Varieties of Learning Automata: An Overview

M. A. L. Thathachar, *Fellow, IEEE,* and P. S. Sastry, *Senior Member, IEEE*

***Abstract*—Automata models of learning systems introduced in the 1960s were popularized as learning automata (LA) in a survey paper in 1974 [1]. Since then, there have been many fundamental advances in the theory as well as applications of these learning models. In the past few years, the structure of LA has been modified in several directions to suit different applications. Concepts such as parameterized learning automata (PLA), generalized learning automata (GLA), and continuous action-set learning automata (CALA) have been proposed, analyzed, and applied to solve many significant learning problems. Furthermore, groups of LA forming teams and feedforward networks have been shown to converge to desired solutions under appropriate learning algorithms. Modules of LA have been used for parallel operation with consequent increase in speed of convergence. All of these concepts and results are relatively new and are scattered in technical literature. An attempt has been made in this paper to bring together the main ideas involved in a unified framework and provide pointers to relevant references.**

***Index Terms*—Continuous action-set learning automata (CALA), generalized learning automata (GLA), modules of learning automata, parameterized learning automata (PLA), teams and networks of learning automata.**

I. Introduction

INVESTIGATION of learning automata (LA) began in the erstwhile Soviet Union with the work of Tsetlin [2], [3]. These early models were referred to as *deterministic and stochastic automata operating in random environments.* Further work was taken up by Fu and others in the U.S. in the 1960s [4]–[6]. In its current form, an LA roughly corresponds to what was called *variable structure stochastic automaton* [7] in the early models. The term *learning automata (LA)* was first publicized in the survey paper by Narendra and Thathachar [1] though earlier uses of this term are known [8], [9]. Since then, the field has seen much development and a number of books and survey papers have appeared [10]–[13]. The notion of reinforcement learning [14], which has received a lot of attention in recent years, represents a development closely related to the work on LA.

Systems built with LA have been successfully employed in many difficult learning situations over the years. This has also led to the concept of LA being generalized in a number of directions in order to handle various learning problems. However, some of these extensions to the structure of LA are not so well known. This paper is aimed at summarizing the main results available on such extended notions of LA.

Manuscript received November 15, 2001; revised March 1, 2002. This work was supported by AICTE through an Emeritus Fellowship. This paper was recommended by Associate Editor M. S. Obaidat.

The authors are with the Department of Electrical Engineering, Indian Institute of Science, Bangalore 560012, India (e-mail: malt@ee.iisc.ernet.in; sastry@ee.iisc.ernet.in).

Publisher Item Identifier S 1083-4419(02)06464-6.

The original notion of LA corresponds to what is called *finite action-set learning automata (FALA).* This type of LA has been studied extensively. In many applications, one needs to use a number of LA and often this leads to teams or networks of LA, which will be discussed in Section II. In such situations, almost all of the current algorithms available for FALA assure convergence only to a local maximizer of the reinforcement signal. One can get convergence to global maximum by modification of the learning algorithm which entails a random walk term being superposed on the probability updating. This in turn needs a parameterization of action probabilities and leads to *parameterized learning automata (PLA),* which are considered in Section III. In a number of random environments which naturally appear in pattern recognition and control problems, the action probabilities have to be updated based upon the *context vector*, which is typically a feature vector or a state vector. This leads to another modified LA structure which has been called *generalized learning automata (GLA)*. This is discussed in Section IV. In situations where the objective of learning is a continuous valued parameter, the values of which cannot be discretized, the action-set corresponds to an interval over the real line. Such continuous action spaces can be handled by *continuous action-set learning automata (CALA),* which are described in Section V. Rapid convergence of LA can be achieved by their parallel operation. Such parallel operation through the use of modules of LA, which result in increased speed of convergence, is the subject of Section VI. In Section VII, we illustrate the utility of some of the LA models on a pattern classification example. Pointers to some of the recent applications of automata models are provided in SectionVIII. Finally, SectionIX concludes the paper.

The common theme running through our discussion of all the models presented here is an optimization view of learning briefly outlined below.

Tsypkin [15] was among the first to formalize a unifying framework for all learning problems as that of optimizing a performance index

$$J(\mathbf{w}) = \int_{\mathcal{X}} \mathcal{R}(\mathbf{x}, \mathbf{w})\, d\mathcal{P} \tag{1}$$

where $\mathcal{R}(\mathbf{x}, \mathbf{w})$ is a functional of the parameter vector $\mathbf{w}$ and observation vector $\mathbf{x}$, and $\mathcal{X}$ is the space of all $\mathbf{x}$. The performance index J is the expectation of $\mathcal{R}$ with respect to the random $\mathbf{x}$, the distribution of which is given by $\mathcal{P}$. The special feature of learning problems is that the probability distribution $\mathcal{P}$ is unknown. Hence, given a $\mathbf{w}$, the value of $J(\mathbf{w})$ cannot be calculated. We are required to find the optimizer of J based on experience with a sequence of $\mathbf{x}$'s drawn according to the unknown distribution $\mathcal{P}$.

As an example of the above structure, a two-class pattern recognition problem can be formulated by choosing

$\mathcal{R}((\mathbf{x}, y), \mathbf{w}) = \ell(G(\mathbf{w}, \mathbf{x}), y)$, where $(\mathbf{x}, y)$ is a (random) training sample with $\mathbf{x}$ as the feature vector and y as its associated class label; $G(\mathbf{w}, \mathbf{x})$ is the output of a classifier, with parameter vector $\mathbf{w}$, on $\mathbf{x}$; and $\ell(\cdot, \cdot)$ is an appropriate loss function. If we consider the case of a 0–1 loss function, then $J(\mathbf{w})$ would be the probability of wrong classification by the classifier corresponding to the parameter vector $\mathbf{w}$. Now the problem is how to choose some "good" parameterization for a desirable family of classifiers and then search for a minimizer of J using only values of $\mathcal{R}$ on some random observations.

The distinguishing characteristic of automata-based learning is that the search for the optimizing parameter vector is conducted in the space of probability distributions defined over the parameter space, rather than in the parameter space itself. While this may appear as complicating the problem further, it means we do not need to assume that the space is isomorphic to some real Euclidean space or even that it is a metric space. This permits a great deal of flexibility and allows one to work with, for example, rich families of classifiers in a pattern recognition context. The different LA models that we discuss in this paper are all motivated by the requirements of such an optimization framework.

II. Finite Action-Set Learning Automata (FALA)

An LA is an adaptive decision-making device that learns the optimal action out of a set of actions through repeated interactions with a random environment. The two characteristic features of LA are that the action choice is based on a probability distribution over the action-set and it is this probability distribution that is updated at each instant based on the reinforcement feedback from the environment. Traditionally, the action-set is always considered to be finite. That is why we termed this traditional model of LA *finite action-set learning automata or FALA.* In this section, we briefly describe the FALA model and how a number of FALA can be configured as teams or networks for solving complex problems.

Let $A = \{\alpha_1, \ldots, \alpha_r\}$, $r < \infty$, be the *set of actions* available. At each instant k, the automaton chooses an *action* $\alpha(k) \in A$, at random, based on its current *action probability distribution* $\mathbf{p}(k) = (p_1(k), \ldots, p_r(k))^t$, $k = 0, 1, \ldots$.[1] (Here, $p_i(k) = \text{Prob}[\alpha(k) = \alpha_i]$ and $\sum_{i=1}^{r} p_i(k) = 1, \forall k$). The action chosen by the automaton is the input to the environment which responds with a stochastic *reaction* or *reinforcement*, $\beta(k) \in R \subseteq [0, 1]$ where R is the set of possible reactions. Higher values of the reinforcement signal are assumed more desirable. Let d_i denote the expected value of $\beta(k)$ given $\alpha(k) = \alpha_i$. Then d_i is called the *reward probability* associated with action α_i, $1 \leq i \leq r$. Define the index m by $d_m = \max_i\{d_i\}$. Then the action α_m is called the *optimal action*. In the above, we have implicitly assumed that d_i, $1 \leq i \leq r$, and hence the identity of the optimal action are not time-varying. In this case, the environment is said to be *stationary.* Otherwise, the environment is said to be *nonstationary.*

The LA has no knowledge of the reward probabilities. The objective for the automaton is to identify the optimal action. This is to be achieved through a learning algorithm that updates, at each instant k, the action probability distribution $\mathbf{p}(k)$ into $\mathbf{p}(k+1)$ using the most recent interaction with the environment, namely, the pair $(\alpha(k), \beta(k))$.

Definition 2.1: A learning algorithm is said to be *ϵ-optimal* if given any $\epsilon > 0$, we can choose parameters of the learning algorithm such that with probability greater than $1 - \epsilon$

$$\text{Lim} \inf_{k \to \infty} p_m(k) > 1 - \epsilon.$$

From the above definition, it is easily seen that ϵ-optimality is achieved if and only if $\text{Lim}\inf_{k\to\infty} E[\beta(k)] > d_m - \epsilon$. Thus, the objective of the learning scheme is to maximize the expected value of the reinforcement received from the environment. Hence, an equivalent way of characterizing the goal of an automata algorithm is

$$\text{maximize} f(\mathbf{p}) = E[\beta(k) \,|\, \mathbf{p}(k) = \mathbf{p}]$$

where the optimization is over the r-dimensional simplex, that is, over all possible action probability vectors. (For brevity, such expectations will be denoted as $E[\beta|\mathbf{p}]$ in the sequel.) This is the manner in which LA are most often used in applications. The actions of the automaton represent the possible values of the parameters and the automata algorithms search in the space of probability distributions defined over the parameter space. This special characteristic of the automata algorithms defines the niche in the application domain where the automata algorithms are most useful.

There are many learning algorithms that are proven to be ϵ-optimal, e.g., linear reward inaction (L_{R-I}), estimator algorithms such as the pursuit algorithm, etc. In these algorithms, the action probabilities are treated as continuous variables. Another class of FALA are the so-called discrete automata where the action probabilities are quantized. There are discrete versions of L_{R-I} and pursuit algorithms which are ϵ-optimal and which, generally, exhibit faster rate of convergence (see, for example, [16]–[21] for more details of FALA algorithms).

A. Games of FALA

As briefly outlined earlier in this paper, a single automaton is generally sufficient for learning the optimal value of one parameter. However, for multidimensional optimization problems, we need a system consisting of as many automata as there are parameters. One possible configuration for such a system of automata is a game of automata.

Let $A_1, \ldots, A_N$ be the automata involved in an N-player game. Each play of the game consists of each of the automata players choosing an action and then getting the *payoffs* or reinforcements from the environment for this choice of actions by the group of LA. Let $\mathbf{p}_1(k), \ldots, \mathbf{p}_N(k)$ be the action probability distributions of the N automata. Then, at each instant k, each of the automata A_i chooses an action $\alpha^i(k)$ independently and at random according to $\mathbf{p}_i(k)$, $1 \leq i \leq N$. This set of N actions is input to the environment which responds with N random payoffs, which are supplied as reinforcements to the corresponding automata. Let $\beta^i(k)$ denote the reinforcement to automaton A_i for the choice of actions made at instant k. We

[1]The superscript t denotes transpose.

assume $\beta^i(k) \in [0, 1], \forall i, k$. Let S_i denote the action-set of A_i. Define functions $d^i: \Pi_{j=1}^N S_i \rightarrow [0, 1], 1 \leq i \leq N$ by

$$d^i(x_1, \ldots, x_N) = E[\beta^i(k) \,|\, \alpha^i(k) = x_i,\ 1 \leq i \leq N]. \quad (2)$$

The function d^i is called the *payoff function* for automaton A_i. Essentially, automata want to maximize their payoff. Since there are multiple payoff functions, one way of defining the objective of learning here is to reach a *Nash equilibrium*.

Definition 2.2: The N-tuple of actions $(a_1, \ldots, a_N)$ is called a *Nash equilibrium* of this game if, for each $i, 1 \leq i \leq N$

$$d^i(a_1, \ldots, a_{i-1}, x, a_{i+1}, \ldots, a_N) \leq d^i(a_1, \ldots, a_N) \quad \forall x \in S_i. \quad (3)$$

It is known that if each of the automata in the team uses an L_{R-I} algorithm for updating action probabilities, then the team would converge to a Nash equilibrium [22].

A useful special case of this model is a game with *common payoff*. Here, all the automata get the same payoff from the environment (that is, $\beta^i = \beta$). Hence, there is only one payoff function, e.g., d. Since all automata here have only finite action-sets, we can represent d as a hyper-matrix $D = [d_{j_1 \cdots j_N}]$ of dimension $r(1) \times \cdots \times r(N)$, where

$$d_{j_1 \cdots j_N} = E[\beta(k) \,|\, \alpha^i(k) = \alpha^i_{j_i},\ 1 \leq i \leq N]. \quad (4)$$

Here, we used the notation $S_i = \{\alpha^i_1, \ldots, \alpha^i_{r(i)}\}$ is the set of actions of automaton $A_i, 1 \leq i \leq N$. D is called the reward probability matrix of the game and it is unknown to the automata. The objective for the team is to maximize the expected value of the common reinforcement β. Define $\mathcal{S} = \prod_{i=1}^N S_i$. For any $\mathbf{a} = (a_1, \ldots, a_N) \in \mathcal{S}$, we define its neighborhood in $\mathcal{S}$ as

$$\mathcal{N}(\mathbf{a}) = \{\mathbf{x} \in \mathcal{S} \,|\, \exists\, j, \text{ s.t. } x_i = a_i,\ \forall\, i \neq j, \text{ and } x_j \neq a_j\}.$$

Thus, the neighbors of any N-tuple of actions is the set of all action choices that differ only in one action.

Now it is easy to see that a specific N-tuple of actions $\mathbf{a} \in \mathcal{S}$ is a Nash equilibrium for this game with common payoff if and only if $d(\mathbf{a}) \geq d(\mathbf{x}), \forall \mathbf{x} \in \mathcal{N}(\mathbf{a})$. The corresponding element of the reward probability matrix is called a mode of the matrix. As it is easy to see, a mode would be an element that is simultaneously maximum along each of the hyper rows to which it belongs. It is known that if all the automata involved in a common payoff game use L_{R-I} (with sufficiently small value for the step size parameter), then the team would converge to a mode of the reward probability matrix [22].

Let $\mathbf{P} = (\mathbf{p}^t_1, \ldots, \mathbf{p}^t_N)^t \in \Re^{r(1)+\cdots+r(N)}$ denote the tuple of all action probabilities of all the automata in the team. Then it is easy to see that once again, the automata team is solving the optimization problem of

$$\text{maximize } f(\mathbf{P}) = E[\beta \,|\, \mathbf{P}]$$

by searching over the N-fold product of the $r(i)$-dimensional simplexes. By searching over this space, the algorithm finds an N-tuple of actions that is a mode. By our earlier explanation of the mode, the corresponding tuple of action probabilities would be a local maximum of the above problem. Thus, by using L_{R-I}, we can locate a local maximum of the expectation of reinforcement in the automata game. It is possible to converge to the global maximum (that is, the action tuple corresponding to the largest entry in the reward probability matrix) by employing the estimator algorithms [23]. The estimator algorithms also exhibit superior speed of convergence. However, these algorithms have large memory overhead, especially in cases where N is large.

Such a game model is useful for optimizing a function over N variables or parameters using only noise corrupted values of the function. The action-sets of different automata represent the possible values of different parameters. Since we are considering only FALA here, to use this model we should quantize the values of different parameters. LA in a common payoff game are often referred to as a team of LA.

B. Networks of FALA

There are many situations where it is convenient to formulate the learning problem in such a way that the optimal action depends on the *state* of the environment made available to the learning system as an input vector. This input is usually designated as the *context vector*. Examples of context vectors include feature vector in a pattern recognition problem and the state of the system in a control problem. We consider the pattern recognition problem to explain the different ways in which LA systems can tackle such problems.

One can think of the requirements of a pattern recognition system as that of outputting a class label for a feature vector that is input to the system. One way of using automata models for this problem is to think of actions of automata as possible class labels. Then, an action is optimal only in the *context* of certain feature vectors. Such problems have been termed associative reinforcement learning problems [24] because here the objective is to learn to associate different *inputs* with different actions. To tackle the problem in this way, we need to have an automata model whereby the probability of choosing an action depends also on the context vector input from the environment. Such an LA model is called a *GLA* and will be discussed in Section IV.

Another way of solving such problems is through teams of automata. We first formulate an appropriately parameterized class of discriminant functions. The learning problem is to obtain the optimal values of the parameters. If we think of actions of automata as parameter values, then we can define optimal actions without reference to any context. As was already discussed, one can use teams of FALA to learn the optimal parameters in such problems.

Since we need to ultimately find different class regions in the feature space, our parameterized class of discriminant functions should be capable of representing them. Depending on the complexity of the problem we may want to employ a number of teams of FALA.

In such cases, it would be convenient to organize the teams in an orderly manner so that there is some clarity about the role played by each team. One such organization is a feedforward network of learning units where each unit is a team of FALA involved in a common payoff game [25], [26]. A schematic of a network is shown in Fig. 1.

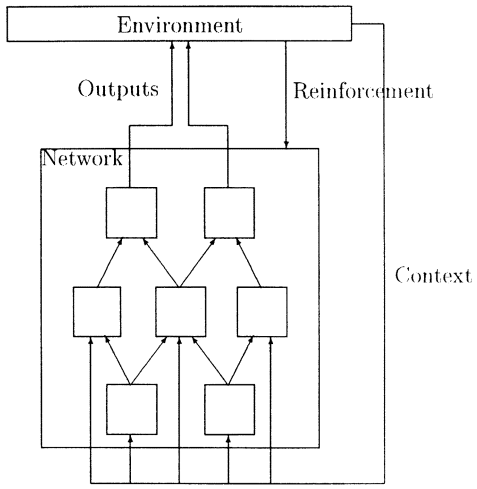

Fig. 1. Network of learning automata. Each of the boxes corresponds to a team of FALA.

The network superficially resembles a feedforward neural network, but differs considerably in operation. The input to each unit could be the context vector from the environment as well as outputs of other units appearing earlier in the network. The output of a unit is determined by the actions chosen by a group of FALA in the unit. Finally, the output of the network to the external environment is determined by the outputs of some designated units. The sequence of events in such a system is as follows.

At each instant, the environment generates a context vector which is input to the LA network. Based on its internal state (i.e., the action probability distributions of all of the FALA in the network) and the context vector, the network outputs an action. The environment then generates a reinforcement β, which indicates the appropriateness of the action for the particular context vector. This reinforcement is made available to every FALA in each unit. The network then updates its internal state so as to improve its performance. Unlike a neural network, the updating of all FALA in all units is based on the same reinforcement signal.

A team of FALA which forms a unit is able to handle the context vector input as follows. Suppose we would like the output of the unit (e.g., y) to be binary with

$$\begin{aligned} y &= 1 \qquad \text{if } G(\mathbf{x}, \mathbf{w}) \geq 0 \\ &= 0 \qquad \text{if } G(\mathbf{x}, \mathbf{w}) < 0 \end{aligned} \tag{5}$$

where G is a known function of $\mathbf{x}$, the context vector input, and $\mathbf{w}$, a parameter vector. (Choice of the function G is part of the architectural design of the network.) Then, the tuples of actions that can be chosen by the team correspond to possible values of the parameter vector $\mathbf{w}$. For every value of $\mathbf{w}$ so chosen by the team, the value of $G(\mathbf{x}, \mathbf{w})$ is used to compute y using (5). If the number of possible actions is r, then we can have functions $G_j(\mathbf{x}, \mathbf{w})$, $j = 1, \ldots, r$, and choose ith action if $G_i(\mathbf{x}, \mathbf{w}) = \max_j G_j(\mathbf{x}, \mathbf{w})$. To update the state of the network, each of the FALA in each unit update their action probabilities using the globally supplied reinforcement.

Suppose the network consists of a total of N number of FALA. (This number includes the FALA in all units of the network.) Let $\mathbf{p}_i(k)$ denote the action probability distribution of the ith FALA at instant k, $1 \leq i \leq N$. Then, $\mathbf{P}(k) = [\mathbf{p}_1^t(k), \ldots, \mathbf{p}_N^t(k)]^t$ represents the internal state of the network at k. The goal for the learning algorithm in this network is to maximize $f(\mathbf{P}) = E[\beta \,|\, \mathbf{P}]$.

It can be shown that if each FALA uses the L_{R-I} algorithm with the reinforcement that is commonly supplied to all automata, then the network converges to a local maximum of $f(\mathbf{P})$ under assumptions that are not restrictive in most applications [25], [26]. How well a local maximum of f corresponds to the desired mapping of context vectors to the best actions depends both on the complexity of the problem as well as the architecture of the network [25].

An interesting example of a network of FALA is the three-layer network extensively used in pattern recognition [26], [27]. The objective here is to divide the feature space into two regions (not necessarily connected) such that each region corresponds to a pattern class. The first layer of the network forms hyperplanes in the feature space and the second layer forms convex regions bounded by these hyperplanes using AND logic. The third layer forms the final class regions (which could be nonconvex and disconnected) by using OR logic on the output of the second-layer units. Here, the parameter vector $\mathbf{w}$ in each first-layer unit is to represent a hyperplane and thus the G function would be a linear discriminant function. The parameter vector for each second-layer unit would be a binary vector with as many components as the number of units in the first layer. Hence, in a second-layer unit all FALA would have two actions and a specific choice of actions indicates which of the hyperplanes from the first layer should be used for making the convex region. Such a feedforward network of FALA would be useful in any two-class pattern recognition problem because one can well approximate arbitrary regions of feature space by union of convex polyhedral sets [27]. It may be noted here that such a network is actually an alternate representation for the so-called oblique decision tree classifiers used extensively in pattern recognition [28].

III. Parameterized Learning Automata (PLA)

The basic limitation of a team as well as a network of automata is that with a decentralized learning algorithm such as L_{R-I}, they can converge only to a local maximum of $E[\beta \,|\, \mathbf{P}]$. In order to facilitate convergence to the global maximum, one has to change the learning algorithm.

It is known that estimator algorithms such as the pursuit algorithm lead to the global maximum, but it has a large memory overhead. Another approach would be to use an algorithm similar to simulated annealing for global optimization [29]. This would mean imposing a random perturbation while updating the action probability vector $\mathbf{p}(k)$ so that the learning process moves out of local maxima. Introducing a random term directly in the updating equations is difficult for two reasons. First, it is cumbersome to ensure that the resulting vector after the updating remains a probability vector. Second, the resulting *diffusion* would be on a manifold rather than the entire space, thus making the analysis difficult.

One way of overcoming such difficulties is to parameterize the action probabilities in terms of some real numbers and up-

date these real numbers based on the reinforcement received. Such a FALA is referred to as a *parameterized learning automaton (PLA).*

A PLA will have an internal state vector $\mathbf{u}$ of real numbers, which is not necessarily a probability vector. The probabilities of various actions are calculated, based on the value of $\mathbf{u}$, using a *probability generating function* $\overline{g}(\cdot, \cdot)$. Teams and networks of PLA can be formed along the same lines as FALA. We explain next the learning algorithm to be used with a PLA under the notation of a network of automata.

Consider the ith PLA in a network. We denote by $\mathbf{u}_i = (u_{i1}, \ldots, u_{ir(i)})^t$ the vector of real numbers that constitutes the *state* of this PLA. The probability of the ith PLA choosing the jth action p_{ij} can be generated from $\mathbf{u}_i$ as

$$\begin{aligned} p_{ij} &= \overline{g}_i(\mathbf{u}_i, j) \\ &= \frac{\exp(u_{ij})}{\sum_l \exp(u_{il})}. \end{aligned} \tag{6}$$

Now, $\mathbf{p}_i$ will be inside the probability simplex irrespective of the values taken by $\mathbf{u}_i$. Other types of probability generating functions are also possible.

The optimization problem addressed by PLA has to be slightly modified in relation to that connected with FALA as it is the $\mathbf{u}_i$ that is updated now. This problem can be stated as follows:

maximize

$$f(\mathbf{U}) = E[\beta \,|\, \mathbf{U}]$$

subject to

$$|u_{ij}| \le L, \qquad j = 1, \ldots, r(i); \quad i = 1, \ldots, N.$$

Here, $\mathbf{U} = (\mathbf{u}_1^t, \ldots, \mathbf{u}_N^t)^t \in \Re^{r(1)+\cdots+r(N)}$ represents the tuple of all state vectors of all PLA in the system. The constant $L > 0$ is introduced to avoid unbounded behavior of the learning algorithm. However, we can choose L to be sufficiently large so that there is insignificant difference between the constrained and unconstrained global maxima.

A learning algorithm for updating the state of the ith PLA, which ensures convergence to the global maximum is described below

$$\begin{aligned} u_{ij}(k+1) = u_{ij}(k) &+ \lambda\beta(k)\frac{\partial \ln \overline{g}_i}{\partial u_{ij}}(\mathbf{u}_i(k), \alpha_i(k)) \\ &+ \lambda h'(u_{ij}(k)) + \sqrt{\lambda} s_{ij}(k). \end{aligned} \tag{7}$$

In the above algorithm

- $h'(\cdot)$ is the derivative of $h(\cdot)$, which is defined as

$$\begin{aligned} h(x) &= -K(x-L)^{2J} && \text{for } x \ge L \\ &= 0 && \text{for } |x| \le L \\ &= -K(x+L)^{2J} && \text{for } x \le -L \end{aligned} \tag{8}$$

- $\{s_{ij}(k)\colon 1 \le j \le r(i),\ 1 \le i \le N,\ k = 0, 1, \ldots\}$ is a set of *iid* random variables with zero mean and variance σ^2; L and K are positive real constants; J is a positive integer; and $\lambda > 0$ is the learning parameter.

The second term on the right-hand side of (7) is a gradient term. The third term containing $h'(\cdot)$ is introduced to keep u_{ij} bounded. This is similar to the penalty term in many constrained optimization algorithms. The last term containing $s_{ij}(\cdot)$ is the random perturbation term which is responsible for the algorithm getting out of local maxima that are not global maxima. This is similar to the simulated annealing-type algorithms for global optimization over $\Re^n$ except for the fact that here we keep the variance of perturbations constant.

It can be shown that for sufficiently small values of the learning parameter λ, the behavior of the algorithm can be well approximated by the Langevin equation. More precisely, one can show that an appropriately interpolated continuous-time version of the state of the network $\mathbf{U}(t)$ converges to the solution of the stochastic differential equation of the Langevin-type given by

$$d\mathbf{U} = \nabla H(\mathbf{U}) + \sigma d\mathcal{W} \tag{9}$$

where

$$H(\mathbf{U}) = E[\beta \,|\, \mathbf{U}] + \sum_{i,j} h(u_{ij}) \tag{10}$$

and $\mathcal{W}$ is the standard Brownian motion process of appropriate dimension. It is known that the solutions of the Langevin equation concentrate on the global maximum of H as σ tends to zero. Because of our choice of the function $h(\cdot)$, this means that (for sufficiently small values of λ and σ) the algorithm would converge to a state that is the global maximum of the expected reinforcement [that is, the function $f(\mathbf{U})$] if the global maximum state vector is such that each component is less than L. Otherwise, the algorithm would find a *constrained* global maximum of $f(\cdot)$ inside the bounded region allowed for the algorithm [30].

IV. Generalized Learning Automata (GLA)

As mentioned in Section II-B, one method of handling associative reinforcement learning problems is to use a GLA, where the structure of LA is modified to allow for context vector input.

A single GLA is described by the tuple $\langle X, Y, R, \mathbf{u}, g, T\rangle$. Here, X is the set of all context vectors that can be input to the GLA; Y is the (finite) set of outputs or actions of GLA; R is the set of values that the reinforcement signal can take (which is usually taken to be the interval $[0, 1]$); g is the probability generating function; and $\mathbf{u}$ is the internal state which is a vector of real numbers. T is the learning algorithm that updates $\mathbf{u}$.

Let the action-set of the GLA be $Y = \{y_1, \ldots, y_r\}$. The action probabilities of a GLA are generated by

$$\text{Prob}[\alpha(k) = y_i \,|\, \mathbf{u}, \mathbf{x}] = g(\mathbf{x}, y_i, \mathbf{u}) \tag{11}$$

where the function g satisfies $g(\mathbf{x}, y_i, \mathbf{u}) \ge 0$, $\forall y_i, \mathbf{u}, \mathbf{x}$, and $\sum_{i=1}^r g(\mathbf{x}, y_i, \mathbf{u}) = 1\ \forall \mathbf{u}, \mathbf{x}$. At each instant k, the learning algorithm T updates $\mathbf{u}(k)$ based on the current values of $\mathbf{x}(k)$, $\mathbf{u}(k)$, the reinforcement signal $\beta(k)$, and the action chosen by the GLA $\alpha(k)$. Dependence of the updating on the context vector $\mathbf{x}(k)$ is the main characteristic of GLA.

The motivation for defining GLA is to be able to tackle associative reinforcement learning problems directly. Hence, with the same state vector $\mathbf{u}$, the probabilities with which a GLA chooses different actions can (and most often, would) be dependent on the context vector. This is why the probability gener-

ating function of the GLA is dependent on both the state $\mathbf{u}$ and the context vector $\mathbf{x}$. This may be contrasted with that of PLA, where the probability generating function is dependent only on $\mathbf{u}$. In a PLA, the state $\mathbf{u}$ is only a *representation* for the action probability distribution. In a GLA, the state $\mathbf{u}$ (along with the function g), is a *representation* for a mapping from set of context vectors to set of action probability distributions.

A simple example of a probability generating function for a GLA is as follows. Suppose the context vector $\mathbf{x}$ belongs to $\Re^n$. We choose the internal state $\mathbf{u}$ to also be n-dimensional. Suppose there are only two actions so that $Y = \{y_1, y_2\}$. Then a probability generating function for the GLA could be

$$g(\mathbf{x}, y_1, \mathbf{u}) = 1 - g(\mathbf{x}, y_2, \mathbf{u}) = \frac{1}{1+\exp(-\mathbf{x}^t\mathbf{u})}.$$

It is easy to see that learning an "optimal" $\mathbf{u}$ with such a GLA is like learning an "optimal" linear discriminant function in a two-class pattern recognition problem.

In general, the goal for a GLA is to learn the desired mapping from context vectors to actions. Since the probability generating function is fixed, chosen by us, and since the environment is unknown, it is not possible to know whether there exists a vector $\mathbf{u}$ so that the desired mapping can be represented by a GLA. Hence, to be able to prove that a GLA learns properly in this fashion, we may have to make very stringent assumptions on the problem [24]. A better choice for the objective of learning is to maximize $f(\mathbf{u}) = E[\beta \,|\, \mathbf{u}]$. This would ensure learning of the best mapping that is allowed after having chosen a probability generating function [25].

From the example of a probability generating function given earlier, it is easy to see that a single GLA is like a team of automata. A network of GLA can be formed just like a network of FALA with the change that each unit (which was a team of FALA) now corresponds to one GLA. The context vector input for each GLA in the network could consist of the external context vector from the environment or the outputs (actions) of other GLA connected to this GLA or a combination of both.

Consider the ith GLA in the network. Its state vector is $\mathbf{u}_i = (u_{i1}, \ldots, u_{ir(i)})^t$. The objective of the network is to maximize $E[\beta \,|\, \mathbf{U}]$, where $\mathbf{U} = (\mathbf{u}_1^t, \ldots, \mathbf{u}_N^t)^t$, subject to boundedness of the components of $\mathbf{U}$. A learning algorithm which ensures the boundedness of solutions is as follows. (This is a modification of the REINFORCE algorithm [32]). Let $\mathbf{u}_i(k)$ denote the state of ith GLA at k; $y_i(k)$ its output at k; and $\mathbf{x}_i(k)$ its context input at k. Let $\beta(k)$ be the reinforcement obtained by the network from the environment at instant k

$$u_{ij}(k+1) = u_{ij}(k) + \lambda\beta(k)\frac{\partial \ln g_i}{\partial u_{ij}}(\mathbf{x}_i(k), y_i(k), \underline{h}(\mathbf{u}_i(k))) + \lambda K[h(u_{ij}(k)) - u_{ij}(k)] \quad (12)$$

where

$$\underline{h}(\mathbf{u}_i) = \left[h(u_{i1}), \ldots, h\left(u_{ir(i)}\right)\right]^t$$

with

$$\begin{aligned} h(\eta) &= L && \text{for } \eta \geq L \\ &= \eta && \text{for } |\eta| \leq L \\ &= -L && \text{for } \eta \leq -L \end{aligned}$$

where

g_i	probability generating function of the ith GLA;
$\lambda > 0$	learning parameter;
$L, K > 0$	constants.

Analysis of the network of GLA can be carried out using weak convergence techniques [25], [33]. It can be shown that under some broad assumptions, the network of GLA using the above algorithm converges to a constrained local maximum of $E[\beta \,|\, \mathbf{U}]$.

The network can be made to converge to the global maximum of the expected reinforcement function by introducing perturbation terms in the updating which are similar to those with a PLA. This global algorithm is as follows:

$$u_{ij}(k+1) = u_{ij}(k) + \lambda\beta(k)\frac{\partial \ln g_i}{\partial u_{ij}}(\mathbf{x}_i(k), y_i(k), \mathbf{u}_i(k)) + \lambda h'(u_{ij}(k)) + \sqrt{\lambda}s_{ij}(k) \quad (13)$$

where

h'	derivative of the function h defined by (8);
$\lambda > 0$	learning parameter;
$s_{ij}(k)$	sequence of *iid* random variables with zero mean and variance σ^2.

The analysis of the global algorithm is similar to that of the global algorithm for PLA via approximation using the Langevin equation.

As mentioned earlier, a single GLA is essentially like a team of FALA. However, there are interesting applications where a GLA is the more natural choice. For example, a GLA can be used as a gating network in models such as adaptive mixture of local experts [31]. Here, each input (context vector) from the environment would be sent to all the experts and the final output from the system is determined by the gating network which decides how to combine the outputs of different expert networks. Such models are useful for learning mixture densities or for learning a pattern classification system in which different (types of) classifiers are to be used in different regions of the feature space. In such problems, a team of FALA does not appear suitable for the task of a gating network.

V. Continuous Action-Set Learning Automata (CALA)

So far, we have considered the LA model where the set of actions is finite. Hence, while finding the optimal parameter values to maximize a performance index, we need to discretize the parameter space so that actions of LA can be possible values of parameters. A more satisfying solution would be to employ an LA model where the action-set can be continuous. Such a model is called *continuous action-set learning automaton or CALA.*

The action-set of CALA is the real line. The action probability distribution at k is $N(\mu(k), \sigma(k))$, the normal distribution with mean $\mu(k)$ and standard deviation $\sigma(k)$. At each instant, the CALA updates its action probability distribution by updating $\mu(k)$ and $\sigma(k)$. As before, let $\alpha(k) \in \Re$ be the action chosen at k and let $\beta(k)$ be the reinforcement at k. Here, instead of reward probabilities for various actions, we now have a reward function f: $\Re \rightarrow \Re$ defined by $f(x) = E[\beta(k) \,|\, \alpha(k) = x]$. We shall denote the reinforcement in response to action x as β_x and thus $f(x) = E\beta_x$.

The objective for CALA is to learn the value of x at which f attains a maximum. That is, we want the action probability distribution $N(\mu(k), \sigma(k))$ to converge to $N(x_o, 0)$ where x_o is a maximum of f. However, we do not let $\sigma(k)$ converge to zero to ensure that the algorithm does not get stuck at a nonoptimal point. Therefore, we use another parameter, $\sigma_\ell > 0$ (with σ_ℓ sufficiently small) and keep the objective of learning as $\sigma(k)$ converging to σ_ℓ and $\mu(k)$ converging to a maximum of f.

The learning algorithm for CALA is described next. Since the updating given for $\sigma(k)$ does not automatically guarantee that $\sigma(k) > \sigma_\ell$, we always use a projected version of $\sigma(k)$, denoted by $\phi(\sigma(k))$, while choosing actions. Furthermore, unlike FALA, CALA interacts with the environment through a choice of two actions at each instant.

At each instant k, CALA chooses a $x(k) \in \Re$ at random from its current action probability distribution $N(\mu(k), \phi(\sigma(k)))$, where ϕ is the function specified below. Then, it gets the reinforcement from the environment for the two actions: $\mu(k)$ and $x(k)$. Let these reinforcements be β_μ and β_x. Then, the distribution is updated as follows:

$$\begin{aligned}\mu(k+1) &= \mu(k) + \lambda \frac{(\beta_x - \beta_\mu)}{\phi(\sigma(k))} \frac{(x(k) - \mu(k))}{\phi(\sigma(k))} \\ \sigma(k+1) &= \sigma(k) + \lambda \frac{(\beta_x - \beta_\mu)}{\phi(\sigma(k))} \left[\left(\frac{(x(k) - \mu(k))}{\phi(\sigma(k))} \right)^2 - 1 \right] \\ &\quad + \lambda \{ C[\sigma_\ell - \sigma(k)] \} \end{aligned} \tag{14}$$

where

$$\begin{aligned}\phi(\sigma) &= \sigma_\ell \qquad \text{for } \sigma \le \sigma_\ell \\ &= \sigma \qquad \text{for } \sigma > \sigma_\ell \end{aligned} \tag{15}$$

and

λ step size parameter for learning ($0 < \lambda < 1$);
C large positive constant;
σ_ℓ lower bound on standard deviation, as explained earlier.

The CALA algorithm can be used as an optimization technique without discretizing the parameter space. It is similar to stochastic approximation algorithms [34] though here the randomness in choosing the next parameter value makes the algorithm explore better search directions. Furthemore, unlike the classical stochastic approximation algorithms, here we do not explicitly estimate the gradient. For this algorithm, it is proven that with arbitrarily large probability, $\mu(k)$ will converge close to a maximum of $f(\cdot)$ and $\phi(\sigma(k))$ will converge close to σ_ℓ, if we choose λ and σ_ℓ sufficiently small [35], [36].

As in the case of FALA, we can consider, for example, a common payoff game played by N number of CALA. Since the action-set of each automaton is the real line, the payoff function would have domain $\Re^N$. It can be shown that if each CALA in the team uses the algorithm described above (with sufficiently small values for the learning parameters), then the team would converge to a local maximum (in the standard Euclidean sense) of the payoff function [35]. Such a team model would be useful for optimizing a function of N variables using only noise corrupted values of the function and without needing (or explicitly estimating) any gradient information [37]. Unlike the case with FALA teams, we do not need to discretize the parameter space.

Another interesting model is a game with N number of FALA and M number of CALA. Now, the payoff function is defined over $N + M$ variables out of which M are discrete and N are continuous. We can define optimal points of the game to be those which are like Nash equilibria with respect to the discrete part and local maxima (in the Euclidean sense) with respect to the continuous part. It can be shown that if each of the FALA uses L_{R-I} algorithm and each of the CALA uses the algorithm given earlier in this section, then the team would converge to one of the optimal points [38]. Such optimization problems, where the objective function is defined over some discrete and some continuous variables, are useful in applications such as learning concepts in the form of logic expressions [38], [39].

It is possible to conceive of networks of CALA along the same lines as networks of FALA. However, at present, there are no convergence results available for networks of CALA.

VI. Modules of Learning Automata

A decisive aspect of any learning system is its rate of learning or equivalently, speed of convergence. It is decisive because most learning systems operate in slowly changing environments and the learning process should be completed well before significant changes take place in the environment; otherwise learning is ineffective. In the case of LA, speed of convergence can be increased by increasing the value of the learning parameter. However, this results in reduced accuracy in terms of probability of convergence to the correct action. The problem therefore is to increase speed without reducing accuracy. This is not possible in the models considered so far because a single parameter controls both speed and accuracy.

Parallel operation is known to increase the speed of convergence in general. In order to conceive of parallel operation of LA, one has to change the sequential nature of the models considered earlier. If we have a number of LA acting in parallel, each of these automata generates its own action and gets a corresponding reinforcement signal from the environment simultaneously.[2]

The basic idea in such a parallel operation of LA is that, since the environmental responses are stochastic, updating of action probabilities based on several responses would have less expected error than that based on a single response and would facilitate faster convergence.

Consider n FALA operating in parallel in the place of a single LA. These LA could be said to form a *module*. The schematic of such a module is shown in Fig. 2. The main features of the module are the following. The action-set is $\{\alpha_1, \ldots, \alpha_r\}$. The action probability vector $\mathbf{p}(k)$ is common to all the n automata of the module. Each LA (e.g., the ith) in the module selects an action $(\alpha^i(k))$ based on the common action probability vector

[2]Such a model would be useful in applications such as pattern recognition where actions of automata are possible parameter values and one can test several parameter vectors simultaneously using the next available example. On the other hand, in applications such as routing in communication networks where the action corresponds to choice of a route, simultaneous choice of several actions is not possible unless one is working with a simulation model of the system. Hence, in general, whether or not parallel operation is feasible depends on the application.

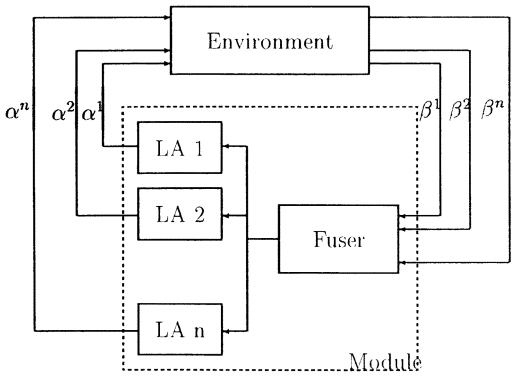

Fig. 2. Module of learning automata.

and obtains its reinforcement signal $(\beta^i(k))$. The common action probability vector is updated depending on all the actions selected and the reinforcements obtained. A *fuser* combines all this information into quantities that are used in probability updating. It is assumed that $\beta^i(k) \in [0, 1], \forall\, i,\ k$. The fuser computes the following quantities at each instant.

- The total response to α^i at k is

$$q_i(k) = \sum_{j=1}^{n} \beta^j(k) I\{\alpha^j(k) = \alpha_i\}$$

  where $I\{A\}$ is the indicator function of event A.
- The total response at instant k is

$$q(k) = \sum_{j=1}^{n} \beta^j(k) = \sum_{i=1}^{r} q_i(k).$$

The learning algorithm is given by

$$p_i(k+1) = p_i(k) + \tilde{\lambda}(q_i(k) - q(k)p_i(k)) \qquad i = 1, \ldots, r \tag{16}$$

where $\lambda \in (0, 1]$ is the learning parameter, and $\tilde{\lambda} = \lambda/n$ is called its *normalized value.*

To get an intuitive feel for the updating, we note that the quantity $q_i(k)/q(k)$ can be considered as a figure of merit for the action α_i. The algorithm moves $p_i(k)$ toward $q_i(k)/q(k)$. Computationally, $\mathbf{p}(k)$ is updated only once and not necessarily by each LA. The updated value $\mathbf{p}(k+1)$ is shared by all the LA in the module for their next action selection. It can be shown that the algorithm is ϵ-optimal in all stationary random environments [40].

In this algorithm, the accuracy of the learning algorithm is controlled by $\tilde{\lambda} = \lambda/n$. The speed of convergence is controlled by λ. Hence, for the required accuracy, n and λ can be varied to control the speed. Higher speed of convergence for a given accuracy needs higher value of n, that is, more members in the module. However, since we need $\lambda \leq 1$, there is a natural limitation on the achievable speed-up.

Simulation results indicate that the speed of convergence increases almost linearly with n, the size of the module [40]. Other LA algorithms such as the pursuit algorithm can also be parallelized in a similar fashion. In fact, in the case of pursuit algorithm, one can achieve convergence in a single step for a sufficiently large size of the module [41].

The common payoff game of a team of FALA also has a parallel version where each LA is replaced by a module of n automata. Thus, there are n parallel teams here with the ith team being formed by the ith LA of each module. Using the above algorithm for each member of each of the parallel teams, we can ensure convergence to local optima of the payoff function while increasing rate of convergence. The same idea can be extended to feedforward networks of units where each unit is a team of automata. Similar properties of increased rate of convergence without sacrificing accuracy follow for such parallel networks. Modules of other types of LA, such as PLA, GLA, and CALA, can also be formed along similar lines [41].

VII. Pattern Classification Using LA Models

In this section, we use the pattern classification problem as an example to illustrate how the different LA models can be employed in an application. We consider learning the "optimal" discriminant function (from a given parameterized class of discriminant functions) in a two-class pattern recognition problem. We denote the feature vector by $\mathbf{x}$ and the parameter vector (of any discriminant function) by $\mathbf{w}$.

We first need to decide on a parameterized family of discriminant functions. Then, it is straightforward to use a team of N automata to learn the optimal values of N parameters. To use a FALA team, we have to quantize the range of each of the parameters. Then, the action-set of each FALA would be the (finite) set of all possible values for the corresponding parameter. A choice of action-tuple by the team results in the choice of a specific classifier. We classify the next training pattern with this classifier and supply a "1" or "0" as common reinforcement to all FALA, based on whether or not this classification agrees with that given in the training sample. From the results given in Section II-A, it is clear that, if we use the L_{R-I} algorithm, we converge to a local optimum. We can alternatively use the PLA model and the corresponding global algorithm to converge to the global optimum. We can use a team of CALA instead of a team of FALA if we do not want to quantize the parameters. In all these cases, it is easily shown that we can tolerate up to 50% noise in the classification of the training samples [23].

In the above, the learning algorithm does not impose any restrictions on the discriminant function. Thus, the method can easily handle discriminant functions that are nonliner both in $\mathbf{x}$ and $\mathbf{w}$. For example, the classifier chosen can be a feedforward neural network and the CALA team algorithm then would be an alternative to the standard backpropagation, as well as being able to handle noisy samples.

In general, we can choose for our discriminant function an arbitrary logic expression, the literals of which are algebraic inequalities (involving feature vector components and some parameters). This can be handled by a network of FALA. For example, we can employ a three-layer network of FALA (which is briefly described in Section II-B) and then we can learn a

TABLE I
SIMULATION RESULTS FOR IRIS DATA. THE ENTRY IN THE FOURTH COLUMN REFERS TO RMS ERROR FOR BPM AND PROBABILITY OF MISCLASSIFICATION FOR L_{R-I}. (NC STANDS FOR NO CONVERGENCE)

Algorithm	Structure	% Noise	Error	Steps
BPM	9 3 1	0	2.0	66,600
BPM	9 3 1	20	–	NC
BPM	9 3 1	40	–	NC
BPM	8 8 8 1	0	2.0	65,800
BPM	8 8 8 1	20	–	NC
BPM	8 8 8 1	40	–	NC
L_{R-I}	9 3 1	0	0.1	78,000
L_{R-I}	9 3 1	20	0.1	143,000
L_{R-I}	9 3 1	40	0.15	200,000

logic expression, the literals of which are linear inequalities. We can also handle it by a network of GLA. In both of the network models, we can use the global algorithms so as to converge to the global optima. Finally, in any of these models, we can replace a single automaton by a module of automata to improve the speed of convergence without sacrificing the accuracy. A more detailed account of LA models for pattern classification can be found in [42].

In the remaining part of this section, we describe some simulation results obtained with the network of LA model on a pattern recognition problem to get a flavor of the type of possible results.

Example: In this example, a two-class version of the iris data is considered. The data was obtained from the machine learning databases maintained at the University of California at Irvine. This is a three-class, four-feature problem. The three classes are: 1) iris-setosa; 2) iris-versicolor; and 3) iris-virginica. Of these, setosa is linearly separable from the other two. Since we are considering only two-class problems here, setosa was ignored and the problem was reduced to that of classifying versicolor and virginica. The data used was 50 samples of each class with the correct classification.

The network consisted of nine first-layer units and three second-layer units. Each first-layer unit has five automata (since this is a four-feature problem). Each automaton had nine actions which were $\{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$. Uniform initial conditions were used. The learning parameters were 0.005 in the first layer and 0.002 in the second layer.

For a comparison of the performance achieved by the automata network, a standard feedforward neural network trained using backpropagation with momentum term (BPM) is considered. The network has four input nodes and one output node. The different network architectures tried were: two hidden layers with nine and three nodes, and three hidden layers with eight nodes each. Initial weights for the network were generated randomly. The step size for the momentum term was set at 0.9 and the results reported are for the best choice of step size for gradient term.

Simulations were conducted for perfect data (0% noise) and noisy cases. Noise was introduced by changing the known classification of the feature vector at each instant by a fixed probability. Noise levels of 20% and 40% were considered.

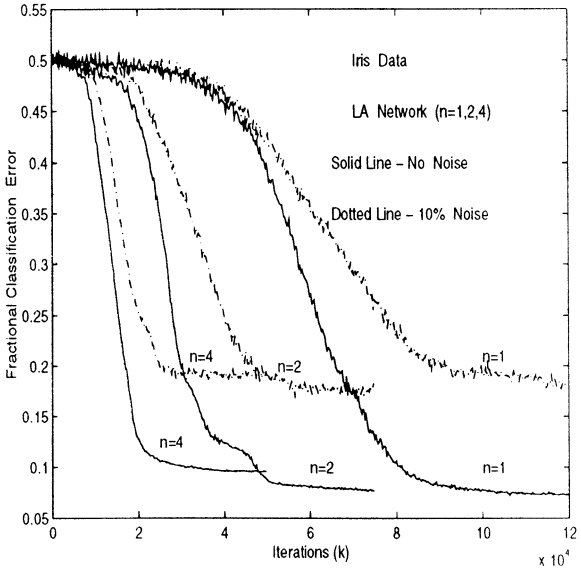

Fig. 3. Learning curves for classification of iris data using modules of LA in a three-layer network.

The results obtained are summarized in Table I. These are averages over ten runs. The error reported in the table for the backpropagation algorithm is the rms error, while that for the automata network is the probability of misclassification on a test set. While they cannot be directly compared, the performance was about the same at the values reported.

The results show that in the noise-free case, the BPM converges about 20% faster. However, this algorithm fails to converge even when only 20% noise is added. The LA network continues to converge even with 40% noise and there is only slight degradation of performance with noise [26].

To illustrate the improvement in speed of convergence through the use of module of automata, we consider the iris data problem with same three-layer network structure, but with each automaton replaced by a module. The results (which are averages over ten runs) are shown in Fig. 3. This figure clearly establishes the faster speed of convergence for the larger module sizes [40].

VIII. APPLICATIONS

In this section, we present a brief overview of applications of LA models in a variety of areas. As explained in Section I, most learning problems can be thought of as optimization of the expectation of a random function where the underlying probability distributions are unknown. LA models prove to be quite useful for handling many such stochastic optimization problems [43]. One example of such optimization is the problem of learning discriminant functions for pattern recognition, as was discussed in Section VII. Another generic problem where LA models would be useful is one that involves adaptive decision making where one needs to choose an action (among several alternatives) on-line to optimize system performance, but without complete knowledge of how actions affect performance. An early example of such an application is that of using LA for making routing decisions in a telephone network. A good survey

of many of the earlier applications in areas such as routing and process control are available in [12] and [13]. A detailed review of recent applications of LA models is not feasible here due to limitations of space. However, we provide pointers to some of the recent applications. We would like to emphasize that this list is only indicative and is, by no means, exhaustive.

As illustrated briefly in Section VII, automata algorithms have been used for learning rich classes of pattern classifiers [42]. Due to the fact that the action-sets of automata need not have any algebraic structure on them, similar algorithms are seen to be useful for concept learning [38], [39]. LA models have also been used successfully for many problems involving adaptive decision making in communication systems. Examples include bus arbitration schemes for ATM switches [44], conflict avoidance in star networks [45], and dynamic channel allocation [46]. Another large application area where LA models are found to be useful is adaptive control and signal processing [47]–[51]. Even in problems with no random components, automata algorithms can prove to be useful as stochastic search techniques; an example being the problem of graph partitioning [52]. Some of the other areas in which LA are useful include image processing [53]–[55], intelligent vehicle control [56], pruning decision trees [28], object partitioning [57], string taxonomy [58], and learning rule bases of fuzzy systems [59]. LA models provide a fairly general purpose technique for adaptive decision making, optimization, and control. Many of the recent developments in LA models (some of which are outlined in this paper), along with some recent techniques for implementing these algorithms in an efficient manner [60], should provide further impetus for employing LA in many different applications.

IX. Conclusions

A variety of LA models have been discussed in this paper. We have indicated how collectives of such LA can be configured into structures such as teams and networks in order to handle different applications.

The general objective of a learning system built out of LA is to maximize the reinforcement received from the environment. As we have indicated, almost all problems of learning from examples involve such optimization. A common theme in all LA models is that updating is done over probability distributions defined over the action space. Even in situations where the action space directly corresponds to the parameter space (e.g., a CALA team), it is not necessary to resort to direct updating in the parameter space. Such an approach, though it may appear indirect, has the following advantages.

When the action-set is finite, as in FALA, PLA, and GLA, there is a great deal of freedom in choosing the actions. This provides a lot of flexibility in designing an appropriate learning system in different applications. For example, in a three-layer network of FALA, actions of some automata correspond to real-valued parameters that represent hyperplanes, while actions of some other automata represent logical decisions of which hyperplanes to pick for making appropriate convex regions. However, the learning algorithm itself is completely independent of what the actions represent. Similarly, using a team containing both CALA and FALA, we can optimize a function defined over some discrete and some continuous parameters.

Even when the action-set is the real line (as in CALA), the randomness in choosing the parameter values enables exploration of better search directions, as demonstrated empirically (e.g., [35]).

Working in the probability space rather than the parameter space appears to give these learning models better noise tolerance. LA models have converged in situations where classical schemes have failed (e.g., the results presented on the iris data with noisy samples).

The models and algorithms described in this paper provide a unifying view of LA techniques as a general approach for on-line stochastic optimization. While a variety of models are described here, choice of the type of LA is dictated by the application. These models are not mutually exclusive in the sense that, in many cases, there would be more than one model that could be used. In general, as with any other suite of learning techniques, the choice is largely governed by ease of representing the solutions needed or of the configuration of the system. Further research is needed in clarifying the strengths and weaknesses of different varieties of LA and their connection with related ideas such as soft computing and evolutionary algorithms.

References

[1] K. S. Narendra and M. A. L. Thathachar, "Learning automata: A survey," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-14, pp. 323–334, 1974.
[2] M. L. Tsetlin, "On the behavior of finite automata in random media," *Autom. Remote Control*, vol. 22, pp. 1210–1219, 1962.
[3] M. L. Tsetlin, *Automata Theory and Modeling of Biological Systems*. New York: Academic, 1973.
[4] K. S. Fu and G. J. McMurtry, "A study of stochastic automata as a model for learning and adaptive controllers," *IEEE Trans. Automat. Contr.*, vol. AC-11, pp. 379–387, 1966.
[5] B. Chandrasekaran and D. W. C. Shen, "On expediency and convergence in variable-structure automata," *IEEE Trans. Syst., Sci., Cybern.*, vol. SSC-4, pp. 52–60, 1968.
[6] K. S. Fu, "Learning control systems—Review and outlook," *IEEE Trans. Automat. Contr.*, vol. AC-15, pp. 210–221, 1970.
[7] V. I. Varshavskii and I. P. Vorontsova, "On the behavior of stochastic automata with a variable structure," *Autom. Remote Contr.*, vol. 24, pp. 327–333, 1963.
[8] R. Viswanathan, "Learning automaton: Models and applications," Ph.D. dissertation, Yale Univ., New Haven, CT, 1972.
[9] Ya. Z. Tsypkin and A. S. Poznyak, "Finite learning automata," *Eng. Cybern.*, vol. 10, pp. 478–490, 1972.
[10] K. S. Narendra and S. Lakshmivarahan, "Learning automata: A critique," *J. Cybern. Inf. Sci.*, vol. 1, pp. 53–71, 1977.
[11] S. Lakshmivarahan, *Learning Algorithms: Theory and Applications*. New York: Springer-Verlag, 1981.
[12] K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction*. Englewood Cliffs, NJ: Prentice-Hall, 1989.
[13] K. Najim and A. S. Poznyak, *Learning Automata: Theory and Applications*. New York: Pergamon, 1994.
[14] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.
[15] Ya. Z. Tsypkin, *Adaptation and Learning in Automatic Systems*. New York: Academic, 1971.
[16] M. A. L. Thathachar and P. S. Sastry, "A new approach to designing reinforcement schemes for learning automata," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-15, pp. 168–175, 1985.
[17] ——, "Estimator algorithms for learning automata," in *Proc. Platinum Jubilee Conference Systems and Signal Processing*. Bangalore, India: Dept. Elect. Eng., Indian Inst. Science, Dec. 1986.

[18] B. J. Oommen, "Absorbing and ergodic discretized two-action learning automata," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-16, pp. 282–296, 1986.
[19] J. K. Lanctot and B. J. Oommen, "Discretized estimator learning automata," *IEEE Trans. Syst., Man, Cybern.*, vol. 22, pp. 1473–1483, 1992.
[20] G. I. Papadimitriou, "A new approach to the design of reinforcement schemes for learning automata: Stochastic estimator learning algorithms," *IEEE Trans. Knowl. Data Eng.*, vol. 6, pp. 649–654, 1994.
[21] B. J. Oommen and M. Agache, "Continuous and discretized pursuit learning schemes: Various algorithms and their comparison," *IEEE Trans. Syst., Man, Cybern. B*, vol. 31, pp. 277–287, June 2001.
[22] P. S. Sastry, V. V. Phansalkar, and M. A. L. Thathachar, "Decentralised learning of Nash equilibria in multiperson stochastic games with incomplete information," *IEEE Trans. Syst., Man, Cybern.*, vol. 24, pp. 769–777, May 1994.
[23] M. A. L. Thathachar and P. S. Sastry, "Learning optimal discriminant functions through a cooperative game of automata," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-17, pp. 73–85, Jan. 1987.
[24] A. G. Barto and P. Anandan, "Pattern-recognizing stochastic learning automata," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC–15, pp. 360–374, May 1985.
[25] V. V. Phansalkar, "Learning automata algorithms for connectionist systems—Local and global convergence," Ph.D. dissertation, Dept. Electr. Eng., Indian Inst. Sci., Bangalore, India, 1991.
[26] M. A. L. Thathachar and V. V. Phansalkar, "Convergence of teams and hierarchies of learning automata in connectionist systems," *IEEE Trans. Syst., Man, Cybern.*, vol. 25, pp. 1459–1469, Nov. 1995.
[27] R. P. Lippman, "An introduction to computing with neural nets," *IEEE Acoust., Speech, Signal Processing Mag.*, pp. 4–22, 1987.
[28] S. Shah and P. S. Sastry, "New algorithms for learning and pruning oblique decision trees," *IEEE Trans. Syst., Man, Cybern. C*, vol. 29, pp. 494–505, Nov. 1999.
[29] T. Chiang, C. Hwang, and S. Sheu, "Diffusion for global optimization in $\Re^n$," *SIAM J. Control Optim.*, vol. 25, no. 3, pp. 737–753, 1987.
[30] M. A. L. Thathachar and V. V. Phansalkar, "Learning the global maximum with parameterised learning automata," *IEEE Trans. Neural Networks*, vol. 6, pp. 398–406, Mar. 1995.
[31] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixture of local experts," *Neural Comput.*, vol. 3, pp. 79–87, 1991.
[32] R. J. Williams, "Simple statistical gradient following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, pp. 229–256, 1992.
[33] V. V. Phansalkar and M. A. L. Thathachar, "Local and global algorithms for generalized learning automata," *Neural Comput.*, vol. 7, pp. 950–973, 1995.
[34] H. Y. Kushner and G. G. Yin, *Stochastic Approximation Algorithms and Applications*. New York: Springer-Verlag, 1997.
[35] G. Santharam, "Distributed learning with connectionist models for optimization and control," Ph.D. dissertation, Dept. Electr. Eng., Indian Inst. Sci., Bangalore, India, May 1994.
[36] G. Santharam, P. S. Sastry, and M. A. L. Thathachar, "Continuous action-set learning automata for stochastic optimization," *J. Franklin Inst.*, vol. 331, pp. 607–628, 1994.
[37] P. S. Sastry and M. A. L. Thathachar, "Learning automata algorithms for pattern classification," *Sadhana*, vol. 24, pp. 261–292, 1999.
[38] K. Rajaraman and P. S. Sastry, "Stochastic optimization over continuous and discrete variables with applications to concept learning under noise," *IEEE Trans. Syst., Man, Cybern. A*, vol. 29, pp. 542–553, 1999.
[39] ——, "A parallel stochastic algorithm for learning logic expressions under noise," *J. Ind. Inst. Sci.*, vol. 77, pp. 15–45, 1997.
[40] M. A. L. Thathachar and M. T. Arvind, "Parallel algorithms for modules of learning automata," *IEEE Trans. Syst., Man, Cybern. B*, vol. 28, pp. 24–33, Feb. 1998.
[41] M. T. Arvind, "Stochastic learning algorithms with improved speed performance," Ph.D. dissertation, Dept. Electr. Eng., Indian Inst. Sci., Bangalore, India, 1996.
[42] M. A. L. Thathachar and P. S. Sastry, "Adaptive stochastic algorithms for pattern classification," in *Pattern Recognition: From Classical to Modern Approaches*, S. K. Pal and A. Pal, Eds. Singapore: World Scientific, 2001, pp. 67–113.
[43] A. S. Poznyak and K. Najim, *Learning Automata and Stochastic Optimization*. New York: Springer-Verlag, 1997.
[44] M. S. Obaidat, G. I. Papadimitriou, and A. S. Pomportsis, "An efficient bus arbitration scheme for scalable shared-medium ATM switch," *Comput. Commun.*, vol. 24, pp. 790–797, 2001.
[45] G. I. Papadimitriou and D. G. Maritas, "'Learning automata-based receiver conflict avoidance algorithms for WDM broadcast-and-select star networks," *IEEE/ACM Trans Networking*, vol. 4, pp. 407–412, June 1996.
[46] G. I. Papadimitriou and A. M. Pomportsis, "On the use of stochastic estimator learning automata for dynamic channel allocation in broadcast networks," in *Proc. IEEE/CEC*, San Diego, CA, July 2000.
[47] C. K. K. Tang and P. Mars, "Games of stochastic learning automata and adaptive signal processing," *IEEE Trans. Syst., Man, Cybern.*, vol. 23, pp. 851–856, 1993.
[48] Q. H. Wu, "Learning coordinated control of power systems using interconnected learning automata," *Int. J. Electr. Power Energy Syst.*, vol. 17, pp. 91–99, 1995.
[49] A. S. Poznyak, K. Najim, and E. Ikone, "Adaptive selection of the optimal order of linear regression models using learning automata," *Int. J. Syst. Sci.*, vol. 27, pp. 151–159, 1996.
[50] X. Zeng, J. Zhou, and C. Vasseur, "A strategy for controlling nonlinear systems using a learning automaton," *Automatica*, vol. 36, pp. 1517–1524, 2000.
[51] T. P. Imthias Ahamed, P. S. Nagendra Rao, and P. S. Sastry, "A reinforcement learning approach to automatic generation control," *Int. J. Electr. Power Syst. Res.*, vol. 63, pp. 9–26, 2002.
[52] B. J. Oommen and T. De St. Croix, "Graph partitioning using learning automata," *IEEE Trans. Comput.*, vol. 45, pp. 195–208, 1995.
[53] S. Sarkar and S. Chavali, "Modeling parameter space behavior of vision systems using Bayesian networks," *Comput. Vis. Image Understand.*, vol. 79, pp. 185–223, 2000.
[54] S. Sarkar and P. Soundararajan, "Supervised learning of large perceptual organization: Graph spectral partitioning and learning automata," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22, pp. 504–525, 2000.
[55] M. A. L. Thathachar and P. S. Sastry, "Relaxation labeling with learning automata," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-8, pp. 256–268, 1986.
[56] C. Unsal, P. Kachroo, and J. S. Bay, "Multiple stochastic learning automata for vehicle path control in an automated highway system," *IEEE Trans. Syst., Man, Cybern. A*, vol. 29, pp. 120–128, Jan. 1999.
[57] B. J. Oommen and D. C. Y. Ma, "Stochastic automata solutions to the object partitioning problem," *Comput. J.*, vol. 35, pp. A105–A120, 1992.
[58] B. J. Oommen and T. De St. Croix, "String taxonomy using learning automata," *IEEE Trans. Syst., Man, Cybern. B*, vol. 27, pp. 354–365, Apr. 1997.
[59] P. Viswanath, "Stability and rule generation in fuzzy systems," Masters thesis, Dept. Comput. Sci. Autom., Indian Inst. Sci., Bangalore, India, Jan. 1995.
[60] M. S. Obaidat, G. I. Papadimitriou, and A. S. Pomportsis, "Fast learning automata for high-speed real-time applications," in *Proc. 7th IEEE Conf. Electronics, Circuits, and Systems*, Kaslik, Lebanon, Dec. 2000.

M. A. L. Thathachar (SM'79–F'91) received the B.E. degree in electrical engineering from the University of Mysore, Mysore, India, in 1959, and the M.E. degree in power engineering and the Ph.D. degree in control systems from the Indian Institute of Science, Bangalore, in 1961 and 1968, respectively.

He was a Member of the Faculty of the Indian Institute of Technology, Madras, from 1961 to 1964. Since 1964, he has been with Indian Institute of Science, Bangalore, where currently he is an Emeritus Fellow (AICTE) in the Department of Electrical Engineering. He has been a Visiting Professor at Yale University, New Haven, CT, Michigan State University, East Lansing, Concordia University, Montreal, PQ, Canada, and the National University of Singapore. His current research interests are in learning automata, neural networks, and fuzzy systems. He is coauthor of the book, *Learning Automata* (Englewood Cliffs, NJ: Prentice-Hall, 1989).

Dr. Thathachar is the recipient of the Alumni Award for Excellence in Research from the Indian Institute of Science and the Jawaharlal Nehru National Award in Engineering and Technology. He is a Fellow of the Indian National Science Academy, the Indian Academy of Sciences, and the Indian National Academy of Engineering.

P. S. Sastry (S'82–M'85–SM'97) received the B.Sc.(Hons.) degree in physics from the Indian Institute of Technology, Kharagpur, in 1978, and the B.E. degree in electrical communications engineering and the Ph.D. degree in electrical engineering from the Indian Institute of Science, Bangalore, in 1981 and 1985, respectively.

Since 1986, he has been on the Faculty of the Department of Electrical Engineering, Indian Institute of Science, Bangalore, where he is currently a Professor. He has held visiting positions at the University of Massachusetts, Amherst, the University of Michigan, Ann Arbor, and General Motors Research Labs, Warren, NJ. His research interests include learning systems, pattern recognition, image processing, and computational neuroscience.

Dr. Sastry is a recipient of the Alumni Medal for Best Thesis from the Division of Electrical Sciences, the Indian Institute of Science (1985), the Indo-U.S. Science and Technology Fellowship (1992), and the Sir C. V. Raman Award in Computer Science (1999).