Efficient Evolution of ART Neural Networks

A. Kaylani, M. Georgiopoulos, M. Mollaghasemi, G. C. Anagnostopoulos

Abstract— **Genetic algorithms have been used to evolve several neural network architectures. In a previous effort, we introduced the evolution of three well known ART architects; Fuzzy ARTMAP (FAM), Ellipsoidal ARTMAP (EAM) and Gaussian ARTMAP (GAM). The resulting architectures were shown to achieve competitive generalization and exceptionally small size. A major concern regarding these architectures, and any evolved neural network architecture in general, is the added overhead in terms of computational time needed to produce the finally evolved network. In this paper we investigate ways of reducing this computational overhead by reducing the computations needed for the calculation of the fitness value of the evolved ART architectures. The results obtained in this paper can be directly extended to many other evolutionary neural network architectures, beyond the studied evolution of ART neural network architectures.**

I. Introduction and Motivation

GENETIC ARTMAP (GART) architectures were introduced in [1] and [2]. It was shown that the these architectures were able to achieve very competitive classification accuracy and exceptionally small-size classifiers when compared to other classifiers in the literature (see [3], [4]). When compared to other ART architectures such as ssFAM, ssEAM, ssGAM [5] and safe-microARTMAP (see [6]), it was shown that these architectures were able to achieve a better generalization and smaller than or equal size network (in almost all problems tested), requiring reduced computational effort to achieve these advantages. In addition, the genetic ART architectures have the advantage of alleviating the need for tweaking algorithm parameters; a well known issue with many other ART and non-ART classifiers.

GART [4] uses a genetic algorithm (GA) [7] to evolve simultaneously the weights, as well a the topology of FAM, EAM or GAM neural networks. Genetic algorithms (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. GA's are capable of finding the global optima rather than the local optima as is the case with gradient descent procedures. Genetic algorithms have been extensively used to evolve (optimize) artificial neural networks [8], [9], [10]. The majority of this work, though, has been focused on MLP neural networks [11]. However, a number of authors proposed using evolutionary optimization algorithms with other neural network models such as RBF neural networks [12], [10]. In [11] a comprehensive literature review was conducted to summarize the prior efforts that aimed at combining evolutionary optimization algorithms with neural networks.

Michael Georgiopoulos is with the School of Electrical Engineering and Computer Science, University of Central Florida, Orlando, FL 32816, USA (phone: 407-823-5338; fax: 407-823-5835; email: michaelg@mail.ucf.edu).

TABLE I
ALLOCATION OF CPU TIME.

Total Time	Training Time Seconds	%	Validation Time Seconds	%	Algorithm Time Seconds	%
19.22	2.63	13.66	16.53	86.02	0.06	0.32
25.61	2.55	9.95	23.05	89.99	0.02	0.06
119.36	2.55	2.13	116.52	97.62	0.30	0.25
30.08	2.59	8.62	27.34	90.91	0.14	0.47
27.88	2.64	9.47	25.08	89.97	0.16	0.56
21.44	2.50	11.66	18.78	87.62	0.16	0.72
20.50	2.56	12.50	17.86	87.12	0.08	0.38
58.69	2.56	4.37	55.90	95.25	0.22	0.38
111.08	2.52	2.27	108.26	97.47	0.30	0.27
22.88	2.55	11.13	20.28	88.67	0.05	0.20
26.14	2.42	9.27	23.67	90.56	0.05	0.18

The evolution of ART architectures with genetic algorithms was introduced for the first time in [1] and extended in [2]. Genetic ART starts with a population of trained ART networks, 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 ART's training rules. To this initial population of networks, GA operators are applied to modify these trained networks (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.

During the $Evaluation$ step, the genetic algorithm estimates the performance (prediction error) of a solution (ART network) by measuring the classification error rate on a validation set. It was found that the majority of the CPU time spent by the genetic algorithm, is used for measuring the classification error (more than 80%) as shown in Table I. Therefore, techniques that can reduce this time component have the potential of reducing the overall convergence time of the GA, when used to evolve ARTMAP NNs. Such improvement can be effective to the viability of many evolved NN architectures.

Table I shows the allocation of CPU time for 10 replications (each row is an identical replication using a different random number seed of the evolutionary process) for running the Genetic ARTMAP algorithm using a sample database. It can be observed that the majority of time is spent during the validation of the solutions (members of the GA population).

The overall run time of the evolved ART architectures can be expressed as follows,

$$T_{total} = T_{training} + T_{validation} + T_{alg} \quad (1)$$

978-1-4244-1823-7/08/$25.00©2008 IEEE

The training time, $T_{training}$, is time required to train the initial population (using ART training rules). The component designated as T_{alg} refers to the overall housekeeping time for the algorithm. The validation time, $T_{validation}$, is the total time required to estimate the classification error (on a validation set) of ART networks in the population, in successive generations, until convergence. The validation time, $T_{validation}$ is dependent on the number of generations until convergence, G_{conv}, population size, λ, number of validation examples, n_v, average number of categories in each ART network, $\overline{size(x)}$ (taken over all generations and all networks in the population), and the time it takes to calculate the output signal (category match function and category choice function) for a given category and a given validation pattern , t_{CMF};

$$T_{validation} = G_{conv} * \lambda * n_v * \overline{size(x)} * t_{CMF} \qquad (2)$$

In this paper we try to reduce the validation time by reducing the number of validation points, n_v. Genetic algorithms are known for their suitability in problems where the fitness evaluation is not reliable or noisy. The noise in estimating objectives (such as classification error) has the effect of causing noisy selection. This in turn might slow down the convergence or convergence to a sub-optimal solution. The noise in the estimation of classification error might be hard to reduce, since we are limited by the data availability. However, when a large amount of data is available for cross-validation (our case of interest in this paper), it becomes time consuming to obtain an estimate for classification error using all the data. Therefore, evaluating the classification error for a given network in a given generation using a randomly sampled subset of the validation data might be beneficial in reducing the convergence time.

Reducing the number of validation points used in estimating the classification error of the produced ARTMAP solutions leads to more noisy evaluations of the GA fitness function. However, as suggested by [13], in some cases, the use of fitness estimates with higher variance could actually increase the quality of solutions obtained from the GA for a fixed computational budget. If time is saved by making faster, but noisier, estimates of the fitness function, then this time can be effectively used to converge to a better solution. To make a good judgement about this approach, one must understand the operation of GA, and how the variance of the estimate of the fitness function affects the progress rate of finding optimal solutions. The effect of noise in GAs has been studied by a number of researchers ([14], [15], [13], etc). Their results provide insight to convergence properties under noisy estimation of the fitness function.

In [16] the author presents a comprehensive survey of fitness function approximation in evolutionary algorithms. Fitness approximation, also referred to as evaluation relaxation, is applied when the fitness function is computationally expensive, noisy, or difficult to define and evaluate. The author identifies three levels of approximation and refer to them as problem approximation, functional approximation and evolutionary approximation. In problem approximation, which is the approach adopted in this paper, the original fitness function is replaced by a less expensive but less accurate one such as in [13]. In function approximation, an expression is constructed to approximate the evaluation of the original fitness function. Evolutionary approximation are methods that are specific to evolutionary algorithms such as fitness inheritance [17], where the fitness value of the offspring is estimated from that of their parents. This approach relies on surrogate functions, which are used to construct a relationship between offspring and parent fitness values. In [18] the authors propose surrogate functions that automatically adapts to the problem structure where the structural form of the surrogate is inferred using a probabilistic model and the coefficients of the surrogate are estimated using a least squares method.

In this paper, we use a less expensive fitness function by means of sampling. The sample size is controlled throughout the evolution in such a way to keep a certain level of confidence in the fitness value. The organization of the paper is as follows: In section II we present some of the preliminaries of the ART architectures (FAM, EAM, and GAM) that we genetically evolve; however we assume that the reader is somewhat familiar with the ART neural networks. In section III we explain how the FAM, EAM and GAM architectures are evolved to produce GFAM (genetic FAM), GEAM (genetic EAM) and GGAM (genetic GAM). In section IV, we provide an estimate for the number of validation points that could be used in the evolution of ART neural networks, and we justify, experimentally, that this estimate leads us to computational savings for the evolutionary process needed to produce GFAM, GEAM and GGAM. In section V, we provide some arguments that verify that the experimental results produced in section IV are justified on the premise of the legitimacy of noisy evaluations of fitness functions in the evolutionary process. In this section we find two sources of variance of the evaluated fitness function and show that time can be saved by properly sampling the validation set to estimate the fitness function. Finally, in Section VI we summarize our findings.

II. The ARTMAP Architectures

The Fuzzy ARTMAP (FAM) neural network architecture was introduced by Carpenter and Grossberg in their seminal paper [19]. Since its introduction, other ART architectures have been introduced into the literature. The focus in this paper is on Fuzzy ARTMAP and two other ART architectures: Ellipsoidal ARTMAP [20] and Gaussian ARTMAP [21]. 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 III.

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

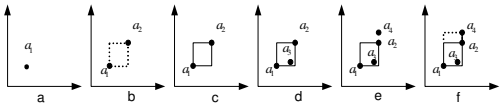

Fig. 1. FAM Learning (2-D Example). (a) A category with 0 size; (b) Introducing a new pattern $\mathbf{I}_2$, represented by $\mathbf{a}_2$; (c) The category expands to include $\mathbf{a}_2$; (d) Since new pattern $\mathbf{I}_3$, represented by $\mathbf{a}_3$ is inside the category, it doesn't change 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

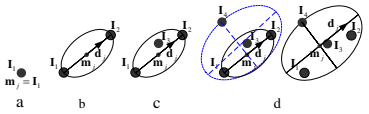

Fig. 2. EAM Learning (2-D Example). (a) A category with 0 size; (b) Introducing a new pattern $\mathbf{I}_2$; the category expands to include $\mathbf{I}_2$; (c) Introducing a new pattern $\mathbf{I}_3$; since the category includes $\mathbf{I}_3$, it does not change its size; (d) Pattern $\mathbf{I}_4$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

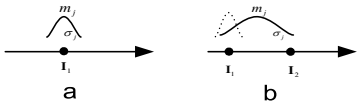

Fig. 3. GAM Learning (1-D example). (a). A category with 0 size ; (b) Introducing a new pattern $\mathbf{I}_2$; the category characteristics (mean, standard deviation, of the Gaussian curve, as well as number of points encoded by the Gaussian curve) change to include the new knowledge that the new input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

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, $\mathbf{I}_1, label(\mathbf{I}_1)$, $\mathbf{I}_2, label(\mathbf{I}_2)$, ..., $\mathbf{I}_{PT}, label(\mathbf{I}_{PT})$ (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 Figure 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 Figure 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 Figure 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 Figure 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 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, a 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.

III. Genetic ART Architectures

GFAM, GEAM, and GGAM 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. GART uses an evolutionary approach to find networks that achieve optimal performance in terms of two objectives: maximizing classification accuracy and minimizing complexity (size) of ARTMAP classifier. Figure 4 lists the pseudo-code for the basic steps of GART.

$P(0) \leftarrow$ Generate-Initial-Population();
for $t \leftarrow 1$ **to** Gen_{max} **do**
  Evaluation();
  if *stopping criteria met* **then** exit for;
  $P'(t) \leftarrow$ Selection($P(t)$);
  $P(t) \leftarrow$ Reproduction($P'(t)$);
end
return *Best Network in* $P(t)$;

Fig. 4. Pseudo Code of GART Algorithm

The algorithm starts by generating an initial population, $P(0)$, of 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 our implementation we fixed the population size, $Pop_{size} = 20$. The networks are encoded into chromosomes, where each component (gene) represents a category (hidden node) of an ART network. Each component contains the weight information for the category. The chromosomes in GART are variable length, where the length is equal to the number of categories in the network represented by the chromosome (see Figures 5, 6, 7).

A weighed 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 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 [22]).

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:

$$fit(p) = p_{err}(p) + \alpha \frac{size(p) - Cat_{min}}{size_{max}} \qquad (3)$$

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

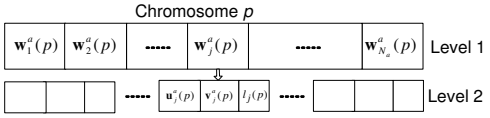

Fig. 5. GFAM chromosome structure. At level 2, the category's weight $\mathbf{w}_j^a$ contains the information about the lower end-point, $\mathbf{u}_j^a$, and the upper end-point, $\mathbf{v}_j^a$, of the hyperbox corresponding to the category, as well as the label l_j of the category.

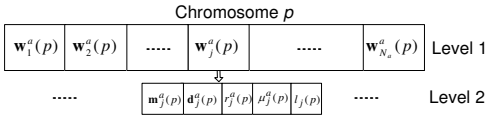

Fig. 6. GEAM chromosome structure. At level 2, the category's weight $\mathbf{w}_j^a$ contains the information of the center, $\mathbf{m}_j^a$, the direction vector of the major axis, $\mathbf{d}_j^a$, the radius (half length) of the major axis, r_j^a, and the ratio of the lengths of the minor axes over the length of the major axis, μ_j^a, of the ellipsoid corresponding to this category, as well as the label l_j of the category.

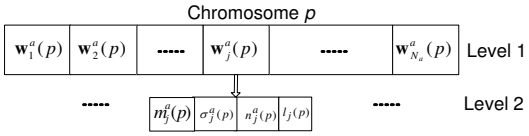

Fig. 7. GGAM chromosome structure. At level 2, the category's weight $\mathbf{w}_j^a$ contains the information of the center of the Gaussian curve, $\mathbf{m}_j^a$, the standard deviation vector of the Gaussian curve, σ_j^a, and the number of points represented by the Gaussian curve, n_j^a, as well as the label l_j of the category.

equation that if the GA drops the size from $size_{max}$ to Cat_{min}, the fitness drops by approximately α. Therefore, the maximum sacrifice in classification accuracy is equal to α. In our work here we use a value of $\alpha = 0.01$. This indicate a maximum possible sacrifice of about 1% from the best accuracy in the initial population. Extensive experimentation showed that this value works well on a wide range of datasets.

The algorithm runs for a maximum number of generations defined by Gen_{max}. In our implementation we set $Gen_{max} = 500$. However, to avoid running GART for unnecessarily large number of generations, the evolution is also stopped when no significant improvement is achieved for 50 consecutive generations.

The selection process creates a temporary population 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, $P(t+1)$ without change. In our implementation we choose a value of $NC_{best} = 3$.

Once the selection step determines the parents, reproduction operators are used to create individuals for the next generation. 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 the weights of the categories of the ARTMAP network. The second operator, referred to as the *Prune operator*, prunes a network by deleting a number of categories from that network (structural mutation).

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 (see [23]). The confidence factor is a metric that measures the performance at the category level. The performance of a category is defined in terms of its accuracy and relative frequency of selection.

$$CF_j^k(p) = 0.5A_j^k(p) + 0.5S_j^k(p) \quad (4)$$

where $A_j^k(p)$ is the accuracy of classification achieved by category j, in the $p-th$ network, that is mapped to label k, relative to the best accuracy achieved by any category in the same network that is mapped to the same label. Furthermore, $S_j^k(p)$ is the probability of selection of category j in the $p-th$ network, that is mapped to label k, relative to the maximally selected category in the same network that is mapped to the same label.

Once CF is calculated for each category, with probability of $1-CF_j^k(p)$, we delete categories from every chromosome in the temporary population $P'(t)$. Also, the weights of every category are mutated using a gaussian distribution that has a mean of 0 and standard deviation of $0.05(1 - CF_j^k(p))$. Therefore, categories with low CF are more likely to be eliminated or more severely mutated.

The remaining (after elitism) chromosomes in $P(t+1)$ are then replaced by chromosomes created by performing crossing over pairs of parents in $P'(t)$. For each parent, p, p', a random cross-over 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. As mentioned above, the evolutionary process continues until one (of the two) stopping criterion is triggered.

IV. Experimental Results

In this section we present an experimental demonstration of the ideas outlined in this paper. Our claim is that computation time can be saved by taking a sample from the validation set to estimate the error rate of a classification rule during the genetic evolution. In evolving neural networks, the error rate is the only (or the main) objective to be optimized. Without loss of generality, we take evolution of ART networks as example. ART networks are evolved by repeatedly applying genetic operators to a population of ART networks. In every generation the selection process determines the probability of survival and breeding of an individual (network) based on its performance. Similar to a number of evolved neural network architectures, the evolved ART architectures introduced in [4], the performance is measured using a fitness function that is a linear combination of the error rate measured on the validation set and the network complexity measured in terms of the number of hidden nodes present in that network. The claim presented in this paper is that making a faster but noisier estimation of the validation error might eventually achieve similar solution quality at reduced computational cost. The fast estimation of classification error is made by randomly sampling a subset of the available validation samples each time we need to estimate the classification error for a given network.

To come up with a reasonable estimate of the number of samples (n_v) that we want to sample from the available validation data to estimate the value of the fitness function, we proceed as follows. Given a classification rule *R* and a classification problem *C*, define p_{err} as the error rate of *R* as a result of classifying patterns in *C*. If the class labels of patterns in *C* are known, then applying *R* to *C* and comparing the predicted and actual class labels, would result in a population whose members are either "correct"

or "error", resulting in the following equation that calculates p_{err}:

$$p_{err} = \frac{\#errors}{\#patterns} \quad (5)$$

The above evaluation of p_{err} is accurate if we have infinitely many patterns with known labels that we pass through our classification rule *R*. However, while evolving ART networks, each network (or solution), corresponding to a classification rule *R(x)*, is evaluated by applying it to a sample of size n_v patterns that belong to a classification problem, *C*. Therefore, an estimate of p_{err} is obtained, denoted as $\hat{p}_{err}$, as follows:

$$\hat{p}_{err} = \frac{\#error(n_v)}{n_v} \quad (6)$$

The variance of this estimate is (see [24], page 388):

$$Var(\hat{p}_{err}) = \frac{p_{err}(1 - p_{err})}{n_v} \quad (7)$$

It can be shown that estimate $\hat{p}_{err}$ is normally distributed according to $N(p_{err}, Var(\hat{p}_{err})$ for sufficiently large n_v. In our experiments we chose a half width of 0.01 for the estimation of the classification error and confidence level of 99%, and used equation 8 to determine the number of patterns to be randomly chosen from the set of available validation samples. However, other reasonable choices of half width values and confidence levels can be used as well.

$$n_v = (\frac{z_{\alpha/2}}{Halfwidth})^2 \hat{p}_{err}(1 - \hat{p}_{err}) \quad (8)$$

The experiments presented in this section compare the genetically engineered ART architectures introduced in [4], which use all the validation patterns in every evaluation, and the same genetically engineered ART architectures that sample the validation set in every evaluation, as designated by equation (8). The genetically engineered ART architectures introduced in [4] include three architectures: Genetic Fuzzy ARTMAP (GFAM), Genetic Ellipsoidal ARTMAP (GEAM) and Genetic Gaussian ARTMAP (GGAM). The results pertaining to GART without sampling the validation patterns are referred as "GFAM Old", "GEAM Old", and "GGAM Old", while the results using the sampling according to equation (8) are referred to as "GFAM New", "GEAM New", and "GGAM New".

We have experimented with 9 databases, of which 4 are simulated databases and 5 are real databases. Each database was randomly divided into three subsets; training, validation and testing. The simulated databases include 2 Gaussian databases: G4C-25 and G6C-15. These are, 2-dimensional databases with 4-classes and 6-classes, and 15% and 25% overlap, between the classes. The database denoted by 1Ci/Sq is the benchmark one circle in a square problem, 2-dimensional, two class classification problem. The probability of finding a data point within a circle or inside the square and outside the circle is equal to 1/2. The rest of the databases were obtained from the UCI repository (see [25]) and they include: Modified Iris, Page blocks (PAGE), Pendigits, Satellite Image (SAT), Image Segmentation (SEG), and Waveform (WAV). More details about these databases can be found there.

For each of the 9 databases, we ran the "old" and "new" architectures 10 times for 10 different initial seeds of the GA optimization process. In Table II we compare the average time (over the 10 replications) required for convergence for the "old" and "new" architectures. In particular, we compare the "old" GFAM to the "new" GFAM, the "old" GEAM to the "new" GEAM, and the "old" GGAM to the "new" GGAM. It can be observed from Table II that the time saved in the "new" implementation can reach up to 80% of time that the "old" took to converge, justifying the merit of the proposed technique. We also record the best performance achieved by both these implementations, in terms of percent of correct classification (PCC) and network size, in Table III. In this table it can be seen that both implementations were able to converge to similar quality of solutions. Therefore, the saving in time did *not* require a sacrifice in the quality of the solutions achieved.

V. Discussion

When the evaluation of solutions is not reliable, the genetic algorithm may suffer from selection error. The result of selection error is reduction in the efficiency of the genetic algorithm and the possibility of reduction in the quality of solutions returned. Selection error happens because the estimation of the fitness function is not accurate. The inaccuracy in the calculation of the fitness function estimation results in incorrect assignment of fitness for solutions. Therefore, the selection operator proceeds in a manner that is different than the way it would have if the fitness function evaluations were accurate.

In [26], [27] the operation of genetic algorithm is explained by understanding the allocation of trials to solutions. This allocation of trials is determined by the selection mechanism in genetic algorithms. It is expected that the genetic algorithm would allocate more trials to more promising areas in the solution space. Let $P(t)$ denote a population of solutions, $x \in P(t)$, at generation t, and P is of size λ. A hyperplane H, also referred to as *schema*, defines a subset of all the possible solutions of the problem, where the solutions are referred to as instances of H. In binary coding, a schema can be represented as a string of ones, zeros and asterisks, where the asterisks serve as wildcards. For example, the schema $H = 1 * * * 0$ represents all strings that start with 1 and end with 0, and strings "10110" and "10010" are said to be instances of H. Note that these two strings are also instances of many other possible schemas. Therefore, when the genetic algorithm is explicitly evaluating solutions in each population of every generation, it is implicitly estimating the average fitness of a much larger number of schemas, where the average fitness of a schema is defined as the average of fitness of all possible instances in that schema. This is referred to as the *implicit parallelism* in genetic algorithms (see [28], [7], [26]). Let $\mu(P, t)$ be the average fitness of

TABLE II

COMPARING THE TRAINING TIME OF GENETIC ART WITH AND WITHOUT SAMPLING OF THE CROSS-VALIDATION SET.($\alpha = 0.01$, TAKE BEST OF 10 REPS)

	GFAM			GEAM			GGAM		
	Old	New	% Time Saved	Old	New	% Time Saved	Old	New	% Time Saved
1Ci/Sq	68.51	22.12	67.72%	152.13	68.15	55.20%	142.29	28.31	80.10%
G4C-25	52.59	23.33	55.63%	82.99	47.21	43.11%	89.32	32.49	63.62%
G6C-15	92.01	27.49	70.12%	127.15	43.44	65.83%	137.82	34.67	74.84%
Iris	13.48	3.47	74.22%	21.56	7.51	65.18%	22.02	4.65	78.89%
page	52.97	13.36	74.77%	60.91	36.23	40.52%	75.03	20.78	72.30%
pendigits	1142.43	874.87	23.42%	4304.84	3923.03	8.87%	537.62	311.02	42.15%
sat	508.21	236.86	53.39%	1234.62	1037.92	15.93%	329.07	192.67	41.45%
seg	41.93	29.82	28.89%	88.45	81.31	8.07%	64.13	63.32	1.26%
wav	147.23	112.85	23.35%	369.38	338.54	8.35%	83.94	56.67	32.48%

TABLE III

COMPARING THE PERFORMANCE OF GENETIC ART WITH AND WITHOUT SAMPLING OF THE CROSS-VALIDATION SET

	Old						New					
	GFAM		GEAM		GGAM		GFAM		GEAM		GGAM	
	PCC	size	PCC	size	PCC	size	PCC	size	PCC	size	PCC	size
1Ci/Sq	98.07	31	99.70	2	99.83	2	97.67	31	99.27	2	98.93	2
G4C-25	74.94	4	75.14	4	75.24	4	74.98	4	74.96	4	75.22	4
G6C-15	84.75	6	85.01	6	84.97	6	84.85	6	85.09	6	85.11	6
Iris	94.96	2	95.04	2	94.75	2	95.08	2	95.04	2	94.94	2
page	96.59	5	95.09	5	96.34	6	96.56	5	95.30	5	95.56	5
pendigits	98.20	282	98.31	331	97.83	108	97.86	276	96.97	175	97.86	129
sat	88.90	310	87.85	203	88.35	118	88.40	184	87.30	173	88.15	127
seg	94.86	22	93.71	128	92.71	17	95.86	35	94.43	129	91.57	13
wav	85.90	4	87.15	4	87.50	3	84.05	4	86.60	8	87.65	4

solutions $x \in P(t)$ at generation t, and similarly, $\mu(H,t)$ is the average fitness of solutions that are instances of schema H at generation t. Let $M(H,t)$ be the number of solutions in $P(t)$ that are also instances of H at generation t. The allocation of the search trials to schemas can be measured by calculating the expected value of $M(H,t+1)$, that is, the number of solutions that are instances of H in the next generation (see [26]):

$$E(M(H,t+1)) = M(H,t)\frac{\mu(H,t)}{\mu(P,t)} \tag{9}$$

This result indicates that the number of trials allocated to an above-average hyperplane H grows exponentially over time. It was shown (see [26]) that this exploration rate is optimal (or close to optimal) search strategy. This result also indicates that the ability of genetic algorithm to find good solutions is determined by the ability to accurately estimate the average fitness of a given hyperplane H over successive generations. The accurate evaluation of the average fitness of a given hyperplane is equivalent to making accurate selection. The reliability of (implicitly) evaluating the fitness of a hyperplane can be measured by its variance. The larger the variance, the larger is the selection error and therefore, the less efficient is the genetic algorithm.

The average fitness of solutions that are instances of H is estimated based on a sample from all the possible instances of H. Therefore, the sampling distribution has variance $\sigma_r^2 = \frac{\sigma_H^2}{r}$ where r is the number of trials allocated to schema H, and σ_H^2 is the variance of the fitness of all instances of H. The number of trials, r, is dependent on the population size and the number of generations the algorithm runs. Therefore, increasing r can be done by increasing either the population size or the maximum number of generations, or both. Increasing r would result in improved estimation of the average fitness of H and therefore, would result in the GA finding better solutions. However, if the measurement of each of the r trials is not reliable (i.e., n_v is not large enough), then there is another source of variance. The larger the variance of estimating the fitness in each of the r trials is, the larger the variance of the estimation of the average fitness of a given hyperplane becomes, and consequently the worse is the selection error. To improve the performance of the genetic algorithm, the variance of the estimation of the average fitness of a given hyperplane should be reduced. It can be shown (see [13]) that the variance of the estimation of the fitness of a given hyperplane, $\hat{p}_{err}^H$, is given by the following formula:

$$Var[\hat{p}_{err}^H] = \frac{\sigma_H^2}{r} + \frac{\overline{p_{err}(1-p_{err})}}{rn_v} \tag{10}$$

where, in the above equation $\overline{p_{err}(1-p_{err})}$ is the variance of the binomial distribution of the error rate averaged over all possible instances of H.

From the above equation it is obvious that we can decrease the variance of the estimate of the average hyperplane fitness value by increasing r, or increasing n_v, or increasing both. Increasing n_v is an expensive proposition (see Table I). A more reasonable approach is to increase r, and decrease n_v so that rn_v remains approximately constant. Our discussion above, and equation 10 indicate that the favorable

experimental results that were obtained in Section IV for the evolved ART neural networks, where an appropriate size (n_v) of a validation set was chosen, would extend to other datasets (beyond the ones that we experimented with in Section IV) and to other neural network architectures (beyond the paradigm of ART neural network architectures that we emphasized in this paper).

VI. Conclusion

In this paper we have introduced a technique that can be used to significantly improve the efficiency of many evolved neural network architectures. The technique proposed capitalizes on the ability of genetic algorithm to operate effectively in noisy environments. It was shown experimentally, and argued qualitatively, that relying on faster, but noisier, estimation of classification error during the evolution of neural networks might be beneficial to the overall computational cost of evolving neural network architectures. Some of the time saved by making fast evaluations of the classification error of the evolved neural networks is used to allow the evolutionary process to reach the desired level of solution quality faster, despite the fact that more generations might be needed to achieve this goal.

The merit of the proposed technique was illustrated using a family of evolved ART neural network architectures. It was shown, using the technique proposed in this paper, that significant amount of computational time (as much as 80% in some cases) can be saved if we rely on noisy calculations of the classification error but allow the GA process to evolve over a higher number of generations. We also demonstrated that this improvement in efficiency did not affect the quality (accuracy and size) of the classifier network produced. Our claim though is that these beneficial results would extend to other classification problems (beyond the ones we experimented with in this paper) and to other neural network architectures (beyond the ART neural networks considered in this paper).

Acknowledgment

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, and G. C. Anagnostopoulos, "GFAM: Evolving Fuzzy ARTMAP neural networks," *Neural Networks*, vol. 20, no. 8, p. 873891, October 2007.

[2] A. Kaylani, M. Georgiopoulos, M. Mollaghasemi, and G. Anagnostopoulos, "m-gfam: An elegant approach to genetically optimize fuzzy artmap neural network architectures," in *Proceedings of the 8th International Conference on Natural Computing*, July 2007, pp. 1617 – 1623.

[3] A. Kaylani, A. Al-Daraiseh, M. Georgiopoulos, M. Mollaghasemi, G. C. Anagnostopoulos, and A. S. Wu, "Genetic optimization of art neural network architectures," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, August 2007, pp. 1114–1119.

[4] A. Kaylani, M. Georgiopoulos, M. Mollaghasemi, and G. Anagnostopoulos, "Genetic optimization of art neural network architectures," in *Proceeding of Artificial Intelligence and Soft Computing (ASC) 2007 Conference*, 2007, pp. 225–230.

[5] G. C. Anagnostopoulos, M. Bharadwaj, M. Georgiopoulos, S. J. Verzi, and 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, pp. 2782–2787.

[6] E. Gomez-Sanchez, Y. Dimitriadis, J. Cano-Izquierdo, and 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, pp. 1197–1202.

[7] D. Goldberg, *Genetic Algorithms in search, optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[8] K. P. Ferentinos, "Biological engineering applications of feedforward neural networks designed and parameterized by genetic algorithms," *Neural Netw.*, vol. 18, no. 7, pp. 934–950, 2005.

[9] T. H. P. P. Palmes and S. Usui, "Mutation-based genetic neural network," *IEEE Transactions on Neural Networks*, vol. 16, no. 3, 2005.

[10] B. A. Whitehead and T. D. Choate, "Cooperative-competitive genetic evolution of radial basis function centers and widths for time series prediction," *IEEE Transactions on Neural Networks*, vol. 7, no. 4, pp. 869–880, July 1996.

[11] X. Yao, "Evolving artificial neural networks," *PIEEE: Proceedings of the IEEE*, vol. 87, pp. 1423–1447, 1999.

[12] X. Fu and L. Wang, "A ga-based rbf classifier with class-dependent features," in *CEC '02: Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress*. Washington, DC, USA: IEEE Computer Society, 2002, pp. 1890–1894.

[13] J. M. Fitzpatrick and J. J. Grefenstette, "Genetic algorithms in noisy environments," *Machine Learning Journal*, vol. 3, no. 2-3, pp. 101–120, October 1988.

[14] B. L. Miller, "Noise, sampling, and efficient genetic algorithms," Ph.D. dissertation, Champaign, IL, USA, 1997.

[15] D. E. Goldberg, K. Deb, and J. Clark, "Genetic algorithms, noise, and the sizing of populations," *Complex Systems*, vol. 6, pp. 333–362, 1992.

[16] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *Soft Computing*, vol. 9, no. 1, pp. 3–12, 2005.

[17] R. E. Smith, B. A. Dike, and S. A. Stegmann, "Fitness inheritance in genetic algorithms," in *SAC '95: Proceedings of the 1995 ACM symposium on Applied computing*. New York, NY, USA: ACM, 1995, pp. 345–350.

[18] K. Sastry, C. F. Lima, and D. E. Goldberg, "Evaluation relaxation using substructural information and linear estimation," in *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2006, pp. 419–426.

[19] G. A. Carpenter, S. Grossberg, N. Markuzon, J. H. Reynolds, and D. B. Rosen, "Fuzzy ARTMAP: A neural network architectur for incremental supervised learning of analog multidimensional maps," *IEEE Transactions on Neural Networks*, vol. 3, pp. 698–713, 1992.

[20] G. Anagnostopoulos, "Novel approaches in adaptive resonance theory for machine learning," Ph.D. dissertation, University of Central Florida, Orlando, May 2001.

[21] J. R. Williamson, "Gaussian ARTMAP: a neural network for fast incremental learning of noisy multidimensional maps," *Neural Networks*, vol. 9, no. 5, pp. 881–897, 1996.

[22] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Wadsworth, 1984.

[23] G. A. Carpenter and H. A. Tan, "Rule extraction: From neural architecture to symbolic representation," *Connection Science*, vol. 7, pp. 3–27, 1995.

[24] W. Mendenhall and T. L. Sincich, *Statistics for Engineering and the Sciences*. Prentice Hall, 1995.

[25] D. J. Newman, S. Hettich, C. L. Blake, and C. J. Merz, "UCI repository of machine learning databases," 1998. [Online]. Available: http://www.ics.uci.edu/∼mlearn/MLRepository.html

[26] M. Mitchell, *An Introduction to Genetic Algorithms*. MIT Press, 1996.

[27] K. A. D. Jong, "An analysis of the behavior of a class of genetic adaptive systems." Ph.D. dissertation, 1975.

[28] J. Holland, *Adaptation In Natural and Artificial Systems*. Ann Arbor: The University of Michigan Press, 1975.