*Neural Processing Letters* **17:** 273–316, 2003.
© 2003 *Kluwer Academic Publishers. Printed in the Netherlands.*

# A Fast Simplified Fuzzy ARTMAP Network

MOHAMMAD-TAGHI VAKIL-BAGHMISHEH* and NIKOLA PAVEŠIĆ
*Laboratory of Artificial Perception, Systems and Cybernetics, Faculty of Electrical Engineering, University of Ljubljana, Slovenia. e-mail: vakil@luz.fe.uni-lj.si; nikola.pavesic@fe.uni-lj.si*

**Abstract.** We present an algorithmic variant of the simplified fuzzy ARTMAP (SFAM) network, whose structure resembles those of feed-forward networks. Its difference with Kasuba's model is discussed, and their performances are compared on two benchmarks. We show that our algorithm is much faster than Kasuba's algorithm, and by increasing the number of training samples, the difference in speed grows enormously.

The performances of the SFAM and the MLP (multilayer perceptron) are compared on three problems: the two benchmarks, and the Farsi optical character recognition (OCR) problem. For training the MLP two different variants of the backpropagation algorithm are used: the BPLRF algorithm (backpropagation with plummeting learning rate factor) for the benchmarks, and the BST algorithm (backpropagation with selective training) for the Farsi OCR problem.

The results obtained on all of the three case studies with the MLP and the SFAM, embedded in their customized systems, show that the SFAM's convergence in fast-training mode, is faster than that of MLP, and online operation of the MLP is faster than that of the SFAM. On the benchmark problems the MLP has much better recognition rate than the SFAM. On the Farsi OCR problem, the recognition error of the SFAM is higher than that of the MLP on ill-engineered datasets, but equal on well-engineered ones. The flexible configuration of the SFAM, i.e. its capability to increase the size of the network in order to learn new patterns, as well as its simple parameter adjustment, remain unchallenged by the MLP.

**Key words.** backpropagation with selective training, backpropagation with plummeting learning rate factor, comparative study, Farsi optical character recognition, multilayer perceptron, neural networks, simplified fuzzy ARTMAP

## 1. Introduction

Neural networks (NNs) have been used in a wide range of applications, including: pattern classification, pattern completion, function approximation, optimization, prediction, and automatic control. In many cases they even outperform their classical counterparts. In spite of different structures and training algorithms, all NNs perform essentially the same function: vector mapping. Putting it another way, all NN applications are special cases of vector mapping. The development of detailed mathematical models for NNs began in 1943 with the work of McCulloch and Pitts [1] and was continued by others.

*Corresponding Author.

## 1.1. THE FUZZY ARTMAP NETWORK

ART stands for 'Adaptive Resonance Theory', it was invented by Grossberg [2], in 1976, as a theory of human cognitive information processing. In 1987, in an attempt to solve the stability-plasticity dilemma[1], Carpenter and Grossberg introduced the first member of the ART family (ART1) [3]. However, because of its mathematical intricacy and because it was only useful for self-organized clustering of digital patterns, it did not gain much attention. Furthermore, in comparison with reintroduced backpropagation training for multilayer perceptron (MLP) it had no practical value.

The years between 1987 and 1992 saw the introduction of other members of the ART family [4–8] in the form of a wide variety of supervised and unsupervised NNs. The most advanced model of the ART family, fuzzy ARTMAP (FAM) [8], could handle both binary and analogue data in a supervised manner. The main drawback to the ART family networks, which prevented others from using them, was their intricacy: the inventors had introduced complicated architectures for their networks instead of presenting them as simple algorithms. This problem later was recognized by the inventors and they introduced algorithmic versions of their networks [9, 10]. Others [11–23] also presented modified models of ART family networks or simplified ones, an example of which is Kasuba's SFAM network [11], which is an incomparably simpler variant of the original FAM. However, Kasuba's model has some alteration from the original FAM network, which only slow down the training of the network. In this paper a genuine simplification of FAM network, which has been independently developed by the authors, is presented. Its differences with Kasuba's model are discussed theoretically, and their performances are compared on two benchmarks: circle-in-the-square and sphere-in-the-cube problems.

## 1.2. THE MULTILAYER PERCEPTRON (MLP) NETWORK

Among all the various NN structures the MLP has the highest practical value. It is a feed-forward layered network with one input layer, one output layer, and some hidden layers (Figure 3). The method for training the MLP is based on the minimization of a suitable cost function, and is called the backpropagation algorithm. The first version of this algorithm, which was based on the gradient descent method, was independently proposed by Werbos [24] and Parker [25], but only gained popularity after the publication of the seminal book by Rumelhart and McClelland [26]. Since then, many modifications have been put forward by others, and Jondarr [27] has reviewed 65 varieties of backpropagation algorithm. In this paper two variants of the backpropagation algorithm are presented: backpropagation with plummeting learning rate factor (BPLRF), and backpropagation with selective training (BST). The first one is evaluated on the benchmark problems, and the latter one is used in comparing MLP's performance against that of the SFAM on the Farsi OCR problem.

[1]While learning new data, forgetting already learnt data is called the stability-plasticity dilemma, which is the main drawback in most of the NN models.

### 1.3. SHORTCOMINGS OF THE COMPARATIVE STUDIES

Comparing different methods and algorithms that are designed to do a specific task, is common in all fields of science, and NN models and their training algorithms are no exception. When doing so, it is absolutely necessary to know exactly what is being compared with what and under which conditions. In addition, the report should provide all the necessary information for reproducing the results by others.

There is no simple answer to the question as to which feature extraction method or which NN model is the best for any particular application.

There are different stages in any pattern recognition system, in the simplest case these are: data acquisition, preprocessing, feature generation and classification. Different methods can be adopted for realizing any of these stages. Since all of these methods have some particular requirements, choosing a certain method for one stage limits, or dictates, the nature, or the output, of another stage or stages. Consequently, different steps cannot be designed independently.

For designing a state-of-the-art recognition system, more than one method should be considered for each stage, and by considering the requirements of each step, various combinations of different methods must be experimentally evaluated.

It must be emphasized that although there are similarities between NN models, in general they are different, and each of them has its own special features and requirements. Ignoring this fact by simply replacing classifiers in a system that is designed or optimized for a certain classifier could lead to invalid conclusions about the transcendence of the classifiers.

A search of the Inspec database, will reveal up to 42 papers that have compared the performances of the FAM network and the MLP network on various datasets from different fields. Among these papers, 17 comparative works [28–44] were examined by the authors. There are some common shortcomings in these works that can call their findings into question. In the hope that these pitfalls will be avoided in future studies, we have listed the most important of them below:

(1) Mixing up the name of the network model and the training algorithm. It should be pointed out that backpropagation is not a network, it is an algorithm with many variants for training many NN models, including multilayer perceptron.
(2) Some authors think that backpropagation is a very specific method and simply stating its name will be enough. It seems that they are unaware that at least 65 variants of the backpropagation algorithm exist [27].
(3) Using non-standard and vague terms for the name of the network model and causing confusion in the literature, examples include using: 'multilayer sigmoidal perceptron', 'multilayer perceptrons', 'backpropagation multilayer perceptron', 'backpropagation network', 'typical error backpropagation network', 'an elementary back propagation network' for 'multilayer perceptron'.

(4) Eliminating the details of the network's model, which could be a sign that the authors are unaware of the various parameters in the model or their importance. For instance, adding the threshold term to the net input of sigmoidal neuron must be examined and decided for every dataset, individually; because if it is added unnecessarily the training could be hindered considerably. Or adding the sigmoid prime offset[2] could accelerate the training by a factor of 100 in some cases. In these reports, it is rare to find any remark about the selected activity functions.

(5) Using commercial NN software without having access to the source codes and knowing the variants of their algorithms. It would be more appropriate if these works were to be called 'an evaluation study on a specific software'. Even in this case, these commercial software applications should have been compared with other rival software, on some standard datasets certified by authorized organizations. No authorized organization has certified these commercial software applications as the standard implementations of certain NN models or training algorithms.

(6) From the reported results, i.e. iteration of the backpropagation training algorithm up to 32000 epochs, we can conclude that not only have they used a very early variant of the backpropagation algorithm, which was introduced almost three decades ago and is quite inefficient, but also that they have failed to adjust the network properly. The oldest of these comparative works dates back to 1991, and there is no excuse for using the very primary algorithm at that time. Some of these works have even been dated as late as 2001.

(7) Using improper coding for the output vector, i.e. one output cell for a two class classification problem, instead of using one output cell for each class[3].

(8) Extracting unequal numbers of samples for various classes. Probably, this approach does not affect the performance of FAM or its impact is trivial, but its impact on the performance of the MLP can be awesome.

(9) No mention of the local minima trap, which is considered to be one of the major drawbacks of backpropagation algorithms. In these papers, there is hardly a word about the occurrence of the local minima problem, and it seems that these authors never encountered this problem. Whatever the case, this must be reported in the paper, explicitly.

(10) Using only one method for feature extraction, and comparing all the networks on the resulted data, heedless to the fact that some classification methods can perform better than others on a certain feature extraction method and the pertinent dataset. In fact, even the number of extracted features could have an impact on the results. In this case the obtained results on NN models are biased by the dataset and the feature extraction method.

[2]For an explanation of the sigmoid prime offset, see Section 4.1-Remark 4.

[3]For an explanation, see Section 4.1, Remark 3, and Footnote 8.

(11) Simply replacing classifiers in a pattern recognition system that has been designed, or optimized, for a specific classifier.
(12) Explaining and simulating the very complicated original FAM in a pure classification problem, even though its much simpler and faster variant, i.e. SFAM algorithm, was introduced in 1993.
(13) Failing to report the data extraction method in a reproducible way.
(14) Incomplete reporting of the obtained results. For instance, presenting only the number of training epochs, or simply stating that one network was faster or slower than another one, instead of providing quantitative data for both the simulation time and the number of training epochs. The reader should be provided with enough data to reach to the same conclusions as the author.
(15) Assigning unbalanced numbers of pages for different models and pertinent results. This is only acceptable when introducing new models or new training algorithms.
(16) Comparing an advanced variant of one network against an elementary or unknown variant of another one.

### Organization of the Paper

A variant of the SFAM algorithm is presented in Section 2. Section 3 is assigned to comparing our model and that of Kasuba. First we point out the eminent features of our model, and then the shortcomings of Kasuba's model that do not exist in our model are analyzed. In Section 4, the MLP network, the BPLRF and the BST algorithms are reviewed. In addition to the Farsi OCR datasets, two benchmarks, i.e. the circle-in-the-square problem and the sphere-in-the-cube problem, are used for evaluation of the networks and their training algorithms. Section 5, is assigned for explaining these benchmarks, and datasets. In Section 6, the simulation results obtained on the two benchmarks are presented and analyzed. In Section 7, the simulation results obtained on the Farsi OCR datasets are presented and analyzed. The conclusions are reviewed in Section 8. The Letter also includes an appendix, which provides more information about the feature extraction methods used for creating the Farsi OCR datasets.

REMARK: Considering that in the classifier every pattern is represented by its feature vector as the input vector to the classifier, classifying the input vector is equivalent to classifying the corresponding pattern. In this Letter, the vector that is to be classified, is frequently referred to by the input pattern and vice versa.

## 2. Fuzzy ARTMAP

Originally, ART networks were defined in terms of differential equations, but in practice they are implemented using approximations or analytical solutions to these equations, in the limit. ART networks have their own special terminology. The main idea of unsupervised ART networks is as follows:

(1) Find the nearest cluster prototype that '**resonates**' with the input pattern.
(2) Update this cluster prototype to be closer to the input.

Here '**resonance**' simply means being within a certain threshold of a second similarity measure. The learning algorithms of ART networks are prototype-based methods, in contrast to those of MLP networks which are error-based methods. Essentially, prototype-based methods, construct a nearest neighbor look-up table. Thus, a new input selects the closest entry in the table and the corresponding output.

In any kind of ART networks, prototypes are created as needed. ART networks solve the stability-plasticity dilemma by allowing an input pattern to modify a prototype, only if the pattern is sufficiently close to the prototype (the resonance state), otherwise a new prototype (and accordingly a new subclass) is formed.

Via the vigilance factor the granularity of subclasses is controlled, or in other terms the resonance is defined mathematically. Supervised ART networks have the suffix 'MAP' as an extension to their names.

### 2.1. THE SIMPLIFIED FUZZY ARTMAP (SFAM)

The FAM is an ART network for the association of analog patterns in a supervised mode. Its original architecture [8], and even algorithmic version [10], are too complicated and have a lot of redundancy for a simple classification task where the output patterns are simple class labels. Hence, we made it much simpler by removing redundancies, so the name simplified fuzzy ARTMAP (SFAM). The SFAM is much faster than the FAM and easier to understand and simulate. Here, it should be made clear that the SFAM can be used only for classification.

It must be stated that in 1993, a similar work had been done by Kasuba [11]. Although the main idea of our SFAM and that of Kasuba is the same, there are differences in the algorithms. In the following we present our algorithm, later in Section 3, we will discuss its major specifications and its differences with Kasuba's model.

The network configuration is shown in Figure 1. The main idea of our SFAM is as follows:

(1) Find the nearest subclass prototype that 'resonates' with the input pattern (winner).
(2) If the labels of the subclass and the input pattern match, update the prototype to be closer to the input pattern.
(3) Otherwise, reset the winner, temporarily increase the resonance threshold ($\rho$), and try the next winner.
(4) If the winner is uncommitted, create a new subclass (assign the input vector to be the prototype pattern of the winner, and label it as the class label of the input).

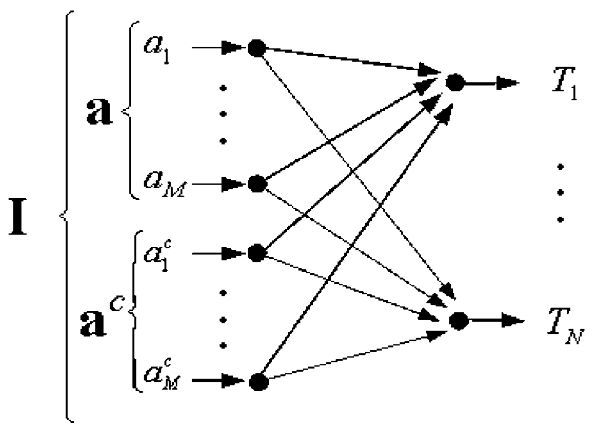

*Figure 1.* Configuration of the SFAM network.

Before presenting the algorithm, the following definitions and assumptions are made:

(1) All inputs are in the complement-coded format, i.e. if $\mathbf{a} \in \mathbb{R}^M$ is an analog input pattern and $a_i \in [0, 1]$, then $\mathbf{I} \in \mathbb{R}^{2M}$ will be the complement-coded input[4] and is defined as:

$$\mathbf{I} = [\mathbf{a}, \mathbf{a}^c] \equiv [a_1, \ldots, a_M, a_1^c, \ldots, a_M^c] \tag{1}$$

where $a_i^c \equiv 1 - a_i$. (2)

(2) The operator $\wedge$ is the fuzzy AND operator, defined by:
$(\mathbf{p} \wedge \mathbf{q})_i \equiv \min(p_i, q_i)$ (3)

(3) The norm $|\cdot|$ is $L_1$ norm defined by : $|\mathbf{p}| \equiv \sum_{i=1}^{M} |p_i|$ (4)

(4) Based on the definitions given above, the norms of all inputs are equal, $|\mathbf{I}| = M$.

(5) $\mathbf{w}_j$ is the weight vector of the $j$th neuron in the second layer:
$\mathbf{w}_j = [w_{1j}, w_{2j}, \ldots, w_{2Mj}]$ (5)

(6) An uncommitted neuron means a neuron that is not labeled and has its weights set equal to one: $w_{ij} = 1 \quad \forall i$, and its activity level $T_0$ is a parameter to choose ($0 \leqslant T_0 \leqslant 1$). After a neuron is assigned to a class, it is labeled and is called committed. When a neuron for the first time becomes committed, its weight vector $\mathbf{w}_J$ will be set equal to the current input vector $\mathbf{I}$, and its label will be the class of the current input.

(7) There is a one-to-one relationship between a committed neuron, and a subclass. In other word, any committed neuron is the representative of a subclass, and its weight vector is the prototype of that subclass. The activity level of a committed neuron for any input is calculated using the Equation (7).

[4]The condition $a_i \in [0, 1]$ is not compulsory, but later we will see that violating it on our OCR datasets, causes prototype proliferation.

(8) The second layer neurons are winner-takes-all cells.
(9) $\alpha$ is the tie-breaker or choice parameter: ($0.001 < \alpha < 10$), see Remarks 5 and 7.
(10) $N$ is the total number of neurons in the second layer.
(11) $\beta$ is the learning factor ($0 < \beta \leqslant 1$), as explained in Remark number 7.
(12) $\rho \in [0, 1]$ is the vigilance factor, see Remarks 3 and 4.
(13) The algorithm starts with one uncommitted neuron.

SFAM ALGORITHM:

(1) Set the vigilance factor equal to its baseline value: $\rho = \bar{\rho}$. (6)
(2) Insert input, and calculate second layer activities:

$$T_j(\mathbf{I}) = \frac{|\mathbf{I} \wedge \mathbf{w}_j|}{\alpha + |\mathbf{w}_j|} \quad \text{for } j = 1, \ldots, N-1; \tag{7}$$

and for the uncommitted neuron: $T_N = T_0$

(3) Find the winner $J = \arg\,[\underset{j}{\mathrm{Max}}\,(T_j)]$ (8)

If the winner neuron is uncommitted, go to step 7.
(4) Check the resonance condition, i.e. if the input is similar enough to the winner's prototype:

$$\frac{|\mathbf{I} \wedge \mathbf{w}_J|}{|\mathbf{I}|} = \frac{|\mathbf{I} \wedge \mathbf{w}_J|}{M} \geqslant \rho \tag{9}$$

If this condition is fulfilled, go to step 5.
If this condition is not fulfilled, reset the winner ($T_J = -1$), go to the step 3, and check the next winner.
(5) If the class label of the winner matches with the class label of input, update the prototype pattern to be closer to the input pattern:

$$\mathbf{w}_J^{(\text{new})} = \beta(\mathbf{I} \wedge \mathbf{w}_J^{(\text{old})}) + (1 - \beta)\mathbf{w}_J^{(\text{old})} \tag{10}$$

and go to step 9,
otherwise reset the winner ($T_J = -1$), temporarily increase the vigilance factor so as to violate the condition of Equation (9), i.e. set $\rho$ equal to:

$$\rho = \frac{|\mathbf{I} \wedge \mathbf{w}_J|}{M} + \varepsilon, \tag{11}$$

(where $\varepsilon$ is a small positive number, i.e. $\varepsilon \approx 0.001$).
(6) If $\rho > 1$, terminate the training for this input pattern in the current epoch (data mismatch), and go to step 9, otherwise go to step 3, and try the next winner.
(7) Create a new subclass, i.e. assign the input vector as the prototype pattern of the

winner neuron:

$$\mathbf{w}_N = \mathbf{I} \tag{12}$$

and set the class label of the winner neuron to be as the class label of input pattern.

(8) Create a new uncommitted neuron, and: $N \leftarrow N + 1$.

(9) Go to the step 1, and repeat the Algorithm for the next input.

The flow chart of the SFAM Algorithm is presented in Figure 2. □

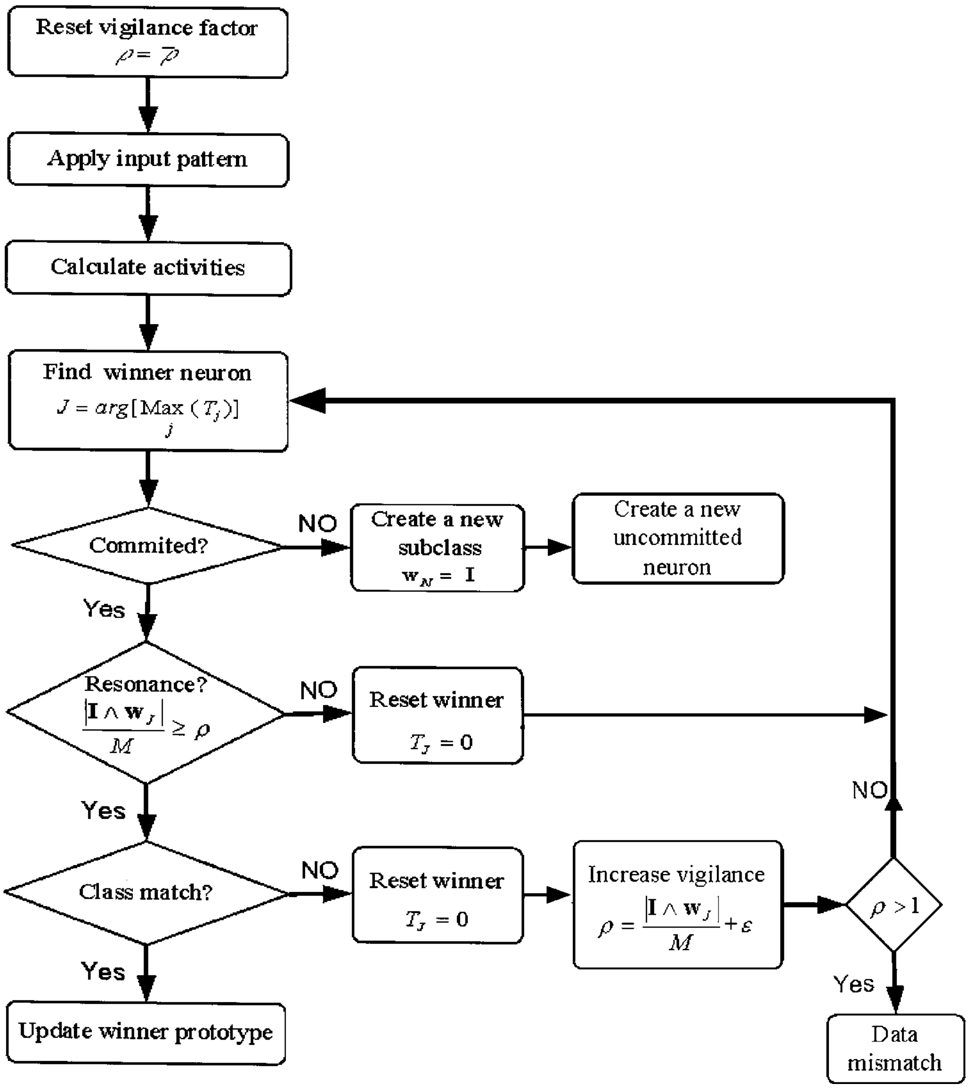

*Figure 2.* Flow chart of the SFAM training Algorithm for one input pattern in one epoch of training.

REMARKS:

(1) If the uncommitted neuron is selected as the winner, the resonance condition, i.e. Equation (9), will be fulfilled, and the search for the winner neuron will end.

(2) We will always have one uncommitted neuron, which is the last neuron. Its activity level is independent of input, and is set equal to $T_0$, and $0 \leqslant T_0 \leqslant 1$. If the uncommitted neuron becomes committed, we create a new uncommitted neuron.

(3) The initial value of $\rho$ is set to the baseline value $\bar{\rho}$, for every training pattern and in every training epoch. When a new input pattern, $\mathbf{I}$, is about to be associated with a wrong prototype, the vigilance factor will be temporarily increased to rule out the association with that prototype, and next time the algorithm will see $\mathbf{I}$ close to the newly formed prototype. This is called 'direct access property'. It means that the first prototype, that will be the winner, is guaranteed to satisfy the test in other steps.

(4) The legitimate range for $\bar{\rho}$ is the interval [0, 1]. By increasing $\bar{\rho}$ the number of created subclasses (and their prototypes) will increase. A larger number of subclasses will result in a fine-grained classification, which sometimes can improve recognition rate on test set of noisy or ill-engineered datasets. On the other hand having more prototypes will decrease both training and online operation speed.

(5) Increasing the value of $\alpha$ will increase the number of subclasses (and prototypes). Therefore, it is advisable to set it equal to 0.001. In our simulation, values smaller than 0.001 did not make any difference.

(6) Increasing the value of $\varepsilon$ will increase the number of prototypes. In addition, a large value of $\varepsilon$ could trigger a false data mismatch alarm, by causing the value of $\rho$ to go beyond 1. In the cases where some real data mismatch exist, larger values of $\varepsilon$ will increase the error. Therefore a value of $\varepsilon$ equal to 0.001 is advisable.

(7) The SFAM can be trained in two modes: fast learning and slow learning[5]. In the fast-learning mode $\beta = 1$. In the slow-learning mode $\beta = 1$, if the neuron is uncommitted; otherwise $0 < \beta < 1$. In the slow-learning mode more prototypes will be created. This mode has been recommended for noisy data.

But sometimes slow learning prevents the network to learn all training data, especially when larger values of $\alpha$ or $\varepsilon$ is used. In other words a phenomenon, which is similar to local minimum trap in MLP, is observed.

(8) In addition to the method of complement coding that was discussed, it is also possible to use other normalization methods. The advantage of the complement coding over other methods is that it preserves amplitude information, without which we will face the prototype proliferation problem, i.e. the number of created prototypes will increase enormously.

[5]In the original FAM, the slow-learning mode is called fast-commit slow-recode.

(9) The training algorithm is repeated until the steady state is reached. As it was explained in the second paragraph of Remark 7, sometimes the network converges to a local minimum.

(10) After training phase, if the network is not supposed to continue learning during online operation, the last uncommitted neuron is discarded (see also Remark 2).

We can keep the last uncommitted neuron during the test phase. If the uncommitted neuron wins the competition, a no-prediction status is attributed to the input. This feature does not exist in the original FAM network.

(11) Before testing the network on test data, all the spurious prototypes which are defined as the prototypes that after training will never be selected as the winner, should be identified and destroyed.

(12) In test mode, the complement-coded input is inserted to the network, and by using Equation (7) second layer activities are calculated. The neuron with maximum activity is the winner, and its label is the predicted class for the input pattern.

(13) The order of data presentation can affect the performance of the network. That is to say, for different permutations of a given dataset, the created prototypes and their number vary. This observation is the basis of ARTMAP *voting strategy*. In voting strategy, an ARTMAP system is trained several times on different permutations of the same training set. Since the prediction can vary from one simulation to the next, all predictions are registered, and the final class prediction for a given input pattern, is the one with majority vote. Carpenter et al. [8] report that by applying the voting strategy, they obtained a better recognition rate.

(14) This voting strategy can be used to assign confidence measures to predictions on small, noisy, or incomplete datasets. Voting strategy can be used with any other classifier whose performance has a stochastic nature, i.e. it is sensitive to the data presentation order, or the initial starting point. Therefore, it should not be considered a specific feature of the FAM networks.

(15) Selecting the initial activity level of uncommitted neuron $T_0$, is not a feature of original FAM network, and should be regarded as our innovation. In fact, Carpenter et al. [8] have failed to notice its importance.

## 3. The Main Features of Our Model, and a Critical Look at Kasuba's Model

Our model and training algorithm is a genuine simplification of the original FAM network: it has been obtained by merely eliminating the sections that are redundant for a classification task, with only one deviation from the original FAM, which is selecting an initial value for the activity level of uncommitted neuron. Kasuba's algorithm has three major alterations from the original model, and as we will see, these alterations are only blunders that slow down the training.

In this section, first we outline the most important features of our model, then we review the deviations of Kasuba's model from the original FAM model and discuss their impacts on the performance of the network.

### 3.1. THE MAIN FEATURES OF OUR MODEL

(1) Our network and pertinent training algorithm is a stand-alone one, which relieves the reader from referring to the complicated original FAM network and its training algorithm.
(2) To shed the fear of the newcomers, we have tried to make it as similar to other NN models as possible. To this end, we turned its peculiar architecture to a much familiar architecture, like those of feedforward networks as the learning vector quantization (LVQ), or the MLP.
(3) The basic concepts are presented in the training algorithm, and not in the architecture, as it is the case with the LVQ, the MLP, and almost every other NN model.
(4) For equivalent concepts, we have used the common terminology of pattern recognition and neural networks. (Also see Section 3.2, item 6).
(5) The only innovation in our model, which does not exist in the original FAM model, is introducing $T_0$ as a training parameter.

### 3.2. A CRITICAL LOOK AT KASUBA'S SFAM MODEL

(1) The first step in the Kasuba's algorithm is controlling the class of input, to see if it is from a new class, from which no pattern has ever been seen before. If this is the case, the training algorithm will end for that input by creating a new subclass and assigning this input to be its prototype. Although this step does not exist in the original FAM model, it may seem a useful innovation, saving a lot of time and effort. Regrettably, this is not the case.

    The number of times that this operation will return a positive answer, is equal to the number of classes, e.g. $L$. And among them, only $L-1$ times this positive answer could be useful and could make a difference (the first time, which happens in the beginning of the training algorithm, should be excluded). For instance, in the circle-in-the-square problem, on which Kasuba has carried out his simulations, this operation is carried out 10000 times (five training epochs over 2000 training samples), while it returns a positive answer only twice, and only the second time it is useful. What a useless intricacy and waste of time!
(2) After calculating the activities of the neurons, Kasuba sorts the outputs in descending order of their activities. This step does not exist in the original FAM model. In addition, this is a very expensive and useless operation, because in most of the cases the right neuron will be chosen as the winner (especially after the first epoch of training). And even if it is necessary to find the next winner, we

can do it just by resetting the current winner and finding the next winner, as it is the case in the original model. In this second case only a small part of sorting operations will be performed, and considering that sorting is repeated after presenting every pattern and in every epoch of training, it slows down the training enormously. Besides that, Kasuba instructs to use stable sorting algorithms[6], which are incomparably slower than unstable sorting algorithms. Later, we will see that in the Kasuba's model if an unstable sorting algorithm is used, the network's training speed can increase substantially, without any negative impact on the performance of the network.

(3) Kasuba has eliminated the concept of the uncommitted neuron and its chance to win the competition. In this way, before creating a new neuron, all the neurons with any dissimilarity degree to the input, and with an activity function as low as zero, will lie in the queue of winners and will be tried to see if they can pass two other tests. Meanwhile, in the original FAM model the uncommitted neuron exists. Although Carpenter et al. have kept silent about the activity level of this uncommitted neuron, one can simply use the same formula that is used for committed neurons, i.e. for any input the activity of uncommitted neuron will be:

$$T_j(\mathbf{I}) = \frac{|\mathbf{I} \wedge \mathbf{w}_j|}{\alpha + |\mathbf{w}_j|} = \frac{|\mathbf{I}|}{\alpha + 2M} = \frac{M}{\alpha + 2M} \tag{13}$$

and if $\alpha \ll 2M \rightarrow T_j(\mathbf{I}) \simeq 0.5$
and any neuron with an activity less than this value, will not be chosen as the winner, and in this way a lot of time will be saved. Later, we see that by defining a larger value for activity of uncommitted neurons, many improvements can be achieved.

(4) By eliminating the uncommitted neuron, the sorting operation gets more detrimental, because all the committed neurons are included in the sorting. But by having an uncommitted neuron, which is always ready to compete with the committed neurons, and has a chance to be the winner, the number of neurons that can qualify to win the competition decreases substantially (all the neurons with an activity function less than $\approx 0.5$ are eliminated), therefore our search to find the qualified winner is ended much earlier than that in Kasuba's model.

(5) Kasuba has overlooked the state of ($\rho > 1$), which happens in three cases:

- Having a large value of $\varepsilon$, e.g. $\varepsilon \approx 0.1$ which can be considered quite large for some datasets.
- Having many and quite different samples from a specific class, with an average value for $\varepsilon$, e.g. $\varepsilon \approx 0.001$.
- Having conflicting data, i.e. the same inputs with different class labels.

[6]In a stable sorting algorithm the relative order of the equal elements is preserved.

In fact, this defect can be traced back to the original FAM model, however, the programmers of many downloadable source codes and applications have solved this problem with their own common sense.

(6) There is a confusion in Kasuba's model and article, and that is using the word 'category' for three different concepts, i.e. class, subclass, and prototype.

The root of this problem can be traced back to the original FAM model [8], where in different locations, 'category' has two different meanings: 'subclass' and 'prototype'. In fact, because of one-to-one relationship between subclasses and their prototypes, sometimes these two words can be used interchangeably, also using a third word (i.e. category) instead of both of them will not change the meaning. Nonetheless, subclass and prototype are two different concepts, and are not interchangeable everywhere.

To solve these confusion, as well as to ease the problem for newcomers, the authors decided to use the common terminology of pattern recognition. Thus, in different locations we substituted the word 'category' with one of these words that is the most appropriate: class, subclass, or prototype.

(7) Finally, there is a typographical error in the definition of class mismatch (see [11], page 23, line 30), where $\geqslant$ must be changed to $<$.

## 4. Training Algorithms For the MLP

In this section first we review the basic backpropagation (BB) algorithm. Then two modified variants of backpropagation algorithm are presented: BPLRF and BST algorithms.

### 4.1. BASIC BACKPROPAGATION ALGORITHM

The configuration of the MLP with one hidden layer is shown in Figure 3. The goal is to adapt the parameters of the network in order to bring the actual output $\mathbf{z}^q$ close to the corresponding desired output $\mathbf{d}^q$ (for $q = 1, \ldots, Q$), where $Q$ is the number of training patterns.

In addition, the network should perform well for vector pairs outside the training set (generalization property). The task of every neuron is to compute a weighted sum of its inputs and pass the sum through a soft nonlinearity. This soft nonlinearity (activity function) should be nondecreasing and differentiable.

Before proceeding, the following assumptions are made:

(1) The network has only one hidden layer, extending the algorithm to the multi-layer case is straightforward [45].

(2) The total sum-squared error (TSSE) is considered as the cost function, i.e.:

(3) $$\mathrm{TSSE} = \sum_q E_q, \quad E_q = \sum_k (d_k^q - z_k^q)^2 \quad (q = 1, \ldots, Q) \tag{14}$$

where $d_k^q$ and $z_k^q$ are the $k$th components of desired and actual output vectors, respectively.

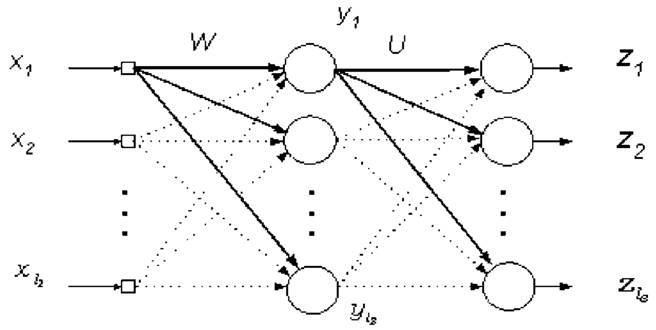

*Figure 3.* Configuration of the MLP with one hidden layer.

(4) Training will be carried out in pattern mode[7], and for simplicity of notation the subscript $q$ from $E_q$ is eliminated.
(5) The activity function of the neurons is the unipolar sigmoid, i.e.: $f(\theta) = \frac{1}{1+e^{-\theta}}$

ALGORITHM:

(1) Set parameters and initialize network
$Q$ = number of training vectors
$l_1$ = dimension of input vector
$l_2$ = number of neurons in the hidden layer
$l_3$ = dimension of the output vector
$w_{ij}$ *and* $u_{jk}$, are set to small random values in the interval [−0.25, +0.25]
(2) Insert input and desired output
(3) Forward pass—Compute the network outputs by proceeding forward through the network, layer by layer:

$$\begin{cases} net_j = \sum_i x_i w_{ij} \\ y_j = \dfrac{1}{1+e^{-net_j}} \end{cases}, \quad j = 1,\ldots,l_2 \tag{16}$$

$$\begin{cases} net_k = \sum_j y_j u_{jk} \\ z_k = \dfrac{1}{1+e^{-net_k}} \end{cases}, \quad k = 1,\ldots,l_3 \tag{17}$$

(4) Backward pass—Calculate the error gradients versus weights values, layer by layer, starting from the output layer and proceeding backwards:

$$\frac{\partial E}{\partial w_{ij}}, \quad \begin{cases} i = 1,\ldots,l_1 \\ j = 1,\ldots,l_2 \end{cases} \tag{18}$$

[7]Training can be carried out in two modes: pattern mode and batch mode. Pattern mode is preferred for three reasons: faster convergence, easier implementation, and less demand on memory.

$$\frac{\partial E}{\partial u_{jk}}, \quad \begin{cases} j = 1, \dots, l_2 \\ k = 1, \dots, l_3 \end{cases} \tag{19}$$

$$\text{where } E = \sum_k (d_k - z_k)^2 \tag{20}$$

(5) Update weights:

$$u_{jk}(n+1) = u_{jk}(n) - \eta \times \frac{\partial E}{\partial u_{jk}} \tag{21}$$

$$w_{ij}(n+1) = w_{ij}(n) - \eta \times \frac{\partial E}{\partial w_{ij}} \tag{22}$$

where $\eta$ is the step-size, and $0 < \eta < 1$.

(6) $TSSE = TSSE + E$;

(7) If $(q = Q)$ check stopping criterion (see Remarks 8 and 9), otherwise $(q = q + 1)$ go to step 2 and repeat.

REMARKS:

(1) In the case of the unipolar sigmoid function, the gradients will be:

$$\text{for the output layer:} \begin{cases} \dfrac{\partial E}{\partial u_{jk}} = -y_j \delta_k, \\ \delta_k = \underbrace{z_k(1 - z_k)}_{f'(\theta)}(d_k - z_k) \end{cases} \tag{23}$$

$$\text{for the hidden layer:} \begin{cases} \dfrac{\partial E}{\partial w_{ij}} = -x_i \delta_j, \\ \delta_j = \underbrace{y_j(1 - y_j)}_{f'(\theta)} \sum_k w_{jk} \delta_k \end{cases} \tag{24}$$

(2) Sometimes, in the net input of the sigmoid function a constant sentence is also considered (called the threshold term), which is implemented using a constant input (equal to $-1$). In some cases this term triggers the *moving target phenomenon* [46] and hinders training, and in some other cases there is no solution without it. Therefore, it must be examined separately for every case. As we will see later, the Farsi OCR application is an example for the first case, while the circle-in-the-square, and the sphere-in-the-cube problems are examples for the second case.

(3) Although there is no range limitation for input vector components, different data normalization methods should be tried to achieve optimal performance. For the unipolar sigmoid function the components of the desired output vector should be in the range [0, 1]. The number of output cells depends on the number of classes and the approach of coding, however, it

is recommended to make it equal to the number of classes, even for a two-class problem[8].

(4) As the outputs of the neurons approach extreme values (0 or 1) there will be little or no learning. A solution to this problem is to add a small offset ($\approx 0.1$) to the derivatives $f'(\theta)$ in equations (23) and (24), which is called '*the sigmoid prime offset*' [47], thus $f'(\theta)$ never reaches zero. The sigmoid prime offset has pivotal impact on the training, and it never should be neglected.

(5) The range, from which the initial values of the connection weights are chosen, could have an effect on the convergence speed.

(6) The backpropagation method suffers from getting stuck in local minima. Our experience shows that this is mostly a problem of parameter tuning, and a careful parameter tuning (including proper selection of network size) can alleviate this problem.

(7) In order to speed up the convergence, as well as alleviating the local minima problem, an extra term called the momentum term is added to the weights update formulas, so the final equations for the weights update will be:

$$u_{jk}(n+1) = u_{jk}(n) - \eta \times \frac{\partial E}{\partial u_{jk}} + \alpha(u_{jk}(n) - u_{jk}(n-1)) \tag{25}$$

$$w_{ij}(n+1) = w_{ij}(n) - \eta \times \frac{\partial E}{\partial w_{ij}} + \alpha(w_{ij}(n) - w_{ij}(n-1)) \tag{26}$$

(8) Our experience shows that the learning rate factor, the momentum factor, the network size and the training set size have interactive impacts on the training, and this makes parameter tuning a difficult task.

(9) The decision to stop training should be based on some test results on the network, which is carried out every $N$ epochs after ($\mathrm{TSSE} = \sum_q E_q < C$).

[8]Although, in most of the literature one finds that one output cell is used for a two-class problem, we suggest using two outputs for such a case. Our explanation is as follows:
In the case of using one output, we should set a strict definition for class membership, for instance:

$$\begin{cases} \text{if } z^j > 0.6, & j\text{th input pattern belongs to class A} \\ \text{if } z^j < 0.4, & j\text{th input pattern belongs to class B} \\ \text{if } 0.4 \leqslant z^j \leqslant 0.6, & \text{no-prediction} \end{cases}$$

where $z^j$ is the value of output cell for the $j$th input pattern.
In the case of using two outputs, a very versatile definition of class membership is enough, for instance:

$$\begin{cases} \text{if } z_1^j > z_2^j + 0.2, & j\text{th input pattern belongs to class A} \\ \text{if } z_2^j > z_1^j + 0.2, & j\text{th input pattern belongs to class B} \\ \text{if } |z_1^j - z_2^j| \leqslant 0.2, & \text{no-prediction} \end{cases}$$

where $z_1^j$ and $z_2^j$ are the values of the first and the second outputs for the $j$th input pattern, respectively. In this second case, because of flexible definitions for acceptable outputs, the number of possible solutions (sets of network parameters which create the correct outputs for all or most of the input patterns, by fulfilling the pertinent inequalities for correct predictions) is much more than the first case. And convergence to a solution happens easier. In other terms: parameter tuning will be easier, the chance of encountering a local minimum trap decreases, and convergence will be faster.

(10) Formerly, it was thought that using the cross validation method [48] as a stopping criterion can achieve better generalization performance. Since this method stops training on both learnt and unlearnt patterns, it is unsatisfactory and unconvincing[9], and nowadays one hardly can find any paper which reports its usage. In addition, there is no report in using the cross validation technique in training other NN models.

(11) The number of neurons in the hidden layer, as well as the number of hidden layers, are parameters to choose, but theoretically the MLP with one hidden layer is enough to solve any problem that the MLP with two hidden layers can solve [49].

(12) Some resources have recommended using a slope factor in the sigmoid activity function, i.e. a value unequal to one for a in:

$$f(\theta) = \frac{1}{1 + e^{-a\theta}}. \tag{27}$$

Our experience shows that although it can speed up convergence at the beginning of training, at the end it slows down the training substantially, therefore, in all our experiments we set it equal to one.

(13) Data presentation order can be crucially important to the performance of the MLP. More specifically:

- The number of samples for various classes must be equal. If for any reason one cannot extract equal numbers of different samples for all classes, the samples of classes that are outnumbered should be duplicated in order to make their numbers equal with that of other classes.
- In every epoch of training, one complete set of patterns, i.e. one pattern from each class, should be presented to the network.
- The same presentation order of patterns should be kept throughout the training.

(14) In the rest of this article, the name *basic backpropagation algorithm (BB)*is assigned for the backpropagation algorithm with the momentum term and sigmoid prime offset.

## 4.2. BACKPROPAGATION WITH PLUMMETING LEARNING RATE FACTOR (BPLRF)

The BPLRF algorithm has a minor difference with BB algorithm. In this algorithm we end the training with diminished training parameters, as explained in the following:

Ten epochs before ending the training, suddenly decrease learning rate and momentum factors to very small values, e.g. make them 10 times smaller. As we will see later, in this way a sudden decrease in the values of cost function and

[9]Since in this method, the reason of stopping the training is overtraining on already learnt patterns, there is no explanation to stop training on unlearned patterns. Our suggestion is simply to stop training only on the learnt pattern and to continue on the unlearnt ones. For a full explanation see Section 4.3.

recognition errors (on both training set and test set) occurs. Our explanation for this phenomenon is as follows:

After approaching the minimum well, by using large values of learning rate and momentum factor we step over the well. But by decreasing the step size we put our leg in the middle of the minimum well and face a free fall to the bottom.

### 4.3. BACKPROPAGATION WITH SELECTIVE TRAINING (BST)

The difference between the BST algorithm and the BB algorithm lies in the selective training, which is appended to the BB algorithm. When most of the vector associations have been learnt, every input vector should be checked individually, and if it is learnt there should be no training on that input, otherwise training will be carried out. In doing so, a side effect will arise: the stability problem. That is to say, when we continue training on only some inputs, the network usually forgets the other input-output associations which were already learnt, and in the next epoch of training it will make wrong predictions for some of the inputs that were already classified correctly. The solution to this side effect consists of considering a 'stability margin' for the definition of the correct classification in the training step. In this way we also carry out training on marginally learnt inputs, which are on the verge of being misclassified.

Selective training has its own limitations, and cannot be used on conflicting data, or on a dataset with overlapping classes. Based on our experience, using the BST algorithm on an error free OCR dataset has the following advantages:

- prevents from overtraining,
- de-emphasizes the overtrained patterns,
- enables the network to learn the last percent of unlearnt associations in a short period of time.

#### BST ALGORITHM:

(1) Start training with the BB algorithm, which includes two steps:

- forward propagation,
- backward propagation.

(2) When the network has learnt most of the vector mappings and the training procedure has slowed down, i.e. the TSSE becomes smaller than a threshold value $C$, stop the main algorithm and continue with selective training.
(3) For every input perform forward propagation and examine the prediction of the

network:

$$\begin{cases} z_J = \max_j(z_j) & \forall j \\ \text{if } (z_J \geqslant z_j + \zeta) & \forall j \neq J \rightarrow J \text{ is the predicted class} \\ \text{if } (z_J < z_j + \zeta) & \exists j \neq J \rightarrow \text{no-prediction} \end{cases} \tag{28}$$

where $\zeta$ is a small positive constant, and is called the stability margin.

(4) If the network makes a correct prediction, do nothing, go back to step 3 and repeat the algorithm for the next input,
otherwise, network does not make a correct prediction (including the no-prediction case), carry out the backward propagation.

(5) If the epoch is not complete, go to step 3, otherwise, check the total number of wrong predictions: if its trend is declining, go to step 3 and continue the training, otherwise stop training.

(6) Examine network performance on the test set:

do only forward propagation, and then:

$$\text{for any input}: z_J = \max(z_j) \quad \forall j = 1, \ldots, l_3 \quad \rightarrow \quad J \text{ is the predicted class} \tag{29}$$

REMARKS :

(1) An alternative condition for starting the selective training is as follows:

After TSSE becomes smaller than a threshold value $C$, every $T$ epochs carry out a recognition test on the training set, and if it fulfills a second threshold condition, i.e. (recognition error $< C_1$), start selective training. The recommended value for $T$ is $3 \leqslant T \leqslant 5$.

(2) If $\zeta$ is chosen as large, training will be continued for most of the learnt inputs, and this will make our method ineffective. On the other hand, if $\zeta$ is chosen too small, during training we will face a stability problem, i.e. with a small change in the weights values, which happens in every epoch, a considerable number of associations will be forgotten, thus the network will oscillate and training will not proceed. After training, with a small change in the feature vector that causes a small change in output values, the prediction of the network will change, or for feature vectors from the same class but with minor differences, we will have different predictions. This also causes vulnerability to noise and weak performance on both the test set and on the real world data out of both the training set and the test set. The optimum value of $\zeta$ should be small enough to prevent training on learnt inputs, but not so small as to give way to changing the winner neuron with minor changes in weights values or input values. Our simulation results shows that for our datasets a value of 0.05 is the optimum value.

(3) It is also possible to consider a no-prediction state in the final test, i.e.:

$$\begin{cases} z_J = \max_j(z_j) \\ \text{if } (z_J \geqslant z_j + \zeta_1) \quad \forall j \neq J \quad \rightarrow \quad J \text{ is the predicted class} \\ \text{if } (z_J < z_j + \zeta_1) \quad \exists j \neq J \quad \rightarrow \quad \text{no} - \text{prediction} \end{cases} \tag{30}$$

in which $0 < \zeta_1 < \zeta$.

This no-prediction state will decrease the error rate at the cost of decreasing the correct prediction rate.

## 5. Datasets

For comparing the networks and their training algorithms, three problems were chosen: the circle-in-the-square problem, the sphere-in-the-cube problem, and the Farsi OCR problem.

### 5.1. THE CIRCLE-IN-THE-SQUARE PROBLEM

This is a well-known benchmark, which requires a system to identify which points of a square lie inside and which points lie outside a circle whose center coincides with that of the square and its area equals half that of the square. Since this benchmark has been used by Carpenter et al. [8] for explanation and evaluation of the original fuzzy ARTMAP network, we also use it to evaluate our training algorithms.

For evaluation of SFAM algorithms on this problem we created 10 datasets, each composed of 20000 random samples. The reported results for the SFAM algorithms are the averaged results obtained over all of these 10 datasets. Then, one dataset on which all of the SFAM algorithms had a recognition rate close to the average recognition rate, was chosen for evaluation of the MLP network.

### 5.2. THE SPHERE-IN-THE-CUBE PROBLEM

This problem is a three dimensional variant of the circle-in-the-square problem, and for the first time was presented in [50]. Later, we will see that it is a much more difficult problem than its two dimensional variant.

For evaluation of the SFAM algorithms on this benchmark, we created two groups of datasets:

1- 10 datasets, each composed of 2000 random samples
2- 10 datasets, each composed of 20000 random samples

The reported results for the SFAM algorithms are the averaged results obtained over all of these 10 datasets.

Then, from each group, one dataset on which all of the SFAM algorithms had a recognition rate close to the average recognition rate, was chosen for evaluation of the MLP network.

REMARK 1. In all of the datsets of these two benchmarks, the number of samples for both of the classes are equal. Randomly created samples are ordered sequentially, i.e. every other sample belongs to the same class. The same presentation order of patterns is kept throughout the whole training. In all of the simulations, the first half of the samples were assigned for the training set, and the second half of the samples were assigned for the test set.

REMARK 2. Datasets that are used for evaluation of the SFAM algorithms, are normalized, i.e. all their components have been mapped into the interval [0,1]. While in the unnormalized datasets, which are used for training the MLP, the radii of both circle and sphere are equal to one, and the shapes are symmetrical around the origin.

### 5.3. FARSI OCR DATASETS

A total of 18 datasets composed of the feature vectors of 32 isolated characters of the Farsi alphabet, sorted in three groups, were created through various feature extraction methods, including: principal component analysis (PCA); vertical, horizontal and diagonal projections; zoning; pixel change coding; and some combinations of these, with the number of features varying from 4 to 78 per character, according to Table 1.

For creating these datasets, 34 Farsi fonts that are used for publishing online newspapers and web sites, were downloaded from the Internet. Figure 4 shows a whole set of isolated characters from one sample font printed as text. Figure 5

*Table 1.* Brief information about the Farsi OCR datasets.

| | Dataset | Extraction method | Explanation | No. of features per character |
|---|---|---|---|---|
| Group A | db1 | PCA | | 72 |
| | db2 | PCA | | 54 |
| | db3 | PCA | | 72 |
| | db4 | PCA | | 64 |
| | db5 | PCA | | 48 |
| Group B | dbn1 to dbn5 | PCA | normalized versions of db1 to db5 | |
| Group C | db6 | zoning | | 4 |
| | db7 | pixel change coding | | 48 |
| | db8 | projection | horizontal and vertical | 48 |
| | db9 | projection | diagonal | 30 |
| | db10 | projection | db8 + db9 | 78 |
| | db11 | | db6 + db7 | 52 |
| | db12 | | db6 + db8 | 52 |
| | db13 | | db6 + db7 + parts of db8 | 72 |

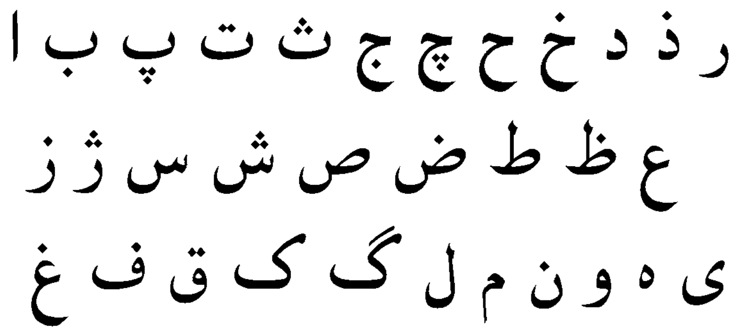

*Figure 4.* A sample set of machine-printed Farsi characters (isolated case).

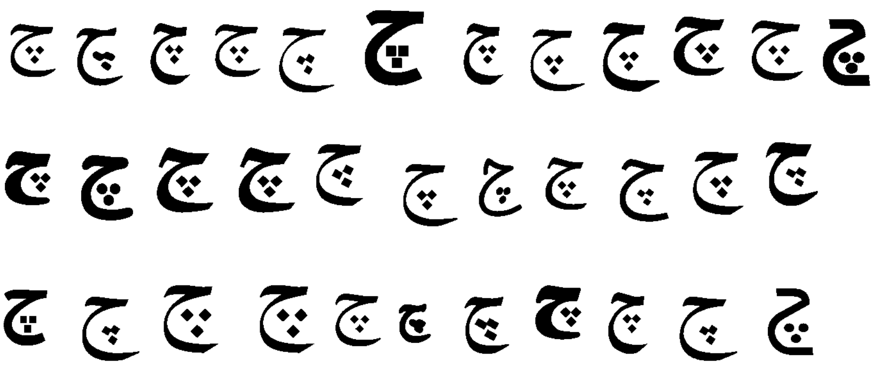

*Figure 5.* Variations of one sample character among different fonts.

demonstrates variations of one sample character among different fonts. Next, 32 isolated characters from these 34 Farsi fonts were printed in an image file; eleven sets of these fonts were boldface and one set was italic. In the next step, by printing the image file and scanning it with various brightness and contrast levels, two additional image files were created. Then, using a 65 $\times$ 65 pixel window, the character images were separated into the images of isolated characters. After applying a shift-radius invariant image normalization [51], and reducing the sizes of the character images to 24 $\times$ 24 pixel, the features vectors were extracted as described in the appendix. Figure 6 demonstrates the same characters of Figure 5, after normalization.

## 6. Simulation Results Obtained on the Benchmark Problems

Simulation results on the benchmark problems, are presented in three subsections:

(1) The SFAM algorithms are evaluated and compared with each other.

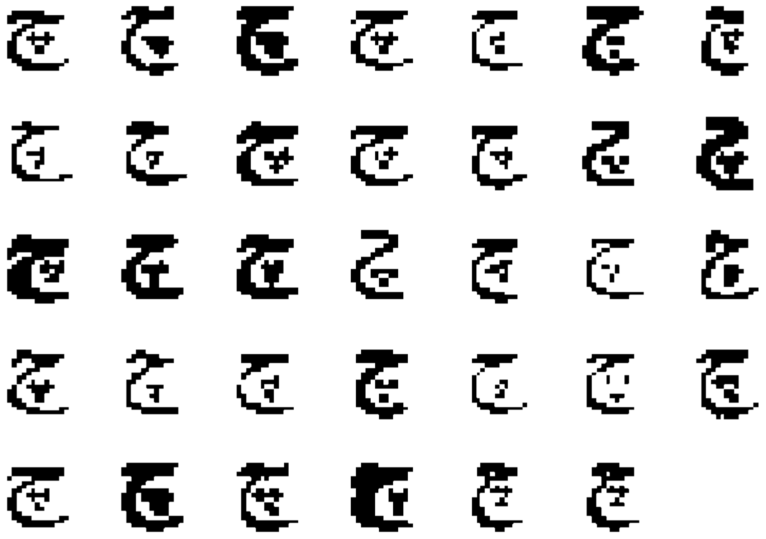
*Figure 6.* The same characters of figure 5, after normalization.

(2) The backpropagation algorithms are evaluated and compared with each other.
(3) Finally, the SFAM and the MLP networks are compared based on their best results.

## 6.1. RESULTS OBTAINED BY THE SFAM ALGORITHMS

The following definitions are used in the rest of this letter:

*SFAM 1.0*, for the Kasuba's algorithm, in which an indexing algorithm based on the *Selection sort* method is used.

*SFAM 1.1*, for a modified variant of Kasuba's algorithm, in which an indexing algorithm based on the *Quicksort* method, is used.

*SFAM 2.0*, for our SFAM algorithm, with the activity level of uncommitted neurons calculated using Equation (13), i.e. $T_0 = \frac{M}{\alpha+2\,M}$.

*SFAM 2.1*, for the modified variant of our algorithm, i.e. the activity level of the uncommitted neurons is treated as training parameter ($0 \leqslant T_0 \leqslant 1$).

REMARKS:

(1) A sorting algorithm that preserves the relative order of the equal elements, is called stable sorting, otherwise it is called unstable. In Kasuba's model after presenting each training pattern, the second layer cells should be stable-sorted. While, in the original FAM, and our algorithms SFAM 2.0 and SFAM 2.1, there is no sorting operation.

(2) The selection sort is a stable sorting algorithm, which categorically are slow. The Quicksort is one of the fastest sorting algorithms, but an unstable one.
(3) Our purpose of designing these algorithms and simulations is to show that:

- Using unstable but fast sorting algorithms will yield the same results that are obtained by using the stable sorting algorithms.
- There is no need for sorting at all; using any kind of sorting only will slow down the training.
- Allowing an uncommitted neuron to compete with committed ones, and using the activity level of uncommitted neuron $T_0$, as a parameter to determine the condition of its win, can improve the performance of the SFAM in many aspects.

## SETTINGS

We set the parameters as follows:

$\beta_0 = 1, \beta = 1, \bar{\rho} = 0, \alpha = 0.001, \varepsilon = 0.001$, and $T_0 = 0.9$ for the SFAM 2.1.

On all of the simulations, the training was continued until reaching zero dynamic recognition error[10] on the training sets. The averaged results obtained by four variants of the SFAM, are presented in Table 2 (A, B, and C).

## ANALYSIS

(1) The most eminent point in this table is the high recognition error on the sphere-in-the-cube problem (Table 2-B, and Table 2-C). Also eminent in these Tables is the substantial increase in the number of created prototypes, when the number of samples in the sphere-in-the-cube problem is increased from 2000 to 20000. These results suggest that the sphere-in-the-cube problem is much more difficult than the circle-in-the-square problem.
(2) We see that in the circle-in-the-square problem with 20000 samples (Table 2-A), our training algorithms, i.e., SFAM 2.1, SFAM 2.0, and SFAM 1.1, are on the average, 4.45, 3.71, and 1.56 times, respectively, faster than SFAM 1.0.
These numbers demonstrate the ratios of average training times. If we calculate the average of the ratios of training times (not presented in the table), these numbers will change to 4.52, 3.68, and 1.66 times, respectively.
(3) In the sphere-in-the-cube problem, we see that by increasing the number of training samples and consequently the number of prototypes, these ratios of training times grow substantially. More specifically, they grow from 2.34, 2.26, and 1.25

[10]Dynamic recognition error is obtained while the network is under training, and is defined as being equal to the number of input patterns for which the first winner neuron is not acceptable (for either failing to fulfill the resonance condition or class mismatch with the input).

*Table 2.* The simulation results obtained by four variants of SFAM algorithm on two benchmark problems (Averaged over 10 random datasets). A: Results obtained on circle-in-the-square problem; total no. of samples = 20000. B: Results obtained on sphere-in-the-cube problem; total no. of samples = 2000. C: Results obtained on sphere-in-the-cube problem; total no. of samples = 2000.

| | Training algorithm | No. of prototypes | Training epoches | Recog. error (%) | Training time (s.) | Testing time (s.) | Spurious prototypes |
|---|---|---|---|---|---|---|---|
| A | SFAM 2.1 | 78 | 5.7 | 1.74 | 4 | 1.4 | 0 |
| | SFAM 2.0 | 71 | 7.7 | 1.92 | 4.8 | 1.3 | 0.8 |
| | SFAM 1.1 | 73 | 8.4 | 1.93 | 11.3 | 1.3 | 0.7 |
| | SFAM 1.0 | 73 | 8.4 | 1.93 | 17.8 | 1.3 | 0.7 |
| B | SFAM 2.1 | 78 | 4.3 | 14.30 | 0.453 | 0.19 | 0 |
| | SFAM 2.0 | 55 | 6.8 | 15.30 | 0.468 | 0.14 | 1 |
| | SFAM 1.1 | 57 | 6.7 | 15.50 | 0.85 | 0.14 | 1 |
| | SFAM 1.0 | 57 | 6.7 | 15.50 | 1.06 | 0.14 | 1 |
| C | SFAM 2.1 | 267 | 7.4 | 8.15 | 20.4 | 6.3 | 0 |
| | SFAM 2.0 | 388 | 9 | 9.06 | 36.6 | 9.4 | 11 |
| | SFAM 1.1 | 392 | 8.9 | 8.96 | 81.9 | 9.6 | 12.5 |
| | SFAM 1.0 | 391 | 9.1 | 8.94 | 383.3 | 9.6 | 12 |

in the 2000-sample set simulations (Table 2-B), to 18.8, 10.47, and 4.68 in 20000-sample sets simulations (Table 2-C), respectively.

(4) The results obtained by the SFAM 1.0 and the SFAM 1.1 models are essentially the same: in Table 2-A and Table 2-B, excluding the training times, their results are exactly the same; in Table 2-C, where the numbers of prototypes have increased considerably, their obtained results are slightly different. This proves that in the Kasuba's algorithm, i.e. the SFAM 1.0, the stability of the sorting method is not useful.

(5) Excluding the training times, the results obtained by the SMAF 2.0 and the SFAM 1.0 are slightly different:

In the circle-in-the-square problem (Table 2-A), only on 2 out of 10 datasets, the results varied. On the other 8 datasets all the results were exactly the same. The same was correct for the sphere-in-the-cube-problem, while the number of samples was 2000 (Table 2-B). When the number of samples increased to 20000 (Table 2-C), identical results were obtained on only 4 out of 10 datasets, and on the other 6 datasets obtained results were slightly different.

(6) The results obtained by the SFAM 2.1, in all the cases were different with those obtained by the other models. The major specifications of these results are:
   (1) Decreased number of training epochs, which decreases as much as 36%.
   (2) Decreased training time, which decreases up to 18 times.
   (3) No creation of spurious prototypes, which in the other models amount up to 5% of all created prototypes.

(4) Decreased error rate on test sets, which reaches on the average up to 10%.
(5) The most interesting phenomenon, brought about by the SFAM 2.1, is the impact of activity level of the uncommitted neuron ($T_0$) on the number of created prototypes: setting $T_0$ equal to a larger value (in our experiments equal to 0.9) plays the role of an equalizer for the number of prototypes. In Table 2-A and Table 2-B, where the numbers of prototypes created by the other models are low, the SFAM 2.1 increases their numbers (on the average 10%, and 37%, respectively), but in Table 2-C where the number of prototypes are very high, it decreases their numbers, on the average up to 32%.
(6) The only disadvantage that can be attributed to the SFAM 2.1 is an increased testing time in Table 2-B, due to an increase in the number of prototypes.

(7) In Table 2-C, the average error rate of the SFAM 2.0 is slightly more than that of the SFAM 1.0 algorithm; this problem was caused by high error rate of the SFAM 2.0 on only one dataset. It must be underlined that the difference originates from the activity level of the uncommitted neuron in the SFAM 2.0. More explicitly, if we set the activity level of the uncommitted neurons as small as 0, excluding the training times, all the results obtained by the SFAM 2.0 the SFAM 1.0 will be identical. However, the SFAM 2.0 will be much faster.
(8) Notice should be made that increasing the values of $\alpha$ and $\varepsilon$, in spite of increasing the number of prototypes, does not improve the recognition errors.
(9) When the number of samples are high, by increasing the value of $\varepsilon$, in all models of the SFAM a phenomenon similar to the local minima trap of the MLP occurs. More specifically, after a few training epochs, the recognition error on training set, either does not decrease or oscillates. By increasing the value of $\alpha$ and decreasing the value of $\beta$ (i.e. in slow-learning mode), this problem worsens. For instance in the circle-in-the square problem, setting $\varepsilon = 0.01$, $\alpha = 0.01$, and $\beta = 1$, is enough to make this phenomenon happens.

### 6.2. RESULTS OBTAINED BY THE MLP

The results obtained by the MLP, trained with the BB and the BPLRF algorithms, are presented in Table 3. In this table the recognition errors are the average values of the results obtained on 10 simulations, each starting from a different random initial point.

### THE SETTINGS

The configuration of the network is $l_1 - l_2 - l_3$; where $l_1$ is equal to the dimension of the input patterns, $l_2$ is a parameter that determines the number of cells in the hidden layer and the network size, and is set equal to 7 and 12; and $l_3 = 2$ is the number of cells in the output layer, equal to the number of classes. Since without the threshold term in the activity functions of the hidden-layer cells, none of these benchmarks can

*Table 3*. Training times, and average recognition errors (%) obtained by the MLP on two benchmarks over 10 runs, each run starting from a different random initial point. A: Circle-in-the-square problem; total no. of samples = 20000. B: Cube-in-the-sphere problem; total no. of samples = 2000. C: Cube-in-the-sphere problem; total no. of samples = 20000.

| | | | | | Error on training set | | Error on test set | |
|---|---|---|---|---|---|---|---|---|
| Experiment | $L_2$ | Training epochs | Training time (Sec.) | Testing time (Sec.) | BB | BPLRF | BB | BPLRF |
| A | 7 | 100 | 30 | 0.3 | 2.57% | 0.89% | 2.61% | 1.03% |
| | 12 | 100 | 43 | 0.4 | 1.68% | 0.58% | 1.71% | 0.76% |
| B | 7 | 150 | 5.1 | 0.04 | 5.61% | 3.82% | 6.61% | 5.17% |
| | 12 | 150 | 8.4 | 0.06 | 5.40% | 3.64% | 6.73% | 4.93% |
| C | 7 | 150 | 52 | 0.4 | 3.42% | 2.84% | 3.68% | 2.96% |
| | 12 | 200 | 112 | 0.7 | 2.41% | 1.87% | 2.59% | 2.07% |

be solved by the MLP, we set $l_1$ equal to 3 for the circle-in-the-square problem, and equal to 4 for the sphere-in-the-cube problem.

We found out that on both of these two benchmarks, encountering a local minima mainly depends on the network size (i.e., $l_2$), and parameters tuning (i.e., $\eta$ and $\alpha$). In other words, by selecting improper values for $l_2$, $\eta$, and $\alpha$, there exists a great chance of ending the training in a local minima, and vice versa. That is, by selecting proper values for the network size and the training parameters, the local minima trap can be avoided. For $l_2$ values smaller than 7, at least one in ten simulations ended in a local minima. In the case of choosing one output cell (i.e., $l_3 = 1$), the chance of getting stuck in a local minimum increases. Besides that, the parameter tuning gets more difficult.

For the circle-in-the-square problem, we set $\eta = 0.2$, and $\alpha = 0.05$. For the sphere-in-the-cube-problem we set $\eta = 0.02$, and $\alpha = 0.02$. With smaller values of $\eta$, some of the simulations ended in the local minima, and larger values of $\eta$ destabilized the network.

The number of training epochs were chosen based on the values of TSSE: when the network converges and the TSSE does change considerably, we recorded the recognition errors (results recorded under BB column), and we decreased the values of $\eta$ and $\alpha$ (10 times), and continued the training for another 10 epochs. It means the training for the BB algorithm has been ended 10 epochs earlier than what has been registered in the Table 3.

## ANALYSIS

(1) The most eminent point in this table is high recognition errors obtained on the sphere-in-the-cube problem (Table 3-B & Table 3-C). This observation confirms the suggestion that we made based on the results obtained by the SFAM algo-

rithms: the sphere-in-the-cube problem is much more difficult than its two dimensional variant, the circle-in-the-square problem.

(2) Then is evident the effectiveness of the BPLRF algorithm in decreasing the recognition error. Maximum decrease in average recognition error is observed on the circle-in-the-square problem (Table 3-A), with 65% and 60%, on training set and test set, respectively. The minimum decrease is observed on sphere-in-the-cube-problem with 20000 samples (Table 3-C), which is equal to 16% and 19%, on training set and test set, respectively.

(3) Figure 7 is demonstrating two convergence diagrams, both obtained on the circle-in-the-square problem. In this figure we see that a sudden decrease in the values of training parameters ($\eta$ and $\alpha$) results in a sudden decrease in the values of the cost functions, which improves the recognition rate considerably.

(4) Also we see that by increasing the number of samples, the recognition rate improves.

(5) In Table 3-B, the impact of increasing the network size on the recognition error is trivial. But in Table 3-A, and in Table 3-C, increasing the network size decreases recognition error, by 25%, and 30%, respectively. Considering the number of samples, we can conclude that for a small training set (1000 training samples in Table 3-B), a network with $l_2 = 7$ is large enough, but with 10000 training samples, using a larger network is more appropriate. Therefore, by increasing the number of samples, the size of network also should increase.

(6) For a certain training set (with a fixed number of samples), increasing the value of $l_2$ allows using larger values for training factor ($\eta$), without destabilizing the network.

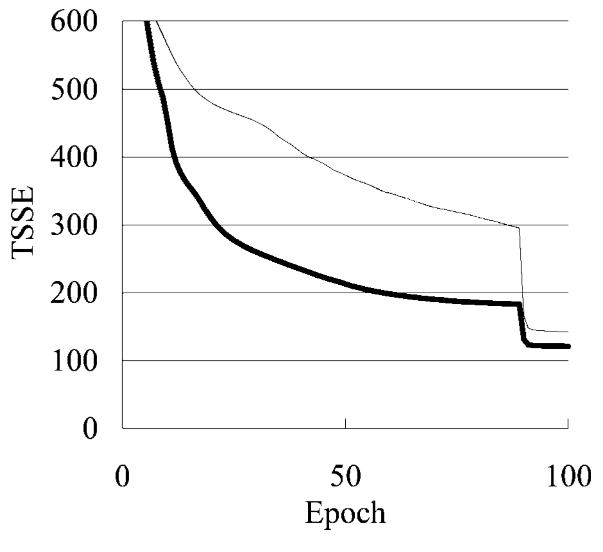

*Figure 7.* The impact of plummeting learning parameters on the cost function, obtained on the circle-in-the-square problem.

### 6.3. COMPARISON OF THE SFAM AND THE MLP NETWORKS ON THE BENCHMARK PROBLEMS

(1) On the circle-in-the-square problem:

The performance of the SFAM 2.1 and the BB algorithm are similar (in terms of recognition error), but the BPLRF algorithm outperforms the SFAM2.1 by 2.2 times less recognition error. The training of the SFAM 2.1 is 10 times faster than that of the MLP, and the MLP's testing is 3 times faster than the SFAM 2.1.

(2) On the sphere-in-the-cube problem:

- The MLP (trained with the BB algorithm) outperforms the SFAM 2.1 algorithm by having 2.8 times less recognition error (on 2000-sample set). This ratio reaches 3.9 times, on 20000-sample set.
- The Training of the SFAM 2.1 is at least 11 times, and 5 times, faster than that of the MLP, on the 2000, and 20000 sample sets, respectively.
- The online operation of the MLP is almost 4.8 times, and 9 times faster than that of the SFAM 2.1, on the 2000, and 20000 sample sets, respectively.

(3) Overall, the SFAM 2.1 has faster training, slower online operation, and higher recognition error, than the MLP. On the other hand, the SFAM algorithms, has much simpler parameter tuning.

## 7. Simulation Results Obtained on the Farsi OCR Problem

In all the simulations, two-thirds of the feature vectors, obtained from the original image file and the first scanned image, were assigned to the training set and two-thirds of them, obtained from the second scanned image were assigned to the test set. This means that 68 samples per character were used for the training and 34 samples per character for the testing; the total number of samples in the training and test sets are 2176 and 1088, respectively. The simulation results obtained on the Farsi OCR datasets are presented in three subsections:

(1) Results obtained by the MLP network
(2) Results obtained by the SFAM network
(3) The SFAM and MLP networks are compared based on their best results.

### 7.1. RESULTS OBTAINED WITH THE BST ALGORITHM

The results obtained by the MLP trained with the BST algorithm on datasets of groups A, C, and dbn4 and dbn5 from group B, are presented in Table 4.

*Table 4.* Simulation Results obtained with the MLP; phase 1: unselective training; phase 2: selective training; MLP1: MLP while $l_2 = \mathbf{20}$; MLP2: MLP while $l_2 = \mathbf{25}$; no. of training samples = 2176, no. of test samples = 1088.

| | MLP2 | | | MLP1 | | | Parameters | | |
|---|---|---|---|---|---|---|---|---|---|
| | Epoch | Error | | Epoch | Error | | $\eta$, $\alpha$ | | |
| Dataset | n, N | Train | Test | n, N | Train | Test | Phase 1 | Phase 2 | Conditions for starting phase 2 |
| db1 | 15, 27 | 0 | 21 | 25, 35 | 0 | 26 | 0.2, 0.2 | 0.3, 0 | (TSSE<60) and (error<25) |
| db2 | 46, 64 | 0 | 29 | 65, 86 | 0 | 39 | 0.2, 0.2 | 0.3, 0.2 | (TSSE<45) and (error<15) |
| db3 | 20, 37 | 0 | 0 | 40, 49 | 0 | 0 | 0.3, 0.4 | 0.2, 0.2 | (TSSE<40) and (error<20) |
| db4 | 20, 24 | 0 | 0 | 30, 35 | 0 | 0 | 0.2, 0.3 | 0.3, 0 | (TSSE<40) and (error<30) |
| db5 | 25, 43 | 0 | 0 | 35, 52 | 0 | 0 | 0.4, 0.1 | 0.4, 0 | (TSSE<40) and (error<20) |
| dbn4 | 75, 107 | 0 | 0 | 90, 122 | 0 | 0 | 0.2, 0.4 | 0.3, 0.2 | |
| dbn5 | 110, 130 | 0 | 0 | 140, 177 | 0 | 0 | 0.2, 0.2 | 0.1, 0 | (TSSE<30) and (error<15) |
| db6 | 500, – | 1060 | 597 | – | – | – | 0.4, 0.06 | – | – |
| db7 | 100, 130 | 0 | 0 | 160, 185 | 0 | 0 | 0.7, 0.3 | 0.2, 0.2 | (TSSE<30) and (error<15) |
| db8 | 95, 125 | 0 | 0 | 160, 188 | 0 | 0 | 0.6, 0.3 | 0.3, 0 | |
| db9 | 500, – | 210 | 113 | – | – | – | 0.15, 0.1 | – | – |
| db10 | 40, 60 | 0 | 0 | 85, 103 | 0 | 0 | 0.5, 0.3 | 0.3, 0 | (TSSE<30) and (error<15) |
| db11 | 80, 117 | 0 | 0 | 105, 130 | 0 | 0 | 0.7, 0.3 | 0.3, 0 | |
| db12 | 75, 78 | 0 | 0 | 150, 190 | 0 | 0 | 0.5, 0.3 | 0.3, 0 | |
| db13 | 30, 53 | 0 | 0 | 60, 68 | 0 | 0 | 0.5, 0.3 | 0.3, 0 | |

The training times and the testing times on some selected datasets are presented in Table 8. In these tables MLP1 and MLP2 represent the MLP with $l_2 = 20$ and $l_2 = 25$, respectively.

## THE SETTINGS

The configuration of the network is $l_1 - l_2 - l_3$; where $l_1$ is the dimension of the input vector that is equal to the number of features per character (see Table 1); $l_2$ is a parameter that determines the number of cells in the hidden layer and the network size, and is set equal to 20 and 25; and $l_3 = 32$ is the number of cells in the output layer, equal to the number of classes.

Adding the threshold term to the activity functions triggered the ‘moving target phenomenon’, so we eliminated it. The stability margin $\zeta$ was set equal to 0.05 in all selective trainings.

A threshold value for the TSSE was chosen, after reaching this value training with the BB algorithm slows down. Thereafter, the network was tested every 5 epochs; if the recognition error on the training set was less than a threshold value the values of the learning parameters (i.e., $\eta$ and $\alpha$) were decreased, and the training algorithm was changed from unselective to selective training and was continued for a maximum of

40 epochs for all datasets, excluding db6 and db9. Training was stopped if the dynamic recognition error[11] on the training set was zero, or if 40 epochs of training were completed. Selective training was not performed on db6 and db9, because selective training is not effective when the error has not decreased enough. In Table 4, *n* and *N* represent the epochs at which selective training starts and ends.

## ANALYSIS

(1) The recognition errors on all test sets are zero, excluding db1, db2, db6 and db9.
(2) We notice that the performance on test sets of db1 and db2 are weaker than those on the corresponding training sets of these datasets. This must be attributed to the inappropriate implementation of the feature extraction method.
(3) The data normalization applied on dbn1 to dbn5 slows down the learning procedure, substantially. In Table 4, the results obtained on dbn4 and dbn5 have been presented as samples. We notice that by data normalization, training has slowed down by more than a factor of 3. (see Table 8 for training times). The same happens on other datasets (i.e., db1, db2, and db3), if normalized.
(4) Another point is the slow convergence on all datasets of group C (db7 to db13), although on these datasets we can use much larger training parameters.
(5) db6 is not suitable for direct classification with the MLP (also with any other classifier) due to a lack of sufficient extracted features. The best recognition rates obtained on the training set and the test set of db6 do not exceed 53% and 51%, respectively, regardless of training time, network size and settings.
(6) The recognition rates on the training and test sets of db9 does not exceed 93%, and 90%, respectively, regardless of training time. This weak performance could be attributed to improper data normalization, improper network size, and the type of extracted features and their insufficient number and consequently insufficient discrimination power of feature vectors. Later we will talk more about this dataset.
(7) For evaluating and ranking datasets, in addition to the recognition error, we adopted two other measures: the number of training epochs and the number of features, which together determine training time and the speed of online operation. Regarding training epochs, the best datasets are: db4, db5, db3, db13, db10, db11, db12, db8 and db7, although in some cases the differences are too slight to be meaningful, and db10, db3 and db13 should be ruled out

[11]Dynamic recognition error is obtained while the network is under training, and after presenting any pattern connection weights will probably change. Therefore, it is different from the recognition error which is obtained after training. Generally, in selective training, the dynamic recognition error on the training set will be larger than the recognition error on the training set, therefore we could stop training earlier by performing a test on the training set after every few epochs of training, but this will violate the stability condition. Although, in our case study this does not cause any problem, it can increase recognition error on the test set and during real online operation.

because of the high dimension of their feature vectors. All in all, db5, db4 and db11 are the best datasets among different groups.

(8) Although db4 needs less training iterations than db5, its whole training time is 40% longer than that of db5 because of more extracted features. The speed of online operation on db4 is half of that on db5 (see Table 8).

(9) We see that with a larger network ($l_2 = 25$), the number of training epochs decreases substantially, at the cost of a longer training time per epoch and a slower online operation. From Table 8 we see that a longer training time per epoch compensates the effect of fewer training epochs and that the training times obtained by $l_2 = 20$ and $l_2 = 25$ are almost equal on all datasets. Therefore, the impact of increasing $l_2$ on the speed of the network is only a slow online operation. However, parameter tuning is easier with larger values of $l_2$.

(10) In order to improve the performance of the MLP on db9 we tried some normalization methods. The most effective method, based on which dbn9 was created, is as follows:

First we applied the data normalization, which was applied in creating dbn1 to dbn5. Then, the range of components was symmetrized[12] by applying the following formula to all the components of the feature vectors:

$$\text{new value} = 2 \times (\text{old value}) - 1. \tag{31}$$

(11) Table 5 compares the results obtained on db9 and its normalized variant dbn9 for two values of $l_2$, i.e., 25 and 40.

Surprisingly, we see that the same normalization method that had a very negative impact on the performance of the system when it was applied on feature vectors extracted by the PCA methods (dbn1 to dbn5), has a very positive impact if it is applied on the feature vectors extracted by diagonal projection. It

*Table 5.* The simulation results obtained with MLP on db9 and dbn9, phase 1: unselective training; phase 2: selective training; in the third line selective training is applied after 400 epochs of unselective training; in the fourth line conditions for starting selective training are: TSSE<60 and error<25.

| | Epoch | Error | | $\eta$, $\alpha$ | | |
|---|---|---|---|---|---|---|
| Dataset | n, N | Train | Test | Phase 1 | Phase 2 | $l_2$ |
| db9 | 500, – | 210 | 113 | 0.15, 0.1 | – | 25 |
| db9 | 500, – | 131 | 70 | 0.15, 0.1 | – | 40 |
| dbn9 | 400, 500 | 75 | 44 | 0.15, 0.1 | 0.08, 0 | 25 |
| dbn9 | 385, 481 | 0 | 0 | 0.15, 0.1 | 0.1, 0 | 40 |

[12]Theoretically, the impact of symmetrization could be achieved by adding threshold terms to the activity functions of the neurons of the hidden layer, however, in this case convergence would be slower.

is even more effective than increasing the network size by 60% (compare the results obtained on db9 while $l_2 = 40$ and the results obtained on dbn9 while $l_2 = 25$). □

## CONSIDERATIONS

(1) We did not encounter the local minima problem in any of our simulations on the Farsi OCR datasets.
(2) By starting from a different initial point the number of training epochs will change slightly, but not so much as to affect the general conclusions.
(3) If larger threshold values for the TSSE and the recognition error are chosen, selective training will be started earlier, and although the recognition error decreases, it will not reach the desired minimum, as is the case with db6 and db9 in Tables 4 and 5.
(4) The reader should recall that in the selective training mode the calculation for weight updating (or backpropagating), which is the most time-consuming step of training, is only carried out for misclassified patterns, whose number at the beginning of the selective training is less than 30, thus one epoch of selective training is at least 10 times faster than that of unselective training, and by decreasing misclassified patterns through time it gets even faster.
(5) Decreasing the learning parameters during selective training stabilizes the network and speeds up training by preventing repeated overtraining and unlearning on some patterns.
(6) The decision about the size of the network should be a compromise based on four considerations:

- The network size should not be less than a minimum, whereby the network will not be able to learn the mappings.
- Speed of training, which increases by increasing the size of the network beyond the minimum size, but by going too far it decreases due to the longer training time per epoch.
- Speed of online operation, which decreases by increasing the size of the network.
- Simplicity of parameter tuning, which improves by increasing the network size.

### 7.2. RESULTS OBTAINED WITH THE SFAM

Since on the Farsi OCR datasets, the number of training samples is low, and the training on most of the datasets ends in 2 epochs (in the final training epoch dynamic recognition error is zero), setting a large value for $T_0$ does not improve the performance of the SFAM. Thus, in this section our purpose of SFAM is the SFAM 2.0, i.e., $T_0$ is determined by equation (13).

*A-The Results Obtained by Fast Training*

We set the parameters as follows:

$$\beta_0 = 1, \beta = 1, \bar{\rho} = 0, \alpha = 0.001, \varepsilon = 0.001$$

and training was continued until a steady state was reached. On all datasets, excluding db6, the dynamic recognition error reached zero; and on db6 continuation of training did not improve the results after the fourth epoch. The results are presented in Table 4.

*B-The results obtained by slow training* $(0.75 < \beta < 1)$

These experiments were only carried out on the datasets of group B and group C; the datasets of group A, db6 and db9 were excluded from the training because the SFAM did not perform well on them. We let $\beta$ vary from 0.75 to 1 in steps of 0.1, and we limited the number of training epochs to 11. Table 5 compares the results with those obtained with $\beta = 1$.

## ANALYSIS

(1) The most eminent point in Table 6 is the positive impact of data normalization on reducing the number of prototype patterns (dbn1 to dbn5, are the normalized

*Table 6.* Simulation results obtained with the SFAM 2.0 in fast-learning mode.

| Dataset | Error | | No. of protoypes | Training epochs |
|---|---|---|---|---|
| | Train | Test | | |
| db1 | 0 | 30 | 105 | 3 |
| db2 | 0 | 49 | 121 | 3 |
| db3 | 0 | 0 | 107 | 3 |
| db4 | 0 | 0 | 161 | 3 |
| db5 | 0 | 0 | 175 | 3 |
| dbn1 | 0 | 43 | 71 | 3 |
| dbn2 | 0 | 61 | 79 | 3 |
| dbn3 | 0 | 0 | 80 | 3 |
| dbn4 | 0 | 0 | 72 | 3 |
| dbn5 | 0 | 0 | 71 | 3 |
| db6 | 14 | 559 | 815 | 4 |
| db7 | 0 | 0 | 75 | 3 |
| db8 | 0 | 0 | 105 | 4 |
| db9 | 0 | 0 | 241 | 5 |
| db10 | 0 | 0 | 84 | 3 |
| db11 | 0 | 0 | 75 | 3 |
| db12 | 0 | 0 | 93 | 3 |
| db13 | 0 | 0 | 78 | 3 |

*Table 7*. Simulation results obtained with the SFAM 2.0 in slow-learning mode, $\beta$ varying from 0.75 to 1.0.

| Dataset | | | $\beta$ values | | | |
|---|---|---|---|---|---|---|
| | | | 0.75 | 0.85 | 0.95 | 1 |
| dbn1 | no.of prototypes. | | 68 | 71 | 73 | 71 |
| | training epochs | | 11 | 8 | 6 | 4 |
| | Error | train | 0 | 0 | 0 | 0 |
| | | test | 44 | 42 | 41 | 43 |
| dbn2 | no. of cat. | | 84 | 81 | 81 | 79 |
| | runs | | 10 | 7 | 6 | 3 |
| | Error | train | 0 | 0 | 0 | 0 |
| | | test | 58 | 50 | 59 | 61 |
| dbn3 | no. of cat. | | 76 | 74 | 81 | 80 |
| | runs | | 11 | 9 | 6 | 3 |
| | Error | train | 2 | 0 | 0 | 0 |
| | | test | 2 | 0 | 0 | 0 |
| dbn4 | no. of cat. | | 73 | 73 | 84 | 82 |
| | runs | | 10 | 8 | 6 | 3 |
| | Error | train | 0 | 0 | 0 | 0 |
| | | test | 0 | 0 | 0 | 0 |
| dbn5 | no. of cat. | | 78 | 76 | 70 | 71 |
| | runs | | 9 | 8 | 10 | 3 |
| | Error | train | 0 | 0 | 0 | 0 |
| | | test | 0 | 0 | 0 | 0 |
| db7 | no. of cat. | | 75 | 74 | 72 | 75 |
| | runs | | 11 | 11 | 8 | 3 |
| | Error | train | 1 | 2 | 0 | 0 |
| | | test | 1 | 2 | 0 | 0 |
| db8 | no. of cat. | | 99 | 96 | 98 | 105 |
| | runs | | 11 | 11 | 11 | 4 |
| | Error | train | 26 | 7 | 0 | 0 |
| | | test | 13 | 4 | 0 | 0 |
| db10 | no. of cat. | | 95 | 96 | 92 | 84 |
| | runs | | 11 | 11 | 10 | 3 |
| | Error | train | 10 | 3 | 0 | 0 |
| | | test | 5 | 2 | 0 | 0 |
| db11 | no. of cat. | | 74 | 69 | 72 | 75 |
| | runs | | 11 | 9 | 8 | 3 |
| | Error | train | 2 | 0 | 0 | 0 |
| | | test | 1 | 0 | 0 | 0 |
| db12 | no. of cat. | | 100 | 100 | 93 | 93 |
| | runs | | 11 | 11 | 8 | 3 |
| | Error | train | 36 | 0 | 0 | 0 |
| | | test | 18 | 0 | 0 | 0 |
| db13 | no. of cat. | | 82 | 81 | 80 | 78 |
| | runs | | 11 | 11 | 10 | 3 |
| | Error | train | 4 | 2 | 0 | 0 |
| | | test | 3 | 2 | 0 | 0 |

variants of db1 to db5). And it gets more remarkable when we recall that this network operates as a look-up table, thus the number of prototypes determines the speed of this network for both training and online operation. This table shows that by normalizing the data, the number of prototypes reduces as little as 20%, up to 60%. Notwithstanding, the data normalization has increased the recognition error on the test sets of the ill-engineered datasets, i.e. dbn1 and dbn2, by as much as 44%.

(2) A $\beta$ value smaller than one increases the number of necessary training epochs substantially, but its impact on the number of prototypes is less than 10% and unpredictable.

(3) The SFAM shows a 100% recognition rate on db9, at the cost of creating too many prototypes and consequently slow training and online operation. For instance, the training and the online operation on db9, are 3.4 and 2.3 times slower, respectively, than those on the SFAM's optimum dataset dbn5.

(4) The performance of the SFAM on dbn9 is not significantly different from that on db9, even by eliminating symmetrization. In this latter case the number of training epochs and the training time decrease to 4 and 42.0 sec., respectively, from 5 and 54 sec.; the number of prototypes increases from 241 to 246, and the testing time increases to from 16.2 to 16.7 sec.

(5) By changing the training parameters a recognition rate of 53% can be obtained on the test set of db6, which is comparable with that obtained by the MLP.

(6) For ranking datasets, based on the performance of the SFAM, we adopted four measures:

    (1) Recognition error on the training and test sets,
    (2) Number of prototypes,
    (3) Number of features,
    (4) Number of training epochs.

By a simple examination of the obtained results and ignoring the trivial differences between them, it is clear that dbn5 and db11 are the best datasets.

## CONSIDERATIONS

(1) The normalization applied on datasets of group A, to create dbn1 to dbn5, is different from *the complement coding* normalization of the SFAM, which is internal and part of the training algorithm. Therefore, on datasets dbn1 to dbn5, and dbn9, two kinds of data normalization are applied.

(2) We tried some other methods of internal normalization as it is permitted by Carpenter et al. [8], e.g., Euclidean norm, but the result was prototype proliferation.

(3) In all of our simulations we did not encounter any spurious prototypes.
(4) The only dataset that has conflicting data is db6, with 14 data mismatches in the training set.

## 7.3. COMPARING THE PERFORMANCES OF THE MLP AND SFAM ON THE FARSI OCR DATASETS

For comparing training times and speeds of online operation we chose 5 datasets: db4, db5, dbn4, dbn5, and db11, on which the best performances of either the MLP or SFAM were obtained. The results are presented in Table 8. In this table the testing time reflects the time spent for testing the network on the whole dataset, i.e., on 3264 samples. Since we have already compared the results obtained with MLP2 and MLP1, in this section only the results obtained with SFAM and MLP1 are compared.

(1) The first point to note in this table is the shorter training time of the SFAM on all datasets. We see that the best training times were obtained, in ascending order, on dbn5, db11, dbn4, db5, and db4. Obviously, db5 and db4 are not suitable for the SFAM.
(2) Then we see that with the MLP the best training times were obtained, in ascending order, on db5, db4, dbn11, dbn5, and dbn4. Obviously, db11, dbn5 and dbn4 are not suitable for the MLP.
(3) The MLP's training time is longer than that of the SFAM on all datasets. On dbn5, which is the optimum dataset of the SFAM, this ratio is 15 times. Notwithstanding, on db5, which is the optimum dataset for the MLP, this ratio is less than 2 times. And on dbn9, which is not the best dataset for any of these networks, this ratio is as high as 21 times (the training times are 11597 sec. and 54.9 sec., for the MLP and the SFAM, respectively).

*Table 8.* The training and testing times of the SFAM 2.0 and the MLP on 5 selected datasets (in second); MLP1: MLP while $l_2 = 20$; MLP2: MLP while $l_2 = 25$.

| Dataset | Train | | | Test | | |
|---|---|---|---|---|---|---|
| | SFAM | MLP2 | MLP1 | SFAM | MLP2 | MLP1 |
| db4 | 46.2 | 93 | 95.7 | 23.7 | 1.8 | 1.5 |
| db5 | 37.2 | 66.6 | 69 | 19.3 | 0.9 | 0.7 |
| dbn4 | 25.3 | 328.8 | 302 | 12.2 | 1.9 | 1.4 |
| dbn5 | 17 | 259.9 | 258 | 7.8 | 0.9 | 0.8 |
| db11 | 18.2 | 213 | 207.3 | 8.8 | 0.95 | 0.8 |

(4) The next point is the testing time which determines the speed of online operation. The MLP is at least 8.7 times faster than the SFAM on all datasets. On db5, which is the most suitable dataset for the MLP, this ratio exceeds 27 times.

(5) The convergence speed of the MLP on the datasets of group A is at least twice that on datasets of group C, therefore, datasets of group C do not befit the MLP. On the other hand, considering that the performance of the SFAM on datasets of group C is similar to that on datasets of group B, before performing experiments we could predict that the MLP will not do well on datasets of group C, and experimental results would affirm this prediction.

(6) Although applying the data normalization method of dbn9 (without symmetrization) on other datasets of group C speeds up convergence of the MLP on them, the results are far from being comparable to those obtained on datasets of group A.

(7) Therefore, we can generalize the result and say: 'If one dataset suits one of these networks it will be inappropriate for the other one'. Now this question arises that is it possible to build a dataset which suits both of these networks? The best answer, that we can give at the moment is that this dataset will most probably be a compromise, therefore, it will not be the best for either of these networks.

(8) Comparing these two networks on a single dataset will lead to a wrong conclusion. The logical approach is to compare the results obtained with the MLP and the SFAM while embedded in their own optimum systems. By doing this, i.e., comparing the results of the SFAM on dbn5 against the results of the MLP1 on db5, we see that the training of the SFAM is 4 times faster than the training of the MLP, and the online operation of the MLP is 11 times faster than that of the SFAM.

(9) The recognition errors of the SFAM on ill-engineered and erroneous datasets, i.e., db1, db2, and their normalized counterparts dbn1 and dbn2, is more than that of the MLP. In addition, data normalization has increased the recognition error of the SFAM on these datasets by up to 44%.

(10) If we choose the slow learning mode, like some researchers who faced problems with the fast learning mode, the SFAM will lose its fast training advantage. In this case, the SFAM only retains its advantages of flexible configuration and easy parameter tuning. And if the problem that the NNs are being used to solve does not require learning of new patterns, its advantage will be restricted to just easier parameter tuning.

(11) The original FAM is much slower than the SFAM because of its redundant operations, therefore its training will be slower than the MLP.

(12) Considering the short training times of both the SFAM and the MLP on their own optimum datasets, a shorter training time can not be an important factor when comparing these networks.

## 8. Conclusions

We presented a new variant of the SFAM algorithm, and compared its performance against Kasuba's algorithm on two benchmark problems. We showed that our algorithm is much faster than Kasuba's algorithm, and by increasing the number of training samples, the difference in the speed grows enormously.

For training the MLP on the benchmarks, we presented the BPLRF algorithm, and showed that how the performance of the MLP improves by simply plummeting the training parameters at the end of training.

We showed that a classifier should not be evaluated by a simple replacement of classifiers in a system that is not designed or optimized for that classifier. That is to say, if we change a classifier in a pattern recognition system it is necessary to make some changes at other stages of the system in order to obtain the best performance. We demonstrated that if one dataset suits either the MLP or the SFAM it will be inappropriate for the other one.

Comparing the results obtained with the MLP and our SFAM, embedded in their customized systems, reveals that:

- On all of the three case studies, the SFAM's convergence in the fast-training mode is faster than that of the MLP, and the online operation of the MLP is faster than that of the SFAM.
- On the benchmark problems the MLP has much better recognition rate than the SFAM.
- On the Farsi OCR problem, the recognition error of the SFAM is higher than that of the MLP on ill-engineered datasets, but equal on well-engineered ones.
- The flexible configuration of the SFAM, i.e., its capability to increase the size of the network in order to learn new patterns, as well as its simple parameter adjustment, remain unchallenged by the MLP.

## Acknowledgments

This work has been partially supported within the framework of the Slovenian-Iranian Bilateral Scientific Cooperation Treaty. The authors would like to thank Paul McGuiness for linguistic revision. The authors would like to thank the anonymous reviewers for their invaluable suggestions, and comments.

## Appendix

### FEATURE EXTRACTION METHODS

*Group A–db1 to db5*

These datasets were created by extracting the principal components, extracted by a single layer feedforward linear neural network with Generalized Hebbian Training Algorithm (GHA)[48, 52], as summarized in the following:

For extracting principal components, we used $m - l$ single layer feedforward linear network with Generalized Hebbian Training Algorithm (GHA).

*db1* – To train the network: $8 \times 8$ non-overlapping blocks of the image of every character were considered as an input vector. The image was scanned from top left to bottom right and $l$ was set equal to 8. Therefore for every character 72 features were extracted. The training was performed with 34 samples per character and the learning rate was set to $\eta = 7 \times 10^{-3}$. To give enough time for the network to learn the statistics of data, training procedure was repeated for three epochs.

*db2* – The same as db1, but $l$ was set equal to 6, thus for any character 54 features were extracted.

*db3* – The image matrix of any character was converted into a vector, by scanning vertically from top left to bottom right, then this vector was partitioned into 9 vectors which were inserted into the network as 9 input vectors. In this way, 72 features were extracted for every character.

*db4* – Similar to db1, but the dimension of the input blocks was considered to be $24 \times 3$, i.e., every three rows were considered as one input vector. In this way, for any character 64 features were extracted.

*db5* – Similar to db4, but $l$ was set equal to 6, thus for any character 48 features were extracted.

*Group B–dbn1 to dbn5*

These datasets are normalized versions of db1 to db5. After creating any dataset, the feature vectors were normalized by mapping the $i$th component of all the vectors into the interval [0, 1].

*Group C–db6 to db13*

*db6* – This dataset was created by the zoning method. Each character image was divided into four overlapping squares and the percentage of black pixels in each square was obtained. The size of overlap was set to two pixels in each edge, which yields the best recognition rate.

*db7* – Pixel change coding was used to extract the feature vectors of this dataset. The vectors were divided to 8, to assure that all the component values are less then one.

*db8* – The feature vectors of this dataset were extracted by vertical and horizontal projections. The vectors were divided to 24, to assure that all the component values are less then one.

*db9* – The feature vectors of this dataset were extracted by diagonal projection. Ten components from the beginning and seven components from the end were deleted, because their values were zero for all characters. The vectors were divided to 24, to assure that all the component values are less then one.

*db10* – By concatenating the feature vectors of db8 and db9, feature vectors of this dataset were extracted.

*db11* – The feature vectors of this dataset were created by concatenating the feature vectors of db6 and db7.

*db12* – The feature vectors of this dataset were created by concatenating the feature vectors of db6 and db8.

*db13* – The feature vectors of this dataset were created by concatenating the feature vectors of db11 and some selected features from db8, that is 10 features from the middle of both vertical and horizontal projections.

## References

1. McCulloch, W. S. and Pitts, W.: A logical calculus of the ideas imminent in nervous activity, *Bulletin of Mathematical Biophysics*, **5** (1943), 115–133.
2. Grossberg, S.: Adaptive pattern classification and universal recoding, II: feedback, expectation, olfaction and illusions, *Biological Cybernetics*, **23** (1976), 187–202.
3. Carpenter, G. A. and Grossberg, S.: A massively parallel architecture for a self organizing neural pattern recognition machine, *Computer Vision, Graphics and Image Processing*, **37** (1987), 54–115.
4. Carpenter, G. A. and Grossberg, S.: ART2: Self-organization of stable category recognition codes for analog input patterns, *Applied Optics*, **26**(23) (1987), 4919–4930.
5. Carpenter, G. A. and Grossberg, S.: ART3: Hierarchical search using chemical transmitters in self-organizing pattern recognition architecture, *Neural Networks*, **3** (1990), 129–152.
6. Carpenter, G. A., Grossberg G. S. and Reynolds J. H.: ARTMAP: Supervised real-time learning and classification of mon-stationary data by a self-organizing neural network, *Neural Networks*, **4** (1991), 565–588.
7. Carpenter, G. A., Grossberg, S. and Rosen, D. B.: Fuzzy ART: Fast stable learning and categorization of analog patterns by an adaptive resonance system, *Neural Networks*, **4** (1991), 759–771.
8. Carpenter, G. A., Grossberg, S., Markuzon, N., Reynolds, J. H. and Rosen, D. B.: Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps, *IEEE Trans. on Neural Networks*, **3**(5) (1992), 698–713.
9. Carpenter, G. A., Grossberg, S. and Rosen, D. B.: ART2-A: An Adaptive Resonance Algorithm for Rapid Category Learning and Recognition, *Neural Networks*, **4**, (1991), 493–504.
10. Carpenter, G. A. and Grossberg, S.: A self-organizing neural network for supervised learning, recognition, and prediction, *IEEE Communications Mag*., pp. 38–49, Sep. 1992.
11. Kasuba, T.: Simplified Fuzzy ARTMAP, *AI Expert*, **8**(11) (1993), 18–25.
12. Marriott, S. and Harrison, R. F.: A modified fuzzy ARTMAP architecture for the approximation of noisy mappings, *Neural-Networks*, **8**(4) (1995), 619–641.
13. Malkani, A. and Vassiliadis, C. A.: Parallel implementation of the fuzzy ARTMAP neural network paradigm on a hypercube, *Expert Systems*, **12**(1) (1995), 39–53.
14. Chen, B. and Hoberock, L. L.: A Fuzzy Neural Network Architecture for Fuzzy Control and Classification, *ICNN 96, The 1996 IEEE International Conference on Neural Networks* (Cat. No. 96CH35907), IEEE, **2**, pp. 1168–1173 New York, USA, 1996.

15. Blume, M., Van-Blerkom, D. A. and Esener, S. C.: Fuzzy ARTMAP Modifications for Intersecting Class Distributions, *WCNN'96, World Congress on Neural Networks*, International Neural Network Society 1996 Annual Meeting. pp. 250–255. Lawrence Erlbaum Assoc, USA, 1996.
16. Chee, P. L. and Harrison, R. F.: Modified fuzzy ARTMAP approaches bayes optimal classification rates: An empirical demonstration, *Neural Networks*, **10**(4) (1997), 755–774.
17. Srinivasa, N.: Learning and generalization of noisy mappings using a modified PROBART neural network, *IEEE Trans. Sign. Proc.*, **45**(10) (1997), 2533–2550.
18. Hsien, L. T. and Shie, J. L.: A Neural Network Model for Spoken Word Recognition, *IEEE International Conference on Systems, Man, and Cybernetics, Computational Cybernetics and Simulation* (Cat. no. 97CH36088-5), IEEE, **5**, pp. 4029–4034 New York, USA, 1997.
19. Carpenter, G. A. and Markuzon, N.: ARTMAP-IC and medical diagnosis: Instance counting and inconsistent cases, *Neural Networks*, **11**(2) (1998), 323–336.
20. Healy, M. J. and Caudell, T. P.: Guaranteed two-Pass convergence for supervised and inferential learning, *IEEE Tran. Neural Networks*, **9**(1) (1998), 195–204.
21. Jervis, B. W., Garcia, T. and Giahnakis, E. P.: Probabilistic simplified fuzzy ARTMAP (PSFAM) and application to biosignal data, *IEE Proc. Science Measur. and Tech.*, **146**(4) (1999), 165–169.
22. Verzi, S. J., Heileman, G. L., Georgiopoulos, M. and Healy, M. J.: Boosted ARTMAP, *IEEE International Joint Conference on Neural Networks Proce. IEEE World Congress on Computational Intelligence* (Cat. no. 98CH36227), IEEE, **1**, pp. 396–401 New York, USA, 1998.
23. Dagher, I., Georgiopoulos, M., Heileman, G. L. and Bebis, G.: Fuzzy ARTVar: An Improved Fuzzy ARTMAP Algorithm, *IEEE International Joint Conference on Neural Networks Proc. IEEE World Congress on Computational Intelligence* (Cat. no. 98CH36227). IEEE, **3**, pp. 1688–1693 New York, USA, 1998.
24. Werbos, P. J.: Beyond Regression: New Tools for Prediction and Analysis in Behavioral Science, Ph.D Thesis, Harvard University, Cambridge, MA., 1974.
25. Parker, D.B. Learning Logic, Invention Report s81-64, File 1, Office of Technology Licensing, Stanford University, March 1982.
26. Rumelhart, D. E. and McClelland, J. L.: *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, MIT press, 1986.
27. Jondarr, G.: Backpropagation Family Album, *Technical Report* C/TR96-05, Department of Computing, Macquarie University, New South Wales, August, 1996.
28. Aggarwal, R. K., Xuan, Q. Y., Johns, A. T., Furong, L. I. and Bennett, A.: A novel approach to fault diagnosis in multicircuit transmission lines using fuzzy ARTMAP neural networks, *IEEE Trans. on Neural networks*, **10**(5) (1999), 1214–1221.
29. Gardner, J. W., Shin, H. W., Hines, E. L. and Dow, C. S.: An electronic nose system for monitoring the quality of potable water, *Sensors and Actuators B (Chemical)*, **B69**(3) (2000), 336–341.
30. Heinke, D. and Hamker, F. H.: Comparing neural networks: a benchmark on growing neural gas, growing cell structures, and fuzzy ARTMAP, *IEEE Trans. on Neural Networks*, **9**(6) (1998), 1279–1291.
31. Hines, W. L., Llobet, E. and Gardner, J. W.: Neural network based electronic nose for apple ripeness determination, *Electronics Letters*, **35**(10) (1999), 821–823.
32. Keyvan, S., Xiaolong, S. and Kell, M.: Nuclear fuel, pellet inspection using artificial neural networks, *Journal of Nuclear Materials*, **264**(12) (1999), 141–154.

33. Liobet, E., Hines, E. L., Gardner, J. W., Bartlett, P. N. and Mottram, T. T.: Fuzzy ARTMAP based electronic nose data analysis, *Sensors and Actuators B (Chemical)*, **B61**(13) (1999), 183–190.
34. Ludwig, L., Sapozhnikova, E., Lunin, V. and Rosenstiel, W.: Error classification and yield prediction of chips in semiconductor industry applications, *Neural Computing & Applications*, **9**(3) (2000), 202–210.
35. Mills, H., Burton, D. R. and Lalor, M. J.: Applying backpropagation neural networks to fringe analysis, *Optics and Lasers in Engineering*, **23**(5) (1995), 33–41.
36. Muchoney, D. and Williamson, J.: A Gaussian adaptive resonance theory neural network classification algorithm applied to supervised land cover mapping using multitemporal vegetation index data, *IEEE transactions on Geoscience and Remote Sensing*, **39**(9) (2001), 1969–1977.
37. Obaidat, M. S., Khalid, H. and Sadoun, B.: Ultrasonic transducer characterization by neural networks, *Information Sciences*, **107**(14) (1998), 195–215.
38. Obaidat, M. S. and Sadoun, B.: Verification of computer users using keystroke dynamics, *IEEE Trans. on Systems, Man and Cybernetics, Part B (Cybernetics)*, **27**(2) (1997), 261–269.
39. Raveendran, P., Palaniappan, R. and Omatu, S.: Fuzzy ARTMAP classification of invariant features derived using angle of rotation from a neural network, *Information Sciences*, **130**(14) (2000), 67–84.
40. Shin, H. W., Llobet, E., Gardner, J. W., Hines, E. L. and Dow, C. S.: Classification of the strain and growth phase of cyanobacteria in potable water using an electronic nose system, *IEE Proc. Science, Measurement and Technology*, **147**(4) (2000), 158–164.
41. Walczak, S.: Neural network models for a resource allocation problem, *IEEE Trans. on Systems, Man and Cybernetics, Part B (Cybernetics)*, **28**(2) (19998), 276–284.
42. Song, X. H., Hopke, P. K., Bruns, M. A., Bossio, D. A. and Scow, K. M.: A fuzzy adaptive resonance theory supervised predictive mapping neural network applied to the classification of multivariate chemical data, *Chemometrics and Intelligent Laboratory Systems*, **41**(2) (1998), 161–170.
43. Zhang, Z. and Manikopoulos, C.: Neural networks in statistical anomaly intrusion detection, *Neural Network World*, **11**(3) (2001), 305–316.
44. Carpenter, G. A., Gjaja, M. N., Gopal, S. and Woodcock, C. E.: ART Neural Networks for Remote Sensing: Vegetation Classification from Landsat TM and Terrain Data, *IEEE Trans. on Geoscience and Remote Sensing*, **39**(2) (1997), 308–325.
45. Rojas, R.: *Neural Networks: A Systematic Introduction*, New York, Springer-Verlag, 1996.
46. Fahlman, S. C. and Lebiere, C.: The cascade-correlation learning architecture, *Technical Report*, CMU-CS-90-100, Carnegie Mellon University, Pittsburgh, 1988.
47. Fahlman, S. E.: An empirical study of learning speed in backpropagation networks, *Technical Report*, CMU-CS-88-162, Carnegie Mellon University, Pittsburgh, 1988.
48. Haykin, S.: *Neural Networks: A Comprehensive Foundation*, USA, Macmillan, 1994.
49. Theodoridis, S. and Koutroumbas, K. *Pattern Recognition*, USA, Academic Press, 1999.
50. Vakil, M.T. and Pavešic, N. Backpropagation with declining learning rate, In: *Proc. of the 10th Electrotechnical and Computer Science Conference*, pp. 297–300. Portorož, Slovenia, **B**, 2001.
51. Vakil, M.-T.: Farsi Character Recognition Using Artificial Neural Networks, Ph.D Thesis, Faculty of Electrical Engineering, University of Ljubljana, October 2002.
52. Diamantaras, K. I. and Kung, S. Y.: *Principal Component Neural Networks: Theory and Applications*, JohnWiley & Sons, 1996.