et al. (see Joshi, Ramakrishman, Houstics, & Rice, 1997) compared more

* Corresponding author. Tel.: +1 407 823 5338; fax: +1 407 823 5835.
E-mail addresses: jcastro@pegasus.cc.ucf.edu (J. Castro), michaelg@mail.ucf.edu (M. Georgiopoulos), demara@pegasus.cc.ucf.edu (R. Demara), gonzalez@pegasus.cc.ucf.edu (A. Gonzalez).

0893-6080/$ - see front matter © 2005 Elsevier Ltd. All rights reserved.
doi:10.1016/j.neunet.2005.01.007

Nomenclature

$\bar{\rho}_a$	baseline vigilance, $\bar{\rho}_a \in [0, 1]$
α	choice parameter, $\alpha > 0$
ε	small positive constant
$\mathbf{w}$, $\mathbf{w}_j$	weights in the neural network
$T(\mathbf{I}, \mathbf{w}, \alpha)$	activation of the FS-FAM node with template $\mathbf{w}$
$\rho(\mathbf{I}, \mathbf{w})$	vigilance ratio
Γ	repeat factor of matchtracking
κ	compression ratio
$T^{\text{sequential}}$	sequential time of FS-FAM
$T_p^{\text{sequential}}$	(Partitions $= p$) sequential time of FS-FAM with p partitions
T_p^{parallel}	(Partitions $= p$) parallel time of FS-FAM with p partitions
M_a	dimensionality of the input patterns
m	order of approximation of a space-filling curve
N	number of bits in a derived key of a Hilbert index
r	an N-Bit binary Hilbert *derived-key*
γ_j^i	a binary digit in r
γ^i	ith binary byte in r
a_j	a coordinate in dimension j of the point $(a_1, a_2, \ldots, a_j, \ldots, a_{M_a})$
α_j^i	a binary digit in a coordinate a_j
α^i	a concatenation of all the ith entries of the a_j's

than 20 classifiers on seven different machine-learning problems. The conclusion of this study was that Fuzzy Min–Max (Simpson, 1992), a network that shares a lot of similarities with the Fuzzy ARTMAP neural network, gives the best or the second best classification accuracy over all the other algorithms on these machine-learning problems. Also, in Heinke and Hamker (1995), the authors compare the performance of Fuzzy ARTMAP and three other neural network classifiers, that is Growing Neural Gas (GNG), Growing Cell Structures (GCS) and the multi-layer perceptron (MLP) on a number of benchmark datasets. The conclusion of their study is that Fuzzy ARTMAP performance (classification accuracy) is inferior to the performance of all the other neural networks. Nevertheless, Fuzzy ARTMAP required lesser amount of time to converge to a solution, and it created smaller size neural network architectures.

Some of the advantages that Fuzzy ARTMAP has, compared to other neural network classifiers: that it learns the required task fast (especially its faster variants, such as the Fast Simplified Fuzzy ARTMAP), 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 the disadvantages of Fuzzy ARTMAP is its tendency to create large size networks, especially when the data presented to Fuzzy ARTMAP are of noisy and/or overlapping nature. This Fuzzy ARTMAP shortcoming has been coined as 'the category proliferation' problem. Quite often, the category proliferation problem, observed in Fuzzy ARTMAP architectures, is connected with the issue of over-training in Fuzzy ARTMAP. Over-training happens when Fuzzy ARTMAP is trying to learn the training data perfectly at the expense of degraded generalization performance (i.e. classification accuracy on unseen data) and also at the expense of creating many categories to represent the training data. A number of authors have tried to address the category proliferation/over-training problem in Fuzzy ARTMAP. Amongst them we refer to the work by Marriott and Harrison (1995), where the authors eliminate the match tracking mechanism of Fuzzy ARTMAP when dealing with noisy data, the work by Charalampidis, Kasparis, and Georgiopoulos (2001), where the Fuzzy ARTMAP equations are appropriately modified to compensate for noisy data, the work by Anagnostopoulos, Bharadwaj, Georgiopoulos, Verzi, and Heileman (2003), Gomez-Sanchez, Dimitriadis, Cano-Izquierdo, and Lopez-Coronado (2002), and Verzi, Heileman, Georgiopoulos, and Healy (2001), where different ways are introduced of allowing the Fuzzy ARTMAP categories to encode patterns that are not necessarily mapped to the same label, provided that the percentage of patterns corresponding to the majority label exceeds a certain threshold, the work by Koufakou, Georgiopoulos, Anagnostopoulos, & Kasparis (2001), where cross-validation is employed to avoid the over-training/category proliferation problem in Fuzzy ARTMAP, and the work by Carpenter (Carpenter, Milenova, & Noeske, 1998), Parrado-Hernandez, Gomez-Sanchez, and Dimitriadis (2003) and Williamson (1997), where the ART structure is changed from a winner-take-all to a distributed version and simultaneously slow learning is employed with the intent of creating fewer ART categories and reducing the detrimental effects of noisy patterns.

In this paper, our focus is to improve the convergence speed of ART-like structures through a training data-partitioning approach. In order to connect our work with previous work on Fuzzy ARTMAP it is worth emphasizing again the work by Kasuba (1993), where a simplified Fuzzy ARTMAP structure (simplified Fuzzy ARTMAP) is introduced that is simpler and faster than the original Fuzzy ARTMAP structure, and functionally equivalent with Fuzzy ARTMAP for classification problems. Furthermore, Taghi et al. (2003), describe variants of simplified Fuzzy ARTMAP, called Fast Simplified Fuzzy ARTMAP, that reduce some of the redundancies of Simplified Fuzzy ARTMAP and speed up its convergence to a solution, even further. One of the Fuzzy ARTMAP fast algorithmic

variants presented in Taghi et al. (1993) is called, SFAM2.0 and it has the same functionality as Fuzzy ARTMAP (Carpenter et al., 1992) for classification problems. From now we will refer to this variant of Fuzzy ARTMAP as FS-FAM (Fast Simplified Fuzzy ARTMAP). The focus of our paper is FS-FAM. Note that FS-FAM is faster than Fuzzy ARTMAP because it eliminated some of the redundancies of the original Fuzzy ARTMAP that are not necessary when classification problems are considered. Since the functionality of Fuzzy ARTMAP (Carpenter et al., 1992) and FS-FAM (Taghi et al., 2003) are the same for classification problems we will occasionally refer to FS-FAM as Fuzzy ARTMAP. We chose to demonstrate the effectiveness of our proposed data-partitioning approach on the performance of FS-FAM since, if we demonstrate its effectiveness for Fuzzy ARTMAP, its extension to other ART structures can be accomplished without a lot of effort. This is due to the fact that the other ART structures share a lot of similarities with Fuzzy ARTMAP, and as a result, the advantages of the proposed data-partitioning approach can be readily extended to other ART variants (for instance some of the ART variants, mentioned above, that address the category proliferation/over-training issue in Fuzzy ARTMAP).

One of the properties of Fuzzy ARTMAP, 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 Fuzzy ARTMAP to automatically adapt to the database without having to arbitrarily specify its network size. On the other hand though, this same property of Fuzzy ARTMAP has the undesirable effect that on large databases it can produce a large network size (category (node) proliferation problem) that can dramatically slow down its convergence speed. It would be desirable to be able to train Fuzzy ARTMAP on large databases, while keeping Fuzzy ARTMAP's convergence speed reasonable. To address this problem we propose the use of space-filling curves. A space-filling curve is by definition a continuous mapping from a unit hypercube $[0,1]^n$ to the unit interval [0,1]. Skopal, Krátký, and Snášel (2002) analyze different space-filling curves, amongst them the Peano curve, *Z* curve and the Hilbert curve, and also provides measures for their appropriateness. Moon, Jagadish, Faloutsos, and Saltz (2001) argues and proves that the Hilbert space-filling curve (HSFC) is the mapping that provides the *least number of splits* of compact sets from $[0,1]^n$ to [0,1]. This can be interpreted as stating that points that are close on the mapping will also be close on the *n*-dimensional hypercube. Lawder (Lawder & King, 2000) has taken advantage of this property of HSFCs and used them to develop a multi-dimensional indexing technique.

In this paper, we use Hilbert space-filling curves to partition the training data, prior to their presentation to FS-FAM. Through the training set partitioning that the HSFC produces we can train many Fuzzy ARTMAP networks, each one of them with one of the partitions of the training set that the HSFC generates. Through this divide-and-conquer approach, facilitated by the Hilbert space-filling curves, our research has shown that we can dramatically reduce the convergence speed of a single FS-FAM trained on the entire (large) training set. Our experimentation also demonstrates that the many FS-FAMs trained on the smaller training sets cumulatively produce a combined Fuzzy ARTMAP structure, referred to as hFS-FAM (Hilbert Fast Simplified Fuzzy ARTMAP), whose size is comparable to the size of the single FS-FAM trained on the large dataset. Furthermore, our experiments show that hFS-FAM's generalization performance is as good as, and at times better, than the single FS-FAM's performance.

It is worth mentioning that our literature review did not produce any papers that addressed the issue of convergence speed in Fuzzy ARTMAP for large databases through data-partitioning approaches. But data-partitioning approaches have been applied to other neural network architectures with success. For instance, Kerstetter (1998) have used a data-partitioning (clustering) approach that divides the data into smaller datasets in an effort to effectively train a multi-layer feedforward neural network for a target recognition application. The clustering approach utilized in their paper was tailored for the multi-layer feedforward neural network and its associated learning algorithm (back-propagation). Another neural network architecture for which data-partitioning (clustering) has been very beneficial is the probabilistic neural network developed by Specht (1990). One of the major issues with the probabilistic neural network is that it uses too many exemplars (templates) to represent the patterns in the training set, resulting in unnecessary long computations in order to respond with a predicted classification for new patterns that it has never seen before. Radial basis function neural networks (Moody & Darken, 1989) suffer from a similar type of deficiency. Clustering (data-partitioning) approaches to remedy this problem for the PNN and RBF neural networks have been proposed in the literature. For example, in Burrascano (1991), Kohonen's learning vector quantization (LVQ) has been successfully used to find representative exemplars to be used with the PNN. In a similar fashion, Traven (1991) reduces the number of templates needed by the PNN by replacing groups of training patterns with exemplar patterns using an approach that estimates the probability density functions needed in the PNN with Gaussian functions of appropriate mean and covariances. Our intent in this paper was similar with the intent in the aforementioned papers, where data-partitioning was applied. We wanted to reduce the number of templates that Fuzzy ARTMAP had to consider during its training phase. So, instead of training Fuzzy ARTMAP on all the available data we partitioned the data in smaller sets, and we trained many Fuzzy ARTMAPs, in an effort to reduce the training time required by a single Fuzzy ARTMAP. We used the HSFC (Hilbert Space-Filling

Curve) approach to partition the data because it is very fast (it takes $N \log_2(N)$ operations to partition a dataset containing N points), and as a result it is expected to provide significant savings in the required computations that Fuzzy ARTMAP needs to solve a 'large dataset' problem. In our paper, we demonstrate that the performance of Fuzzy ARTMAP (generalization and size) is not affected by the data-partitioning (clustering) that the HSFC enforces.

This paper is organized as follows. First, we review the FS-FAM architecture and its functionality. Then we examine the computational complexity of FS-FAM and analyze how and why a partitioning approach can considerably reduce the algorithm's training time. Subsequently, we discuss space-filling curves in general and the Hilbert space-filling curve in particular; furthermore, we discuss of why this curve can be instrumental in improving FS-FAM's convergence time. Also, experimental results are presented on a sequential machine and on a Beowulf cluster of workstations that illustrate the merit of our approach. The datasets that we have used in our experiments to demonstrate the effectiveness of hFS-FAM, compared to FS-FAM are (i) a two-class, 16-dimensional Gaussianly distributed dataset, and (ii) the Forest database that resides at the UCI repository (University of California, Irvine, 2003). We close the paper with a summary of the findings and suggestions for further research.

2. The FS-FAM architecture

The Fuzzy ARTMAP neural network and its associated architecture was introduced by Carpenter and Grossberg in their seminal paper (Carpenter et al., 1992). Since its introduction, a number of Fuzzy ARTMAP variations and associated successful applications of this ART family of neural networks have appeared in the literature (for instance, ARTEMAP (Carpenter & Ross, 1995), ART-MAP-IC (Carpenter & Markuzon, 1998), Ellipsoid-ART/ARTMAP (Anagnostopoulos & Georgiopoulos, 2001), Fuzzy Min–Max (Simpson, 1992), LAPART2 (Caudell & Healy, 1999), and σ-FLNMAP (Petridis, Kaburlasos, Fragkou, & Kehagais, 2001), to mention only a few). For the purposes of the discussion that follows in this section it is worth mentioning again the work by Kasuba (1993) and Taghi et al. (2003). In his paper, Kasuba introduces a simpler Fuzzy ARTMAP architecture, called *Simplified Fuzzy ARTMAP*. Kasuba's simpler Fuzzy ARTMAP architecture is valid only for classification problems. Taghi et al. (2003) have eliminated some of the unnecessary computations involved in Kasuba's Simplified Fuzzy ARTMAP, and introduced two faster variants of Simplified Fuzzy ARTMAP, called SFAM2.0 and SFAM2.1. Kasuba's simpler Fuzzy ARTMAP variants were denoted as SFAM1.0 and 1.1 in Taghi's paper. In order to connect the version of Fuzzy ARTMAP, implemented in this paper, with Carpenter's and Grossberg's Fuzzy ARTMAP, Kasuba's simplified Fuzzy ARTMAP (SFAM1.0) and Taghi's simplified Fuzzy ARTMAP versions, such as SFAM1.1, SFAM2.0 and SFAM2.1, it is worth mentioning that in our paper we have implemented the Fuzzy ARTMAP version, called SFAM2.0 in Taghi's paper. As, we have mentioned in Section 1, we refer to this Fuzzy ARTMAP variant as FS-FAM. Once more, FS-FAM is algorithmically equivalent with Fuzzy ARTMAP for classification problems. Classification problems are the only focus in our paper.

The block diagram of FS-FAM is shown in Fig. 1. Notice that this block diagram is different than the block diagram of Fuzzy ARTMAP mentioned in Carpenter et al. (1991), but very similar to the block diagram depicted in Kasuba's work (see Kasuba, 1993). The Fuzzy ARTMAP architecture of the block diagram of Fig. 1 has three major layers. The *input layer* (F_1^a), where the input patterns (designated by $\mathbf{I}$) are presented, the *category representation layer* (F_2^a), where compressed representations of these input patterns are formed (designated as $\mathbf{w}_j^a$, and called *templates*), and the *output layer* (F_2^b) that holds the labels of the categories formed in the category representation layer. Another layer, shown in Fig. 1 and designated by F_0^a, is a pre-processing layer and its functionality is to pre-process the input patterns, prior to their presentation to FS-FAM. This pre-processing operation (called complementary coding is described in more detail below).

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.

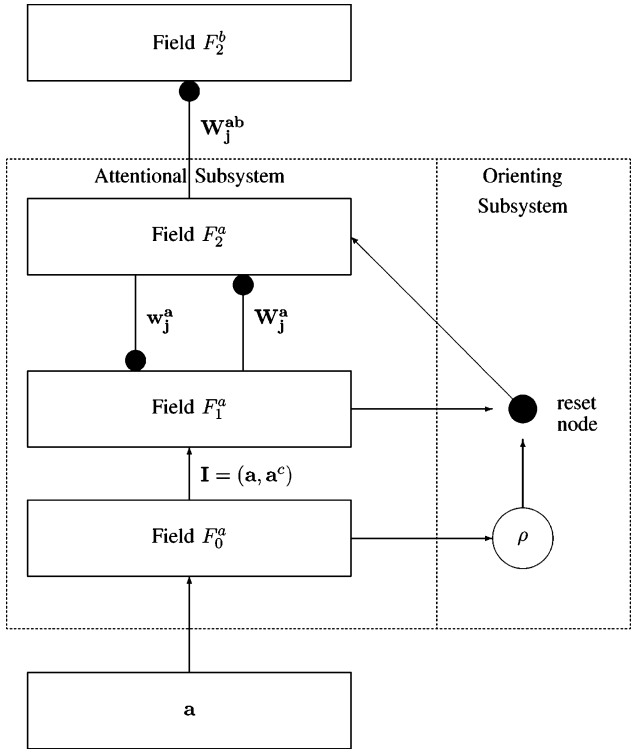


Fig. 1. Block diagram of the FS-FAM architecture.

Given a set of inputs and associated label pairs, $\{(\mathbf{I}^1, \text{label}(\mathbf{I}^1)),\ldots,(\mathbf{I}^r, \text{label}(\mathbf{I}^r)),\ldots,(\mathbf{I}^{PT}, \text{label}(I^{PT}))\}$, we want to train Fuzzy ARTMAP to map every input pattern of the training set to its corresponding label. To achieve the aforementioned goal we present the training set to the Fuzzy ARTMAP architecture repeatedly. That is, we present $\mathbf{I}^1$ to F_1^a, label($\mathbf{I}^1$) to F_2^b, $\mathbf{I}^2$ to F_1^a label($\mathbf{I}^2$) to F_2^b and finally $\mathbf{I}^{PT}$ to F_1^a, and label($\mathbf{O}^{PT}$) to F_2^b. We present the training set 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 training set presentation. The aforementioned training scenario is called *off-line learning*. There is another training scenario, the one considered in this paper, that is called *on-line training*, where each one of the input/label pairs are presented to Fuzzy ARTMAP only once. The performance phase of Fuzzy ARTMAP works as follows. Given a set 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 (label) 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 set to the trained Fuzzy ARTMAP architecture and we observe the network's output.

The training process in FS-FAM is succinctly described in Taghi's et al. paper (Taghi et al., 2003). We repeat it here to give the reader a good, well-explained overview of the operations involved in its training phase.

(1) Find the nearest category in the category representation layer of Fuzzy ARTMAP that 'resonates' with the input pattern.
(2) If the labels of the chosen category and the input pattern match, update the chosen category to be closer to the input pattern.
(3) Otherwise, reset the winner, temporarily increase the resonance threshold (called *vigilance parameter*), and try the next winner.
(4) If the winner is uncommitted, create a new category (assign the representative of the category to be equal to the input pattern, and designate the label of the new category to be equal to the label of the input pattern).

The nearest category to an input pattern $\mathbf{I}^r$ presented to FS-FAM is determined by finding the category that maximizes the function:

$$T_j^a(\mathbf{I}^r, \mathbf{w}_j^a, \alpha) = \frac{|\mathbf{I}^r \wedge \mathbf{w}_j^a|}{\alpha + |\mathbf{w}_j^a|} \tag{1}$$

The above function is called the *bottom-up input* (or choice function) pertaining to the F_2^a node j with category representation (template) equal to the vector $\mathbf{w}_j^a$, due to the presentation of input pattern $\mathbf{I}^r$. This function obviously depends on an FS-FAM network parameter α, called *choice parameter*, that assumes values in the interval $(0, \infty)$. In most simulations of Fuzzy ARTMAP, the useful range of α is the interval (0,10]. Larger values of α create more category nodes in the category representation layer of FS-FAM.

The resonance of a category is determined by examining if the function, called *vigilance ratio*, and defined below

$$\rho(\mathbf{I}^r, \mathbf{w}_j^a) = \frac{|\mathbf{I}^r \wedge \mathbf{w}_j^a|}{|\mathbf{I}^r|} \tag{2}$$

satisfies the following condition:

$$\rho(\mathbf{I}^r, \mathbf{w}_j^a) \geq \rho_a \tag{3}$$

If the above equation is satisfied we say that resonance is achieved. The parameter ρ_a appearing in the above equation is called *vigilance parameter* and assumes values in the interval [0,1]. As the vigilance parameter increases, more category nodes are created in the category representation layer (F_2^a) of Fuzzy ARTMAP. If the label of the input pattern ($\mathbf{I}^r$) is the same as the label of the resonating category, then the category's template ($\mathbf{w}_j^a$) is updated to incorporate the features of this new input pattern ($\mathbf{I}^r$). The update of a category's template ($\mathbf{w}_j^a$) is performed as depicted below:

$$\mathbf{w}_j^a = \mathbf{w}_j^a \wedge \mathbf{I}^r \tag{4}$$

The update of templates, illustrated by the above equation, has been called *fast-learning* in Fuzzy ARTMAP. Our paper is concerned only with the fast-learning Fuzzy ARTMAP.

If the category j is chosen as the winner and it resonates, but the label of this category $\mathbf{w}_j^a$ is different than the label of the input pattern $\mathbf{I}^r$, then this category is reset and the vigilance parameter ρ_a is increased to the level defined in the following equation. Note that the label of a category j, in the category representation layer of FS-FAM, is encoded in the interconnection weight vector $\mathbf{W}_j^{ab}$ emanating from node j of the category representation layer F_2^a and converging to all the nodes in the output layer F_2^b of FS-FAM

$$\frac{|\mathbf{I}^r \wedge \mathbf{w}_j^a|}{|\mathbf{I}^r|} + \varepsilon \tag{5}$$

The parameter ε assumes very small values. Increasing the value of vigilance barely above the level of vigilance ratio of the category that is reset guarantees that after this input/label-of-input pair is learned by FS-FAM, immediate presentation of this input to FS-FAM will result in correct recognition of its label by Fuzzy ARTMAP. It is difficult to correctly set the value of ε so that you can guarantee that after category resets no legitimate categories are missed by FS-FAM. Nevertheless, in practice, typical values of the parameter ε are taken from the interval [0.00001, 0.001]. In our experiments, we took $\varepsilon = 0.001$. After the reset of category j, other categories are searched that maximize the bottom-up input and they satisfy the vigilance (resonate). This process continues until a category is found that maximizes the bottom-up input, satisfies the vigilance and has the same label as the input pattern presented to

FS-FAM. Once this happens, update of the category's template as indicated by Eq. (4) ensues. If through this search process an uncommitted category (an uncommitted category is a category that has not encoded any input pattern before) is chosen, it will pass the vigilance, its label will be set to be equal to the label of the presented input pattern, and the update of the category's template will create a template that is equal to the presented input pattern.

In all of the above equations (Eqs. (1)–(5)), there is specific operand involved, called *fuzzy min operand*, and 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.

It is worth mentioning that an input pattern $\mathbf{I}$ presented at the input layer (F_1^a) of FS-FAM has the following form

$$\mathbf{I} = (\mathbf{a}, \mathbf{a}^c) = (a_1, a_2, \ldots a_{M_a}, a_1^c, a_2^c, \ldots, a_{M_a}^c) \tag{6}$$

where

$$a_i^c = 1 - a_i; \quad \forall\, i \in \{1, 2, \ldots, M_a\} \tag{7}$$

The assumption here is that the input vector $\mathbf{a}$ is such that each one of its components lies in the interval [0, 1]. Any input pattern can be, through appropriate normalization, be represented by the input vector $\mathbf{a}$, where M_a stands for the dimensionality of this input pattern. The above operation that creates $\mathbf{I}$ from $\mathbf{a}$ is called *complementary coding* and it is required for the successful operation of Fuzzy ARTMAP. The number of nodes (templates) created in the F_2^a layer of FS-FAM (category representation layer) is designated by N_a, and it is not a parameter that need to be defined by the user before training commences; N_a is parameter, whose value is dictated by the needs of the problem that FS-FAM is trained with and the setting of the choice parameter (α) and baseline vigilance parameter $\bar{\rho}_a$. The *baseline vigilance parameter* is a parameter set by the user as a value in the interval [0,1]. The vigilance parameter ρ_a, mentioned earlier (see Eq. (3)), is related with the baseline vigilance $\bar{\rho}_a$ since at the beginning of training with a new input/label pattern pair, the vigilance parameter is set equal to the baseline vigilance parameter; during training with this input/label pattern pair the vigilance parameter could be raised above the baseline vigilance parameter (see Eq. (5)), only to be reset back the baseline vigilance parameter value once a new input/label pattern pair appears.

Prior to initiating the training phase of FS-FAM the user has to set the values for the choice parameter α (chosen as a value in the interval [0,10]), baseline vigilance parameter value $\bar{\rho}_a$ (chosen as a value in the interval [0,1]).

In the performance phase of FS-FAM, a test input is presented to FS-FAM and the category node in F_2^a of FS-FAM that has the maximum bottom-up input is chosen. The label of the chosen F_2^a category is the label that FS-FAM predicts for this test input. By knowing the correct labels of test inputs belonging to a test set allows us, in this manner, to calculate the classification error of FS-FAM for this test set.

2.1. A FS-FAM pseudocode

In this paper, we primarily focus on one-epoch FS-FAM training, or as it was referred before the 'on-line training' FS-FAM. Whatever speedup we achieve for the 'on-line training' FS-FAM, it will also be applicable to the 'off-line training' FS-FAM, since the 'off-line training' FS-FAM is an 'on-line training' FS-FAM, where after an on-line training cycle is completed, another cycle starts with the same set of training input patterns/label pairs. These 'on-line training' FS-FAM cycles are repeated for as long as it is necessary for the FS-FAM network to learn the required mapping. The following pseudocode pertains to the 'on-line training' FS-FAM.

```
FAM-TRAINING-PHASE(Patterns, ρ̄_a, α, ε)
templates ← {}
for each I^r in Patterns
  do ρ ← ρ̄_a
     repeat
          T_max ← M_a / (2M_a + α)
          status ← FoundNone
          for each w_j^a in templates
            do if ρ(I^r, w_j^a) ≥ ρ and T(I^r, w_j^a, α) > T_max
                 then
                      T_max ← T(I^r, w_j^a, α)
                      j_max ← j
                      status ← FoundOne
          if status = FoundOne
            then if label(I^r) = label(w_{j_max}^a)
                   then status ← Allocated
                   else status ← TryAgain
                        ρ ← ρ(I^r, w_{j_max}^a) + ε
     until status ≠ TryAgain
     if status = Allocated
       then
            w_{j_max}^a ← w_{j_max}^a ∧ I^r
       else
            templates ← templates ∪ {I^r}

return templates
```

The step-by-step procedure of the training phase of FS-FAM (considered in this paper, and outlined above in terms of its associated pseudocode) is described in every detail in Taghi et al. (2003). It is only repeated here, in less detail, to assure completeness of the manuscript.

2.2. On-line FS-FAM complexity analysis

We can see from the pseudocode that the algorithm tests every input pattern $\mathbf{I}$ 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 **repeat** loop in the pseudocode is executed for each input pattern. Notice that this Γ parameter is not specified as a numerical value but it is introduced as an unspecified constant to help us produce an analytical formula that describes the complexity of the on-line training FS-FAM. Our experiments with many datasets indicate that this parameter is small compared to the number of templates that FS-FAM creates during its training phase. As a result, the number of times that a given input pattern **I** passes through the code will be:

$$\text{Time}(\mathbf{I}) = \text{O}(\Gamma \times |\text{templates}|) \tag{8}$$

Also, under the unrealistic condition that the number of templates does not change during training it is easy to see that the time complexity of the algorithm is:

$$\text{Time}(\text{FS-FAM}) = \text{O}(\Gamma \times PT \times |\text{templates}|) \tag{9}$$

Usually for a fixed database the FS-FAM algorithm achieves a certain *compression ratio*. This means that the number of templates created is actually a fraction of the number of patterns PT in the training set

$$|\text{templates}| = \kappa PT \tag{10}$$

and

$$\text{Time}(\text{FS-FAM}) = \text{O}(\Gamma PT \kappa PT) = \text{O}(\kappa \Gamma PT^2) \tag{11}$$

Note that the parameter κ is also an unspecified numerical value. As it was the case with Γ, the introduction of κ as an unspecified constant helps us in deriving an analytical formula for the on-line complexity of FS-FAM. The actual value of the parameter κ, defined in the above equation, depends on $\bar{\rho}_a$, α and obviously on the training set that is presented to FS-FAM. Assuming that $\bar{\rho}_a$ and α are kept fixed, the value of κ decreases the more redundant data the training set contains. For a fixed training set, the value of κ increases as α, or $\bar{\rho}_a$, or both increase.

Now if we divide the training set into p partitions this will reduce the number of patterns in each partition to (PT/p) and the number of templates in each partition to $(\kappa PT/p)$, on the average. On a sequential machine the speedup obtained by partitioning the training set into p subsets will be proportional to

$$\frac{T^{\text{sequential}}}{T_p^{\text{sequential}}(\text{Partitions} = p)} \tag{12}$$

which using our assumptions above simplifies to

$$\frac{T^{\text{sequential}}}{T_p^{\text{sequential}}(\text{Partitions} = p)} = \frac{\kappa \Gamma PT^2}{p\Gamma \frac{\kappa PT}{p} \frac{PT}{p}} = p \tag{13}$$

and on a parallel machine with p processors the speedup will be proportional to

$$\frac{T^{\text{sequential}}}{T_p^{\text{parallel}}(\text{Partitions} = p)} \tag{14}$$

and again with our assumptions above simplifies to:

$$\frac{T^{\text{sequential}}}{T_p^{\text{parallel}}(\text{Partitions} = p)} = \frac{\kappa \Gamma PT^2}{\Gamma \frac{\kappa PT}{p} \frac{PT}{p}} = p^2 \tag{15}$$

The previous speedup (i.e. Eqs. (13) and (15)) still hold even if we do not assume that the number of times that the repeat loop is executed (i.e. the parameter Γ) is fixed. In that case we expect Γ to get smaller as the number of templates that a pattern has to go through decreases. So we expect the Γ in the numerator of Eqs. (13) and (15) to be larger than the Γ in the denominator of these equations; thus resulting in higher than the $p|p^2$ speedup that Eqs. (13) and (15) predict. Furthermore, we also assumed that the parameter κ is fixed. In reality, it is expected that the parameter κ will become smaller as the size PT and the number of templates is reduced. So it is reasonable to expect speedups greater than p for p partitions run in sequence, or p^2 for p partitions run in parallel.

Furthermore, additional assumptions that were needed to make the speedup equations valid are as follows:

(1) The partitioning scheme is *well balanced* and distributes the learning task fairly amongst the different partitions.
(2) The partitioning scheme is not computationally expensive so as to outweigh its benefits.

2.3. Partitioned FS-FAM versus non-partitioned FS-FAM

It is worth mentioning how the data-partitioning works. Whatever approach is used to partition the data in the training set will lead us into a collection of smaller training sets. Each one of these sets will be used to independently train a different FS-FAM network. The resulting collection of trained FS-FAM partitions is what we refer to as hFS-FAM for reasons that will become apparent in Section 3. In the performance phase, a test input is presented to hFS-FAM and this test input activates only the templates of the partition to which the input pattern belongs. This is accomplished by finding the Hilbert space index of the test pattern and then matching it with the Hilbert space-filling curve indices of the training patterns associated with one of the trained FS-FAMs in the hFS-FAM collection. Hence, a test input pattern is presented only to one of the trained FAMs in the hFS-FAM collection. The most active category that passes the vigilance criterion will produce the predicted label of the input pattern. It is apparent from the above statements, that the hFS-FAM and the single FS-FAM trained on the original dataset are two distinct methods for solving the same classification problem. Our intent, with hFS-FAM, in addition to achieving convergence speedup compared to FS-FAM, is to create a trained hFS-FAM whose size is not larger than the size of FS-FAM, and whose generalization

performance is comparable to the generalization performance of FS-FAM.

3. Space-filling curves

A space-filling curve is a mapping from a unit hypercube $[0,1]^{M_a}$ to the unit interval [0,1]. Mokbel and Aref (2001) describe them as a 'thread that goes through all the points in a space but visiting every point only once'. A space-filling curve S is defined to as the limit of an infinite sequence of curves $\{\mathcal{S}_m^{M_a} : m \in 0,1,\ldots\}$ so that $\mathcal{S} \stackrel{\text{def}}{=} \lim_{m\to\infty} \mathcal{S}_m^{M_a}$, where $\mathcal{S}_m^{M_a}$ is the mth-order approximation of the space-filling curve $\mathcal{S}$ in the M_a-dimensional space. Every mth-order space-filling curve approximation has a finite number of segments and connects a finite number of points in the multi-dimensional space. The *grid size* N of the space-filling curve will be the number of divisions into which each of the M_a dimensions is split.

There are many space-filling curves available, amongst them we have the Peano curve, the Z curve, the Hilbert curve, the Sweep, the scan and the gray curves. An M_a-dimensional space-filling curve with grid size N connects N^{M_a} points and has $N^{M_a}-1$ segments. Fig. 2 shows the Sweep and Peano space-filling curves, respectively. The grid size in these examples is 4, the number of dimensions $M_a=2$, the number of points that they connect is $4^2=16$, and the number of segments is 15.

A curve $\mathcal{S}$ is space-filling iff:

$$\mathcal{S} \stackrel{\text{def}}{=} \lim_{m\to\infty} \mathcal{S}_m^{M_a} = [0,1]^{M_a} \tag{16}$$

The above equation states that a space-filling curve is such that, as its order becomes larger and larger, it ends up connecting more and more points in the space (of dimensionality M_a) that it is trying to represent, and eventually in the limit (as $m \to \infty$) it connects all the points in the M_a-dimensional space.

Peano was the first to use space-filling curves, and Hilbert generalized the definition to arbitrary number of dimensions. To be able to characterize their properties, Mokbel & Aref (2001) concentrate on the nature of the segments that connect adjacent points in the space-filling curve by cataloging them as either a *jump*, *contiguity*, *reverse*, *forward* or *still*. Different applications will require different space-filling curves. If, for example, we wanted to access a database by an index in which the order is relevant then a space-filling curve that preserves the order of the dimension will probably be best (low *reverse*, i.e. the Peano curve).

3.1. The Hilbert space-filling curve

Moon et al. (2001) concentrates on the Hilbert space-filling curve (HSFC) and show that for range queries the HSFC curve produces the least number of splits in an index. This result is a consequence of the property that the HSFC does not have any jumps, it is continuous, and that it does not have a bias towards any dimension.

We will denote the mth-order approximation of the M_a-dimensional Hilbert space-filling curve as $\mathcal{H}_m^{M_a}$. Examples of the first four approximations of the two-dimensional Hilbert space-filling curve can be seen in Figs. 3 and 4. The mth-order approximation $\mathcal{H}_m^{M_a}$ of the HSFC has a grid size of $N=2^m$. In practice, $\mathcal{H}_m^{M_a}$ divides the M_a-dimensional space into 2^{mM_a} boxes and orders them in a contiguous sequence. For a more detailed exposition of the clustering properties of this curve we refer the reader to Moon et al. (2001).

3.2. The Hilbert space-filling curve index calculation

There are various algorithms for calculating the Hilbert index of a given M_a-dimensional data point. The one we present here is based on Lawder (2000) which is a modification of an iterative algorithm originally found in

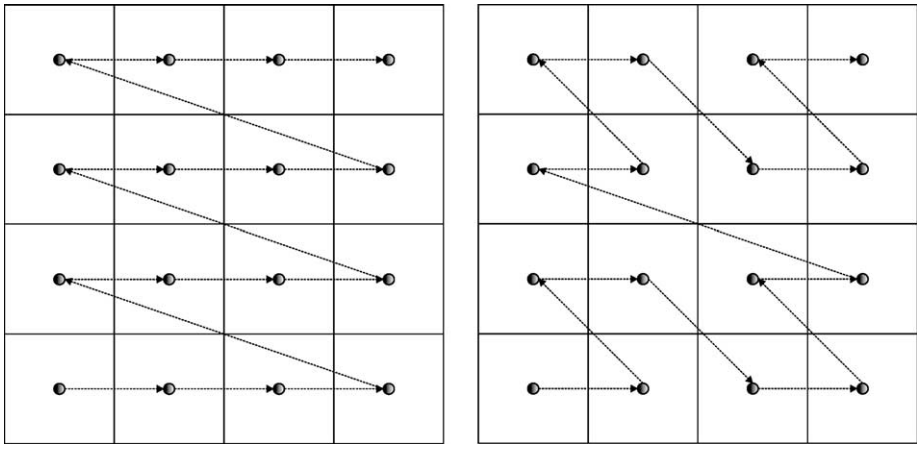


Fig. 2. Second-order Sweep and Peano space-filling curves with a grid size of $N=4$, $2^N=16$ partitions and $2^N-1=15$ line segments each.

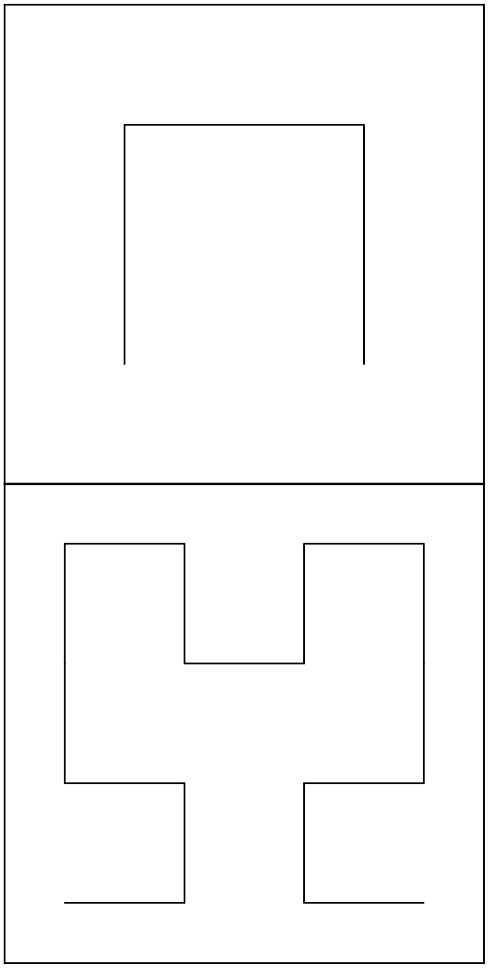


Fig. 3. First and second-order approximations of the two-dimensional Hilbert space-filling curve.

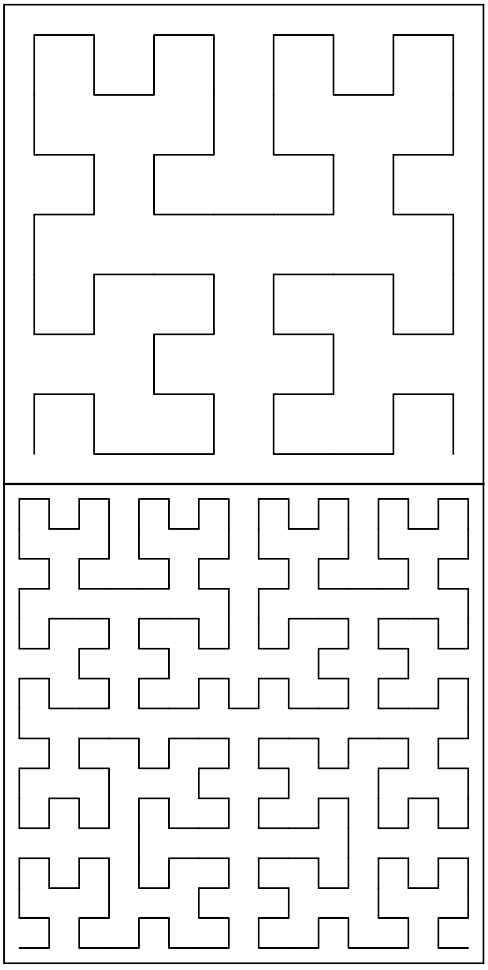


Fig. 4. Third and fourth-order approximations of the two-dimensional Hilbert space-filling curve.

Butz (1971). This is a table driven algorithm. It assumes that we are mapping binary numbers into binary numbers, and the precision of the mapping is limited by the order of the approximation m. Any precision can be achieved by increasing the approximation order m. Also this algorithm assumes that the number of bits in the full representation of the M_a-dimensional data point is the same as the number of bits in the resulting key $r \in [0, 1)$. All operations are fully reversible, and as a result, the HSFC is a one-to-one mapping.

The algorithm requires the following definitions.

- M_a: number of dimensions
- m: the order of approximation
- N: the number of bits in a *derived-key*, $N = mM_a$
- r: an N-Bit binary Hilbert *derived-key* expressed as a real number in the range [0, 1).
- *byte*: a word containing M_a bits
- $\gamma_j^i : i \in \{1, \ldots, m\}, j \in \{1, \ldots, M_a\}$ a binary digit in r such that

  $r = 0 \cdot \gamma_1^1 \gamma_2^1 \ldots \gamma_{M_a}^1 \gamma_1^2 \gamma_2^2 \ldots \gamma_{M_a}^2 \gamma_1^3 \ldots \gamma_{M_a}^m$

- γ^i: ith binary byte in r, $\gamma^i = \gamma_1^i \gamma_2^i \ldots \gamma_j^i \ldots \gamma_{M_a}^i$
- a_j: a coordinate in dimension j of the point

  $(a_1, a_2, \ldots, a_j, \ldots, a_{M_a})$

  whose *derived-key* is r, (each $a_j \in [0, 1)$)
- α_j^i: a binary digit in a coordinate a_j

  $a_j = 0 \cdot \alpha_j^1 \alpha_j^2 \ldots \alpha_j^i \ldots \alpha_j^m$

- α^i: a concatenation of all the ith entries of the a_j's

  $\alpha^i = \alpha_1^i \alpha_2^i \ldots \alpha_{M_a}^i$

- *principal position*: the least significant position of a byte that is different from the last position of the byte. If all positions are equal then the principal position is M_a (the first or most significant bit is bit 1).
- *parity*: number of bits in a byte that are equal to 1.

Given these definitions we can succinctly state the HSFC mapping as

$$\mathcal{H}_m^{M_a}(a_1, a_2, \ldots, a_{M_a}) = 0 \cdot \gamma^1 \gamma^2 \ldots \gamma^m$$

where the γ^i's are calculated using the following algorithm:

HILBERT$((a_1, a_2, \cdots, a_{M_a}), m)$
1 $\omega^0 \leftarrow (0, 0, \cdots, 0)$
2 $\tilde{\tau}^0 \leftarrow (0, 0, \cdots, 0)$
3 **for** $i \in \{1, 2, \ldots, m\}$
4 **do**
5 $\omega^i \leftarrow \omega^{i-1} \oplus \tilde{\tau}^{i-1}$
6 $\tilde{\sigma}^i \leftarrow \alpha^i \oplus \omega^i$
7 $\sigma^i \leftarrow$ shift $\tilde{\sigma}^i$ *left* circular $\Sigma_{k=1}^{i-1}(J_k - 1)$ times
8 $\gamma^i \leftarrow (\gamma_1^i = \sigma_1^i,\ \gamma_2^i = \sigma_2^i \oplus \gamma_1^i,\ \cdots,\ \gamma_{M_a}^i = \sigma_{M_a}^i \oplus \gamma_{M_a-1}^i)$
9 $J_i \leftarrow$ principal position of γ^i
10 $\tau^i \leftarrow$ a byte of M_a bits obtained by complementing σ^i in the M_a^{th} position
11 and iff it is of odd parity then complement it in the principal position
12 $\tilde{\tau}^i \leftarrow$ shift τ^i *left* circular $\Sigma_{k=1}^{i-1}(J_k - 1)$ times
13 **return** $0.\gamma^1\gamma^2\cdots\gamma^m$

Here, $\oplus$ is taken to mean the bitwise Exclusive-Or (XOR) operation. Notice that all operations can be performed in constant time, and if we fix the order of approximation m we can safely state that $\mathcal{H}_m^{M_a}(\cdot)$ is a constant time operation. In practice, the efficiency of the XOR operation makes the time spent in the calculation of the Hilbert index negligible even for large databases.

3.3. The Hilbert space-filling curve for data-partitioning in FS-FAM

The FS-FAM algorithm is a distance based algorithm in which all dimensions are treated equally. Fuzzy ART (unsupervised learning), on which FS-FAM is based, is a clustering algorithm that uses the distance function as a means of selecting its templates. Our interest in providing a data-partitioning method for the FS-FAM algorithm made us contemplate different options. Naïvely dividing the space into hyper-boxes has the disadvantage of having to decide which dimension to select. Using all dimensions is not viable in high-dimensional spaces, since the amount of partitions would be at least 2^{M_a} where M_a is the number of dimensions. The entropy measure used in decision trees can be used here to select dimensions and split points (Quinlan, 1993) per dimension. However, this approach, which gives priority to some dimensions over others, runs counter to FS-FAM's learning process in which, using an L_1 distance function, all dimensions are treated impartially. On the other hand, the Hilbert space-filling curve has been successfully used by Lawder (2000) for distance queries. Moon et al. (2001) prove that the Hilbert space-filling curve does not have a bias towards any dimension. We concentrate on the HSFC because its properties make it more compatible with the characteristics of the FS-FAM algorithm. Our claim is that the M_a-dimensional distance function is best preserved by a space-filling curve like the HSFC. Points that are close in the index will be close in the M_a-dimensional space (the converse is not necessarily true, though).

Our approach is as follows: we take the set of training pairs $(\mathbf{I}^r, \mathbf{O}^r)$, apply the Hilbert index $r = \mathcal{H}_m^{M_a}(\mathbf{a})$, where $\mathbf{a}$ is the non-complement coded part of $\mathbf{I} = (\mathbf{a}, \mathbf{a}^c)$. The resulting values are added to an index file and sorted. Once sorted, the index is split into p contiguous and equally sized partitions, each partition is processed independently. The complexity of the partitioning operation is equal to the complexity of the sorting algorithm used. For any reasonable sorting algorithm this is $O(PT \log(PT))$ and therefore does not add to the complexity of the FS-FAM learning process itself (which was found to be at least $O(PT^2)$). In our paper, we used the quicksort sorting algorithm.

The p partitions obtained will be completely balanced in the number of patterns they process, although they may have different number of templates depending on the complexity of the classification task in each partition. On the experimental level we found that the calculation of a fifth-order Hilbert index for 581,012 patterns of dimensionality 12 of the Forest Covertype database (see Section 4) took about 2 s. This is a negligible amount of time compared to the training time of the FS-FAM algorithm on the same data. We also found that a fifth-order curve for the Forest Covertype database and the Gaussian databases was sufficient to map every point of the training dataset to a unique Hilbert index. This could also be a rule of thumb of choosing an order for the Hilbert curve when we are confronted with other datasets (other than the ones used in this paper).

4. Design of experiments

Experiments were conducted on three databases: one real-world database (Forest Covertype) and two artificially-generated databases (16-dimensional, two-class Gaussianly distributed data with 5 and 15% overlap between classes).

In particular, the first database used for testing hFS-FAM versus FS-FAM was the Forest Covertype database provided by Blackard (1999), and donated to the UCI Machine-Learning Repository (University of California, Irvine, 2003). The database consists of a total of 581,012 patterns each one associated with 1 of 7 different forest tree cover types. The number of attributes of each pattern is 54, but this number is misleading since attributes 11–14 are

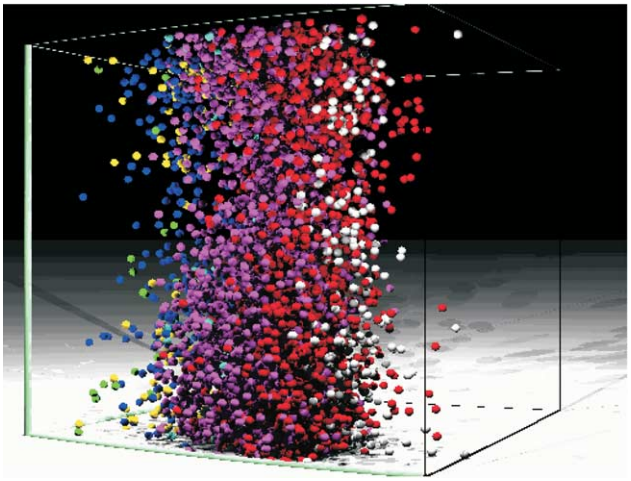


Fig. 5. A random sample of 5000 Forest Covertype data points out of the available 581,012 data points is shown. The data points are projected to the first three dimensions of the database. Different colors for the data points represent different class labels. (For interpretation of the reference to color in this legend, the reader is referred to the web version of this article.)

actually a binary tabulation of the attribute `Wilderness-Area`, and attributes 15–54 (40 of them) are a binary tabulation of the attribute `Soil-Type`. The original database values are not normalized to fit in the unit hypercube. Thus, we transformed the data to achieve this. There are no omitted values in the data. For the purpose of generating the Hilbert index the binary attributes, `Wilderness-Area` and `Soil-Type` were re-packed and the database was treated as if it consisted of only 12 real valued attributes. In the FS-FAM training, the input patterns had their original dimension of 54. Patterns 1–512,000 were used for training. The test set for all trials were patterns 561,001–581,000. A visualization of the first three dimensions of the Forest Covertype database can be seen in Fig. 5. Different tones correspond to different classes. As it can be seen from the figure the class boundaries are quite complex. Classification performance of different machine-learning algorithms for this database has been reported in the range of 75% (Blackard, 1999).

Furthermore, we tested the hFS-FAM versus FS-FAM for simulated data (Gaussianly distributed data). The Gaussian data was artificially generated using the polar form of the Box–Muller transform with the R250 random number generator by Kirkpatrick and Scholl (1981). We generated two-class, 16-dimensional data. All the dimensions are identically distributed with the same mean μ and variance σ^2 except one. The discriminating dimension has offset means so that the overlap between the Gaussian curves is at 5% for one database and at 15% for the other. Five hundred and thirty-two thousand patterns were generated for each Gaussian database. Five hundred and twelve thousand patterns were used for training; the remaining 20,000 patterns were used for testing.

Training set sizes of 1000×2^i, $i\in\{0, 1,\ldots,9\}$, that is 1000–512,000 patterns were used for the training of FS-FAM and hFS-FAM. The test set size was fixed at 20,000 patterns. The number of partitions varied from $p=1$ (FS-FAM) to $p=32$ (hFS-FAM). Partition sizes were also increased in powers of 2.

To avoid additional computational complexities in the experiments (beyond the one that the size of the training set brings along) the values of the FS-FAM network parameters $\bar{\rho}_a$, and α were fixed (i.e. the values chosen were ones that gave reasonable results for the database of focus). In particular, we used a $\bar{\rho}_a$ value of 0.94 for the Forest Covertype database and a $\bar{\rho}_a$ value of 0.8 for the Gaussian databases. The value of the choice parameter α was chosen equal to 0.01 for all databases. For each database and for every combination of (p, PT) = (partition, training set size) values, we conducted 32 independent experiments (training and performance phases), corresponding to different orders of pattern presentations within the training set. As a reminder FS-FAM's performance depends on the values of the network parameters $\bar{\rho}_a$, and α, as well as the order of pattern presentation within the training set.

All the tests where conducted on the `SCEROLA` Beowulf cluster of workstations (Micikevicius, 2003) of the Institute for Simulation and Training. This cluster consists of 64 900 MHz machines running with 250 MBytes of RAM each. In the sequential implementation of hFS-FAM, one of these machines was used for the training of the p FS-FAMs. In the parallel implementation of hFS-FAM, p of these machines were used for the training of the p FS-FAMs (one machine for the training of one of the p FS-FAMs). Obviously, one of these machines was used for the training of the FS-FAM (this FS-FAM was trained on the entire training set). Since there is no communication between processors for the parallel implementation of the p FS-FAMs on p machines, the time required to train hFS-FAM on the parallel (Beowulf cluster) machine was taken to be the maximum time required to train any of the p FS-FAMs.

The metrics used to measure the performance of our Hilbert Space-Filling Curves partitioning approach (hFS-FAM) were:

(1) Classification performance of hFS-FAM compared with the classification performance of FS-FAM (Higher classification performance is better).
(2) Size of the trained hFS-FAM compared against the size of FS-FAM (smaller size is better).
(3) Speedup of hFS-FAM versus FS-FAM, calculated experimentally for the sequential and parallel implementations, respectively.

To calculate the speedup we used two measures, the first one is total CPU time of each test and the second one is the total number of iterations of the FS-FAM main loop (lines 8–17 of the pseudocode). This approach allowed us to check how closely the wall clock speedup values are correlated with the number of computations performed by the algorithm. Wall clock time refers to the time in milliseconds

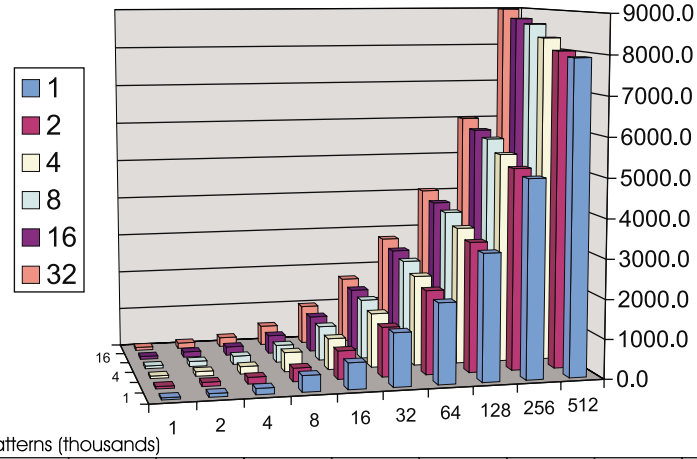
that it took for the algorithm to run from start of computation to finish. Wall clock time was measured using the system's clock. A discrepancy between these measures would mean that we might be obtaining speedup from other implementation dependent sources (like cache non-linearities or operating system dependent issues).

5. Experimental results

The first set of results reports on the network sizes created by hFS-FAM and FS-FAM. Note that the network size of hFS-FAM is the sum of the sizes of all the FS-FAM networks trained on the partitions of the data produced by the HSFC data-partitioning technique. On the other hand, the size of FS-FAM is the network size created by FS-FAM, when it is trained with all the data. In Fig. 6, we show a bar graph of the number of templates on the Z axis, the training set size on the X axis increasing from left to right (in thousands of patterns), and the number of partitions of the training dataset on the Y axis increasing from front to back. All the following graphs, unless otherwise stated, have this format with the measured variable of interest in the Z dimension and the controlled variables training set size and number of partitions in the X and Y dimensions, respectively. It is evident from this graph that the number of data partitions, up to the quantity tested, does not significantly affect the size of hFS-FAM. In other words, we see that the ratio of the number of templates (representing the size of the ART architecture) in hFS-FAM versus the number of templates in FS-FAM is slightly higher than 1. Nevertheless, it is worth noting that hFS-FAM consistently creates larger size ART architectures compared to the sizes of the architectures that FS-FAM creates. The fact that the number of data partitions has little effect on the size of the hFS-FAM architecture created is also true for the Gaussian data, and confirmed in Fig. 7.

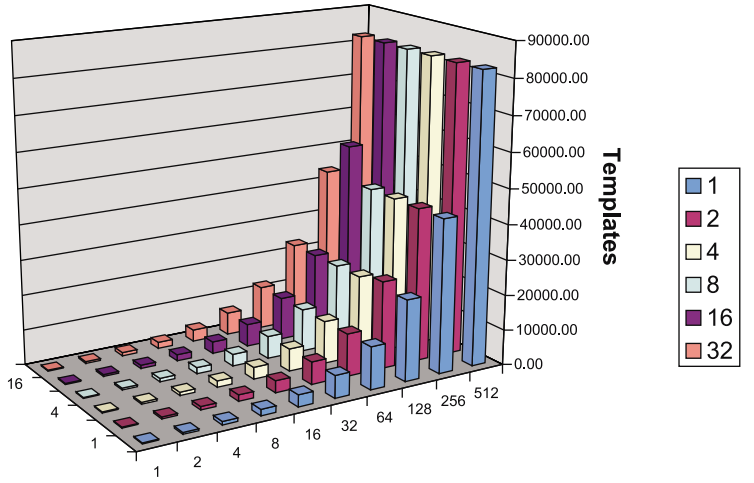
The generalization performance of the forest type database can be seen in Fig. 8. The classification performance of hFS-FAM is very similar with the classification performance of FS-FAM. In fact, the classification performance curve for the hFS-FAM is smoother than that of FS-FAM. Also, the hFS-FAM attained a slightly better classification than FS-FAM, reaching a peak of 76.63% for 32 data-partitions and 512,000 patterns. The Tree Covertype database classification performance consistently improves up to 512,000 patterns. This phenomenon clearly indicates that there is useful information in all of the data-points of the Forest Covertype database. Hence, training with more and more data-points from this dataset consistently improves the trained network's generalization performance on an independent test set. This behavior is not observed with the Gaussian artificial data (5 or 15% data), whose classification performance peaks at 32,000 patterns (Fig. 9). Beyond this number of training patterns the classification performance graph is flat. Also, classification improved considerably by partitioning regardless of the amount of patterns used for training. This is a consequence of the artificial nature of the data since the best split point for the classes coincides with the first split point obtained by Hilbert partitioning.

The turn-around time of the Forest Covertype data is presented in Fig. 10. We can compare this graph with the graph counting the number of iterations in Fig. 11 and observe that their shape is almost identical. Also, the difference in training time for the algorithm with 512,000 patterns and 32 partitions (26 s) and FS-FAM (4 h 7 min) is very dramatic.



Partitions	1	2	4	8	16	32	64	128	256	512
1	50.0	90.4	156.8	410.3	663.9	1347.9	2031.8	3208.6	5004.7	7923.5
2	60.0	103.3	170.5	341.1	715.9	1257.3	2129.4	3303.8	5124.7	8037.5
4	63.0	117.3	188.4	489.4	790.4	1384.4	2298.4	3498.7	5364.3	8322.0
8	70.0	129.7	211.5	438.0	877.9	1528.3	2505.8	3755.0	5660.8	8637.6
16	74.0	138.2	220.7	463.5	920.2	1592.3	2609.3	3866.0	5789.8	8750.6
32	80.0	144.6	237.3	497.4	989.5	1698.7	2761.5	4055.4	6013.1	8986.0

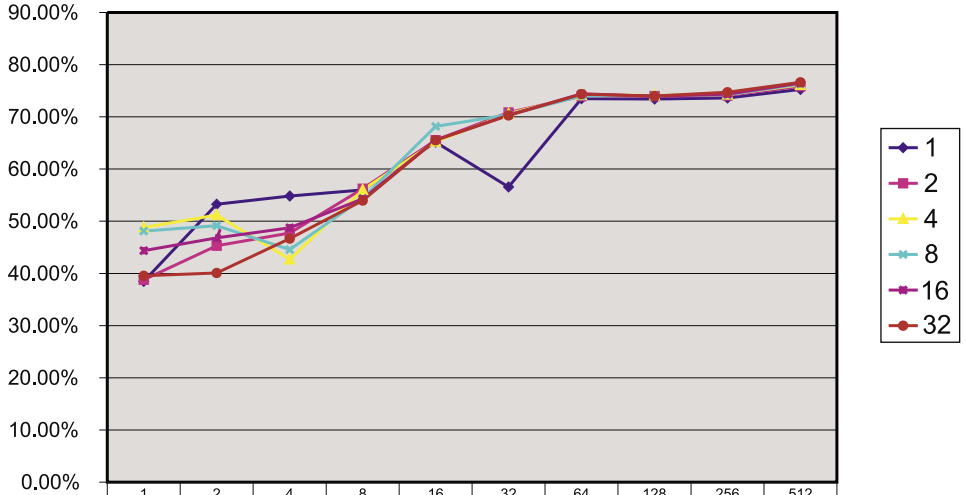
Fig. 6. Number of templates in hFS-FAM and FS-FAM trained with the Covertype Data; X axis shows thousands of training patterns, Y axis shows number of partitions and Z axis shows number of templates.



Partitions \ Patterns (thousands)	1	2	4	8	16	32	64	128	256	512
1	283.00	520.00	950.19	1781.22	3366.19	6401.75	12157.97	23099.66	43727.59	82609.69
2	283.00	520.00	950.19	1781.22	3366.19	6401.66	12158.06	24899.47	43729.19	82506.75
4	283.00	520.03	949.41	1781.88	3366.59	6401.53	12156.84	23101.50	43724.84	82498.59
8	275.19	520.03	949.53	1782.00	3367.03	6402.00	12156.16	23101.34	43722.59	82498.53
16	283.00	520.00	949.47	1782.34	3367.09	6401.69	12157.72	23100.72	53664.41	82506.00
32	283.00	520.00	949.69	1781.88	3366.88	6401.28	12156.28	23099.66	43723.78	82505.31

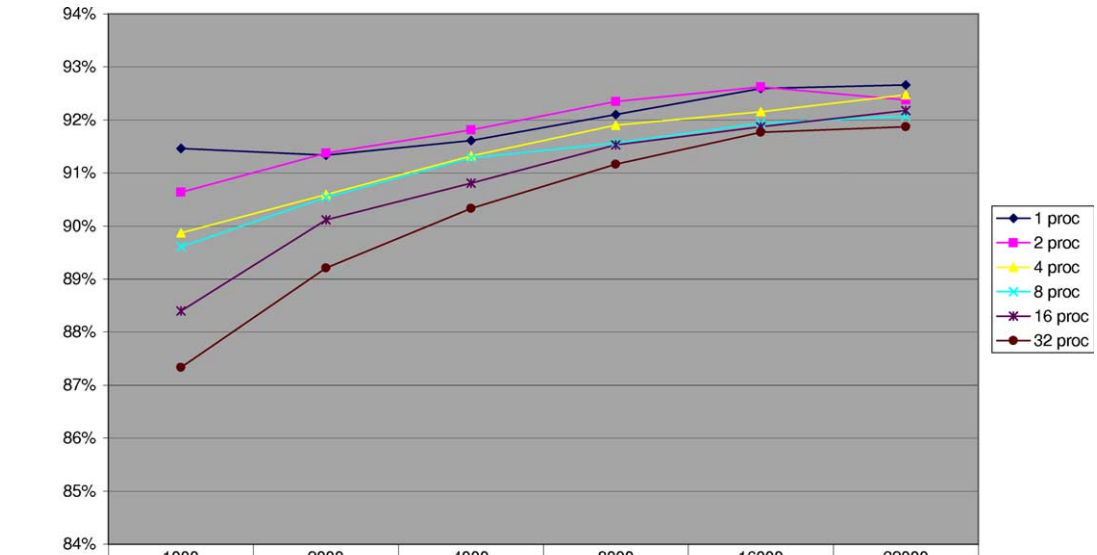
Fig. 7. Number of templates in hFS-FAM and FS-FAM trained with the 5% overlap Gaussian data; *X* axis shows thousands of training patterns, *Y* axis shows number of partitions and *Z* axis shows number of templates.

Fig. 12 shows the speedup in turn-around time of hFS-FAM with p partitions running in parallel. The best speedups obtained were in the order of 565. If we look at the table of this figure for the speedup values when the database size is 512,000, we can see that the numbers seem to increase quadratically as the number of partitions p is increased from 1 up to 16. At $p=32$, there seems to be an anomaly and a dip in the speedup. The speedup measured using number of iterations can be seen in Fig. 13, where the best speedup is in the order of 100 and is not obviously quadratic. Nevertheless, we provide the log–log graph of the same data in Fig. 14. This figure clearly shows a slope on the graph close to 2 for the largest (and most representative for our purposes) training set size, which indicates that



	1	2	4	8	16	32	64	128	256	512
1	38.48%	53.26%	54.82%	56.00%	65.15%	56.58%	73.50%	73.41%	73.62%	75.25%
2	38.77%	45.32%	47.74%	56.22%	65.52%	70.84%	73.96%	73.97%	74.09%	75.84%
4	48.81%	51.20%	42.70%	56.00%	65.26%	70.63%	74.19%	74.13%	74.30%	76.13%
8	48.13%	49.16%	44.58%	54.39%	68.17%	70.41%	74.00%	74.07%	74.22%	76.28%
16	44.35%	46.80%	48.74%	54.30%	65.56%	70.33%	74.43%	73.94%	74.37%	76.47%
32	39.55%	40.08%	46.66%	53.96%	65.51%	70.27%	74.34%	73.99%	74.72%	76.63%

Fig. 8. Classification (generalization) performance of hFS-FAM and FS-FAM with the Forest Cover database; *X* axis shows thousands of training patterns, *Y* axis shows the % of correctly classified patterns in the test set.

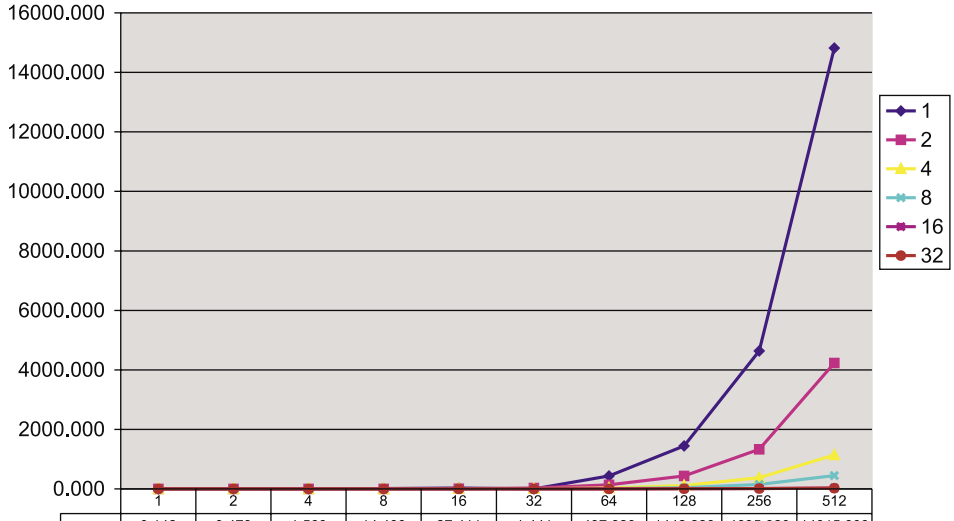


	1000	2000	4000	8000	16000	32000
1 proc	91%	91%	92%	92%	93%	93%
2 proc	91%	91%	92%	92%	93%	92%
4 proc	90%	91%	91%	92%	92%	92%
8 proc	90%	91%	91%	92%	92%	92%
16 proc	88%	90%	91%	92%	92%	92%
32 proc	87%	89%	90%	91%	92%	92%

Fig. 9. Classification (generalization) performance of hFS-FAM and FS-FAM for the Gaussian 5% Overlap Data; *X* axis shows thousands of training patterns, *Y* axis shows the % of correctly classified patterns in test set.

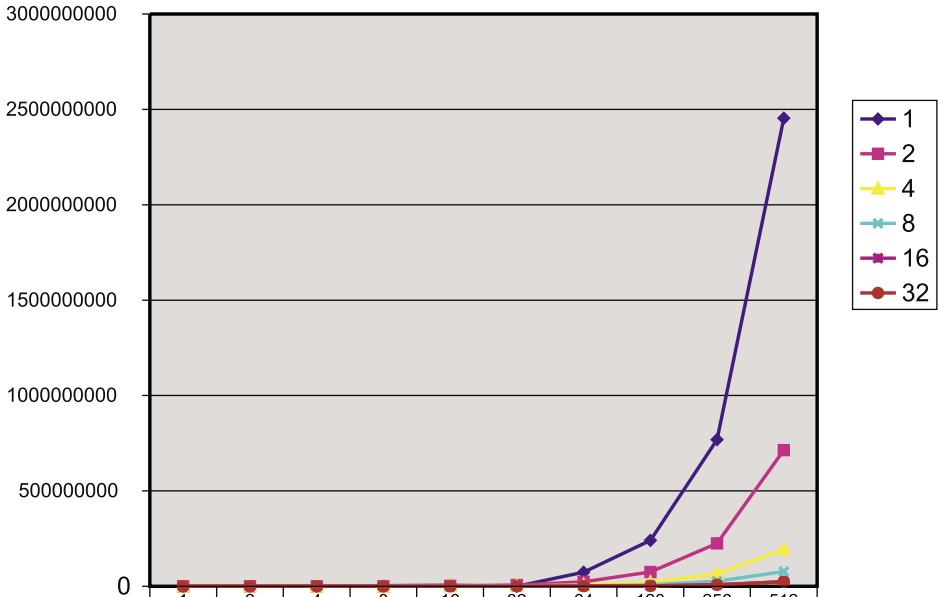
the speedup in terms of the number of iterations is as predicted close to p^2 for the parallel implementation. We speculate that the difference in speedup obtained between these two measures is due to platform restrictions that slow the sequential FS-FAM algorithm when the size of the database is too large.

The speedup measured in iterations for the same data using a sequential processing machine can be seen in Fig. 15. We can see from this figure that the speedup for a single processor is in the order of p. Again the speedup observed from 16 to 32 partitions does not follow the same progression for this data.



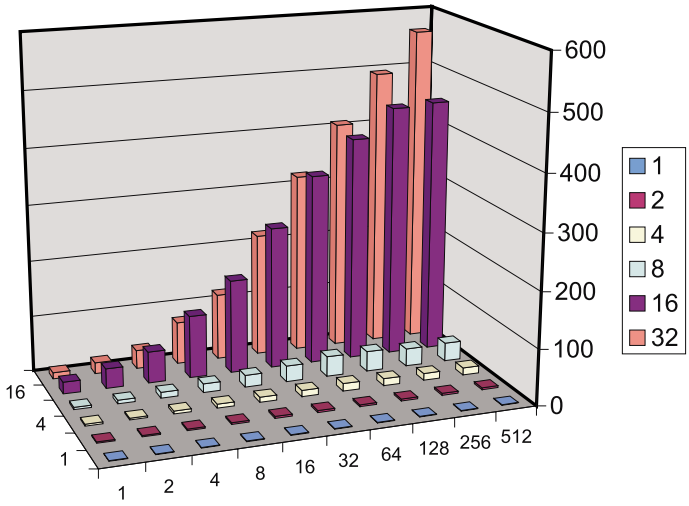
	1	2	4	8	16	32	64	128	256	512
1	0.142	0.473	1.568	14.489	27.411	1.441	437.820	1442.228	4635.820	14815.803
2	0.063	0.201	0.612	2.270	7.627	31.968	135.418	436.814	1330.048	4228.651
4	0.043	0.157	0.301	1.659	3.018	9.870	32.087	114.338	374.651	1140.852
8	0.036	0.075	0.150	0.390	1.298	3.802	12.174	40.219	147.358	449.125
16	0.007	0.013	0.028	0.066	0.164	0.452	1.288	3.614	10.338	32.707
32	0.012	0.024	0.047	0.103	0.232	0.551	1.367	3.531	9.376	26.213

Fig. 10. hFS-FAM versus FS-FAM parallel partitioning absolute elapsed time in seconds for the Covertype Data; *X* axis shows thousands of training patterns, *Y* axis shows the number of seconds.



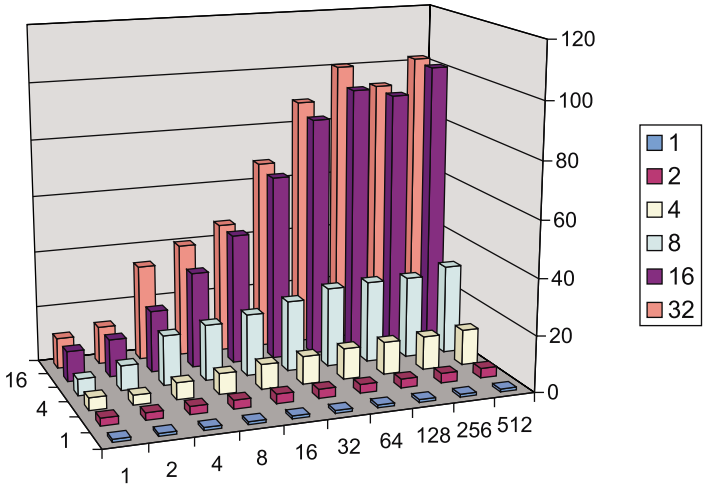
	1	2	4	8	16	32	64	128	256	512
1	31513	106415	352504	3E+06	6E+06	134951	7E+07	2E+08	8E+08	2E+09
2	11657	40832	122187	469456	2E+06	6E+06	2E+07	7E+07	2E+08	7E+08
4	7096	30651	58928	353780	648631	2E+06	7E+06	2E+07	6E+07	2E+08
8	5356	12106	19910	65649	261839	794535	3E+06	8E+06	3E+07	8E+07
16	2913	7986.6	16088	47387	123934	352401	866473	3E+06	8E+06	2E+07
32	2913	7986.6	10390	32985	123590	352349	831808	2E+06	8E+06	2E+07

Fig. 11. hFS-FAM versus FS-FAM parallel partitioning absolute number of iterations for the Covertype Data; *X* axis shows thousands of training patterns, *Y* axis shows the number of iterations.



	1	2	4	8	16	32	64	128	256	512
1	1	1	1	1	1	1	1	1	1	1
2	2.24084788	2.3511748	2.56077995	3.07729603	3.59381211	3.41345529	3.23309847	3.30169614	3.48545365	3.50367135
4	3.27985122	3.01738471	5.21308764	7.14803269	9.08297774	11.3639602	13.6449427	12.6137074	12.3736932	12.9866177
8	3.98756019	6.31102046	10.4348825	15.7777386	21.1205946	28.5416291	35.9626636	35.8594421	31.459568	32.9881161
16	21.0862284	35.3848352	56.2727084	111.745463	167.218218	253.549296	339.880375	399.083307	448.436003	452.988533
32	11.3818677	19.7871519	33.0698528	75.6583199	118.246787	219.253637	320.260487	408.443315	494.437882	565.200076

Fig. 12. hFS-FAM versus FS-FAM parallel partitioning speedup on the Covertype Data using the elapsed time in seconds to calculate speedup; *X* axis shows thousands of training patterns, *Y* axis shows number of partitions and *Z* axis shows the parallel speedup.



	1	2	4	8	16	32	64	128	256	512
1	1	1	1	1	1	1	1	1	1	1
2	2.70335421	2.60613375	2.88494441	3.14324113	3.40153784	3.24398659	3.08643535	3.21508314	3.41203687	3.43843016
4	4.44095265	3.471831	5.98190071	7.41074027	8.83957984	9.9229699	11.00636	11.5825847	11.8993436	12.6690953
8	5.88368185	8.79030434	17.705088	19.8012914	21.8974948	25.0504714	28.2034481	28.9736828	29.0443754	31.7315716
16	10.818057	13.3242399	21.9105562	34.0871443	46.2637323	65.7674264	85.2711204	94.8683111	91.9605498	101.166108
32	10.818057	13.3242399	33.9263073	40.1593525	46.3923976	67.6085845	88.8247713	100.753257	92.9185287	102.063429

Fig. 13. hFS-FAM versus FS-FAM parallel partitioning speedup on the Covertype Data using the number of iterations to calculate speedup; *X* axis shows thousands of training patterns, *Y* axis shows number of partitions and *Z* axis shows the parallel speedup.

6. Conclusions

We observed that FS-FAM's training time tends to slow considerably when FS-FAM is applied to large classification problems. After analyzing the complexity of the algorithm, we conjectured that by partitioning the dataset we would theoretically be able to reach a speedup of p in the sequential machine and p^2 in an efficient parallel machine with sufficient number of processors. We proposed the use of Hilbert space-filling curves to enforce some sort of data-partitioning (the resulting scheme was called hFS-FAM). Experimental results on three databases confirmed our expectations. The classification performance of hFS-FAM is not affected, as compared to the classification performance of FS-FAM. The size of the resulting hFS-FAM is slightly higher than the corresponding FS-FAM size. Finally, and most importantly, the convergence time of hFS-FAM is improved linearly on the sequential machine and quadratically on the parallel machine. Nevertheless, there is still room for improvement.

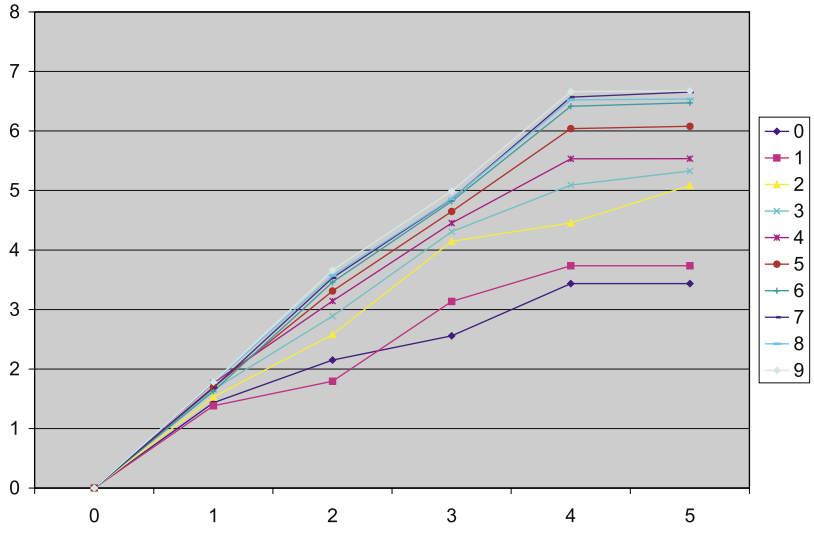
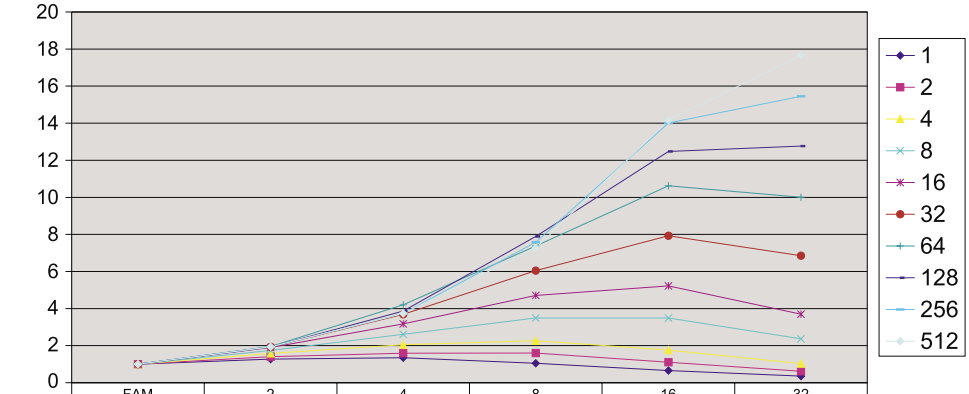


Fig. 14. hFS-FAM versus FS-FAM parallel partitioning speedup on log–log graph using the number of iterations to calculate speedup for the Forest Covertype Database; *X* axis shows the logarithm of the number of partitions, and *Y* axis shows the $\log_2$(speedup). Database sizes are represented as 1000×2^i, $i\in\{0,2,\ldots,9\}$.



FAM	2	4	8	16	32
1	1	1.274	1.347	1.045	0.659	0.356
2	1	1.402	1.592	1.601	1.106	0.618
4	1	1.582	2.048	2.262	1.759	1.033
8	1	1.739	2.610	3.487	3.492	2.364
16	1	1.897	3.171	4.711	5.226	3.695
32	1	1.928	3.688	6.044	7.923	6.852
64	1	1.960	4.205	7.378	10.621	10.008
128	1	1.930	3.866	7.890	12.471	12.764
256	1	1.927	3.730	7.581	14.014	15.451
512	1	1.936	3.733	7.264	14.156	17.663

Fig. 15. hFS-FAM versus FS-FAM sequential partitioning speedup on log–log graph using the number of iterations to calculate speedup for the Forest Covertype Database; *X* axis shows thousands of training patterns, *Y* axis shows the speedup.

Analysis of the workload balance in the parallel machine indicates that not all processes are being utilized to the maximum level. This is because even though the number of patterns processed by each partition is the same, the number of templates varies considerably from one processor to another depending on the complexity of the region it has to classify. We believe that combining our data-partitioning approach with a networking partitioning approach will help us achieve optimal workload balance in the parallel implementation of the algorithm. This is one of the directions for our future research.

Acknowledgements

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. CRCD:0203446 and by the National Science Foundation, under grant no. CCLI 0341601.

References

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In J. B. Bocca, M. Jarke, & C. Zaniolo (Eds.), *Proceedings of the twentieth international conference on very large databases, September 12–15* (pp. 487–499). Santiago, Chile: Morgan Kaufmann.

Anagnostopoulos, G. C., Bharadwaj, M., Georgiopoulos, M., Verzi, S. J., & Heileman, G. L. (2003). Exemplar-based pattern recognition via semi-supervised learning. In *International joint conference on neural networks* (pp. 2782–2787). Portland, Oregon: IEEE–INNS–ENNS.

Anagnostopoulos, G. C., & Georgiopoulos, M. (2001). *Ellipsoid ART and ARTMAP for incremental unsupervised and supervised learning. Proceedings of the IEEE–INNS–ENNS*, Vol. 2 (pp. 1221–1226). Washington, DC: IEEE–INNS–ENNS.

Blackard, J. A. (1999). *Comparison of neural networks and discriminant analysis in predicting forest cover types*. Unpublished doctoral dissertation, Department of Forest Sciences, Colorado State University.

Burrascano, P. (1991). Learning vector quantization for the probabilistic neural network. *IEEE Transactions on Neural Networks*, *2*, 458–461.

Butz, A. R. (1971). Alternative algorithm for Hilbert's space-filling curve. *IEEE Transactions on Computers*, *April*, 424–426.

Carpenter, G. A., Grossberg, S., Markuzon, N., Reynolds, J. H., & Rosen, D. B. (1992). Fuzzy ARTMAP: A neural network architecture for incremental learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, *3*(5), 698–713.

Carpenter, G. A., Grossberg, S., & Reynolds, J. H. (1991). Fuzzy ART: An adaptive resonance algorithm for rapid, stable classification of analog patterns. In *International joint conference on neural networks, IJCNN'91* (Vol. II, pp. 411–416). Seattle, Washington: IEEE–INNS–ENNS.

Carpenter, G. A., & Markuzon, N. (1998). ARTMAP-IC and medical diagnosis: Instance counting and inconsistent cases. *Neural Networks*, *11*, 793–813.

Carpenter, G. A., Milenova, B. L., & Noeske, B. W. N. (1998). Distributed ARTMAP: A neural network for fast distributed supervised learning. *Neural Networks*, *11*(2), 323–336.

Carpenter, G. A., & Ross, W. D. (1995). ART-EMAP: A neural network architecture for object recognition by evidence accumulation. *IEEE Transactions on Neural Networks*, *6*(5), 805–818.

Caudell, T. P., & Healy, M. J. (1999). Studies of generalization for the LAPART-2 architecture. In *International joint conference on neural networks* (Vol. 3, pp. 1979–1982). Washington, DC: IEEE–INNS–ENNS.

Charalampidis, D., Kasparis, T., & Georgiopoulos, M. (2001). Classification of noisy signals using Fuzzy ARTMAP neural networks. *IEEE Transactions on Neural Networks*, *12*(5), 1023–1036.

Gomez-Sanchez, E., Dimitriadis, Y. A., Cano-Izquierdo, J. M., & Lopez-Coronado, J. (2002). ARTMAP: Use of mutual information for category

reduction in Fuzzy ARTMAP. *IEEE Transcations on Neural Networks*, *23*(1), 58–69.

Heinke, D., & Hamker, F. H. (1995). Comparing neural networks: A benchmark on growing neural gas, growing cell structures and Fuzzy ARTMAP. *IEEE Transactions on Neural Networks*, *9*(6), 1279–1291.

Joshi, A., Ramakrishman, N., Houstis, E. N., & Rice, J. R. (1997). On neurobiological, neurofuzzy, machine learning, and statistical pattern recognition techniques. *IEEE Transactions on Neural Networks*, *8*(1), 18–31.

Kasuba, T. (1993). Simplified Fuzzy ARTMAP. *AI Expert*, *November*, 18–25.

Kerstetter, T. (1998). Recursively partitioning neural networks for radar target recognition. In *1998 IEEE world congress on computational intelligence* (pp. 3208–3212). Anchorage, AL.

King, R., Feng, C., & Shutherland, A. (1995). STATLOG: Comparison of classification algorithms on large real-world problems. *Applied Artificial Intelligence*, *9*(3), 259–287.

Kirkpatrick, S., & Stoll, E. (1981). A very fast shift-register sequence random number generator. *Journal of Computational Physics*, *40*, 517–526.

Koufakou, A., Georgiopoulos, M., Anagnostopoulos, G., & Kasparis, T. (2001). Cross-validation in Fuzzy ARTMAP for large databases. *Neural Networks*, *14*, 1279–1291.

Lawder, J. K. (2000). *Calculation of mappings between one and n-dimensional values using the Hilbert space-filling curve (Tech. Rep.)*. London, UK: School of CS and Information Systems, Brickwell University.

Lawder, J. K., & King, P. J. H. (2000). Using space-filling curves for multi-dimensional indexing. *Lecture notes in computer science 1832* .

Marriott, S., & Harrison, R. F. (1995). A modified Fuzzy ARTMAP architecture for the approximation of noisy mappings. *Neural Networks*, *8*(4), 619–641.

Mehta, M., Agrawal, R., & Rissanen, J. (1996). SLIQ: A fast scalable classifier for data mining. In *Extending database technology* (pp. 18–32). Avignon, France: Springer.

Micikevicius, P. (2003). *Scerola parallel cluster* (http://www.cs.ucf.edu/courses/cda5110/scerola/guidescerola.html).

Mokbel, M. F., & Aref, W. G. (2001). Irregularity in multi-dimensional space-filling curves with applications in multimedia databases. In *International conference on information and knowledge management*, November. Atlanta, GA: ACM.

Moody, J., & Darken, C. (1989). Fast learning in networks of locally-tuned processing units. *Neural Computation*, *1*, 281–294.

Moon, B., Jagadish, H., Faloutsos, C., & Saltz, J. H. (2001). Analysis of the clustering properties of the Hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering*, *13*(1).

Parrado-Hernandez, E., Gomez-Sanchez, E., & Dimitriadis, Y. A. (2003). Study of distributed learning as a solution to category proliferation in Fuzzy ARTMAP based neural systems. *Neural Networks*, *16*, 1039–1057.

Petridis, V., Kaburlasos, V. G., Fragkou, V. G., & Kehagais, A. (2001). Text classification using the σ-FLNMAP neural network. In *Proceedings of the international joint conference on neural networks* (Vol. 2, pp. 1362–1367). Washington, DC: IEEE–INNS–ENNS.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo, California: Morgan Kaufmann.

Shafer, J. C., Agrawal, R., & Mehta, M. (1996). SPRINT: A scalable parallel classifier for data mining. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, & N. L. Sarda (Eds.), *Proceedings of 22nd international conference on very large databases, VLDB* (pp. 544–555). Bombay, India: Morgan Kaufmann.

Simpson, P. K. (1992). Fuzzy Min–Max neural networks—Part 1: Classification. *IEEE Transactions on Neural Networks*, *3*(5), 776–786.

Skopal, T., Krátký, M., & Snášel, V. (2002). *Properties of space filling curves and usage with UB-trees*. High Fatra, Slovakia.

Specht, D. F. (1990). Probabilistic neural networks. *Neural Networks*, *3*, 109–118.

Taghi, M., Baghmisheh, V., & Pavesic, N. (2003). A fast simplified Fuzzy ARTMAP network. *Neural Processing Letters*, *17*, 273–316.

Traven, H. G. C. (1991). A neural network approach to statistical pattern classification by semiparametric estimation of probability density functions. *IEEE Transactions on Neural Networks*, *2*, 366–377.

University of California, Irvine (2003). *UCI repository of machine learning databases* (http://www.icf.uci.edu/mlearn/MLRepository.html).

Verzi, S. J., Heileman, G. L., Georgiopoulos, M., & Healy, M. H. (2001). Rademacher penalization applied to Fuzzy ARTMAP and Boosted ARTMAP. In *Proceedings of the IEEE–INNS–ENNS international joint conference on neural networks (IJCNN01)* (Vol. 2, pp. 1191–1196). Washington, DC.

Williamson, J. R. (1997). A constructive, incremental-learning network for mixture modeling and classification. *Neural Computation*, *9*, 1517–1543.