Neurocomputing 72 (2009) 2079–2092

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

AG-ART: An adaptive approach to evolving ART architectures

A. Kaylani [a], M. Georgiopoulos [a,*], M. Mollaghasemi [a], G.C. Anagnostopoulos [b]

[a] School of Electrical Engineering and Computer Science, University of Central Florida, Orlando, FL 32816, USA
[b] Florida Institute of Technology, Melbourne, FL 32901, USA

ARTICLE INFO

Available online 16 December 2008

Keywords:
Machine learning
Classification
ARTMAP
Genetic algorithms
Genetic operators
Category proliferation

ABSTRACT

This paper focuses on classification problems, and in particular on the evolution of ARTMAP architectures using genetic algorithms, with the objective of improving generalization performance and alleviating the adaptive resonance theory (ART) category proliferation problem. In a previous effort, we introduced evolutionary fuzzy ARTMAP (FAM), referred to as genetic Fuzzy ARTMAP (GFAM). In this paper we apply an improved genetic algorithm to FAM and extend these ideas to two other ART architectures; ellipsoidal ARTMAP (EAM) and Gaussian ARTMAP (GAM). One of the major advantages of the proposed improved genetic algorithm is that it adapts the GA parameters automatically, and in a way that takes into consideration the intricacies of the classification problem under consideration. The resulting genetically engineered ART architectures are justifiably referred to as AG-FAM, AG-EAM and AG-GAM or collectively as AG-ART (adaptive genetically engineered ART). We compare the performance (in terms of accuracy, size, and computational cost) of the AG-ART architectures with GFAM, and other ART architectures that have appeared in the literature and attempted to solve the category proliferation problem. Our results demonstrate that AG-ART architectures exhibit better performance than their other ART counterparts (semi-supervised ART) and better performance than GFAM. We also compare AG-ART's performance to other related results published in the classification literature, and demonstrate that AG-ART architectures exhibit competitive generalization performance and, quite often, produce smaller size classifiers in solving the same classification problems. We also show that AG-ART's performance gains are achieved within a reasonable computational budget.

1. Introduction

The adaptive resonance theory (ART) was developed by Grossberg [19]. Some of the ART architectures that have appeared in the literature include fuzzy ARTMAP (FAM) [7], ellipsoidal ARTMAP (EAM) [2], and Gaussian ARTMAP (GAM) [36]. All of these ART architectures possess a number of desirable properties, such as they can solve arbitrarily complex classification problems, they converge quickly to a solution (within a few presentations of the list of input/output patterns belonging to the training set), they have the ability to recognize novelty in the input patterns presented to them, they can operate in an on-line fashion (new input patterns can be learned by the ART system without retraining with the old input/output patterns), and they produce answers that can be explained with relative ease.

One of the limitations of these ART architectures that has been repeatedly reported in the literature is the category proliferation problem. This refers to the problem where ART, in the process of solving a classification problem, creates unnecessarily large architectures. This problem is more amplified when the data in the classification problem are noisy, and/or significantly overlapping. Another limitation of these ART architectures is the dependence of their performance on the parameters chosen in the training phase (e.g., vigilance parameter, choice parameter, order of training pattern presentation). Good choices for these parameters is problem-dependent, thus requiring experimentation with various parameter choices (an expensive proposition) in order to obtain the best possibly performing ART networks.

To alleviate these problems, genetic fuzzy ARTMAP (GFAM) was introduced in [1]. GFAM uses a genetic algorithm (GA) (see [16]) to evolve simultaneously the weights, as well as the topology of FAM neural networks. It starts with a population of trained FAMs, whose number of nodes in the hidden layer and the values of the interconnection weights converging to these nodes are fully determined (at the beginning of the evolution) by ARTs training rules. To this initial population of FAM networks, GA operators are applied to modify these trained FAM architectures (i.e., number of nodes in the hidden layer, and values of the interconnection weights) in a way that encourages better generalization and smaller size architectures.

In this paper we propose the use of an improved GA for the evolution of ART architectures. The proposed GA relies on adaptive GA parameter control mechanisms. While GFAM required the user

* Corresponding author. Tel.: +1 407 823 5338; fax: +1 407 823 5835.
E-mail address: michaelg@mail.ucf.edu (M. Georgiopoulos).
URLS: http://www.ucf.edu (M. Georgiopoulos), http://www.fit.edu (G.C. Anagnostopoulos).

0925-2312/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2008.09.016

to choose values for the probability of an ART category deletion, and for the probability of a category mutation, AG-ART finds good values for these parameters through an adaptation mechanism that takes into consideration the specificities of the problem under consideration. Hence, not only AG-ART is more elegant (requires minimal user intervention) than GFAM, but as our experiments illustrate it is faster than GFAM, because the GA parameters are more wisely chosen through appropriate adaptation mechanisms.

In this paper, we also extend the use of GAs to solve the ART category proliferation problem, and simultaneously improve ARTs generalization performance to two other architectures: EAM and GAM. The evolution of these trained ART networks produces a network, referred to as AG-EAM or AG-GAM, which have better generalization performance and smaller size than the ART networks that we started with in the initial population. AG-FAM, AG-EAM or AG-GAM is the FAM, EAM or GAM network, respectively, that at the last generation of the evolutionary process attained the highest fitness function value. In the evolution of ART trained networks, in addition to the common GA operators, such as crossover and mutation, we use a unique (and needed) genetic operator, referred to as *Prune* operator. This operator, allows us to destroy ART categories, thus leading us to ART networks of smaller size.

The ultimate measure of the justification for the existence of the AG-ART collection of architectures is how well they fare in comparison with other ART architectures (including GFAM), as well as other than ART classifiers. Consequently, in this paper we compare AG-ART with GFAM, ssFAM, ssEAM, ssGAM (see [3]) and a few other non-ART-based classifiers. This comparison is based on the accuracy, size and computational complexity of the classifiers. In particular, our results show that AG-FAM, AG-EAM and AG-GAM perform well on a number of classification problems, and optimally on some of these problems. For instance, AG-ART gave a better generalization performance and a smaller than, or equal, size network (in almost all problems tested), compared to ssFAM, ssEAM and ssGAM networks, requiring reduced (sometimes significantly reduced) computational effort to achieve these advantages. Furthermore, the AG-FAM accuracy and size results are similar with the GFAM results, but AG-FAM attains these good results at a reduced computational cost, compared to GFAM. Finally, AG-ART architectures compared very favorably with a number of other classifiers, whose results have been published in the literature. In a nutshell, this comparison demonstrates that AG-ART compares very favorably with a number of other classification approaches, proposed in the literature.

The organization of this paper is as follows: in Section 2 we present a detailed overview of the relevant literature. In this section we discuss in detail the issue of GA parameter adaptation, which is a central issue in the development of the AG-ART architectures. In this section we also present a brief overview (only the absolutely necessary information) of the ART architectures under consideration (FAM, EAM, GAM). In Section 3 we describe all the important elements pertinent to the evolution of the ARTMAP architectures, according to the suggested AG-ART specifications. In Section 4, we describe the experiments, the datasets used in the experimentation, and we provide the performance comparisons between the AG-ART and other classifiers (GFAM, other ART-based classifiers and non-ART-based classifiers). In Section 5, we summarize our contributions and findings.

2. Literature review

As mentioned above, one of the objectives of AG-ART introduced in this paper is to solve a major limitation of ART, that has been repeatedly reported in the literature, and known as *category proliferation problem*. This refers to the creation of unnecessarily large number of ART categories to solve a classification problem under consideration, a phenomenon that is mostly observed when the data are noisy and/or of overlapping nature. Quite often the category proliferation problem, observed in ART, is connected with the issue of over-training. Over-training happens when ART is trying to learn the training data perfectly at the expense of degraded generalization performance (i.e., classification accuracy on unseen data) and also at the expense of creating many categories to represent the training data (leading to the category proliferation problem). Also, it has been connected to other related ART limitations, such as the representative inefficiency of the categories or the excessive triggering of the match tracking mechanism due to existence of noise.

Since the early 1990s a number of ART-related papers were published in the literature with the intent of addressing these ART inefficiencies. In [27], PROBART eliminates the match tracking mechanism and instead stores probability information in the map field, thus eliminating one of the causes of ART over-training. Micro-ARTMAP [17] introduces the idea of an entropy of a category and allows the creation of ART categories that encode patterns, whose labels can be mixed, provided that the category entropy is less than a designated threshold. Safe Micro-ARTMAP [18], introduced later, adds a mechanism to limit the growth of a category in response to a single pattern. Semi-supervised learning of ART was introduced [3] in 2003, where it allows, with a certain tolerance, categories to encode patterns that are not mapped to the same label. Three semi-supervised architectures were introduced in [3]: ssFAM, ssGAM and ssEAM. In [23] the authors suggest the use of crossvalidation to prevent over-training and therefore avoid the creation of unnecessary categories. In addition, in the work by Carpenter et al. [8], Williamson [37] and Parrado-Hernadez et al. [31] the ART structure is changed from a winner-take-all to a distributed version, and simultaneously slow learning is employed, with the intent of creating fewer ART categories and reducing the effects of noisy patterns.

In our work here, we propose (yet another!) method (AG-ART) to solve the ART category proliferation problem, while simultaneously improving the generalization performance in ART. This method uses GAs [16] to optimize the size of ART structures created, as well as their generalization, and it does this more elegantly and more efficiently than the recently introduced GFAM architecture [1].

GAs are a class of population-based stochastic search algorithms that are developed from ideas and principles of natural evolution. An important feature of these algorithms is their population-based search strategy. Individuals in a population compete and exchange information with each other in order to perform certain tasks. GAs are capable of finding the global optima rather than the local optima as is the case with gradient descent procedures. Also, GAs are not as sensitive to the initialization of weights and they do not require a continuous or differentiable fitness function to operate. GAs [16] have been extensively used to evolve (optimize) artificial neural networks.

2.1. Evolutionary neural network architectures

The literature is rich with articles applying evolutionary optimization algorithms to train neural networks [10,15,30,35]. The majority of this work is focused on MLP neural networks [38]. However, a number of authors proposed using evolutionary optimization algorithms with other neural network models such as RBF neural networks [15,35]. In [38] a comprehensive literature review was conducted to summarize the prior efforts that aimed

at combining evolutionary optimization algorithms with neural networks. The author divides these efforts into three main classes; evolution of connection weights, evolution of architectures and evolution of learning rules.

Evolution of weights of neural networks often uses Gaussian perturbation in the mutation operator to bring about changes in the weights [14]. When relying on mutation to change the network structure and/or weights, some researchers incorporated an adaptive mechanism to control the severity of mutations [4,39]. This was done by adopting a simulated annealing-based strategy, in which mutations are allowed to be aggressive at the beginning of the evolutionary process, while only mild mutations are permitted as the process progresses towards the optimum. Evolution of architectures often uses mutation of the architectural structure that includes addition and deletion of connections and nodes. In this work, structural additions are avoided as the probability of adding a viable component is very low.

Many authors [4,30,38] point out to the problems associated with using crossover when evolving the neural network structure. The crossover has a destructive effect as it may combine parents that result in non-viable offspring solutions. The probability of producing offspring solutions with worse fitness than the parents is relatively high when crossover is used. This significantly reduces the effectiveness of the EA. To combat this problem, some authors [4,30] eliminate the use of crossover and rely only on mutation.

As it will be explained later, in evolving ART architectures a 2-level encoding scheme is adopted. The 2-level encoding scheme has the advantage of being able to apply viable crossover operations at level 1 of the representation. On the other hand, adaptive mutations are applied at both levels, as will be shown later. Mutations at level 1 cause structural modifications (referred to as Pruning), while mutations applied at level 2 cause weight modifications.

To the best of our knowledge, work that utilizes GAs and ART neural network architectures is rather limited. For instance, in [26], a GA was employed to appropriately weigh attributes of input patterns before they were fed into the input layer of FAM. The results reported reveal that this attribute weighting was beneficial because it produced a trained ART architecture of improved generalization. In our work, we do not use the GAs to optimize the weight with which each attribute of the input patterns is affecting the input layer of the ART architecture. Instead, as mentioned above, we use GAs to optimize the topology and the weights of trained ART architectures.

2.2. Adaptation in GAs

When applying a GA to solve an optimization problem, we not only need to choose the algorithm, representation, and operators for the problem, but we also need to choose parameter values and operator probabilities for the GA so that it will not only find the solution, but also find the solution efficiently. In many cases, researchers choose these parameter values and operator probabilities based on experience or experimentation on a specific problem. The choice of the parameters and operator probabilities has attracted a significant amount of research. A number of researchers suggested good parameter values as a result of extensive experimentation on a range of optimization problems. For examples, Jong [21] proposed to set $p_m = 0.001$ and $p_c = 0.6$, where p_m and p_c stand for the probability of mutation and crossover, respectively. In [28] the authors propose $p_m = 1/\beta$ and in [5] $p_m = 1.75/n * \beta^{1/2}$, where n represents the population size and β represents the bit-length of a chromosome.

Finding good GA parameter values for a certain optimization problem is a time consuming process. It is prone to human error which may lead to suboptimal results. Moreover, what might be initially considered as a good parameter setting could, in the progress of evolution, prove not to be as good, since the search may move to different regions of the solution space. As a result, an emphasis was placed on designing GAs where the parameters automatically adapt to the problem at hand.

The GA parameters that affect its performance include environmental parameters such as population size and the objective (fitness) function. The adaptation can be applied to global parameters such as mutation rate, mutation strength or crossover rate that are affecting all individuals in the population, or applied to local parameters where the parameter value is customized for each individual. Also, some research proposes the customization of the parameter setting at the component level (part of the individual).

The majority of the research though focuses on adapting the mutation rate or mutation strength. For real-valued representations, the term mutation strength, sometimes called mutation step size, refers to the magnitude of change in each mutated variable. This is different in binary representation schemes, where the term mutation rate is used to express how probable it is for a certain binary variable to be changed (inverted).

The adaptation approaches can be distinguished in three groups, in order of increasing complexity: *deterministic*, *adaptive*, and *self-adaptive* [20]. Each of these approaches is described below.

- Deterministic: Deterministic adaptation refers to the dynamic adjustment of a GA parameter using a deterministic rule, and without feedback from the quality of the solution achieved by the evolutionary process. This rule can be based on a schedule (similar to simulated annealing) or number of generations (see [13]). In general, the objective in this approach is to alter the GA parameters in such a way that result in a wide-spread search at the beginning of the optimization, and increasingly localized search at later stages. An example of this approach can be found in [24], where the mutation step sizes are discounted by a constant factor each time an offspring is produced.
- Adaptive: This approach uses some form of feedback from the GA that is used to determine the direction and/or magnitude of the change to the GA parameter. In [4], the standard deviation of the Gaussian mutation was varied based on a temperature parameter. The temperature parameter was defined as $T(\eta) = 1 - f(\eta)/f_{max}$, where f_{max} is the maximum fitness for a given task. Thus, the temperature of a solution is determined by how close the solution is to being an optimal solution for the task under consideration. Solutions with a high temperature are mutated severely, and those with a low temperature are mutated only slightly. This allows a coarse-grained search initially, and a progressively finer-grained search as the GA approaches a solution for the assigned task.
  In [32] the authors suggest the use of a feedback signal that is defined by the difference $fit_{max}(P,t) - \mu(P,t)$, where $fit_{max}(P,t)$ is the maximum fitness and $\mu(P,t)$ is the average fitness of solutions in population $P(t)$ at generation t. This difference is used as an indication of closeness to convergence. This difference is likely to be less for a population that has converged to an optimal solution than that for a random population, scattered in the solution space. The authors define the adaptive rates of crossover and mutation for all chromosomes to be inversely proportional to this difference. To make this mechanism less disruptive for good solutions, the

mechanism is adjusted to have low values of parameters for high fitness values and high values of parameters for low fitness values, as follows: $p_m(x) = k_1(fit_{max}(P,t) - fit(x,t))/(fit_{max}(P,t) - \mu(P,t))$, and $p_c = k_2(fit_{max}(P,t) - fit(x,t))/(fit_{max}(P,t) - \mu(P,t))$. Therefore, in this case, the parameters are controlled at the individual level.

- Self-adaptive: In this approach, the GA parameters undergo evolution. The GA parameters are encoded in the chromosomes and evolved as part of the solution. The encoded parameters will lead to better fitness for individuals with better parameters values, and since these individuals are more likely to survive and reproduce, this mechanism will propagate these better parameter values.

In our implementation genetic optimization of ARTMAPs, referred to as AG-ART, we chose to use an adaptive approach, where a feedback signal is used to determine the operator probability at the component level.

2.3. The ARTMAP architectures

The FAM neural network architecture was introduced by Carpenter and Grossberg in their seminal paper [7]. Since its introduction, other ART architectures have been introduced into the literature. The focus in this paper is on FAM and two other ART architectures: EAM [2] and GAM [36]. We assume that the reader is familiar with the FAM, EAM and GAM architectures. In this section we only provide the necessary information that is needed to understand the evolution of these ART structures, explained in detail in Section 3.

An ART architecture consists of an input layer, where the inputs are applied, a category representation layer, where categories (compressed representations of the input patterns are formed), and an output layer where the correct mapping between the input patterns and their associated labels are established. Training in ART is achieved by presenting a training set to the ART network. Given a set of inputs and associated label pairs, $\mathbf{I}_1, label(\mathbf{I}_1), \mathbf{I}_2, label(\mathbf{I}_2), \ldots, \mathbf{I}_{PT}, label(\mathbf{I}_{PT})$ (called the training set), we want to train ART to map every input pattern of the training set to its corresponding label. To achieve the aforementioned goal we present the training set to the ART architecture repeatedly, as many times as it is necessary for ART to correctly classify these input patterns. The task is considered accomplished (i.e., learning is complete) when the weights in ART do not change during a training set presentation, or after a specific number of list presentations is reached.

The weights in ART correspond to compressed representations of the input patterns presented to the ART network during its training phase. These compressed representations have a geometrical interpretation. In particular, every node (category) in the category representation layer of FAM has weights that completely define the lower and upper endpoints of a hyperbox. At the beginning of training, every category of FAM starts as a trivial hyperbox (equal to a point) and subsequently it expands to incorporate within its boundaries all the input patterns that in the training phase choose this hyperbox as their representative hyperbox, and are encoded by it (see Fig. 1, where the category expansion of FAM is shown for an example dataset). The size of hyperbox is measured as the sum of the lengths of its sides.

Also, every node (category) in the category representation layer of EAM has template weights that completely define an ellipsoid, through its center, direction of major axis, length of the major axis, and ratio of lengths of minor axes to major axis in the ellipsoid. At the beginning of training, every EAM category starts as a trivial ellipsoid (equal to a point) and subsequently it expands to incorporate within its boundaries all the input patterns that in the training phase chose this ellipsoid as their representative ellipsoid, and are encoded by it (see Fig. 2, where the category expansion of EAM is shown for an example dataset). The size of the ellipsoid is measured as the length of the major axis.

Finally, every node (category) in the category representation layer of GAM has template weights that define the mean vector, the standard deviation vector of a multi-dimensional Gaussian distribution, and the number of points that are associated with this Gaussian distribution. At the beginning of training, every category of GAM starts as a collection of Gaussian distributions in every dimension, with mean equal to the input pattern that was first encoded by this category, and a small standard deviation vector (part a of Fig. 3); as training progresses in GAM this GAM category is modified to incorporate the information of the additional input patterns that are encoded by it (see part b of Fig. 3 for an illustration of how the GAM category is modified for an example dataset). At any point in time the mean vector of this

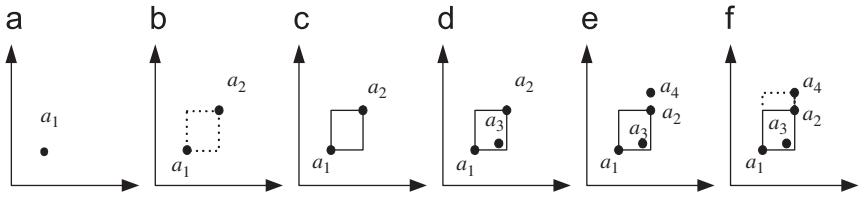


Fig. 1. FAM learning (2-D example). (a) A category with 0 size; (b) introducing a new pattern $\mathbf{I}_2$, represented by $\mathbf{a}_2$; (c) the category expands to include $\mathbf{a}_2$; (d) since new pattern $\mathbf{I}_3$, represented by $\mathbf{a}_3$ is inside the category, it does not change its size; (e) new pattern $\mathbf{I}_4$, represented by $\mathbf{a}_4$ is presented; (f) since $\mathbf{a}_4$ is outside the category, the category is expanded to include $\mathbf{a}_4$, within its boundaries.

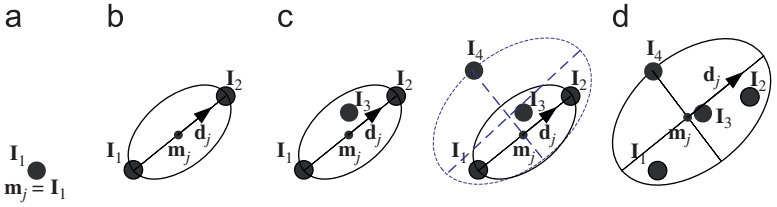


Fig. 2. EAM learning (2-D example). (a) A category with 0 size; (b) introducing a new pattern $\mathbf{I}_2$; the category expands to include $\mathbf{I}_2$; (c) introducing a new pattern $\mathbf{I}_3$; since the category includes $\mathbf{I}_3$, it does not change its size; (d) pattern $\mathbf{I}_4$ is presented; since this pattern is outside the category, the category is expanded to include $\mathbf{I}_4$ within its boundaries.

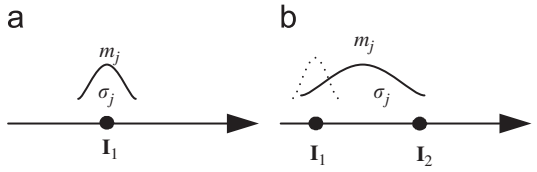


Fig. 3. GAM learning (1-D example). (a) A category with 0 size; (b) introducing a new pattern $\mathbf{I}_2$; the category characteristics (mean, standard deviation, of the Gaussian curve, as well as number of points encoded by the Gaussian curve) change to include the new knowledge that the new input pattern brings.

Gaussian distribution, corresponding to a category, is equal to the mean vector of all the input patterns encoded by the category, and the variance vector of the Gaussian distribution is equal to the variance vector corresponding to the input patterns that were encoded by the category, while the number of the points associated with this Gaussian distribution are the number of points that chose this category as their representative category.

It is also worth mentioning that the categories in FAM, EAM and GAM are allowed to expand up to a point allowed by a threshold, controlled by a network parameter denoted as the baseline vigilance parameter ($\bar{\rho}_a$). This parameter assumes values in the interval $[0,1]$. Small values of this parameter allow the creation of large categories, while large values of this parameter allow the creation of small categories. In the one extreme when $\bar{\rho}_a$ is equal to 0, an FAM or EAM category, equal to the whole input space, could be created, while at the other extreme when $\bar{\rho}_a$ is equal to 1 only point categories are formed. In GAM, small values of this parameter allow more and more patterns to be encoded by a GAM category, while large values of this parameter allow only a few patterns to be encoded by a GAM category. It turns out that this parameter has a significant effect on the number and type of categories formed, and consequently it affects the performance of these networks.

The performance of ART networks is measured in terms of the number of categories created in its training phase (small number of categories is good), and how well it generalizes on unseen data (high generalization accuracy is good). The performance of an ART architecture depends on the choice of the vigilance parameter. It has been a known fact that performance in ART is also affected by the order according to which training data are presented to an ART architecture.

3. The evolution of ARTMAP neural networks

AG-FAM, AG-EAM, and AG-GAM are evolved FAM, EAM, GAM networks, respectively, that are produced by applying, repeatedly, genetic operators on an initial population of trained FAM, EAM, or GAM networks. AG-FAM, AG-EAM, AG-GAM use binary tournament selection, as well as genetic operators, including crossover and mutation. In addition, AG-FAM, AG-EAM and AG-GAM use a special operator, *Prune*; this special operator is needed so that categories could be destroyed in the ART architectures, thus leading us, through evolution, to smaller ART structures.

It is assumed that for the classification problems under consideration we have a training set, a validation set, and a test set. The training set is used for the training of AG-FAM, AG-EAM, and AG-GAM architectures under consideration. The validation set is used to optimize the produced AG-FAM, AG-EAM, or AG-GAM network in ways that will become apparent in the following text. Finally, the test set is used to assess the performance of the optimized AG-FAM, AG-EAM, or AG-GAM network created.

To better understand how ART (FAM or EAM or GAM) is genetically designed, we resort to a step-by-step description of this design. The genetic design of ART can be articulated through a sequence of basic steps, defined succinctly below.

The pseudo-code in Algorithm 1 shows the basic steps of AG-FAM, AG-EAM and AG-GAM:

Algorithm 1. Pseudo-code of AG-ART algorithm.

```
P(0) ← Generate-Initial-Population()
For(t ← 1 To Gen_max)
—  Evaluation()
—  If(stoppingcriteriamet) then exitfor
—  P'(t) ← Selection(P(t))
—  P(t) ← Reproduction(P'(t))
End For
Return (Best Network in P(t))
```

3.1. Generate initial population

The algorithm starts by training Pop_{size} ARTMAP networks (FAM, EAM or GAM), each one of them trained with a different value of the baseline vigilance parameter $\bar{\rho}_a$, and order of training pattern presentation. In particular, we first define $\bar{\rho}_a^{inc} = (\bar{\rho}_a^{max} - \bar{\rho}_a^{min})/(Pop_{size} - 1)$, and then the baseline vigilance parameter of every network is determined by the equation $\bar{\rho}_a^{min} + i\bar{\rho}_a^{inc}$, where $i \in \{1, 2, \ldots, Pop_{size} - 1\}$. In our implementation we fixed $Pop_{size} = 20$. The choice parameter in an FAM network was chosen to be equal to 0.1. The choice parameter in an EAM network was chosen to be equal to 0.01. The ratio of the length of the minor axes to major axes in EAM was chosen equal to 1. The initial value of the standard deviation γ in a GAM network is chosen to be equal to 0.6. In our experiments with AG-FAM and AG-EAM we chose $\bar{\rho}_a^{min} = 0.1$ and $\bar{\rho}_a^{max} = 0.95$, while in our experiments with AG-GAM we chose $\bar{\rho}_a^{min} = 0.1$ and $\bar{\rho}_a^{max} = 0.75$.

We assume that the reader is familiar with how training of FAM, EAM and GAM networks is accomplished, and thus the details here are omitted. Once the Pop_{size} networks are trained, they need to be converted to chromosomes so that they can be manipulated by the genetic operators. AG-FAM, AG-EAM and AG-GAM use a real number representation to encode the networks. Each chromosome consists of two levels, level 1 containing all the categories of the network and level 2 containing the template parameters needed to represent every category in level 1, as well as the label of every category in level 1. The chromosome encoding is explained in more detail in Fig. 4 for AG-FAM, in Fig. 5 for AG-EAM and in Fig. 6 for AG-GAM.

3.2. Evaluation

A weighted sum approach is used to define a fitness function that combines the two objectives of the optimization problem; the error rate, $p_{err}(p)$, and size of the network, $size(p)$. The error rate, $p_{err}(p)$, corresponds to the error rate, exhibited by the p-th network, on the validation set, while $size(p)$ is the number of categories of the p-th network. The use of weighted sum approach is one of the simplest ways of defining a fitness function that

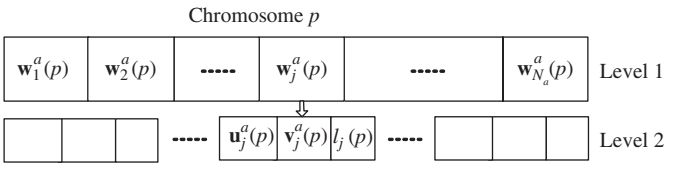


Fig. 4. AG-FAM chromosome structure. At level 2, the category's weight $\mathbf{w}_j^a$ contains the information about the lower end-point, $\mathbf{u}_j^a$, and the upper end-point, $\mathbf{v}_j^a$, of the hyperbox corresponding to the category, as well as the label l_j of the category.

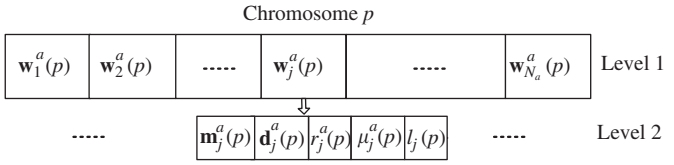


Fig. 5. AG-EAM chromosome structure. At level 2, the category's weight $\mathbf{w}_j^a$ contains the information of the center, $\mathbf{m}_j^a$, the direction vector of the major axis, $\mathbf{d}_j^a$, the radius (half length) of the major axis, r_j, and the ratio of the lengths of the minor axes over the length of the major axis, μ_j, of the ellipsoid corresponding to this category, as well as the label l_j of the category.

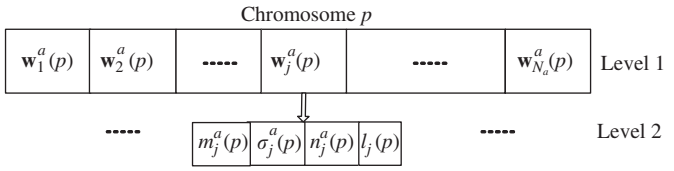


Fig. 6. AG-GAM chromosome structure. At level 2, the category's weight $\mathbf{w}_j^a$ contains the information of the center of the Gaussian curve, $\mathbf{m}_j^a$, the standard deviation vector of the Gaussian curve, σ_j^a, and the number of points represented by the Gaussian curve, n_j, as well as the label l_j of the category.

depends on two measures (generalization of the network and size of the network) and has been extensively adopted in the classification literature (e.g., see [6]).

The use of weighted sum to combine two objectives requires the proper scaling of the objective values so that one objective cannot dominate the other. Since $p_{err}(p)$ is appropriately scaled between 0 and 1.0, it remains to scale $size(p)$. As it is not possible to determine the range of $size(p)$ for every classification problem, we estimate the range from the initial population. We define the size scale factor, $size_{max}$, as the size of the network that minimizes $p_{err}(p)$ in the initial population. Since we expect the GA to improve the error rate and size in subsequent generations, this choice is considered appropriate. The fitness function $fit(p)$ of the p-th network is defined as follows:

$$fit(p) = p_{err}(p) + \alpha size(p) - Cat_{min} size_{max} \quad (1)$$

Obviously, this fitness function decreases as $p_{err}(p)$ decreases or as $size(p)$ decreases (hence the objective is to minimize the above defined fitness function). The value of Cat_{min} is chosen to be equal to the number of classes of the classification problem at hand. It is evident from the fitness equation that if the GA drops the size from $size_{max}$ to Cat_{min}, the fitness drops by approximately α. Therefore, the maximum allowed sacrifice in classification accuracy is equal to α.

3.3. Selection

Initialize a temporary generation P', where the parent chromosomes used to create the next generation are selected. The parents are chosen using a deterministic binary tournament selection, as follows: randomly select two groups of two chromosomes each from the current generation, and use as a parent, from each group, the chromosome with the best fitness value in the group.

The algorithm implements elitism as follows: it finds the best NC_{best} chromosomes (i.e., the chromosomes having the NC_{best} highest fitness values) from the current generation and copies them to the next generation without change. In our implementation we choose a value of $NC_{best} = 3$.

3.4. Reproduction—adaptation

Once the selection step determines the parents, reproduction operators are used to create individuals for the next generation. As expected, the reproduction operators are problem specific. In this section the reproduction operators used in evolving ARTMAP networks are described.

The two well-known operators for reproduction in GAs are crossover and mutation. In this work, in addition to crossover, two mutation-based operators are proposed. The first is referred to as the *Mutation operator*, and it performs Gaussian mutations on chromosomes. The second operator, referred to as the *Prune operator*, prunes a network by deleting a number of categories from that network. The mutation operator applies Gaussian perturbations of the weights at level 2 of the chromosome string (see Figs. 4–6). On the other hand, the Prune operator applies structural mutation at level 1 of the chromosome string.

To avoid the need for finding proper values for the mutation and pruning probabilities, or setting default values that might result in suboptimal operation, an adaptation mechanism was employed to automatically adjust, based on performance, the invocation of reproduction operators. This performance-based adaptation is implemented at the gene (category) level. More specifically, adaptive, performance based, parameters are computed for each component in the individual. The performance feedback relies on a metric defined for each category, referred to as the *confidence factor*, *CF*.

The confidence factor is a metric that measures the performance at the category level. Since our objective is to find a network with good generalization and small size, the performance of a category is defined in terms of accuracy and frequency of selection of the category. We favor accurate and frequently selected (therefore larger) categories. The assumption here is that if categories are frequently selected, the network size would likely to be smaller. The confidence criteria is based on similar confidence criteria designed by other researchers in the field (see [9,33]), whose objective was to prune under-performing ART categories. In [9] the authors proposed the use of a confidence factor that is computed by utilizing a validation set of labeled patterns. The confidence factor defined in [9] consists of two components: the probability of selection and the classification accuracy. The confidence factor is defined, for every category j of the p-th ART network, that is mapped to label k, as follows:

$$CF_j^k(p) = 0.5A_j^k(p) + 0.5S_j^k(p) \quad (2)$$

where $A_j^k(p)$ is a measure of accuracy of classification achieved by category j, in the p-th network, that is mapped to label k. Furthermore, $S_j^k(p)$ is a measure of probability of selection of category j, in the p-th network, that is mapped to label k.

The accuracy measure, $A_j^k(p)$, is defined as follows: the probability of correct classification for category j divided by the maximum probability of correct classification for any category in the same network (p-th network) that predicts the same class label, k. This measure assumes higher values for categories that are performing relatively well. In particular, if the number of validation samples that selected this category, and were correctly classified by it, is denoted by $P_j^k(p)$, and the number of validation samples that selected this category is denoted by $C_j^k(p)$, then

$$A_j^k(p) = \frac{P_j^k(p)/C_j^k(p)}{\max_j(P_j^k(p)/C_j^k(p))} \quad (3)$$

We also define $S_j^k(p)$ as the probability of selection by the validation patterns of a category j, of the p-th network, that is mapped to label k. The probability of selection of category j, of the p-th network, that is mapped to label k, is the number of

validation patterns that selected this category, $C_j^k(p)$, divided by the maximum number of patterns $C_{j_{max}}^k(p)$ that selected any category j that predicts the same classification label, k, for the p-th network:

$$S_j^k(p) = C_j^k(p)/C_{j_{max}}^k(p) \quad (4)$$

This measure achieves higher values for categories that were selected more often using the validation patterns. The scaling ensures that $A_j^k(p) \in [0,1]$, $S_j^k(p) \in [0,1]$ and therefore $CF_j(p) \in [0,1]$. In addition, in every network, at least one category has $A_j(p) = 1$, and at least one category (but not necessarily the same) has $S_j(p) = 1$. Therefore, in every generation the confidence factor is calculated for every category based on the performance on the validation set.

3.4.1. Prune

To be able to create smaller networks using the evolutionary search, we introduced a genetic operator, *Prune*, that deletes categories from a network using some appropriate selection criteria. Pruning is prohibited if it violates the *class inclusion criterion*. The class inclusion criterion dictates that in every network there is at least one category for each class label present in the data. It is obvious that the criterion used for selection of categories to be pruned affects the efficiency of the genetic search. One selection criterion is to randomly prune a category using a user-specified probability. However, this criterion does not exploit the knowledge we have about the performance of a category on the validation set after every generation. It might be beneficial to take this information into consideration when deciding on which categories to be deleted. However, complete reliance on this knowledge would result in a hill-climbing search that would probably end up at local optima. To avoid this situation we resort to a probabilistic approach that increases the chance of deletion of under-performing categories. This way, the search is directed towards better solutions, but not limited from exploring other regions of the solution space. Consequently, in AG-ART, with probability of $1 - CF_j^k(p)$, we delete categories from every chromosome in the temporary generation.

3.4.2. Mutation

Every chromosome gets mutated as follows:

In *AG-FAM*, for each category, either its u or v endpoint is selected randomly (with 50% probability) and then every component of this selected vector gets mutated by adding to it a small number. This number is drawn from a Gaussian distribution. If the component of the chosen vector becomes smaller than 0 or greater than 1 (after mutation), it is set back to 0 or 1, respectively.

In *AG-EAM*, for each category, every component of the ellipsoidal center m gets mutated by adding to it a small number. This number is drawn from a Gaussian distribution. If the component of the chosen vector becomes smaller than 0 or greater than 1 (after mutation), it is set back to 0 or 1, respectively. Furthermore, the mutated category's axis ratio μ or radius r is selected with 50% probability. We add a small number to the axis ratio or the radius. The small number is drawn from a Gaussian distribution. However, if μ or r, due to mutation, becomes larger than 1, they are set back to the value of 1, while if they become smaller than zero we set their value to 0.0001.

In *AG-GAM*, for each category, either its mean vector m or standard deviation vector σ is selected randomly (50% probability). Then every component of this selected vector is mutated by adding to it a small number. This number is drawn from a Gaussian distribution. If the component of the chosen vector becomes smaller than 0 or greater than 1 (after mutation), it is set back to 0 or 1, respectively.

Notice that mutation is applied at level 2 of the chromosome structure. The label of the chromosome is not mutated because our initial GA population consists of trained networks, and consequently we have a lot of confidence in the labels of the categories that these trained networks have discovered through the training process.

The use of Gaussian perturbation for mutation of weights of evolved neural networks has been adopted by several researchers before (e.g., see [4]). We use a Gaussian distribution that has a mean of 0 and a standard deviation that is equal to a severity factor that is calculated for each category based on its performance. A performance-based severity of mutation should be used to impose higher probability of mutation to those categories that do not perform well. We use the following expression to control the severity of mutation:

$$SF_j^k(p) = 0.05(1 - CF_j^k(p)) \quad (5)$$

3.4.3. Crossover operation

The remaining $Pop_{size} - NC_{best}$ chromosomes are created by crossing over pairs of parents. For each parent, p, p', a random crossover point is chosen, designated as n, n', respectively. Then, all the categories with index greater than n' in the chromosome p' and all the categories with index less than or equal to index n in the chromosome with index p are moved into an empty chromosome within the new generation. Notice that crossover is done at level 1 of the chromosome. This operation is pictorially illustrated in Fig. 7.

3.5. Stopping criteria

If a stopping criterion is not met, we go to the next iteration of the genetic evolution. Otherwise, we terminate and we return the best network. There is a need for an automated stopping criterion so that the evolution does not proceed for unnecessarily many generations. Ideally, the evolution should be allowed to proceed for as long as it is necessary, and it should terminate when network performance improvements are not attainable any more. In practice though, there is a trade-off between network performance improvements and computational effort expended to achieve these improvements. It might be beneficial to use multiple stopping criteria to terminate the evolution of ART

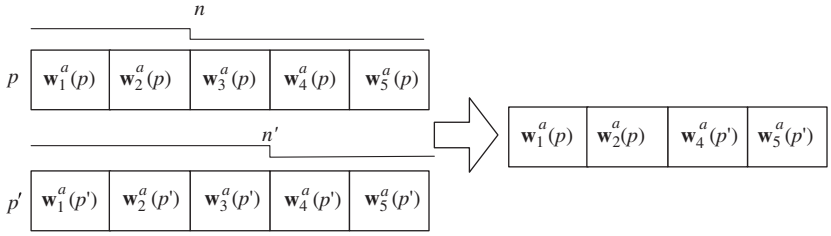


Fig. 7. AG-FAM, AG-EAM, AG-GAM crossover implementation. In crossover the weight vectors of chromosome p, with index smaller than or equal to index n, and the weight vectors of chromosome p' with index larger than n', are combined (concatenated) to produce a new chromosome.

networks. One obvious stopping criterion is to set a threshold for the maximum number of generations, Gen_{max}, that the evolution is allowed to continue. The advantage of having this stopping criterion is that it ensures that the algorithms will always terminate and would not get trapped in an infinite loop if the other stopping criteria are never triggered. The user can always set the maximum number of generations to a large number to allow the algorithm to terminate using other, more appropriate, stopping criteria. Another popular stopping criterion is to stop when no more improvement in fitness is observed. To ensure the lack of improvement is not due to the stochasticity of the search, the evolution is terminated only when no significant network performance improvements are observed for a number of consecutive evolutions. This number of consecutive evolutions can be chosen to be a percentage of the maximum number of generations Gen_{max}. In our experiments we chose $Gen_{max} = 500$, and furthermore we stopped the evolution if 50 generations (10% of Gen_{max}) elapsed without an appreciable network fitness improvement. Appreciable network fitness improvement is an improvement larger than 0.01.

4. Experimental results

In this section we perform experiments to assess the performance of the AG-ART architectures and to compare their performance with other ART and non-ART-based classifiers. First we describe the datasets used in the experimentation. Then, we compare the AG-ART architectures with the GFAM architecture, introduced by Daraiseh et al. [1]. Furthermore, we also compare the AG-ART collection to other ART-based classifiers that addressed the same ART category proliferation problem; this comparison is thorough because we have coded and experimented with these other ART-based classifiers on the same datasets used to assess AG-ARTs performance. Finally, we compare the AG-ART performance to the performance attained by other non-ART-based classifiers.

4.1. Datasets

In this work, experiments were conducted on 11 datasets, of which four are simulated datasets and seven are real datasets. These datasets were chosen to form a diverse group in terms of number of classes, number of features, and also availability of published results. Each dataset was randomly divided into three subsets: training, validation and testing. The summarized specifics of each one of the datasets are depicted in Table 1.

- *Gaussian datasets*: G4c-25, G6c-15: These are artificially generated, 2-D, and Gaussianly distributed datasets, belonging to 4-class and 6-class problems, with 25% and 15% overlap between the classes, respectively. Note that 15% overlap means that the optimal Bayesian classifier would attain a 15% misclassification rate on these data.
- 1*Ci*/*Sq*: This is the circle in a square benchmark problem, a 2-class classification problem. The probability of finding a data-point within a circle or inside the square and outside the circle is equal to $\frac{1}{2}$. The sizes of the areas in the circle, and outside the circle but inside the square are the same. This problem has been extensively used in the pattern classification literature.
- *Modified iris* (*MOD-IRIS*) *dataset*: This is a modified version of the IRIS dataset from the UCI repository (see [29]). The original IRIS dataset (another benchmark classification problem) has 150 data-points and three classes. The data corresponding to the class that is linearly separable was eliminated. This left 100 data-points and two classes. From the four input attributes of the original IRIS dataset only two attributes (attribute 3 and 4) were used because they seem to have enough discriminatory power to separate the 2-class data. Finally, in order to create a reasonable size dataset from these 100 points, data were generated by adding noise around each one of these 100 data-points (the noise was Gaussian of zero mean and small variance) to end up with approximately 10,000 points.
- *Page blocks* (*PAGE*) *dataset*: This database represents the problem of classifying the blocks of the page layout in a document (see [29]). It contains 5473 examples coming from 54 distinct documents. Each example has 10 numerical attributes (e.g., height of the block, length of the block, eccentricity of the block, etc.) and one target (output) attribute, representing the type of the block (text, horizontal line, graphic, vertical line, and picture). One of the noteworthy points about this database is that its major class (text) has a high probability of occurring (above 80%).
- *Pendigits* (*PEN*) *dataset*: This dataset has records representing handwritten digits (see [29]). The dataset was created by collecting 250 samples from 44 writers. The samples written by 30 writers are used for training and crossvalidation and writer-dependent testing, and the digits written by the remaining 14 writers are used for writer-independent testing. This dataset has 16 attributes and 10 classes. The training set has 7494 records and the test set has 3498 records. In this work, the original training set was divided into a training set of 4494 records and a validation set of 3000 records.
- *Satellite image* (*SAT*) *dataset*: This dataset gives the multi-spectral values of pixels within 3×3 neighborhoods in a satellite image, and the classification associated with the central pixel in each neighborhood (see [29]). The aim is to predict the classification given the multi-spectral values. There are six classes and 36 numerical attributes. The training set consists of 4435 records while the test set consists of 2000

Table 1
Datasets used for experimentation, and their characteristics.

Database name	Training instances	Validation instances	Test instances	Attributes	Classes	Major class (%)
G4c-25	500	5000	5000	2	4	25.00
G6c-15	504	5004	5004	2	6	16.67
1Ci/Sq	2000	5000	3000	2	2	50.00
MOD-IRIS	500	4800	4800	2	2	50.00
PAGE	500	2486	2487	10	5	89.80
Pendigits	4494	3000	3498	16	10	10.00
Sat	2000	2436	2000	36	6	24.19
Seg	800	810	700	19	7	14.29
Wav	1000	2000	2000	21	3	33.33
Glass	75	75	64	9	6	35.51
Pima	150	150	232	7	2	66.70

records. The original training set was divided into a training set of 2000 records and a validation set of 2435 records.

- *Image segmentation* (*SEG*) *dataset*: This dataset was used in the StatLog Project (see [29]). The samples are from a dataset of seven outdoor images. The images are hand-segmented to create a classification for every pixel as one of brickface, sky, foliage, cement, window, path, or grass. There are seven classes, 19 numerical attributes and 2310 records in the dataset.
- *Waveform* (*WAV*) *dataset*: This is an artificial 3-class problem based on three wave-forms (see [29]). Each class consists of a random convex combination of two waveforms sampled at designated integer values, with noise added. There are 21 numerical attributes, and 3000 records in the training set. Error rates are estimated from an independent test set of 2000 records. The original training set was divided into a training set of 1000 records and a validation set of 2000 records.
- *Glass* (*GLS*) *dataset*: This dataset is used to classify types of glass (see [29]). It was motivated by a criminological investigation. At the scene of the crime, the glass left can be used as evidence, if it is correctly identified. This dataset has 214 instances, 10 numerical attributes and six classes.
- *Pima-Indian diabetes* (*PIMA*) *dataset*: This dataset classifies the patients that are females, at least 21 years old, of Pima-Indian heritage and living near Phoenix, Arizona, USA (see [29]). The problem is to predict whether a patient would test positive for diabetes, given a number of physiological measurements and medical test results. There are two classes, eight numerical attributes and 768 records. However, many of the attributes, such as serum insulin, contain zero values which are physically impossible. By removing the serum insulin and records that have impossible values in the other attributes, a dataset of seven attributes and 532 records resulted (this approach was followed by other researchers).

4.2. Comparison with GFAM

In this section we compare the performance of the AG-ART collection to that of GFAM introduced in [1]. The purpose of this comparison is to assess the value of the adaptive approach to choose the GA parameters, proposed in this paper, to the static approach to choose these parameters, utilized in GFAM.

In this work we introduced an adaptively defined confidence factor, according to which ART categories are pruned, and an adaptively defined severity factor, according to which ART categories are mutated. In GFAM the probability of deleting an ART category was chosen after expensive experimentation and evaluation of the appropriateness of candidate probability values on a limited collection of classification problems; then these probability values were used for all other classification problems. Hence, the GFAM approach was not only computationally costly, but it was also not dataset-dependent, and it did not change throughout the evolutionary process. All these issues have now been addressed by the AG-ART approach, using the adaptively defined confidence factor, that is dataset-dependent and performance-based varying (relying on the performance of a category at each generation). Furthermore, in GFAM the severity of mutation was fixed, and the only user defined parameter was the probability of mutation. This parameter was chosen in GFAM after expensive experimentation and evaluation of candidate probability values on a limited collection of classification problems; then these probabilities were used for all other classification problems. Hence, the GFAM approach for choosing the mutation probability was also computationally costly, and not dataset-dependent, and did not change throughout the evolutionary process. All these issues have now been addressed by the AG-ART approach, by using the adaptively defined severity of mutation factor, that is dataset-dependent and performance-based varying (relying on the performance of a category at each generation).

Table 2
Total run time for AG-FAM vs. GFAM, in seconds.

Dataset	10 runs GFAM	10 runs AG-FAM	Reduction (%)
G4c-25	131.08	100.69	23.18
G6c-15	179.00	171.14	4.39
1Ci/Sq	387.58	150.78	61.10
Glass	4.44	2.81	36.63
Iris	32.16	26.61	17.25
Page	134.17	92.23	31.26
Pendigits	2684.20	2039.75	24.01
Pima	3.81	1.58	58.59
Sat	1033.17	679.17	34.26
Seg	151.23	79.86	47.19
Wav	288.52	238.77	17.24

In summary, AG-ARTs approach of choosing the category prune probabilities and the severity of mutation is much more elegant, sensible and cost-effective, compared to the approach used in GFAM. The final solutions that GFAM and AG-FAM discover are very similar. The important difference though is that AG-FAM, due to the adaptively chosen GA parameters, converges to this solution faster. Hence, the AG-ART approach is not only more elegant and more cost-effective in defining good values for the GA parameters, but even after the evolution starts AG-ART is more efficient than GFAM (converges to the final solution faster).

To demonstrate our point (about the efficiency of the AG-ART approach compared to the GFAM approach) we compare, in Table 2, the average run time of GFAM, introduced in [1], and AG-FAM discussed in this paper. The total run time of AG-FAM and GFAM are defined as the total times, needed over a number of runs (different initial seeds), by the evolutionary process in AG-FAM and GFAM to converge to a solution, respectively; in our case (see Table 2) AG-FAM and GFAM were run for 10 different initial seeds. Both approaches were able to find solutions of similar quality. However, the AG-FAM approach was able to reduce the evolutionary computation time up to 60% in some cases. Fig. 8 illustrates how parameter adaptation allows the GA to find better solutions more quickly. Therefore, with adaptation, the GA is able to find same quality solutions using a smaller number of generations. For example, the run time in AG-FAM is reduced due to the use of the Prune operator, introduced in this paper. The Prune operator is more efficient than the fixed probability of pruning used in the GFAM paper [1]. The Prune operator in AG-FAM contributes in finding smaller FAM networks faster in its evolutionary process than the corresponding operator in GFAM; since these smaller FAM networks are validated faster we end up with a reduced run time with AG-FAM, compared to GFAM.

4.3. Comparison with other ART architectures

In this section we compare AG-ARTs performance to that of other popular ART architectures, which have been proposed in the literature with the intent of addressing the category proliferation problem, such as ssFAM, ssEAM, and ssGAM. These approaches are based on the principle of semi-supervision, introduced by Anagnostopoulos et al. [3] and Verzi et al. [34]. Semi-supervision is a term attributed to learning in an ART architecture (FAM, EAM or GAM), where categories in ART are allowed to encode patterns of different labels provided that the percentage of patterns that belong to the plurality label exceed a certain threshold.

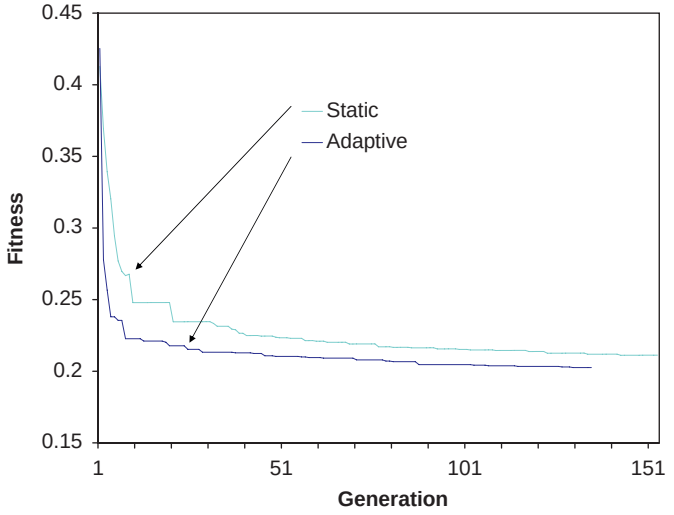


Fig. 8. Fitness as a function of generation for the satellite dataset. The upper curve was produced using a non-adaptive approach. The lower curve was produced using adaptation. It is clear that adaptation allows faster progress of fitness towards optimum.

Our comparison is based on three measures of performance: generalization, size, and computational cost. The results obtained from ssFAM, ssEAM, and ssGAM depend on the setting of the parameters of these networks. The choice of good settings for these parameters depends on the dataset at hand. Therefore to obtain good results from these networks one should experiment with a range of settings for the network parameters. Since the results obtained from AG-FAM, AG-EAM, and AG-GAM is a result of evolving (optimizing) a population of ART networks, it is appropriate to compare their performance to that obtained from the ssFAM, ssEAM, and ssGAM experimentation performed to find their best parameter setting for any given database.

Since in this work we are not only focusing on generalization performance, but also on the size of the network produced, it becomes more complicated to compare and rank networks. To provide a fair comparison, we resort to a comparison approach that considers the two objectives simultaneously. Since the existence of the two, sometimes competing, objectives result in multiple good solutions rather than one "best" solution, in our comparison, we assess multiple solutions (sets of solutions) produced by the different classifiers, under consideration. In other words, for each classification algorithm, we produce a number of solutions that have attained the two objectives (good generalization and small size) at different levels of success. Then we choose the *non-dominated solutions*. A non-dominated solution is defined to be a network, where no other network from the list of found solutions dominates its performance, that is, achieves better generalization utilizing equal or smaller number of categories.

For each of the ssFAM, ssEAM, and ssGAM, and for each of the 11 databases, we performed a number of experiments with different settings of their network parameter values. In particular, the parameter settings that we experimented with ssFAM were: baseline vigilance values ranging from 0.1 to 0.9 with step size of 0.1, choice parameter values of 0.01 and 0.1, maximum allowable mixture threshold values ranging from 0 to 0.9 with step size of 0.1, and 10 different orders of pattern presentations of the training data (resulting in 1800 different parameter settings). Furthermore, the settings for ssEAM were: baseline vigilance values ranging from 0.1 to 0.9 with step size of 0.1, choice parameter values of 0.001 and 0.01, maximum allowable mixture threshold values ranging from 0 to 0.5 with step size of 0.1, minimum axes to maximum axis ratio values ranging from 0.5 to 1 with step size of 0.1, and 10 different orders of pattern presentations of the training data (resulting in 6480 different parameter settings). Also, the settings for ssGAM were: baseline vigilance values ranging from 0 to 0.9 with step size of 0.1, initial standard deviation parameter ranging from 0.5 to 1 with step size of 0.1, maximum allowable mixture threshold values ranging from 0 to 0.9 with step size of 0.1, and 10 different orders of pattern presentations of the training data (resulting in 6000 different parameter settings). It should be emphasized that these parameter ranges were determined by the authors of this paper and they reflect their experience of what are good parameter settings for these ART networks.

For the training of ssFAM, ssEAM, and ssGAM we used the same training set, and validation set as the one used for the AG-FAM, AG-EAM, and AG-GAM networks. We choose the solution networks proposed by each method based on the network size and performance of the network on the validation set. Different network solutions for the ssFAM, ssEAM, and ssGAM network were produced by changing the parameter settings for these networks, as delineated in the previous paragraph. The total computation time required to obtain these network solutions for each database and each method, which is the sum of training and validation CPU times (in seconds) for all the tried settings, is reported in Table 3, and referred to as the *total run time*.

Experiments were also conducted for AG-FAM, AG-EAM, and AG-GAM for each of the 11 databases. To obtain different solution networks, we varied the fitness parameter, α (see Eq. (1)). In particular, the different solutions for AG-FAM, AG-EAM, and AG-GAM were obtained by considering the following α values: 0.01, 0.05, 0.1, 0.2, and 0.5. The total computation time needed to produce these solutions for AG-FAM, AG-EAM, and AG-GA is also referred to as the *total run time*, and reported in Table 3, as well.

A one-to-one comparison of the results reported in Table 3 reveals that the *total run time* of the AG-FAM, AG-EAM, and AG-GAM networks is smaller, sometimes an order of magnitude smaller than the *total run time* of their corresponding counterparts, ssFAM, ssEAM, and ssGAM, respectively. The *total run time* results are also shown in a condensed, pictorial, fashion in Figs. 9–11.

To compare the generalization performance of AG-FAM and ssFAM, AG-EAM and ssEAM, and finally AG-GAM and ssGAM we use a metric that compares the network solutions obtained by the ss-network (for all different parameter settings) and the network solutions obtained by the AG-network (for the five different α values). This metric has been used before in similar situations (see [41,11,12]). This metric is defined as follows:

$$C(A,B) = \frac{|b \in B : \exists a \in A, b \prec a|}{|B|} \tag{6}$$

This metric measures the fraction of members in set B that are dominated by at least one member in set A. Therefore, $C(A,B) = 1$ means all members in B are dominated by members in A. In this case the approach that produced set A is a clear winner. It is obvious that we need to consider also $C(B,A)$ in order to properly compare the two sets. Since the calculated values of $C(A,B)$ and $C(B,A)$ are dependent on the seed used to evolve the population of FAMs, EAMs, and GAMs in the AG-ART approach, we produced network solutions for the five different α parameter values, by changing the seed 10 times. Consequently, 10 different values of C(AG-FAM, ssFAM), and C(ssFAM, AG-FAM), were produced. Similarly, 10 different of C(AG-EAM, ssEAM), and C(ssEAM, AG-EAM), as well as of C(AG-GAM, ssGAM), and C(ssGAM, AG-GAM) were produced. In Table 4 we compare the average values (over the 10 replications) of C(AG-FAM, ssFAM) vs. C(ssFAM, AG-FAM), and C(AG-EAM, ssEAM), vs. C(ssEAM, AG-EAM), and C(AG-GAM,

Table 3
Total run time for AG-FAM, AG-EAM, and AG-GAM compared to total run time for ssFAM, ssEAM, ssGAM.

Database name	AG-FAM	ssFAM	Reduction (%)	AG-EAM	ssEAM	Reduction (%)	AG-GAM	ssGAM	Reduction (%)
G4C-25	52.59	130.92	59.83	82.99	916.78	90.95	89.32	314.49	71.60
G6C-15	92.01	145.16	36.61	127.15	508.22	74.98	137.82	266.77	48.34
1Ci/Sq	68.51	216.42	68.34	152.13	1167.67	86.97	142.29	1431.35	90.06
Glass	1.71	2.82	39.37	2.26	5.72	60.45	2.64	3.52	24.99
Iris	13.48	21.30	36.73	21.56	123.56	82.55	22.02	109.25	79.84
Page	52.97	69.52	23.80	60.91	484.15	87.42	75.03	125.68	40.30
Pendigits	1142.43	7864.27	85.47	4304.84	58 865.05	92.69	537.62	20 050.39	97.32
Pima	0.95	3.68	74.21	2.41	75.88	96.82	1.91	32.75	94.15
Sat	508.21	2034.29	75.02	1234.62	17 162.45	92.81	329.07	4302.16	92.35
Seg	41.93	85.55	50.99	88.45	1331.47	93.36	64.13	509.99	87.43
Wav	147.23	763.99	80.73	369.38	8612.65	95.71	83.94	1199.58	93.00

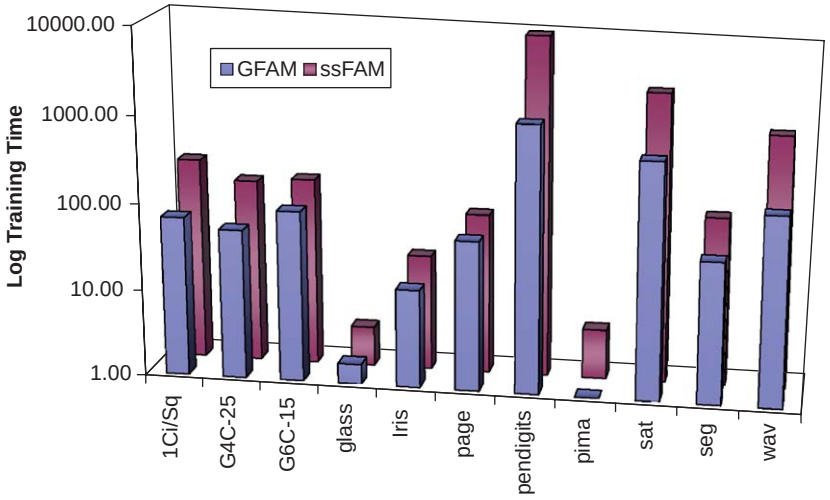


Fig. 9. Total run time of AG-FAM vs. ssFAM.

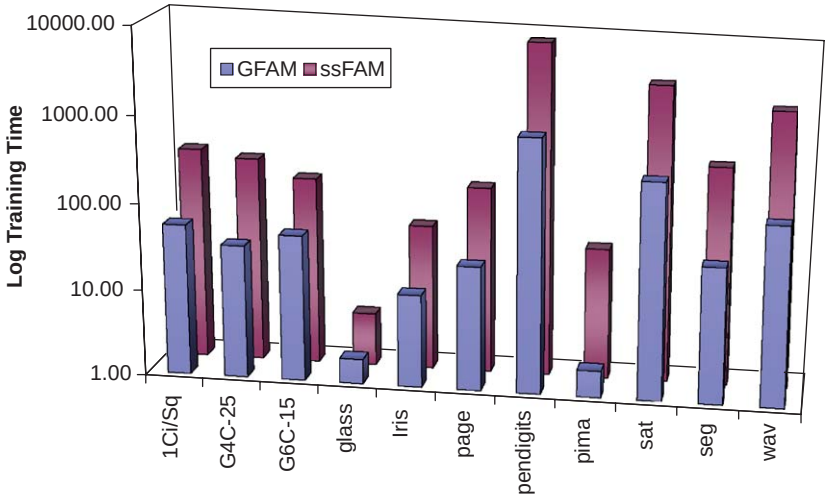


Fig. 10. Total run time of AG-EAM vs. ssEAM.

ssGAM) vs. C(ssGAM, AG-GAM). It is obvious from the table that the average values of C(AG-FAM, ssFAM) are larger than C(ssFAM, AG-FAM) values, which indicates that networks produced by AG-FAM are more likely to dominate networks produced by ssFAM, and therefore, the networks produced by AG-FAM are expected to be of higher quality. The p-value column reported in the table corresponds to the t-test for the two sample means. The p-values in the table indicate that the difference in the means between C(AG-FAM, ssFAM) and C(ssFAM, AG-FAM) is statistically significant. Similar conclusions can be made by comparing the means of C(AG-EAM, ssEAM) and C(ssEAM, AG-EAM), as well as the means of C(AG-GAM, ssGAM) and C(ssGAM, AG-GAM). The box plot shown below the C(AG-FAM, ssFAM) and C(ssFAM, AG-FAM) values of Table 4 compares visually the values of C(AG-FAM, ssFAM) against the C(ssFAM, AG-FAM) values for the segmentation dataset (one of the tested datasets). Furthermore, the box plot shown below the C(AG-EAM, ssEAM) and C(ssEAM, AG-EAM) values of Table 4 compares visually the values

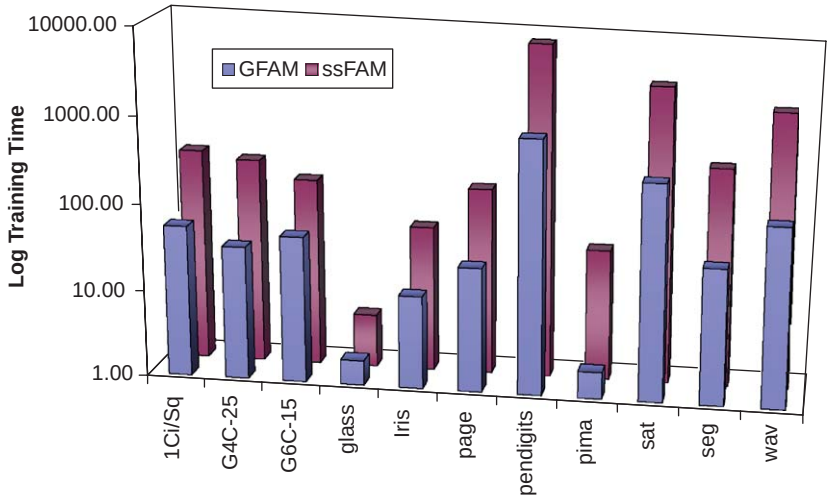
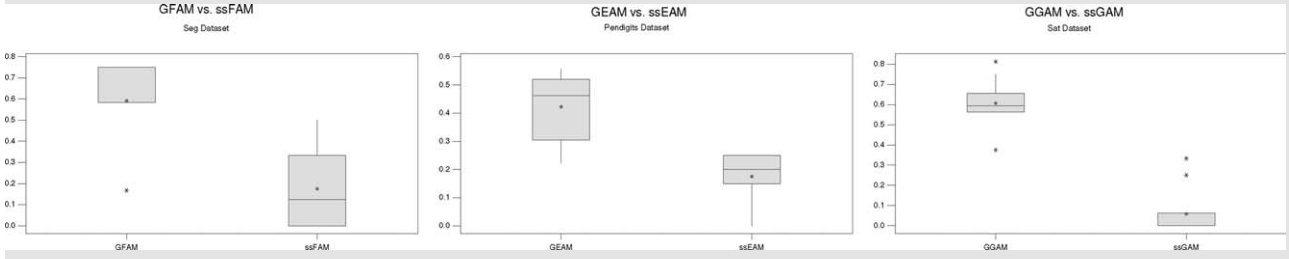


Fig. 11. Total run time of AG-GAM vs. ssGAM.

Table 4
C-metric values. The *p*-value is based on *t*-test for the 10 values of the metric.

	C(AG-FAM, ssFAM)	C(ssFAM, AG-FAM)		C(AG-EAM, ssEAM)	C(ssEAM, AG-EAM)		C(AG-GAM, ssGAM)	C(ssGAM, AG-GAM)	
	Average	Average	*p*-Value	Average	Average	*p*-Value	Average	Average	*p*-Value
G4c-25	0.900	0.050	0.000	1.000	0.000	NA	1.000	0.000	NA
G6c-15	1.000	0.000	NA	1.000	0.000	NA	0.863	0.000	0.000
1Ci/Sq	0.419	0.078	0.000	0.892	0.000	0.000	0.671	0.200	0.000
Glass	0.433	0.350	0.593	0.878	0.075	0.000	0.914	0.000	0.000
Iris	0.533	0.300	0.165	0.800	0.300	0.017	0.857	0.000	NA
Page	1.000	0.000	NA	0.900	0.000	0.000	1.000	0.000	NA
Pendigits	0.458	0.000	0.000	0.422	0.175	0.000	0.468	0.080	0.000
Pima	1.000	0.000	NA	0.860	0.100	0.000	0.983	0.100	0.000
Sat	0.847	0.000	0.000	0.722	0.000	0.000	0.606	0.058	0.000
Seg	0.592	0.175	0.000	0.572	0.243	0.000	0.829	0.025	0.000
Wav	1.000	0.000	NA	1.000	0.000	NA	0.957	0.000	0.000



of C(AG-EAM, ssEAM) against the C(ssEAM, AG-EAM) values for the pendigits dataset (one of the tested datasets). Finally, the box plot shown below the C(AG-GAM, ssGAM) and C(ssGAM, AG-GAM) values of Table 4 compares visually the values of C(AG-GAM, ssGAM) against the values of C(ssGAM, AG-GAM) for the satellite dataset (one of the tested datasets). The box plots convey the same conclusions that the tabular entries in Table 4 convey.

Table 3 shows that the better performance of the AG-FAM, AG-EAM, and AG-GAM networks is attained with reduced computations as compared with the computations needed by the alternate architectures (ssFAM, ssEAM, ssGAM). The computational advantage of genetically engineered ART networks compared to the semi-supervised ART architectures can be explained by the fact that the performance attained by ssFAM, ssEAM, and ssGAM required training these networks for a large number of network parameter settings (at least 1800 experiments) and then choosing the best networks through crossvalidation. In the AG-FAM, AG-EAM, and AG-GAM cases we trained only a small number of these networks ($Pop_{size} = 20$ of them). Furthermore, in AG-FAM, AG-EAM, and AG-GAM cases we evolved the trained networks for at most $Gen_{max} = 500$ generations, each evolution requiring crossvalidating only the $Pop_{size} = 20$ networks. Quite often, the evolutionary process converged after only a few (50) generations, because a satisfactory solution was found.

The accuracy and size advantage of AG-FAM, AG-EAM, AG-GAM compared to ssFAM, ssEAM, and ssGAM can be attributed to the genetic optimization that it employs. This optimization, involving the *Prune* and *Crossover* operators, allows one to construct networks that are not attainable using the original ART training rules. Also, the operation of *Prune* and *Mutation* operators were designed to guide the genetic search to optimal solution faster, resulting in the significant, at times, computational advantages of AG-FAM, AG-EAM, and AG-GAM compared to the semi-supervised ART architectures.

4.4. Comparison with other published results

In this section we compare results obtained using the proposed AG-ART architectures to literature results published by other

Table 5
Performance of AG-FAM, AG-EAM, and AG-GAM for 11 datasets.

Dataset name	AG-FAM		AG-EAM		AG-GAM	
	Accuracy	Categories	Accuracy	Categories	Accuracy	Categories
G4C-25	74.94	4	75.14	4	75.24	4
G6C-15	84.75	6	85.01	6	84.97	6
1Ci/Sq	98.07	31	99.70	2	99.83	2
Glass	76.56	6	75.00	6	73.44	9
Iris	94.96	2	95.04	2	94.75	2
Page	96.59	5	95.09	5	96.34	6
Pendigits	98.20	282	98.31	331	97.83	108
Pima	79.31	2	78.88	3	77.16	2
Sat	88.90	310	87.85	203	88.35	118
Seg	95.86	22	93.71	128	92.71	17
Wav	85.90	4	87.15	4	87.50	3

authors using various classification algorithms. We base the comparison on well-known datasets that many researchers chose to test their algorithm against. Here we focus mainly on the classification accuracy as it is the case with most published work. We test our approach with a fitness function set to maintain high level of accuracy in the classifier. In Table 5 we list the results obtained.

In [22] the authors use the simple Bayes classifier to produce classification accuracy of 85.20% on satellite, 93.12% on segmentation, 78.57% on waveform, 70.11% on glass and 75.9% on pima. In [40], the authors use a decision tree variant to produce classification accuracy of 92% (size: 37) on segmentation, 83% (size: 65) on waveform, 60% (size 14) on glass, 75.2% (size: 3.4) on pima and 95% (size: 37) on pendigits. As it is evident from Table 5 our AG-ART results consistently outperform these results.

In [25], the authors compared the accuracy and size of a 33 classifiers belonging to the tree, statistical and neural types classifiers. Three of the datasets that Lim, Loh and Shih have experimented with are the satellite, the segmentation and the waveform datasets that has been tested in Table 5. The AG-FAM results on the satellite dataset are: 88.9 classification accuracy, needing 310 categories (other AG-FAM solutions include: 83.6% with six categories, 84.5% with 14 categories). The AG-GAM results on the satellite dataset are 88.35 with 118 categories. The accuracy results reported on the satellite dataset by Lim et al. [25] are: minimum classification accuracy of 60% and maximum classification accuracy of 90%. Furthermore the tree type classifiers (22 of them) created a minimum tree size of 8, while the median tree size was 63. Finally, two of the most celebrated decision tree algorithms, such as CART and C4.5 created tree sizes of 63 and 216. The AG-FAM results on the segmentation dataset are: 95.86% classification accuracy, needing 22 categories. The accuracy results reported on the segmentation dataset by Lim et al. [25] are: minimum classification accuracy of 48% and maximum classification accuracy of 98% (achieved by the nearest neighbor classifier, which performs no data compression). Furthermore the tree type classifiers (22 of them) created a minimum tree size of 6, while the median tree size was 39. Finally, two of the most celebrated decision tree algorithms, such as CART and C4.5 created tree sizes of 69 and 42. The AG-FAM results on the waveform dataset are: 85.9% classification accuracy, needing four categories for AG-FAM and 87.5% and three categories for AG-GAM. The accuracy results reported on the waveform dataset by Lim et al. [25] are: minimum classification accuracy of 52% and maximum classification accuracy of 85%. Furthermore the tree type classifiers (22 of them) created a minimum tree size of 3, while the median tree size was 16. Finally, two of the most celebrated decision tree algorithms, such as CART and C4.5 created tree sizes of 14 and 54.

5. Summary and conclusion

In this paper, we have introduced an improved, compared to its predecessor GFAM, genetically engineered FAM neural network referred to as AG-FAM. We also extended the use of this improved GA to two other ART architectures; EAM and GAM. The resulting architectures are referred to as AG-EAM and AG-GAM, or collectively as AG-ART.

Experimental results have shown that AG-FAM is as accurate and creates as small of an architecture as GFAM, introduced in [1], and it does so at reduced computational cost. While in GFAM the probability of deleting an ART category and the probability of mutating a category were chosen after experimentation on a limited collection of classification problems, in AG-FAM an adaptive mechanism was implemented to choose these parameters based on the database at hand and the quality of the solution found. The AG-ART approach is not only more elegant and more cost-effective in defining good values for the GA parameters, but after the evolution starts AG-ART was found to be more efficient than GFAM.

In this paper, we have also presented an extensive comparison between the AG-ART architectures and semi-supervised ART architecture (ss-FAM, ss-EAM, ss-GAM); these semi-supervised ART architectures are architectures that perform very favorably compared to other ART architectures, and quite often compared to other classification approaches. This comparison took into consideration the classification accuracy and size of the classifier at the same time, and it was fair because these semi-supervised ART architectures were coded and tested on the same datasets as the AG-ART architectures were tested. Our experiments reveal clearly that the AG-ART architectures are able to produce "better quality" classifiers than the ssART architectures, at a reduced computational cost. The computational cost reduction was found, in a number of instances, to be significant (more than an order of magnitude). Furthermore, we compared the performance of AG-ART classifiers with the performance of other classifiers (non-ART-based classifiers) that have appeared in the literature; this comparison showed that AG-ART classifiers are very competitive in terms of accuracy and size of a classifier that they produce.

In summary, we introduced a new family of ART-based architectures, called AG-ART, using an elegant evolutionary approach. The introduced architectures are able to produce classifiers of good accuracy and small size, using a reasonable computational budget. They also have the advantage of requiring little user intervention because the algorithm parameters are automatically adapted.

Acknowledgments

This work was supported in part by the NSF Grants: 0341601, 0647018, 0717674, 0717680, 0647120, 0525429, 0203446.

References

[1] A. Al-Daraiseh, A. Kaylani, M. Georgiopoulos, A.S. Wu, M. Mollaghasemi, G.C. Anagnostopoulos, GFAM: evolving fuzzy ARTMAP neural networks, Neural Networks 20 (8) (2007) 873–891.
[2] G. Anagnostopoulos, Novel approaches in adaptive resonance theory for machine learning, Ph.D. Thesis, University of Central Florida, Orlando, May 2001.
[3] G.C. Anagnostopoulos, M. Bharadwaj, M. Georgiopoulos, S.J. Verzi, G.L. Heileman, Exemplar-based pattern recognition via semi-supervised learning, in: Proceedings of the 2003 International Joint Conference on Neural Networks (IJCNN '03), vol. 4, Portland, Oregon, USA, 2003.
[4] P.J. Angeline, G.M. Saunders, J.P. Pollack, An evolutionary algorithm that constructs recurrent neural networks, IEEE Transactions on Neural Networks 5 (1) (1994) 54–65.

[5] T. Back, Self-adaptation in genetic algorithms, in: Proceedings of the First European Conference on Artificial Life, MIT Press, Washington, DC, 1992.
[6] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and Regression Trees, Wadsworth, Belmont, CA, 1984.
[7] G.A. Carpenter, S. Grossberg, N. Markuzon, J.H. Reynolds, D.B. Rosen, Fuzzy ARTMAP: a neural network architecture for incremental supervised learning of analog multidimensional maps, IEEE Transactions on Neural Networks 3 (1992) 698–713.
[8] G.A. Carpenter, B.L. Milenova, B.W. Noeske, Distributed ARTMAP: a neural network for fast distributed supervised learning, Neural Networks 11 (5) (1998) 793–813.
[9] G.A. Carpenter, H.A. Tan, Rule extraction: from neural architecture to symbolic representation, Connection Science 7 (1995) 3–27.
[10] K.P. Ferentinos, Biological engineering applications of feedforward neural networks designed and parameterized by genetic algorithms, Neural Networks 18 (7) (2005) 934–950.
[11] J.E. Fieldsend, R.M. Everson, S. Singh, Using unconstrained elite archives for multiobjective optimization, IEEE Transactions on Evolutionary Computation 7 (3) (2003) 305–323.
[12] J.E. Fieldsend, S. Singh, Pareto evolutionary neural networks, IEEE Transactions on Neural Networks 16 (2) (2005) 338–354.
[13] T.C. Fogarty, Varying the probability of mutation in the genetic algorithm, in: Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989.
[14] D.B. Fogel, Using evolutionary programming to create neural networks that are capable of playing tic-tac-toe, in: Proceedings of the American Power Conference, 1993.
[15] X. Fu, L. Wang, A GA-based RBF classifier with class-dependent features, in: CEC '02: Proceedings of the Evolutionary Computation on 2002. Proceedings of the 2002 Congress, IEEE Computer Society, Washington, DC, USA, 2002.
[16] D. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, MA, 1989.
[17] E. Gomez-Sanchez, Y. Dimitriadis, J. Cano-Izquierdo, J. Lopez-Coronado, MicroARTMAP: use of mutual information for category reduction in fuzzy ARTMAP, IEEE Transactions on Neural Networks 13 (1) (2002) 58–69.
[18] E. Gomez-Sanchez, Y. Dimitriadis, J. Cano-Izquierdo, J. Lopez-Coronado, Safe-μARTMAP: a new solution for reducing category proliferation in fuzzy ARTMAP, in: Proceedings of the 2001 International Joint Conference on Neural Networks (IJCNN '01), vol. 2, Washington, DC, USA, 2001.
[19] S. Grossberg, Adaptive pattern classification and universal recoding, ii: feedback, expectation, olfaction, and illusions, Biological Cybernetics (1976) 187–202.
[20] R. Hinterding, Z. Michalewicz, A.E. Eiben, Adaptation in evolutionary computation: a survey, in: I. Aleksander, E.J. Taylor (Eds.), Proceedings of the IEEE International Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, Amsterdam, Netherlands, 1997.
[21] K.A.D. Jong, An analysis of the behavior of a class of genetic adaptive systems, Ph.D. Thesis, 1975.
[22] R. Kohavi, B. Becker, D. Sommerfield, Improving simple Bayes, in: Proceedings of the European Conference on Machine Learning, 1997.
[23] A. Koufakou, M. Georgiopoulos, G. Anagnostopoulos, T. Kasparis, Cross-validation in fuzzy ARTMAP for large databases, Neural Networks 14 (2001) 1279–1291.
[24] M. Laumanns, L. Thiele, K. Deb, E. Zitzler, Combining convergence and diversity in evolutionary multiobjective optimization, Evolutionary Computation 10 (3) (2002) 263–282.
[25] T.-S. Lim, W.-Y. Loh, Y.-S. Shih, A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms, Machine Learning 40 (3) (2000) 203–228.
[26] H. Liu, Y. Liu, J. Liu, B. Zhang, G. Wu, Impulse force based ART network with GA optimization, in: Proceedings of the 2003 International Conference on Neural Networks and Signal Processing, vol. 1, 2003.
[27] S. Marriott, R.F. Harrison, A modified fuzzy ARTMAP architecture for the approximation of noisy mappings, Neural Networks 8 (1995) 619–641.
[28] H. Muhlenbein, D. Schlierkamp-Voosen, Predictive models for the breeder genetic algorithm I: continuous parameter optimization, Evolutionary Computation 1 (1) (1993) 25–49.
[29] D.J. Newman, S. Hettich, C.L. Blake, C.J. Merz, UCI repository of machine learning databases, 1998, URL ⟨http://www.ics.uci.edu/~mlearn/MLRepository.html⟩.
[30] P.P. Palmes, T. Hayasaka, S. Usui, Mutation-based genetic neural network, IEEE Transactions on Neural Networks 16 (3) (2005) 587–600.
[31] E. Parrado-Hernadez, E. Gomez-Sanchez, Y. Dimitriadis, Study of distributed learning as a solution to category proliferation in fuzzy ARTMAP based neural systems, Neural Networks 16 (2003) 1039–1057.
[32] N. Srinivas, L.M. Patnaik, Adaptive probabilities of crossover and mutation in genetic algorithms, IEEE Transactions on Systems, Man and Cybernetics 24 (4) (1994) 656–667.
[33] S.C. Tan, M.V.C. Rao, C.P.L. Lim, On the reduction of complexity in the architecture of fuzzy ARTMAP with dynamic decay adjustment, Neurocomputing 69 (2006) 2456–2460.
[34] S.J. Verzi, G.L. Heileman, M. Georgiopoulos, M. Healy, Rademacher penalization applied to fuzzy ARTMAP and boosted ARTMAP, in: Proceedings of the IEEE-INNS International Joint Conference on Neural Network, vol. 2, Washington, DC, 2001.
[35] B.A. Whitehead, T.D. Choate, Cooperative-competitive genetic evolution of radial basis function centers and widths for time series prediction, IEEE Transactions on Neural Networks 7 (4) (1996) 869–880.
[36] J.R. Williamson, Gaussian ARTMAP: a neural network for fast incremental learning of noisy multidimensional maps, Neural Networks 9 (5) (1996) 881–897.
[37] J.R. Williamson, A constructive, incremental-learning network for mixture modeling and classification, Neural Computation 9 (7) (1997) 1517–1543.
[38] X. Yao, Evolving artificial neural networks, PIEEE: Proceedings of the IEEE 87 (1999) 1423–1447.
[39] X. Yao, Y. Liu, Making use of population information in evolutionary artificial neural networks, IEEE Transactions on Systems, Man, Cybernetics Part B 28 (1998) 417–425.
[40] O.T. Yildiz, E. Alpaydin, Omnivariate decision trees, IEEE Transactions on Neural Networks 12 (6) (2001) 1539–1546.
[41] E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach, IEEE Transactions on Evolutionary Computation 3 (4) (1999) 257–271.

Assem Kaylani is a Ph.D. candidate in Computer Engineering at the University of Central Florida. He is also a Senior Software Engineer at Productivity Apex, Inc. in Orlando, FL, USA. He has received a B.S. in Electrical Engineering in 1998 from the University of Jordan, an MS in Computer Engineering in 2001 and Certificate Degree in Systems Simulation for Engineers in 2005, from the University of Central Florida. His research focus is on Evolutionary Neural networks. During his professional career, Mr. Kaylani led the development of several engineering software and research projects, funded by NASA and other organizations, focusing on data mining and simulation modeling.

Michael Georgiopoulos has received a Diploma in EE from the National Technical University of Athens, Greece, in 1981, an MS in EE and a Ph.D. in EE from the University of Connecticut, Storrs, CT, in 1983 and 1986, respectively. He joined the University of Central Florida in 1986, where he is currently a Professor in the School of EECS. His research interests lie in the areas of Machine Learning and applications with special emphasis on ART neural networks. He has published his work in over 230 journal and conference venues. He has been an Associate Editor of the IEEE Transactions on Neural Networks from 2002 to 2006 and he is currently serving as an Associate Editor of the Neural Networks journal. He is the general chair of the upcoming S + SSPR 2008 Workshops.

Mansooreh Mollaghasemi received her Bachelor of Science in Chemical Engineering, in 1982, Masters of Science in Chemical Engineering, in 1983, and Ph.D. in Industrial Engineering, in 1991 from the University of Louisville, Louisville, Ky. She has been with the University of Central Florida since 1991. Her research interest are in the areas of multi-criteria decision making, optimization, simulation modeling and analysis, and artificial intelligence. She has an extensive publication record in a number of prestigious journals, such as IIE Transactions, Interfaces, Annals of Operation Research, Transactions of the Society for Computer Simulation, Computers and OR, International Journal of Production Economics, and IEEE Transactions on Engineering Management.

Georgios C. Anagnostopoulos received the M.Sc. and Ph.D. degrees from the School of Electrical Engineering and Computer Science at the University of Central Florida, Orlando, in 1997 and 2001, respectively. Currently, he is an assistant professor in the Department of Electrical and Computer Engineering at the Florida Institute of Technology in Melbourne, Florida. His main research area is machine learning with emphasis on pattern recognition and detection as they apply to data modeling and mining, machine vision, remote sensing, and bioinformatics, among other fields.