Sādhanā, Vol. 24, Parts **4** & **5**, August & October 1999, pp. 261–292. © Printed in India

Learning automata algorithms for pattern classification

P S SASTRY and M A L THATHACHAR

Department of Electrical Engineering, Indian Institute of Science, Bangalore 560012, India
e-mail: [sastry,malt]@ee.iisc.ernet.in

Abstract. This paper considers the problem of learning optimal discriminant functions for pattern classification. The criterion of optimality is minimising the probability of misclassification. No knowledge of the statistics of the pattern classes is assumed and the given classified sample may be noisy. We present a comprehensive review of algorithms based on the model of cooperating systems of learning automata for this problem. Both finite action set automata and continuous action set automata models are considered. All algorithms presented have rigorous convergence proofs. We also present algorithms that converge to global optimum. Simulation results are presented to illustrate the effectiveness of these techniques based on learning automata.

Keywords. Learning automata; games of learning automata; optimisation of regression functions: minimising probability of misclassification; global optimisation.

1. Introduction

In this paper we discuss the problem of learning optimal decision rules for classifying patterns. We survey a number of adaptive stochastic algorithms for finding the decision rule that minimises the probability of misclassification. All these algorithms are based on the learning automata (LA) models (Narendra & Thathachar 1989). The primary motivation for this survey is that while LA methods are effective in solving many pattern recognition (PR) problems, the variety of automata-based techniques available for learning many rich classes of classifiers, are not widely known. Here we present a unified view of LA algorithms for pattern classification.

The LA algorithms that we consider are all essentially optimisation algorithms for finding a maximum or a minimum of a regression functional based on noisy function measurements. Such an optimisation is an important component of learning both in statistical pattern recognition as well as in computational learning theory and the probably approximately correct (PAC) learning of symbolic concepts (see the discussion in Haussler 1992). The LA algorithms that we discuss here are useful both in PR (Thathachar & Sastry 1987; Thathachar & Phansalkar 1995b) and in learning concepts in the forrn of Boolean expressions (Sastry *et al* 1993, Rajaraman & Sastry 1997). To put these algorithms

in proper perspective, we briefly review below the 2-class PR problem and the PAC learning framework as extended by Haussler (1992).

We shall be considering the pattern recognition (PR) problem in the statistical framework. For simplicity of presentation, we shall concentrate only on the 2-class problem. However, all these algorithms can be used in multiclass problems also (e.g., see discussion in Thathachar & Sastry 1987).

Consider a 2-class PR problem. Let $p(X|1)$ and $p(X|2)$ be the two class conditional densities and let p_1 and p_2 be the prior probabilities. Let the discriminant function, $g(X)$, be given by $g(X) = p(X|1)p_1 - p(X|2)p_2$. (Here X is the feature vector). Then, it is well known (Duda & Hart 1973) that the Bayes decision rule:

$$\begin{aligned} &\text{decide } \mathbf{X} \in \text{class} - 1, \quad \text{if } g(\mathbf{X}) > 0, \\ &\text{decide } \mathbf{X} \in \text{class} - 2, \quad \text{otherwise,} \end{aligned} \tag{1}$$

is optimal in the sense that it minimises the probability of error in classification.

Often, in a PR problem we do not know the class conditional densities and prior probabilities. All that is provided is a set of sample patterns along with their correct classification (modulo noise, if present), using which the proper decision rule is to be inferred. One approach is to assume that the form of the class conditional densities is known. Then the sample patterns can be used for estimating the relevant densities, which, in turn, can be used to implement Bayes decision rule (Duda & Hart 1973). This method is somewhat restricted by the class of densities that can be handled. Also it is difficult to relate the errors in classification to errors in estimation of densities.

An alternative approach is to assume some parametric form for the discriminant function and learn the needed parameters. Let $g(\mathbf{X}, \mathbf{W})$ be the discriminant function, where $\mathbf{X}$ is the feature vector and $\mathbf{W}$ is the parameter vector, to be used in a decision rule:

$$\begin{aligned} &\text{decide } \mathbf{X} \in \text{class} - 1, \quad \text{if } g(\mathbf{X}, \mathbf{W}) > 0 \\ &\text{decide } \mathbf{X} \in \text{class} - \mathbf{2}, \quad \text{otherwise.} \end{aligned}$$

Now the problem is one of determining an optimum value for the parameter vector from the sample patterns provided. For this we need to define a criterion function and devise algorithms for determining where the criterion function attains optimum values. We are interested in the case where the class conditional densities are totally unknown and there may be present both pattern noise (in the sense that class conditional densities may be overlapping) and classification noise (in the sense that the classification provided for the sample patterns may occasionally be incorrect). The objective is to determine a parameter vector that results in minimising probability of error in classification.

A popular criterion function for this problem (particularly for neural net algorithms) is the squared error over the sample set. Here we define

$$F(\mathbf{W}) = \sum_{\mathbf{X} \in \mathcal{S}} (Y(\mathbf{W}, \mathbf{X}) - t(\mathbf{X}))^2, \tag{3}$$

where $\mathcal{S}$ is the set of sample patterns. $Y(\mathbf{W}, \mathbf{X})$ denotes the output of the classifier with parameter vector $\mathbf{W}$ on the pattern $\mathbf{X}$, and $t(\mathbf{X})$ is the 'correct' classification (as given in the sample set) for $\mathbf{X}$.

This is the criterion function used with feedforward neural network models for pattern classification. In such a model $Y(.,.)$ is represented by the neural net and $\mathbf{W}$ corresponds to

the weights in the network. If we choose a discriminant function[1] $g(\mathbf{W},\mathbf{X}) = \mathbf{W}^T\mathbf{X}$ and define $Y(.,.)$ by the decision rule given by (2), we get the Perceptron model (Minsky & Papert 1969). The Perceptron learning algorithm guarantees to find a $\mathbf{W}$ at which the value of $F(.)$ given by (3) is zero provided such a $\mathbf{W}$ exists. In general, we can find a $\mathbf{W}$ that minimises $F(.)$ using gradient descent. However, in such a case, $Y(.,.)$ has to be differentiable with respect to its first argument. In feedforward neural net models with sigmoidal activation function, the error backpropagation algorithm (Rumelhart *et al* 1986) implements gradient descent in a parallel and distributed manner,

One of the problems with the criterion function $F(.)$ defined by (3) is that it measures the error of a classifier (given by $\mathbf{W}$) only over the sample set. However, we are interested in the classification error over the entire population. That is, if $\mathbf{W}^*$ is a minimiser of $F(.)$ then we want to know how well a classifier with parameters $\mathbf{W}^*$ performs on a random new pattern. This issue of generalisation is a well studied problem in statistics (Vapnik 1982, 1997). The 'goodness' of the learnt classifier $\mathbf{W}^*$ depends on whether there are 'sufficient' number of 'representative' samples and on the 'complexity' of the class of discriminant functions chosen. For example, in the case where the discriminant function is to be represented by a multilayer perceptron, if the samples are drawn in an *independent identically distributed (iid)* manner, then, using results from computational learning theory (Blumer *et al* 1989), it is possible to put an upper bound on the required number of samples to ensure, with a (preset) high probability, that the classification error with $\mathbf{W}^*$ is no more than (say) twice the average error made by $\mathbf{W}^*$ on the sample set (Baum & Haussler 1989). (See Vapnik 1997 for a comprehensive discussion on this issue.)

In the statistical pattern recognition framework, one can ensure that the criterion function properly takes care of the generalisation problem by defining $F(.)$ by

$$F(\mathbf{W}) = E[Y(\mathbf{W},\mathbf{X}) - t(\mathbf{X})]^2, \tag{4}$$

where E denotes expectation with respect to the (unknown) distribution from which the patterns are drawn. It should be noted that $F(.)$ defined by (4) is not observable. That is, given a $\mathbf{W}$ we cannot get $F(\mathbf{W})$ because the relevant statistical properties of the pattern classes are unknown. Now consider the function, $\hat{F}(\mathbf{W},\mathbf{X}) = (Y(\mathbf{W},\mathbf{X}) - t(\mathbf{x}))^2$ which is observable for all sample patterns. If the sample patterns are drawn in an *iid* manner then we have

$$F(\mathbf{W}) = E\hat{F}(\mathbf{W},\mathbf{X}). \tag{5}$$

From the given sample of patterns (and hence the observations on $\hat{F}$), we can find a $\mathbf{W}$ that minimises $F(.)$ using, e.g., stochastic approximation algorithms (Kiefer & Wolfowitz 1952, Blum 1954, Kashyap *et al* 1970, Kushner & Yin 1997, Borkar 1998). As earlier, we need to assume again that $Y(.,.)$ is a differentiable function of its first argument. Unlike the case of the criterion function defined by (3), here we can also take care of zero-mean independent additive noise in the classification (the expectation operation in (5) can include averaging with respect to the noise also).

There is still one important problem with both the criterion functions defined above. The parameter vector that minimises $F(.)$ does not necessarily minimise probability of misclassification (Sklansky & Wassel 1981). If we assume that probability of misclassification is a more natural criterion for pattern classification problems, then we

[1] Here $\mathbf{W}^T$ denotes transpose of vector $\mathbf{W}$

should define the criterion function by

$$F(\mathbf{W}) = \text{Prob}[\mathbf{W} \text{ misclassifies a random pattern}]. \tag{6}$$

For a two class problem, we can assume both Y and t (defined earlier) to be binary valued. Then, if we assume, as before, *iid* samples and unbiased classification noise, we can rewrite (6) by

$$F(\mathbf{W}) = EI_{\{Y(\mathbf{W},\mathbf{X})\neq t(\mathbf{X})\}}, \tag{7}$$

where I_A is the indicator of event **A,** and E denotes the expectation. Since the indicator function above is observable for all sample patterns, we may think that we can once again use simple stochastic approximation algorithms for minimisation. However, here $Y(.,.)$ needs to be binary valued and would not be differentiable. There are schemes based on stochastic approximation algorithms for tackling this problem (Sklansky & Wassel 1981) though these techniques are rather cumbersome and are computationally intensive.

In finding a W that minimises $F(\cdot)$ given by (7), one uses an iterative algorithm that updates $\mathbf{W}_n$ to $\mathbf{W}_{n+1}$ using the 'noisy' value of $F(\cdot)$, namely, $I_{\{Y(\mathbf{W}_n,\mathbf{X}_n)\neq t(\mathbf{X}_n)\}}$, where $\mathbf{X}_n$ is the nth sample pattern and $\mathbf{W}_n$ is the parameter vector at iteration n. Such algorithms (e.g., stochastic approximation algorithms or the LA algorithms discussed in this paper) converge asymptotically to a minimiser of $F(\cdot)$ given by (7). Such a procedure presupposes an infinite sequence of *iid* samples. Since in practice one has only a finite set of samples, what can we say about the result of such algorithms if ,the same set of samples are used repeatedly till convergence? This question, widely studied in computational learning theory, brings to focus the distinction between the statistical part and the optimisation part inherent in any learning problem. We briefly discuss below a framework due to Haussler (1992) that clarifies this issue.

Let $\mathcal{X}$ be the feature space and let $\mathcal{Y} = \{0, 1\}$ be the output space of our classifier. Let $\mathcal{S} = \{(X_i, y_i),\ 1 \leq i \leq m, X_i \in \mathcal{X}, y_i \in \mathcal{Y}\}$ be a finite set of labelled samples drawn in an *iid* manner with respect to some distribution P on $\mathcal{X} \times \mathcal{Y}$. (Note that since P is arbitrary, noise is automatically taken care of). Every classifier or decision rule is a function from $\mathcal{X}$ to $\mathcal{Y}$ (e.g., Y(W,.) above). Let $\mathcal{H}$ be the set of classifiers of interest. Define a functional on $\mathcal{H}$ by

$$F(h) = E\ell(h(X), y), \quad h \in \mathcal{H}, \tag{8}$$

where E denotes expectation with respect to P and $\mathbf{t}(.,.)$ is called the loss function. On a sample (X,y), where X is the feature vector and y is the classification, $\ell(h(X), y)$ gives the *loss* suffered by the classifier h. The loss function is assumed to be known to the learner. If we consider a loss function given by: $\ell(x, y) = 0$ when $x = y$ and $\ell(x, y) = 1$ when $x \neq y$, $F(.)$ given by (8) is same as that given by (7) and it will be the probability of misclassification with classifier h. Let

$$h^* = \arg\min_{h\in\mathcal{H}} F(h), \tag{9}$$

which is the optimal classifier.

Define another functional on $\mathcal{H}$, $\bar{F}$, by

$$\begin{aligned}\bar{F}(h) &= \bar{E}\ell(h(X), y) \\ &= \frac{1}{m}\sum_{i=1}^{m} \ell(h(X_i), y_i),\end{aligned} \tag{10}$$

where $\bar{E}$ denotes expectation with respect to the empirical distribution defined by the sample set $\mathcal{S} = \{(X_i, y_i), \mathrm{I} \leq \mathrm{i} \leq m\}$. Let

$$\hat{h} = \arg\min_{h \in \mathcal{H}} \bar{F}(h). \tag{11}$$

Suppose it is true that $\bar{F}(h)$ converges to $F(h)$ uniformly over $\mathcal{H}$. This depends on the nature of $\mathcal{H}$ and thus on the structure chosen for the classifier. However, given such a uniform convergence, a close enough approximator to $\hat{h}$ from a sufficiently large sample set will also be a close approximator to h^*. (See lemma 3.1 in Haussler (1992) and the discussion therein.) Now we can think of the problem of learning, to a good approximation, the optimal classifier h^* from a finite sample set as having two parts. One is the statistical problem of establishing the uniform convergence as above and calculating the number of samples needed for a given degree of approximation. The second one is an optimisation problem of finding a good approximator to $\hat{h}$ given a finite set of samples. The computational learning theory literature has concentrated mostly on the statistical part of the problem and many results are available regarding the number of samples needed for learning a good approximation to h^* for various kinds of $\mathcal{H}$ (Vapnik 1982; Natarajan 1991; Haussler 1992; Vapnik 1997). In this paper we concentrate on the second problem, namely, finding a good approximation to $\hat{h}$ given a finite set of samples, for many rich classes $\mathcal{H}$.

We will be discussing learning automata algorithms for minimising $F(\cdot)$ defined by (6) or equivalently by (8). LA methods considered here are all essentially regression function learning algorithms. That is, based on observations of $\ell(h(X_i), y_i)$ from a sequence $\{(X_i, y_i)\}$ of iid samples, the algorithm finds the minimiser of $F(\cdot)$ defined by (8). Hence, given a finite set of sample patterns, $\mathbf{S}$, if samples are presented to the algorithm by uniformly drawing from $\mathbf{S}$, then, the algorithm will (asymptotically) find a minimiser of $\bar{F}$ defined by (10). If $\bar{F}(h)$ converges to $F(h)$ uniformly over $\mathcal{H}$, then as discussed above, the algorithm will find a good approximator to the optimal classifier defined by (9). The structure of classifiers we consider are such that the needed uniform convergence holds. Hence, even though we present the algorithms as if we have an infinite sequence of *iid* samples, in this sense, the algorithm learns well from a sufficiently large finite sample. However, we do not give any sample complexity results for various families of classifiers here because these are easily calculated, e.g., from the results of Haussler (1992) and Vapnik (1997).

We can broadly distinguish between two types of algorithms for optimisation employed in pattern recognition: deterministic and stochastic. The simplest and most often used algorithm is the gradient descent procedure. Here one searches in the parameter space by the following iterative procedure

$$\mathbf{W}(k+1) = \mathbf{W}(k) - \eta \nabla F(\mathbf{W}(k)), \tag{12}$$

where η is the stepsize. For example, in feedforward neural network models, error backpropagation implements gradient descent in **a** distributed fashion. The dynamics of such a procedure traces out a path in the parameter space which generally converges to local minima. If there is noise in the sample set provided, then the calculated gradient direction may be erroneous and one cannot, in general, say how the algorithm behaves.

In stochastic algorithms, the next point in the parameter space is picked randomly (based on the current point and the value of the criterion function). For example, with the criterion function defined by (5), after each sample, we can employ the Robbins–Munro algorithm:

$$\mathbf{W}(k+1) = \mathbf{W}(k) - \eta_k \nabla_W \hat{F}(\mathbf{W}(k), \mathbf{X}(k)), \tag{13}$$

where the stepsize η_k satisfies $\eta_k \geq 0$, $\sum \eta_k = \infty$ and $\sum \eta_k^2 < \infty$ (Kushner & Yin 1997; Borkar 1998). This can be implemented if $Y(.,.)$ is differentiable and an explicit expression for the above gradient is available. Otherwise we can approximate the gradient by finite difference and use, e.g., the Kiefer–Wolfowitz procedure (Kiefer & Wolfowitz 1952). The procedure converges to a local minimum under some conditions on the noise etc. In these algorithms also, the search is in the parameter space though the choice of next point is random because it depends on the randomly drawn next pattern (also, there may be noise in the classification of $\mathbf{X}(k)$).

An alternative approach for stochastic search is provided by the learning automata models. These algorithms maintain, at each instant k, a probability distribution, say, $\mathbf{p}(k)$ over the parameter space. The parameter vector at instant k, $\mathbf{W}(k)$, is a random realisation of this distribution. The (noisy) value of the criterion function at this parameter value is then obtained and is used to update $\mathbf{p}(k)$ into $\mathbf{p}(k+1)$ by employing a learning algorithm. The objective of the learning algorithm is to make the process converge to a distribution that chooses the optimal parameter vector with arbitrarily high probability. Thus, here the search is over the space of probability distributions. The random choice of $\mathbf{W}(k)$ from $\mathbf{p}(k)$ can allow for sufficient exploration of the parameter space. Unlike stochastic approximation based techniques (or other gradient descent methods), here we do not need to explicitly estimate any gradient information and hence these algorithms are numerically more stable. In the rest of this paper we present a few such learning algorithms for pattern classification using learning automata.

2. Learning automata

In this section we briefly explain learning automata (LA) models. The reader is referred to Narendra & Thathachar (1989) for more details.

A learning automaton is an adaptive decision making device that learns the optimal action out of a set of actions through repeated interactions with a random environment. Based on the cardinality of the action set, two kinds of learning automata are distinguished: finite action set learning automata (FALA) and continuous action set learning automata (CALA).

2.1 *Finite action set learning automata*

In a FALA the number of actions available is finite. Let $\mathbf{A} = \{\alpha_1, \ldots, \alpha_r\}, r < \infty$, be the *set of actions* available. The automaton maintains a probability distribution over the action set. At each instant, k, the automaton chooses an *action* $\alpha_k \in \boldsymbol{A}$, at random, based on its current *action probability distribution*, $\mathbf{p}(k) \in \mathbf{R}^r$, $k = 0, 1, \ldots$. Thus, $\mathbf{p}(k) = [p_1(k) \cdots p_r(k)]^T \in \mathbf{R}$, with $p_i(k) = \text{Prob}[\alpha(k) = \alpha_i], \forall k$. The action chosen by the automaton is the input to the environment which responds with a *reaction*, $\beta(k)$. This reaction is stochastic and is also called the *reinforcement*. We have $\beta(k) \in R \subseteq [0,1]$, $\forall k$, where R is the set of possible reactions from the environment. If $R = \{0,1\}$ then the environment is called P-model; if $R = [0,1]$ then it is called S-model; and if $R = [\beta_1, \ldots, \beta q\}$ then it is called Q-model. In all cases, higher values of the reinforcement signal are assumed more desirable. Let $\mathcal{F}_i$ be the distribution from which $\beta(k)$ is drawn when $\alpha(k) = \alpha_i$, $1 \leq i \leq r$. Let d_i denote the expected value of $\beta(k)$ given $\alpha(k) = \alpha_i$ (i.e., the expected value of $\mathcal{F}_i$). Then d_i is called the *reward*

probability[2] 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 discussion, we have implicitly assumed that the distributions $\mathcal{F}_i$ and hence $d_i, 1 \leq i \leq r$, are not time varying and thus the identity of the optimal action is also not time varying. In this case the environment is said to be stationary. If the distribution of the random reaction from the environment for a given choice of action is time varying then the environment is said to be nonstationary. In this section we consider only stationary environments. In the next two sections where **LA** systems are used for pattern classification we will be considering some nonstationary environments.

The learning automaton has no knowledge of the distributions $\mathcal{F}_i$ or of the reward probabilities. The objective for the automaton is to identify the optimal action; that is, to evolve to a state where the optimal action is chosen with probability arbitrarily close to unity. 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 $(\alpha(k), \beta(k))$. Thus if T represents the learning algorithm, then, $\mathbf{p}(k+1) = T(\mathbf{p}(k), \alpha(k), \beta(k))$. The main problem of interest here is the design of learning algorithms with satisfactory asymptotic behaviour. We are interested in algorithms that make $p_m(k)$ converge to a value close to unity in some sense.

DEHNITION 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 $1 - \epsilon$,

$$\liminf_{k \to \infty} p_m(k) > 1 - \epsilon.$$

We will be discussing ϵ-optimal learning algorithms here. We can characterise ϵ-optimality in an alternative way that captures the connection between learning and optimisation. Define *average reward* at k, $G(k)$, by

$$\begin{aligned} G(k) &= E[\beta(k) | \mathbf{p}(k)] \\ &= \sum_i d_i p_i(k). \end{aligned} \tag{14}$$

DEFINITION 2

A learning algorithm is said to be ϵ-optimal if, given any $\epsilon > 0$, it is possible to choose parameters of the algorithm so that

$$\liminf_{k \to \infty} EG(k) > d_m - \epsilon.$$

It is easily seen that the two definitions are equivalent. Thus, the objective of the learning scheme is to maximise the expected value of the reinforcement received from the environment. In the remaining part of this section we present three specific learning algorithms which are used later on.

---

[2] This name has its origin in P-model environments where d_i is the probability of getting a reward (i.e., $\beta = 1$) with action α_i.

2.1a *Linear reward inaction (L_{R-I}) algorithm:* This is one of the most popular algorithms used with LA models. This was originally described in mathematical psychology literature (Bush & Mosteller 1958) but was later independently rediscovered and introduced with proper emphasis by Shapiro & Narendra (1969).

Let the automaton choose action α_i at time k. Then $\mathbf{p}(k)$ is updated as:

$$\mathbf{p}(k+1) = \mathbf{p}(k) + \lambda(\mathbf{e}_i - \mathbf{p}(k))\beta(k), \tag{15}$$

where $0 < \lambda < 1$ is the stepsize parameter and $\mathbf{e}_i$ is a unit probability vector with ith component unity and all others zero. To get an intuitive understanding of the algorithm, consider a P-model environment. When $\beta(k) = 1$, (i.e., a *reward* from the environment), we move $\mathbf{p}(k)$ a little towards $\mathbf{e}_i$ when α_i is the chosen action, thus incrementing the probability of choosing that action and decrementing all others. When $\beta(k) = 0$, (i.e., a *penalty* from the environment), the probabilities are left unchanged. Hence the name of the algorithm.

L_{R-I} is known to be ϵ-optimal in all stationary random environments (Narendra & Thathachar 1989). That is, given any $\epsilon > 0$, we can choose a $\lambda^* > 0$ such that for all $\lambda \leq \lambda^*$, with a large probability, asymptotically $p_m(k)$ will be greater than $1 - \epsilon$. L_{R-I} is very simple to implement and it results in decentralised learning in systems consisting of many automata (Sastry *et al* 1994). However, in such cases it can find only local minima (see the next section) and it may converge rather slowly.

2.1b *Pursuit algorithm:* This belongs to the class of estimator algorithms (Thathachar & Sastry 1985, Rajaraman & Sastry 1996) that were originally proposed to improve the speed of convergence. This algorithm typically converges about 10 to 50 times faster than L_{R-I}.

This improvement in speed is bought at the expense of additional computation and memory requirements. Here, the automaton maintains, in addition to the action probabilities, two more vectors, $\mathbf{Z}(k) = [Z_1(k), \ldots, Z_r(k)]^T$ and $\mathbf{B}(k) = [B_1(k), \ldots, B_r(k)]^T$. $Z_i(k)$ and $B_i(k)$ represent respectively, the number of times action α_i is chosen till k and the total amount of reinforcement obtained with α_i till k, $1 \leq i \leq r$. Then, a natural estimate of the reward probability of ith action, d_i, is $\hat{d}_i(k) = B_i(k)/Z_i(k)$, which is used in the algorithm to update the action probabilities. The algorithm also needs to update the vectors $\mathbf{Z}(k)$ and $\mathbf{B}(k)$ and it is specified below.

Let $\alpha(k) = \alpha_i$ and let $\beta(k)$ be the reinforcement at k. Then,

$$\begin{aligned} B_i(k) &= B_i(k-1) + \beta(k), \\ Z_i(k) &= Z_i(k-1) + 1, \end{aligned} \tag{16}$$

$$\begin{aligned} B_j(k) &= B_j(k-1), \ \forall j \neq i, \\ Z_j(k) &= Z_j(k-1), \ \forall j \neq i, \end{aligned} \tag{17}$$

$$\hat{d}_i(k) = \frac{B_i(k)}{Z_i(k)}, \quad i = 1, \ldots, r.$$

Let the random index H be defined by

$$\hat{d}_H(k) = \max_i \{\hat{d}_i(k)\}.$$

Then,

$$\mathbf{p}(k+1) = \mathbf{p}(k) + \lambda(\mathbf{e}_H - \mathbf{p}(k)), \tag{18}$$

where $\lambda(0 < \lambda < 1)$ is the stepsize parameter and $\mathbf{e}_H$ is the unit probability vector with Hth component unity and all others zero. By the definition of the random index H, α_H is the current estimated best action and (18) biases $\mathbf{p}(k+1)$ more in favour of that action. Since the index H keeps changing as the estimation proceeds, the algorithm keeps pursuing the current estimated best action. A special feature of the algorithm is that the actual reinforcement, $\beta(k)$ does not appear in the updating of $\mathbf{p}(k)$. Hence $\beta(k)$ can take values in any bounded set unlike the case of L_{R-I} where $\beta(k)$ has to be in [0,1] to ensure that $\mathbf{p}(k+1)$ is a probability vector (see (15)). The pursuit algorithm and the other estimator algorithms are ϵ-optimal in all stationary environments.

2.2 *Continuous action set learning automata*

So far we have considered the LA model where the set of actions is finite. Here we consider LA whose action set is the entire real line. To motivate the model, consider the problem of finding the maximum of a function $f : \mathrm{R} \rightarrow \mathrm{R}$, given that we have access only to noisy function values at any chosen point. We can think of f as the probability of misclassification with a single parameter discriminant function. To use the LA model for this problem, we can discretise the domain off into finitely many intervals and take one point from each interval to form the action set of the automaton (Thathachar & Sastry 1987) (see § 3 below). We can supply the noisy function value (normalised if necessary) as the reinforcement. This can solve the optimisation problem but only at a level of resolution which may be poor, based on the coarseness of the discretisation. Also, if we employ too fine a level of discretisation, the resulting LA will have too many actions and the convergence rate will be poor.

A more satisfying solution would be to employ an LA model where the action set can be continuous. Such a model, called continuous action-set learning automaton (CALA) will be discussed in this subsection.

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 (based on its interaction with the environment) by updating $\mu(k)$ and $\sigma(k)$, which is analogous to updating the action probabilities by the traditional LA. As before, let $\alpha(k) \in \mathrm{R}$ 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, $f : \mathrm{R} \rightarrow \mathrm{R}$, 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 whichf 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 off. However, we do not let $\sigma(k)$ converge to zero to ensure that the algorithm does not get stuck at a nonoptimal point. So, we use another parameter, $\sigma_\ell > 0$, and keep the objective of learning as $\sigma(k)$ converging to σ_ℓ and $\mu(k)$ converging to a maximum of f. By choosing σ_ℓ sufficiently small, asymptotically CALA will choose actions sufficiently close to the maximum with probability sufficiently close to unity.

The learning algorithm for CALA is described below. 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. Also, CALA interacts with the environment through choice of two actions at each instant.

At each instant k, **CALA** chooses an $x(k) \in \mathbf{R}$ at random from its current 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:

$$\mu(k+1) = \mu(k) + \lambda \frac{(\beta_x - \beta_\mu)}{\phi(\sigma(k))} \frac{(x(k) - \mu(k))}{\phi(\sigma(k))}$$

$$\sigma(k+1) = \sigma(k) + \lambda \frac{(\beta_x - \beta_\mu)}{\phi(\sigma(k))} \left[\left(\frac{(x(k) - \mu(k))}{\phi(\sigma(k))} \right)^2 - 1 \right] + \lambda \{ C[\sigma_\ell - \sigma(k)] \} \quad (19)$$

where

$$\phi(\sigma) = \sigma_\ell \quad \text{for } \sigma \leq \sigma_\ell$$
$$= \sigma \quad \text{for } \sigma > \sigma_\ell, \quad (20)$$

and

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

As explained at the beginning of this subsection, this **CALA** can be used as an optimisation technique without discretising the parameter space. It is similar to stochastic approximation algorithms (Kushner & Yin 1997; Borkar 1998) though here the randomness in choosing the next parameter value makes the algorithm explore better search directions. For this algorithm it is proved 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 (Santharam 1994; Santharam *et al* 1994).

3. A common payoff game of automata for pattern classification

In this section and the next we will present models using several learning automata for pattern classification. All the algorithms presented here are proved to converge (in some suitable sense) to the optimal solution. To keep the presentation simple, we will not state any of the convergence results in a precise form nor present any proofs. However, we will provide references for each algorithm where the convergence results are proved.

Recall from § 1 that we pose the pattern classification problem as follows. Let $g(\mathbf{X}, \mathbf{W})$ with $\mathbf{X}$ the feature vector and $\mathbf{W}$ the parameter vector, be the discriminant function. We classify a pattern using the classification rule given by **(2).** The form of $g(.,.)$ is assumed known (chosen by the designer). The optimal value for the parameter vector is to be determined by making use of a set of (possibly noisy) *iid* samples from the pattern classes which are preclassified. We are interested in learning $\mathbf{W}$ that maximises

$$\bar{F}(\mathbf{W}) = \text{Prob}[g(., \mathbf{W}) \text{ correctly classifies a random pattern}]. \quad (21)$$

F is defined over $\mathbf{R}^n$ if there are n parameters and we are interested in finding $\mathbf{W}$ that globally maximises F. Define

$$Y(\mathbf{X}, \mathbf{W}) = 1, \quad \text{if } g(X, \mathbf{W}) > 0,$$
$$= 0, \quad \text{otherwise.} \tag{22}$$

For a sample pattern **X,** define

$$t(\mathbf{X}) = 1, \quad \text{if label on } \mathbf{X} \text{ is as Class-1,}$$
$$= 0, \quad \text{otherwise.} \tag{23}$$

Now consider the function $F(\cdot)$ defined by

$$F(\mathbf{W}) = EI_{\{t(\mathbf{X})=Y(\mathbf{X},\mathbf{W})\}}, \tag{24}$$

where E denotes the expectation with respect to distribution of patterns. Since the samples are *iid,* this F will be same as $\bar{F}$ if $t(\mathbf{X})$ defined above is the *true* class of **X.** When there is noise, the F defined above takes care of any pattern noise that is due to the overlapping of class conditional densities[3]. However, if there are random mistakes in the classification of training samples, then the $F(\cdot)$ defined above will only give the probability that the classification of a random pattern by the system agrees with that of the teacher. But we want to actually maximise the probability of correct classification. As defined earlier, let $\bar{F}(\mathbf{W})$ be the probability of correct classification with parameters **W** and let ρ be the probability of correct classification by the teacher (which is assumed to be independent of the class to which the pattern belongs). Then,

$$\bar{F}(\mathbf{W}) = \rho EI_{\{t(\mathbf{X})=Y(\mathbf{X},\mathbf{W})\}} + (1-\rho)E(1 - I_{\{t(\mathbf{X})=Y(\mathbf{X},\mathbf{W})\}})$$
$$= (2\rho - 1)EI_{\{t(\mathbf{X})=Y(\mathbf{X},\mathbf{W})\}} + (1-\rho). \tag{25}$$

Thus, as long as $\rho > 0.5$, $\bar{F}(\mathbf{W})$ and $F(\mathbf{W})$ have the same maxima and hence it is sufficient to maximise F.

In the above we have assumed a uniform classification noise. That is, the probability of the teacher correctly classifying a pattern is same for all patterns. Some of the automata algorithms discussed here can also handle the more general case where the probability of teacher correctly classifying **X** is $\rho(\mathbf{X})$ as long as $\rho(\mathbf{X}) > 0.5$, $\forall \mathbf{X}$, for certain classes of discriminant functions (Nagendra 1997).

3.1 *Common payoff game of LA*

As briefly outlined in § 2.3, a single automaton is sufficient for learning the optimal value of one parameter. But for multidimensional optimisation problems we need a system consisting of as many automata as there are parameters. Here we consider the case where these automata are involved in a cooperative game.

Let $A_1, \ldots, A_N$ be the automata involved in an N-player game. In the terminology of Game Theory, the *choices* or *pure strategies* of each player correspond to the set of actions of each automaton. The action probability distribution of an automaton represents the current *mixed strategy* adopted by the corresponding player. Each play of the game consists of each of the automata players choosing an action and then getting the *payoffs*

[3] It may be noted that this is the case in most real pattern classification problems. The sample patterns are obtained from the respective physical processes that are sought to be distinguished rather than be drawn at random from the feature space and classified by an 'optimal' decision rule.

(reinforcement) from the environment for this choice of actions by the team. The game we consider is a common payoff game and hence all players get the same payoff. 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 a random payoff, $\beta(k)$ which is supplied as the common reinforcement to all automata. The objective for the team is to maximise the payoff. Define

$$d(x_1, \ldots, x_N) = E[\beta(k) | \alpha^i(k) = x_i, 1 \leq i \leq N]. \tag{26}$$

If $A_1, \ldots, A_N$ have all finite action sets then we call $d(x_1, \ldots, x_N)$ the reward probability for that choice of actions, In this case, we can represent the reward probabilities as a hypermatrix $D = [d_{j_1 \ldots j_N}]$ of dimension $r_1 \times \cdots \times r_N$, where

$$d_{j_1 \ldots j_N} = E[\beta(k) | \alpha^i(k) = \alpha^i_{j_i}, 1 \leq i \leq N]. \tag{27}$$

Here $\{\alpha^i_1, \ldots, \alpha^i_{r_i}\}$ 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 automata are to evolve to the optimal set of actions through multiple interactions with the environment (i.e., repeated plays of the game) and updating of their action probability vectors using a learning algorithm. For this case of a game of finite action set automata, the action $\alpha^i_{m_i}$ is the optimal action of automaton A_i, $1 \leq i \leq N$, if

$$d_{m_1 \ldots m_N} = \max\{d_{j_1 \ldots j_N}\}, \tag{28}$$

where the maximum is over all possible values of the indices. It may be noted here that for any single automaton in the team, the environment is nonstationary. This is because the reinforcement to any automaton depends also on the choice of actions by the other automata.

In a similar manner, we can consider a common payoff game played by CALA also. Then the action set of each automaton is the real line, $d(., \ldots, .)$ is a function from $\mathbf{R}^N$ to R, and we are considering a maximisation problem over N-dimensional real Euclidean space. The objective for the automata team again is to find a maximum of d using a learning algorithm. It may be noted again that the automata have no knowledge of the function d and all they get from the environment is the common reinforcement, β (whose expected value for a given choice of actions equals the value of function d at the corresponding parameter values).

3.2 Pattern classification with finite action set LA

We need to learn the optimal value of the parameter vector, $\mathbf{W} = [w_1, \ldots, w_N] \in \mathbf{R}^N$. Let $w_i \in V^i \subset \mathrm{R}$. In any specific problem, knowledge of the parametric form chosen for the discriminant function and knowledge of the region in the feature space where the classes cluster, is to be utilised for deciding on the sets V^i. Partition each of the sets V^i into finitely many intervals V^i_j, $1 \leq j \leq r_i$. Choose one point, v^i_j, from each interval V^i_j, $1 \leq j \leq r_i$, $1 \leq i \leq N$. For learning the N parameters we will employ a team of N automata, $A_1, \ldots, A_N$. The action set of ith automaton is $\{v^i_1, \ldots, v^i_{r_i}\}$. Thus the actions of ith automaton are the possible values for the ith parameter, which are finitely many due to the process of discretisation.

Now consider the following common payoff game played by these N automata, At each instant k, each automaton A_i chooses an action $\alpha^i(k)$ independently and at random according to its action probabilities, $\mathbf{p}_i(k)$. Since actions **of** automata are possible values for parameters, this results in the choice of a specific parameter vector, say $W(k)$ by the automata team. The environment classifies the next sample pattern using this parameter vector, and the correctness or otherwise of this classification is supplied to tlie team as the common reinforcement, $\beta(k)$. Specifically,

$$\begin{aligned} \beta(k) &= 1, \quad \text{if } Y(\mathbf{X}(k), \mathbf{W}(k)) = t(\mathbf{X}(k)), \\ &= 0, \quad \text{otherwise,} \end{aligned} \tag{29}$$

where $\mathbf{X}(\mathbf{k})$ is the sample pattern at k and Y and t are as defined by (22) and *(23).*

It is easy to see from (24) and (29) that the expected value of the common payoff to the team at k is equal to $F(\mathbf{W}(k))$ where $\mathbf{W}(k)$ is the parameter vector chosen by the team at k. Now it follows from (25), (26)–(29) that the optimal set of actions for the team (corresponding to the maximum element in the reward matrix) is the optimal parameter vector that maximises the probability of correct classification. Now what we need is a learning algorithm for the team which will make each automaton in the team converge to its optimal action. We will see below that each of the algorithms for a single automaton specified in § 2 can easily be adapted to the team problem.

Before proceeding further, it should be noted that this method (in the best case) would only converge to the classifier that is optimal from among the *finitely* many classifiers in the set $\prod_{i=1}^{N} V^i$. This can only be an approximation to *the* optimal classifier due to the inherent loss of resolution in the process of discretisation. While this approximation can be improved by finer discretisation, it can result in a large number of actions for each automaton and consequently, slow rate of convergence. One can also improve the precision in the learnt classifier by progressively finer discretisation. That is, we can first learn a rough interval for the parameter and then can choose the V^i set as this interval and further subdivide it and so on. However, the method is most effective in practice mainly in two cases: when there is sufficient knowledge available regarding the unknown parameters so as to make the sets V^i small enough intervals or when it is sufficient to learn the parameter values to a small degree of precision. Since we impose no restrictions on the form of the discriminant function $g(.,.)$, we may be able to choose the discriminant function so as to have some knowledge of the sets V^i. (This is illustrated through an example later on.) In \$3.3 we will employ a team of CALA for solving this problem where no discretisation of parameter ranges would be necessary.

3.2a *L_{R-I} algorithm for the team:* The linear reward inaction algorithm presented in § 2.1 is directly applicable to the automata team. Each automaton in the team uses the reinforcement that is supplied to it to update its action probabilities using (15). This will be a decentralised learning technique for the team. No automaton needs to know the actions selected by other automata or their action probabilities. In fact each automaton is not even aware that it is part of a team because it is updating its action probabilities as if it were interacting alone with the environment. However, since the reinforcement supplied by the environment depends also on the actions selected by others, each automaton experiences a nonstationary environment.

In a common payoff game, if each automaton uses an L_{R-I} algorithm with sufficiently small step size, then the team converges with arbitrarily high probability to a set of actions that is a *mode* of the reward matrix. The concept of a mode is defined below.

DEFINITION 3
The choice of actions, $\alpha^i_{j_i}$, $1 \leq i \leq N$, is called a mode of the reward matrix if the following inequalities hold simultaneously.

$$\begin{aligned} &d_{j_1 \cdots j_N} \geq \max_t \{d_{t j_2 \cdots j_N}\}, \\ &d_{j_1 \cdots j_N} \geq \max_t \{d_{j_1 t \cdots j_N}\}, \\ &\quad \vdots \\ &d_{j_1 \cdots j_N} \geq \max_t \{d_{j_1 \cdots j_{N-1} t}\}, \end{aligned} \tag{30}$$

where the maximum is over all possible values for the index, t.

The mode is a Nash equilibrium in the common payoff game. In our case, from the point of view of optimisation, it amounts to a local maximum. If the reward matrix of the game is unimodal then the automata team using the L_{R-I} algorithm will converge to the optimal classifier. For example, if the class conditional densities are normal and if the discriminant function is linear, then the game matrix would be unimodal. Another example where the automata team with L_{R-I} algorithm is similarly effective is that of learning simple conjunctive concepts (Sastry *et al* 1993; Rajaraman & Sastry 1997). However, in general, with this algorithm the team can converge only to a local maximum of $F(\cdot)$ defined by (24) (Sastry *et al* 1994) and, depending on the specific application, it may or may not be acceptable.

3.2b *Pursuit algorithm for the team:* We can adopt the pursuit algorithm presented in §2.2 for the automata team problem. However, if each automaton simply uses its reinforcement to estimate its effective reward probabilities, the algorithm will not work because each automaton experiences a nonstationary environment. We need to keep an estimated reward probability matrix from which each automaton can derive a quantity which is analogous to the $\hat{d}_i$ used in the pursuit algorithm in § 2.2. The complete algorithm is given below. We use hypermatrices B and Z for updating the estimated reward probability matrix. The vector $\hat{E}^i$ here serves for automaton A_i the same purpose as the vector $\hat{d}$ for the algorithm in *§2.2.*

Let $\alpha^i(k)$, the action chosen by the ith automaton at k, be $\alpha^i_{j_i}$, $1 \leq i \leq N$, and let $\beta(k)$ be the reinforcement. Then

$$\begin{aligned} B_{j_1 \cdots j_N}(k) &= B_{j_1 \cdots j_N}(k-1) + \beta(k), \\ Z_{j_1 \cdots j_N}(k) &= Z_{j_1 \cdots j_N}(k-1) + 1, \end{aligned} \tag{31}$$

$$\begin{aligned} B_{i_1 \cdots i_N}(k) &= B_{i_1 \cdots i_N}(k-1), \ \forall (i_1 \cdots i_N) \neq (j_1 \cdots j_N), \\ Z_{i_1 \cdots i_N}(k) &= Z_{i_1 \cdots i_N}(k-1), \ \forall (i_1 \cdots i_N) \neq (j_1 \cdots j_N), \\ \hat{d}_{i_1 \cdots i_N}(k) &= [B_{i_1 \cdots i_N}(k)]/[Z_{i_1 \cdots i_N}(k)], \ \forall (i_1 \cdots i_N). \end{aligned} \tag{32}$$

For $1 \leq i \leq N$, update the vectors E^i by

$$\hat{E}^i_\ell = \max_{t_s, 1 \leq s \leq N, s \neq i} \{\hat{d}_{t_1 \cdots t_{i-1} \ell t_{i+1} \cdots t_N}\}, \quad 1 \leq \ell \leq r_i.$$

Let the random indices $H(i)$, $1 \leq i \leq N$ be defined by

$$\hat{E}^i_{H(i)}(k) = \max_t \{\hat{E}^i_t(k)\}.$$

Then, the action probabilities are updated as:

$$\mathbf{p}_i(k+1) = \mathbf{p}_i(k) + \lambda(\mathbf{e}_{H(i)} - \mathbf{p}_i(k)), \quad 1 \leq i \leq N, \tag{33}$$

where λ is the step size parameter and $\mathbf{e}_{H(i)}$ is the unit probability vector with $H(i)$th component unity and all others zero.

It is proved by Thathachar & Sastry (1987, 1991) that the automata team employing this algorithm converges to the optimal set of actions even if the game matrix is not unimodal. Thus the automata team with pursuit algorithm learns the globally optimal classifier.

As is easy to see, this algorithm is not decentralised unlike the case with the L_{R-I} algorithm. To maintain the estimated reward probability matrix, we need to know the actions chosen by all the automata at that instant. The algorithm is computationally not very intensive. Only one element of the estimated reward probability matrix changes at each instant and hence one can make obvious computational simplifications while updating the vectors $\hat{E}^i$. However, as the dimensionality of the problem increases, the memory overhead becomes severe due to the need to store the estimated reward probability matrix. However, this algorithm will make the team converge to the maximum element in the reward probability matrix in any general game with common payoff and thus ensure convergence to the global maximiser of probability of correct classification.

3.3 *Pattern classification with CALA*

As earlier, we use a team of N learning automata, $A_1, \ldots, A_N$, for learning an N-dimensional parameter vector. The actions of the automata will be possible values for the parameters. We use N continuous action-set learning automata to lean the N components of the pararneter vector. Since the action set of a CALA is the real line, we need not discretise the parameter space. Each automaton will be using a normal distribution for the action probability distribution as described in § 2.3. The N automata will independently choose actions which results in the choice of a parameter vector by the team. As in the previous section, we classify the next pattern with the chosen discriminant function and supply a 0/1 reinforcement to the team, depending on whether the classification agrees with that of the teacher or not. Each of the automata uses the algorithm described in § 2.3 to update the action probability distribution. This is once again a completely decentralised learning scheme for the team. For the algorithm described by (19) in § 2.3, at each instant the automaton needs to interact with the environment twice. The same situation holds for the team also and thus we classify each pattern with two different parameter vectors to supply the two reinforcement signals needed.

It is proved that the team will converge (with arbitrarily large probability) to a parameter vector value that is arbitrarily close to a local maximum of the function $F(\cdot)$ defined by (24) (Santharam 1994).

3.4 *Simulations*

In this section we present results obtained with the automata team models presented earlier on some pattern classification problems. We present results with the pursuit algorithm and with CALA. For the pursuit algorithm, we need to discretise the parameters and the algorithm finds the global maximum. In the case of CALA, convergence to only local maxima is assured but we need not discretise the parameters. As will be seen below, by

proper choice of initial variance in the CALA algorithm we can obtain very good performance.

We present three problems and all three have 2-dimensional feature vectors. In the first two, the discriminant function is linear while in the third problem we use a quadratic discriminant function. In all problems the prior probabilities of the two classes are equal. Let $p(\mathbf{X}|\omega_i), i = 1, 2$ denote the two class conditional densities.

Example 1. $p(\mathbf{X}|\omega_1)$ has a uniform distribution over the compact set $[2,4] \times [2,4]$ and $p(\mathbf{X}|\omega_2)$ has a uniform distribution over the compact set $[3.5, 5.5] \times [3.5, 5.5]$.

Example 2. The class conditional densities for the two classes are given by Gaussian distributions:

$$\begin{aligned} p(\mathbf{X}|\omega_1) &= N(\mathbf{m}_1, \Sigma), \\ p(\mathbf{X}|\omega_2) &= N(\mathbf{m}_2, \Sigma), \\ \text{where } \mathbf{m}_1 &= [2.0, 2.0]^T, \\ \mathbf{m}_2 &= [3.0, 3.0]^T, \\ \Sigma &= \begin{bmatrix} 0.7 & 0 \\ 0 & 0.7 \end{bmatrix}. \end{aligned}$$

For these two problems we use the same discriminant function whose general form is given by

$$g(\mathbf{X}, \mathbf{W}) = 1 - x_1/w_1 - x_2/w_2$$

where the parameters w_1 and w_2 are unknown. We choose this form for the linear discriminant function to illustrate the fact that we can handle discriminant functions that are nonlinear in its parameters.

In example 1 with uniform densities, the weight values for the optimal linear discriminant function are $w_1 = 7.5$ and $w_2 = 7.5$. The theoretically computed value of the minimum probability of misclassification that can be achieved, with the optimal discriminant function, in the example is equal to 0.06.

In example 2 with Gaussian densities, the weight values for the optimal linear discriminant function are $w_1 = 5.0$ and $w_2 = 5.0$, for which the minimum probability of misclassification is equal to 0.16. The form of the discriminant function and the contours of the class conditional densities for these two problems are provided in figures 1 and 2.

Example 3. The class conditional densities for the two classes are given by Gaussian distributions:

$$\begin{aligned} p(\mathbf{X}|\omega_1) &= N(\mathbf{m}_1, \Sigma_1), \\ \text{where } \mathbf{m}_1 &= [2.0, 2.0]^T, \\ \Sigma_1 &= \begin{bmatrix} 1.0 & -0.25 \\ -0.25 & 1.0 \end{bmatrix}, \\ p(\mathbf{X}|\omega_2) &= N(\mathbf{m}_2, \Sigma_2), \\ \text{where } \mathbf{m}_2 &= [4.0, 4.0]^T, \end{aligned}$$

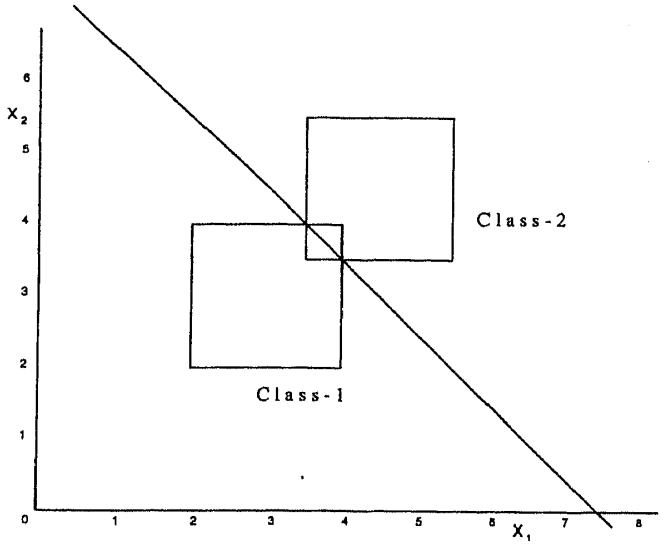

Figure 1. Class conditional densities and the form of the optimal discriminant function in example 1.

$$\Sigma_2 = \begin{bmatrix} 1.5 & -0.25 \\ -0.25 & 1.5 \end{bmatrix}.$$

For this problem, a quadratic discriminant function is considered. The form of the discriminant function is

$$g(\mathbf{X}, \mathbf{W}) = \left[mx_2 + x_1 - (m^2+1)\left(x_0 - \frac{a}{(1+m^2)^{1/2}} \right) \right]^2 \frac{1}{1+m^2} - \left[x_1 - \left(x_0 + \frac{a}{(1+m^2)^{1/2}} \right) \right]^2 - \left[x_2 - m\left(x_0 + \frac{a}{(1+m^2)^{1/2}} \right) \right]^2,$$

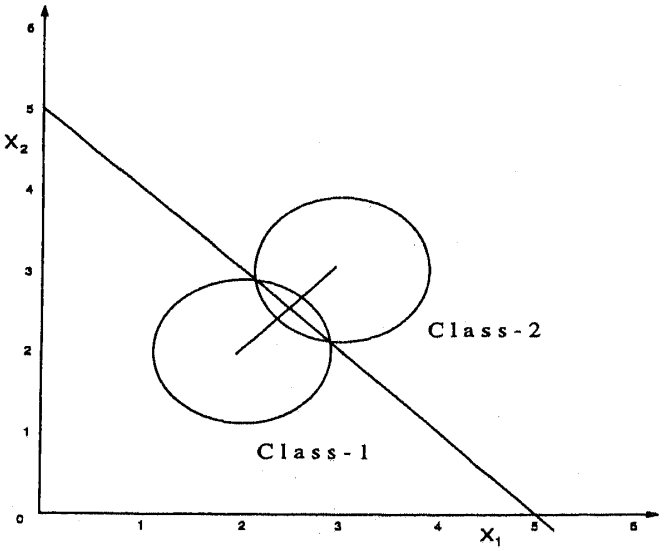

Figure 2. Class conditional densities and the form of the optimal discriminant function in example 2.

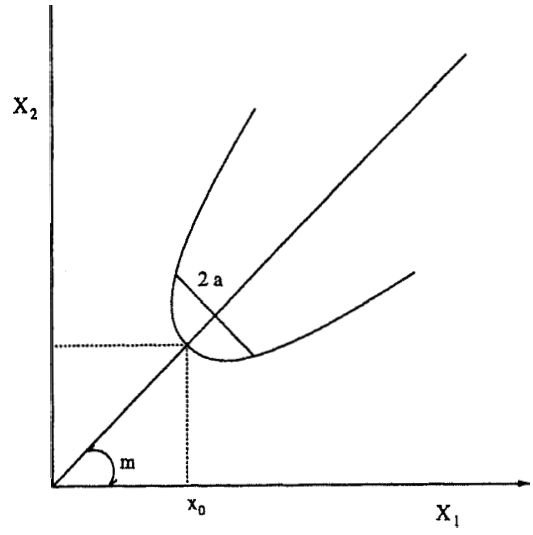

Figure 3. This figure shows the form of the discriminant function used in example 3. It is a parabola specified by three parameters m, x_0 and a.

where $\mathbf{W} = (m, x_0, a)$ is the parameter vector. This is a parabola described by three parameters m, x_0 and a. A sketch of this parabola is shown in figure 3. This form of the equation chosen for the parabola makes it easier to visualize the parabola in terms of the three parameters. As mentioned earlier, in our method we can choose any form for the discriminant function. With the specific form chosen, it is easier to guess the ranges of the parameters based on some knowledge of where in the feature space the two classes cluster. It would be considerably more difficult to guess the parameter ranges if we had chosen a general quadratic expression for our discriminant function. Once again it may be noted that the discriminant function is nonlinear in its parameters.

The parameters of the optimal discriminant function for the above pattern recognition problem (example 3) are:

$$m = 1.0 \; x_0 = 3.0 \; a = 10.0.$$

A sketch of the optimal discriminant function is shown in figure 4.

3.4a *Learning automata team with pursuit algorithm:* For all the three problems, 300 samples of each class are generated. At each instant, one of the patterns from this set is selected at random and given to the learning system. The learning parameter, λ, is set at 0.1

In the first two examples, we used a team of two automata, one for each parameter. The range of the parameters was chosen to be [0, 10] which was discretised into five intervals. Thus each automaton had five actions.

In example 1, 10 experiments were tried. In seven out of 10 runs, the team converged to the optimal actions. The average number of iterations needed for the probability of the optimal action in each automaton to be greater than 0.99 was 830, averaged over the seven runs. In the other three runs, the team converged to a nearby line.

In example 2, the team converged to the optimal actions in eight out of ten experiments and the average number of iterations for convergence was 930. In the other two runs, the team converged to a good suboptimal solution.

In example **3,** a team of three automata was used. The ranges for the parameters, m, x_0 and a were taken to be [0.5, 1.51, [2, 6] and [1, 10] respectively. The range of each

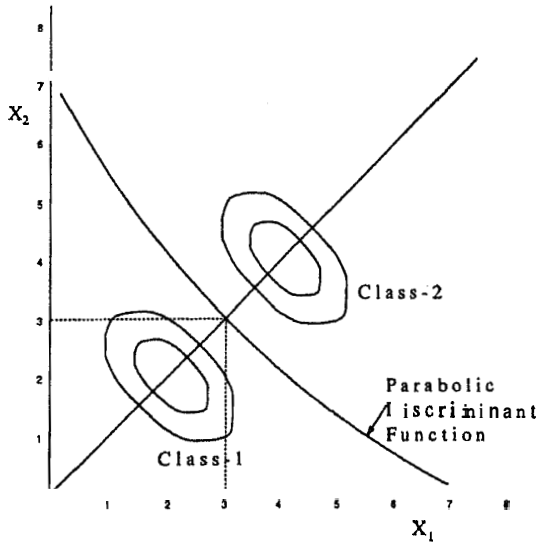

Figure 4. Class conditional densities and the form of the optimal discriminant function in example 3. The discriminant function used in this example is a parabola with 3 unknown parameters.

parameter was discretised into five levels. In seven out of 10 experiments, the team converged to the optimal parameters. In the other three runs, only one of the three automata converged to a wrong action. The average number of iterations needed for convergence was 1970.

3.4b ***CALA** team:* For the first two problems we use a team of two continuous action set automata, one for each parameter w_1 and w_2. For each problem, 300 samples of each class are generated from which a pattern is selected at random for every iteration during training. We also generated 50 sample patterns from each class for testing the optimality of the learned parameters. The results of the simulations for these two problems are presented in tables 1 and 2. We have chosen $\sigma_l = 0.3$ for both problems, and step size $\eta = 0.005$ for example 1 and $\eta = 0.01$ for example 2. In all the simulations, the value of the variance parameter σ in each CALA converged to some value such that $\phi(\sigma)$ was close to σ_l. In table 1 the number of iterations corresponds to the first instant (that is a multiple of 100) after which the number of classification errors for example 1 on the test set is less than 7 (out of

Table 1. Results obtained using **CALA** team with example 1.

Initial values						Final value
μ_0^1	μ_0^2	σ_0^1	σ_0^2	% Error	# Iterations	% Error
5	***5***	9	9	44	1600	6
5	5	11	11	44	2200	6
5	8	8	8	28	1900	6
5	8	12	12	28	1300	6
9	9	8	8	38	1900	6
9	9	12	12	38	1300	6
8	5	9	9	28	700	5
8	5	12	12	28	2500	6

Table 2. Results obtained using CALA team with example 2.

Initial values						Final value
μ_0^1	μ_0^2	σ_0^1	σ_0^2	% Error	# Iterations	% Error
2	2	6	6	49	2200	15
2	2	8	8	49	1300	16
8	8	6	6	46	1900	13
8	8	8	8	46	3400	16
2	7	6	6	40	1000	14
2	7	8	8	40	4500	13
7	3	6	6	35	5500	16
7	3	8	8	35	1900	16

the total of 100 test patterns), and that for example 2 is less than 17 (out of 100). It may be recalled that the minimum probability of misclassification achievable by the optimal discriminant function for example 1 was 0.06 and that for example **2** was 0.16. In the simple CALA algorithm, we can guarantee only a local convergence. However, from the tables given, one can see that the algorithm converges to a classifier with acceptably small error from many different starting points.

For the third problem we use a *team* of *three continuous action set learning automata.* As in the previous example, 300 sample patterns are generated from each class. Also a test set consisting of 100 samples is generated from the two classes. In this problem it is difficult to analytically compute the minimum probability of misclassification. The number of misclassifications on the generated test set of patterns, with the parameter values set to those values corresponding to the optimal discriminant function (mentioned above), was found to be 11 (out of a total of 100 test patterns). The results of the simulations are provided in table 3, for 3 different initial values.

4. Three-layer network consisting of teams of automata for pattern classification

In the previous section we have considered a common payoff game of automata and showed how it can be utilised for pattern classification. There we assumed that the designer has decided on a parametric representation for the discriminant function, $g(\mathbf{W}, \mathbf{X})$. The algorithm itself is independent of what this function is or how it is represented. For example, we could have represented it as an artificial neural network with parameters being the weights and then the algorithm would be converging to the 'optimal' set of weights. By making a specific choice for the discriminant function, we can configure the automata team more intelligently and this is illustrated in this section. We will only be considering teams of finite action learning automata here though the method can be extended to include

Table 3. Results obtained using CALA team with example 3.

Initial values								Final value
μ_0^1	μ_0^2	μ_0^3		σ_0^2	σ_0^3	% Error	# Iterations	% Error
1	0.4	6	6	6	4	32	3700	11
5	2	12	6	6	4	45	6400	12
4	2	12	6	6	4	42	3400	12

CALA. The material in this section follows Phansalkar (1991) and Thathachar & Phansalkar (1995).

In a two-class PR problem, we are interested in finding a surface that appropriately divides the feature space which may be assumed to be a compact subset of $\mathbf{R}^N$. Such a surface is well approximated by a piecewise linear function (Lippmann 1987) and can be implemented using linear threshold units in a three layer feedforward network. In this network the first layer units learn hyperplanes. Units in the second layer perform the *AND* operation on the outputs of *some selected* first layer units and thus learn convex sets with piecewise linear boundaries. The final layer performs an ***OR*** operation on the outputs of the second layer units. Thus this network, with appropriate choice of the internal parameters of the units and connections, can represent any subset of the feature space that is expressed as a union of convex sets with piecewise linear boundaries. The network structure is chosen because any compact subset of $\mathbf{R}^N$ with piecewise linear boundary can be expressed as a union of such convex sets. We now describe how we can configure such a network with each unit being a team of automata.

Let the first layer consist of M units and let the second layer have L units. That means we can learn at most M distinct hyperplanes and L distinct convex pieces. The final layer consists of a single unit. As before, let $\mathbf{X}(k) = [x_1(k),\ldots,x_N(k)]^T \in \mathbf{R}^N$ be the feature vector.

Denote by U_i, $1 \leq \mathrm{i} \leq M$, the units in the first layer, each of wliich should learn an N-dimensional hyperplane. A hyperplane in $\mathbf{R}^N$ can be represented by $(N+1)$ parameters – to represent the normal vector and the distance from the origin of the hyperplane. Hence we will represent each unit, U_i, $1 \leq i \leq M$, by an $(N+1)$-member team of automata, A_{ij}, $0 \leq j \leq N$. The actions of automaton A_{ij} are the possible values of thejth parameter of the ith hyperplane being learnt. Since we are using finite action set automata here[4], as in § 3.1, we discretise the ranges of parameters for making up the action sets of automata. Let W_{ij} be the set of actions of automaton A_{ij} whose elements will be denoted by w_{ijs}, $1 \leq s \leq r_{ij}$, $0 \leq j \leq N$, $1 \leq i \leq M$. Let $\mathbf{p}_{ij}(k)$ be the action probability vector of A_{ij} with components p_{ijs} and

$$\mathrm{Prob}[\alpha_{ij}(k) = w_{ijs}] = p_{ijs}(k),$$

where $\alpha_{ij}(k)$ is the action chosen by the automaton A_{ij} at time k. The output of unit U_i at k is $y_i(k)$ where

$$y_i(k) = 1, \quad \text{if } \sum_j \alpha_{ij}(k)x_j(k) > 0,$$

$$= 0, \quad \text{otherwise.} \tag{34}$$

Let V_i be the ith second layer unit that has connections with $n(i)$ first layer units, $1 \leq i \leq L$. The $n(i)$ first layer units that are connected to V_i are prefixed. Thus V_i can learn a convex set bounded by *ntmost* $n(i)$ hyperplanes. The unit V_i is composed of a team of $n(i)$ automata B_{ij}, $1 \leq j \leq n(i)$, each of which has two actions: 0 and 1. The action probability distribution of B_{ij} at k can be represented by a single real number $q_{ij}(k)$ where,

$$\mathrm{Prob}[z_{ij}(k) = 1] = 1 - \mathrm{Prob}[z_{ij}(k) = 0] = q_{ij}(k),$$

[4] We could also use a team of CALA for each unit in the first layer. We use a FALA team here to demonstrate (in the next subsection) how one can introduce a perturbation term to L_{R-I} type algorithms so as to converge to global optimum.

and $z_{ij}(k)$ is the action selected by B_{ij} at k. Let $a_i(k)$ be the output of V_i at instant k. $a_i(k)$ is the AND of the outputs of all those first layer units which are connected to V_i and are activated, i.e., $z_{ij}(k) = 1$. More formally,

$$a_i(k) = 1, \quad \text{if } y_j(k) = 1 \,\forall j, \quad 1 \le j \le n(i), \quad \text{such that } z_{ij}(k) = 1, \tag{35}$$

$$= 0, \quad \text{otherwise.} \tag{36}$$

The third layer contains only one unit whose output is a Boolean OR of all the outputs of the second layer units. Since here we want to learn a union of convex sets, no learning was needed in the third layer.

This network of automata functions as follows. At each instant k, all the automata in all the first and second layer units choose an action at random based on their current action probability vector. That is, in each U_i, each of the automata A_{ij} chooses an action $\alpha_{ij}(k)$ from the set W_{ij} at random based on the probability vector $\mathbf{p}_{ij}(k)$. This results in a specific parameter vector and hence a specific hyperplane being chosen by each U_i. Then, based on the next pattern vector, $\mathbf{X}(k)$, each unit U_i calculates its output $y_i(k)$ using (34). In each second layer unit V_i, all the $n(i)$ automata B_{ij} choose an action $z_{ij}(k)$ at random based on the probability $q_{ij}(k)$. Using these $z_{ij}(k)$ and the outputs of first layer units, each V_i would calculate its output $a_i(k)$ using (36). Using the outputs of the second layer units, the unit in the final layer will calculate its output which is 1 if any $a_i(k)$ is 1; and 0 otherwise. Let $Y(k)$ denote the output of the final layer unit which is also the output of the network. $Y(k) = 1$ denotes that the pattern is classified as class-1 and $Y(k) = 0$ denotes that the pattern is class-2. For this classification, the environment supplies a reinforcement $\beta(k)$ as

$$\beta(k) = 1, \quad \text{if } Y(k) = t(\mathbf{X}(k)),$$

$$= 0, \quad \text{otherwise,} \tag{37}$$

where $t(\mathbf{X}(k))$ is the classification supplied for the current pattern $\mathbf{X}(k)$ (cf. § 3.1). $\beta(k)$ is supplied as the common reinforcement to all the automata in all the units and then all the automata update their action probability vectors using the L_{R-I} algorithm as below.

For each i,j, $0 \le j \le N$, $1 \le i \le M$, the probability vectors $\mathbf{p}_{ij}$ are updated as

$$p_{ijs}(k+1) = p_{ijs}(k) + \lambda\beta(k)(1 - p_{ijs}(k)), \quad \text{if } \alpha_{ij}(k) = w_{ijs},$$

$$= p_{ijs}(k)(1 - \lambda\beta(k)), \quad \text{otherwise,} \tag{38}$$

For each i, j, $1 \le j \le n(i)$, $1 \le i \le L$, the probabilities q_{ij} are updated as

$$q_{ij}(k+1) = q_{ij}(k) + \lambda\beta(k)(1 - q_{ij}(k)), \quad \text{if } z_{ij}(k) = 1,$$

$$= q_{ij}(k)(1 - \lambda\beta(k)), \quad \text{otherwise.} \tag{39}$$

Let $\mathbf{P}(k)$ denote the internal state of the iietwork. This includes the action probabilities of all the automata in all the units. That is, it includes all $\mathbf{p}_{ij}$ and all q_{ij}. Define

$$\cdot f(\mathbf{p}) = E[\beta(k)|\mathbf{P}(k) = \mathbf{p}]. \tag{40}$$

For this network of teams of automata, the learning algorithm given by (38) and (39) will make $\mathbf{P}(k)$ converge to a local maximum of $f(\cdot)$. Thus the action probabilities of all the automata will converge to values that would (locally) maximise the expected value of the reinforcement.

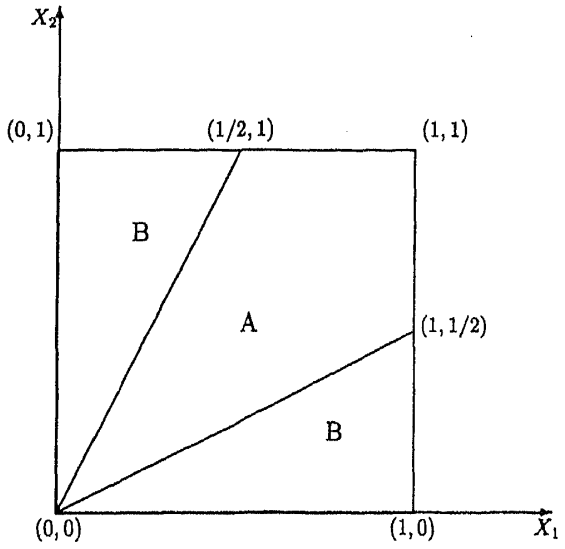

Figure 5. The form of the discriminant function for the pattern classification problem studied in example 4.

It is clear from the earlier discussion in § 3 and from (37), that maximising the expected reinforcement will result in maximising the probability of correct classification. Thus this network will learn a classifier which locally maxiinises the probability of correct classification.

4.1 *Simulations*

We consider two examples here to illustrate the three-layer network presented above. The first one is an artificial problem, while the second one is on a real data set. For the first one we consider a problem where the region in the feature space, in which the optimal decision is class-1, is a convex set with linear boundaries. Once again we consider only 2-dimensional feature vectors because it is easier to visualise the problem. Both these examples are from Thathachar & Phansalkar (1995b).

Example 4. The feature vectors from the environment arrive uniformly from the set [0, 1] x [0,1]. The discriminant function to be learnt is shown in figure 5. Referring to the figure, the optimal decision in region **A** is class-1 and that in region B is class-2. In region **A,**

$$\text{Prob}[\mathbf{X} \in \text{Class-1}] = 1 - \text{Prob}[\mathbf{X} \in \text{Class-2}] = 0.9,$$

and in region B,

$$\text{Prob}[\mathbf{X} \in \text{Class-1}] = 1 - \text{Prob}[\mathbf{X} \in \text{Class-2}] = 0.1.$$

The discriminant function to be learnt is

$$[2x_1 - x_2 > 0] \; AND \; [-x_1 + 2x_2 > 0],$$

where $\mathbf{X} = (x_1 \; x_2)^T$ is the feature vector.

Since we need to learn only one convex set, the network is made up of two first layer units, U_1 and U_2, and one fixed second layer unit. The second layer unit performs AND operation on the outputs of the first layer units. Each first layer unit has two automata. We used two rather than three because the hyperplanes to be learnt pass through the origin.

Each of the automata has four actions which are the possible values of the parameters to represent the hyperplanes. All four automata have the same action set given by $\{-2,-1,1,2\}$. The parameter vector that represents a pair of hyperplanes through the origin will have four components and hence a choice of action by each automaton represents a parameter vector. In this problem, there are two sets of choices of actions by the four automata (or parameter vectors) given by $(-1,2,2,-1)$ and $(2,-1,-1,2)$ at which the global optimum is attained. The learning parameter, λ, is fixed at 0.005 for all automata. The initial action probability distribution is uniform. That is, the initial probability of each of the four actions is 0.25. Twenty simulation runs were conducted and the network converged to one of the two sets of optimal actions in every run. The number of samples generated is 500 and at each instant one pattern chosen randomly from this set is presented to the network. The average number of iterations needed for the probability of the optimal action to be greater than 0.98 for each automaton, is 10,922 steps. (A single run of 20,000 steps took about 3.5 s of CPU time on a VAX 8810.)

Example 5. In this example, a 2-class version of the Iris data (Duda & Hart 1973) was considered. The data was obtained from the machine learning databases maintained at University of California, Irvine. This is a 3-class 4-feature problem. The three classes are iris-setosa, iris-versicolor and iris-viginica. Of these, setosa is linearly separable from the other two. Since we are considering only 2-class problems here, setosa was ignored and the problem was reduced to that of classifying versicolor and viginica. The data used was 50 samples of each class with the correct classification.

The network consisted of 9 first layer units and 3 second layer units. Each first layer unit has 5 automata (since this is a 4-feature problem). Each automaton had 9 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.

In this problem we do not know which are the optimal actions of the automata and hence we have to measure the performance based on the classification error on the data after learning.

For a comparison of the performance achieved by the automata network, we also simulated a standard feedforward neural network where we used backpropagation with momentum term (BPM) for the learning algorithm. The network has four input nodes (to take in the feature vector) and one output node. We tried two and three hidden layers. For the two hidden layer network we used 9 and 3 units in the hidden layers. For the three hidden layer network we used 8 nodes in each hidden layer. Initial weights for the network were generated randomly. In the learning algorithm the learning parameter for the momentum term was set at 0.9 and various values of the learning parameter for the gradient term were considered and the best results are reported here.

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. With 40% noise, each sample has a probability of 0.6 of correct classification.

The results obtained are summarised in table 4. These are averages of over 10 runs. The error reported in the table for the backpropagation algorithm is the root mean square error while that for the automata network is the probability of misclassification. 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 backpropagation with momentum converges about 20% faster. However, this algorithm fails to converge even when only 20%

Table 4. 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}.

Algorithm	Structure	Noise (%)	Error	steps
BPM	9 3 1	0	2.0	66,600
BPM	9 3 1	20	–	No convergence
BPM	9 3 1	40	–	No convergence
BPM	8 8 8 1	0	2.0	65,800
BPM	8 8 8 1	20	–	No convergence
BPM	8 8 8 1	40	–	No convergence
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

noise is added. The learning automata network continues to converge even with 40% noise and there is only slight degradation of performance with noise.

4.2 A *globally convergent algorithm for the network of automata*

In the three-layer network of automata considered above, all automata use the L_{R-I} algorithm (cf. (38) and (39)). As stated earlier, one can establish only the local convergence result for this algorithm. In this subsection we present a modified algorithm which leads to convergence to the global maximum.

One class of algorithms for the automata team that result in convergence to global maximum are the estimator algorithms. As stated in §3.3, these algorithms have a large memory overhead. Here, we follow another approach, similar to the simulated annealing type algorithms for global optimisation (Aluffi-Pentini *et al* 1985, Chiang *et al* 1987), and impose a random perturbation in the update equations. However, unlike in simulated annealing type algorithms, here we keep the variance of perturbations constant and thus our algorithm would be similar to constant heat bath type algorithms. Since a FALA learning algorithm updates the action probabilities, introducing a random term directly in the updating equations is difficult due to two reasons. First, it is not easy 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 analysis difficult. To overcome such difficulties, the learning automaton is *parametrised* here. The automaton will now have an internal state vector, **u,** of real numbers, which is not necessarily a probability vector. The probabilities of various actions are calculated based on the value of **u** using a *probability generating function,* $\bar{g}(.,.)$. The value of $\bar{g}(\mathbf{u}, \alpha_i)$ will give the probability with which the ith action is chosen by the automaton when the state vector is **u.** Such learning automata are referred to as *parametrised learning automata* (PLA) (Phansalkar 1991). The probability generating function that we use is given by

$$p_i = \bar{g}(\mathbf{u}, \alpha_i) = \exp(u_i) \Big/ \sum_j \exp(u_j), \tag{41}$$

where $\mathbf{u} = (u_1 \cdots u_r)'$ is the state vector and $\mathbf{p} = (p_1 \cdots p_r)'$ is the action probability vector.

We will now give the complete learning algorithm for the network of automata following the same notation that was used earlier. Thus p_{ijs} is the probability of sth action, w_{ijs},

of automaton A_{ij} which is the jth automaton in U_i, the ith first layer unit and so on. The functioning of the network is the same as before. However, the learning algorithm now updates the internal state of each automaton and the actual action probabilities are calculated using the probability generating function. Suppose $\mathbf{u}_{ij}$ is the state vector of automaton A_{ij} and has components u_{ijs}. Similarly, $\mathbf{v}_{ij}$ is the state vector of automaton B_{ij} and has components v_{ij0} and v_{ij1}. Let $g_{ij}(.,.)$ be the probability generating function for automaton A_{ij} and let $\&(.,.)$ be the probability generating function for automaton B_{ij}. As indicated by (41), the various action probabilities, p_{ijs} and q_{ij} are now given by

$$p_{ijs} = g_{ij}(\mathbf{u}_{ij}, w_{ijs}) = \frac{\exp(u_{ijs})}{\sum_s \exp(u_{ijs})},$$
$$q_{ij} = \tilde{g}_{ij}(\mathbf{v}_{ij}, 1) = \frac{\exp(v_{ij1})}{\exp(v_{ij1}) + exp(v_{ij0})}. \tag{42}$$

The algorithm given below specifies how the various state vectors should be updated. Unlike in (38) and (39), there is a single updating equation for all the components of the state vector. $\beta(k)$ is the reinforcement obtained at k, which is calculated as before by (37).

For each $i, j, 0 \leq \mathbf{j} \leq N$, $1 \leq i \leq M$, the state vectors $\mathbf{u}_{ij}$ are updated as

$$u_{ijs}(k+1) = u_{ijs}(k) + \lambda\beta(k)\frac{\partial \ln g_{ij}}{\partial u_{ijs}} + \lambda h'(u_{ijs}(k)) + \sqrt{\lambda} s_{ij}(k). \tag{43}$$

For each i, j, $1 \leq \mathbf{j} \leq n(i)$, $1 \leq \mathrm{i} \leq L$, the state vectors $\mathbf{v}_{ij}$ are updated as

$$v_{ijs}(k+1) = v_{ijs}(k) + \lambda\beta(k)\frac{\partial \ln \tilde{g}_{ij}}{\partial v_{ijs}} + \lambda h'(v_{ijs}(k)) + \sqrt{\lambda} s_{ij}(k), \tag{44}$$

where

- The functions $g_{ij}(.,.)$ and its partial derivatives are evaluated at $(\mathbf{u}_{ij}(k), \alpha_{ij}(k))$, the current state vector and the current action of $\mathbf{A}_{ij}$. Similarly, the functions $\tilde{g}_{ij}(.,.)$ and its partial derivatives are evaluated at $(\mathbf{v}_{ij}(k), z_{ij}(k))$.
- $h'(.)$ is the derivative of $h(.)$ defined by

$$\begin{aligned} h(x) &= -K(x - L_1)^{2n}, \quad x \geq L_1, \\ &= 0, \quad |x| \leq L_1, \\ &= -K(x + L_1)^{2n}, \quad x \leq -L_1, \end{aligned} \tag{45}$$

  where K and L_1 are real numbers and n is an integer, all of which are parameters of the algorithm.
- $\{s_{ij}(k)\}$ is a sequence of iid random variables (which also are independent of all the action probabilities, actions chosen etc.) with zero mean and varianace σ^2. σ is a parameter of the algorithm.

In the updating equations given by (43) and (44), the $h'(\cdot)$ term on the right-hand side (*rhs*) is essentially a projection term which ensures that the algorithm exhibits bounded behaviour; and the s_{ij} term on the rhs adds a random ***walk*** to the updating. The $\beta(k)$ term on the rhs is essentially the same updating as the L_{R-I} given earlier. To see this, it may be noted that

$$\frac{1}{g_{ij}(\mathbf{u}_{ij}, w_{ijs})}\left[\frac{\partial g}{\partial u_{ijs}}(\mathbf{u}_{ij}, w_{ijs})\right] = 1 - p_{ijs},$$
$$\frac{1}{g_{ij}(\mathbf{u}_{ij}, w_{ijs})}\left[\frac{\partial g}{\partial u_{ijs'}}(\mathbf{u}_{ij}, w_{ijs})\right] = -p_{ijs'}. \tag{46}$$

For this algorithm it is proved (Thathachar & Phansalkar 1995a) that a continuous-time interpolated version of the state of the network, **U**, given by the states of all automata, converges to a solution of the Langevin equation given by

$$d\mathbf{U} = \nabla \mathbf{H}(\mathbf{U}) + \sigma d\mathcal{W}, \tag{47}$$

where

$$\mathbf{H}(\mathbf{U}) = E[\beta \mid \mathbf{U}] + \sum h(u_{ijs}),$$

and $\mathcal{W}$ is the standard Brownian motion process of appropriate dimension.

As is well-known, the solutions of the Langevin equation concentrate on the global maximum of **H** as σ tends to zero. By the nature of the function $h(.)$, this means the algorithm will converge to a state that globally maximises the expected value of the reinforcement if the global maximum state vector is such that each component is less than L_1 in magnitude. Otherwise it will find a global maximum of **H** (which can be shown to be in a bounded region) and the expected value of reinforcement at this point would be greater than or equal to that inside the bounded region allowed for the algorithm. For more discussion and precise statement of this convergence result, the reader is referred to Thathachar & Phansalkar (1995a).

4.2a *Simulations with the global algorithm:* Here, we briefly give results of simulations with this algorithm on one example considered earlier in § 4.1, namely, example 4.

In this example, we have seen that the global maximum is attained at two parameter vectors $(-1, 2, 2, -1)$ and $(2, -1, -1, 2)$. One of the local maxima in that problem is given by (1,1,1,1). We have seen that the L_{R-I} algorithm converges to the global maximum when started with uniform initial conditions, that is, equal initial probabilities to all actions in all automata. Here we pick the initial conditions such that the effective probability of the parameter vector corresponding to the local maximum is greater than 0.98 and the rest of the probability is distributed among the other parameter vectors. With this much of bias, the L_{R-I} algorithm always converged to the local maximum.

We try the globally convergent algorithm presented above with these initial conditions. The parameters in the $h(.)$ function are set as L_1=3.0, K=1.0, and n=2. The learning parameter is set at 0.05. The value for σ is initially 10 and is reduced as

$$\sigma(k+1) = 0.999\sigma(k), \quad 0 \leq k \leq 5000,$$

and is kept constant thereafter. (Here $\sigma(k)$ is the value of σ used at iteration k.)

Twenty simulations were done and each time the algorithm converged to one of the two global maxima. The average number of iterations needed for convergence was 33,425. This was with a set of 500 samples as in example 4. The time taken for one **run** of 50,000 iterations was 36 s (CPU time) on **VAX** 8810. This, of course, does not compare favourably with the time taken by L_{R-I}. The extra time seems to be mainly due to the fact that the action probabilities are to be computed at each instant and are not directly stored. The extra terms in the algorithm (the term for bounding the algorithm and the random term) do not

seem to slow down the algorithm much. A different choice of the probability generating function may result in a faster algorithm. However, the higher computational time and slower rates of convergence appear to be the price to be paid for convergence to global maximum, as in all annealing type algorithms.

5. Discussion

In this paper we have considered algorithms based on teams of learning automata for pattern classification. In all the algorithms, some parametric representation was chosen for the discriminant function and the objective was to learn the optimal values of parameters from the given set of preclassified samples. We have considered pattern classification problems where no knowledge regarding the distribution of pattern classes is assumed and further, there may be present both pattern noise and classification noise.

The criterion of optimality is maximising probability of correct classification. As mentioned in § 1, algorithms that minimise mean square error do not necessarily maximise probability of correct classification. The problem is to find a classifier, h, that minimises $F(\cdot)$ given by (8) with a 0–1 loss function. While $\ell(h(X), y)$ is observable, $F(h)$ is not and hence we have to solve a regression problem. However, there are two additional complications here. Even if h is parameterised by a real vector and $F(\cdot)$ is differentiable, $\ell(h(X), y)$ may not be differentiable. Hence algorithms (like stochastic approximations) that rely on estimating the gradient information by perturbation methods, are not likely to be robust. In addition, we may choose the structure of the classifier in such a way that it is not easy to define a gradient (e.g. the three-layer network of § 4). The main strength of automata based algorithms (and other reinforcement learning methods) is that they do not explicitly estimate the gradient.

The essence of LA based methods is the following. Let $\mathcal{H}$ be the space of classifiers chosen. Then we construct an automata system such that when each of the automata chooses an action from its action set, this tuple of actions corresponds to a unique classifier, say h, from $\mathcal{H}$. Then we give $1 - \ell(h(X), y)$ (which, for the 0–1 loss function, is simply correctness or otherwise of classifiying the next training pattern with h) as the reinforcement. Since the automata algorithms guarantee to maximise expected reinforcement, with *iid* samples the system will converge to an h that maximises $F(\cdot)$ given by (8). The automata system is such that its state, represented by the action probability distributions of all automata, defines a probability distribution over $\mathcal{H}$. It is this probability distribution that is effectively updated at each instant. Thus the automata techniques would be useful even in cases where $\mathcal{H}$ is not isomorphic to a Euclidean space (or when there is no simple algebraic structure on $\mathcal{H}$).

In the simplest case, if the classifier structure is a discriminant function determined by N real valued parameters, then the actions of automata are possible values of the parameters and we employ a cooperating team of N automata involved in a common payoff game. If we use the traditional (finite action set) learning automata then we have to discretise the parameter space which may result in loss of precision. However, from the results obtained on the Iris data (cf. example 5), it is easy to see that the automata algorithm, even with discretisation of parameters, performs at a level comparable to other techniques such as feedforward neural nets in noise-free cases and outperforms such techniques when noise is present. We have also presented algorithms based on the recent model of continuous action set learning automata **(CALA)** where no discretisation of parameters is needed.

The interesting feature of the automata models is that the actions of automata can be interpreted in many different ways leading to rich possibilities for representing classifiers. In the algorithms presented in §3, the actions of all automata are values of real-valued parameters. The discriminant functions (functions mapping $\mathbf{R}^N$ to $\mathbf{R}$) can be nonlinear in parameters also (as illustrated in the examples) since the form of the discriminant function does not affect the algorithm. In §4, another structure of automata are used to represent unions of convex sets. Here the actions of first level automata are real values denoting parameters to represent hyperplanes. The actions of second level automata are Boolean decisions regarding which hyperplanes to pick to make convex sets. Here the discriminant function is essentially a Boolean expression whose literals are simple linear inequalities. It is easy to see that a network structure like this will also be useful in learning decision tree classifiers. Another example of this flexibility is that the same models discussed in §3 can be used for concept learning where the features may be nonnumeric and the discriminant function is a logic expression (Sastry *et al* 1993, Rajaraman & Sastry 1997.).

All the automata algorithms presented here implement a probabilistic search over the space of classifiers. All the action probabilities of all the automata together determine a probability distribution over $\mathcal{H}$. At each iteration, an $h \in \mathcal{H}$ is chosen (by each of the automata choosing an action) which is a random realisation of this probability distribution. Then the reinforcement obtained is used, in effect, to tune this probability distribution over $\mathcal{H}$. This allows for a type of randomness in the search that helps the algorithms to generally converge to good parameter values even in presence of local minima. The three-layer automata network delivers good performance on the Iris data even under 40% classification noise. The CALA algorithm also achieves good performance (see simulation results in §3.4) though theoretically only convergence to local maxima is assured. In the CALA algorithm, this is achieved by choosing a higher value of the initial variance for the action probability distribution which gives an initial randomness to the search process to better explore the parameter space.

We have also presented algorithms where convergence to global maximum is assured. The pursuit algorithm allows a team of finite action set automata to converge to the global maximum of the reward matrix. However, this algorithm has a large memory overhead to estimate the reward matrix. One can trade such memory overhead for time overhead using a simulated annealing type algorithm. We presented automata algorithms that use a biased random walk in updating the action probability distributions (cf. §4.2) and here the automata team converges to the global maximum with a large probability. A similar modification is possible for the CALA algorithm also so that it can converge to the global maximum.

There are some similarities between the automata approach discussed here and the approach based on genetic algorithms for optimisation. Like in genetic algorithms, in the automata approach, we maintain a population of possible parameter values. However this population is maintained implicitly through the effective action probability distribution of the system of automata. At each instant we evaluate this population and update it into a new population. The updating does not involve combining different parameters to make new ones. However, it may be argued that since we could keep an infinite population such combining may not be necessary. The second difference here is that the updating of the population is based on evaluation of a single randomly chosen member of the population unlike in the genetic algorithms where the updating is based on the evaluation of a large number of possible parameters. This can be incorporated into the automata framework by

making the automaton choose multiple actions and receive multiple reinforcements before updating the action probability distributions. For this, we can think of a parallel module of identical automata interacting with the environment. One can design learning algorithms for such parallel modules of automata which are ϵ-optimal and which result in a large increase in the speed of learning (which increases almost linearly with the number of parallel units). Details of such automata structures can be found in Thathachar & Arvind (1998). In spite of the above similarities, there are many differences between automata algorithms and genetic algorithms. The main strength of the automata models is that all the algorithms discussed in this paper have rigorous convergence proofs. More work is needed to combine the analytical tractability of the automata algorithms with some of the ideas from genetic algorithms to design more flexible learning systems with provable convergence properties.

There are other automata models that have been used for pattern classification. In all the models considered in this paper, the actions of automata are possible values for the parameters. It is possible to envisage an alternative set-up where the actions of the automata are the class labels. However, in such a case, we need to allow for the pattern vector to be somehow input to the automata system. Hence we need to extend the automaton model to include an additional input which we shall call *context.* In the traditional model of learning automata (whether with finite action set or continuous action set), the automaton does not take any input other than the reinforcement feedback from the environment. Within this framework we talk of the optimal action of the automaton without reference to any *context.* For example, when actions of automata are possible values of parameters, it makes sense to ask which is the optimal action. However, when actions of automaton are class labels, one can talk of the optimal action only in the *context* of a pattern vector that is input. Here with different context inputs, different actions may be optimal and hence we should view the objective of the automaton as learning to associate the right action with each context. Such a problem has been called ***associative*** *reinforcement learning* and automata models with provision for a context input, called generalised learning automata (GLA), are studied by many researchers (Barto 1985; Barto & Anandan 1985; Phansalkar 1991; Williams 1992). In contrast, the traditional automaton may be thought of as a model for non-associative reinforcement learning. In a GLA, the action probabilities for various actions would also depend on the current context vector input. Thus, if X is the context input then the probability of GLA taking action y is given by $\bar{g}(X, W, y)$, where $\bar{g}(.,.,.)$ is the action probability function of the GLA and W is a set of internal parameters. The learning algorithm updates the parameters W based on the reinforcement and the objective is to maximise the reinforcement over all context vectors. Due to the provision of the context input into the **GLA,** these automata can be connected together to form a network where outputs of some automata can form part of context input to other automata (Phansalkar 1991; Williams 1992). There are learning algorithms for GLA that guarantee convergence to local maxima of the reinforcement function (Phansalkar 1991). These automata networks can be used for pattern classification very much like the models discussed in this paper.

We thank G Santharam for his help in preparing the figures for this manuscript. This work is supported in part by an Indo-US project under ONR grant number N-00014-J-1324.

References

Aluffi-Pentini F, Parisi V, Zirilli F 1985 Global optimisation and stochastic differential equations. *J. Optim. Theory Appl.* 47: 1–26

Barto A G 1985 Learning by statistical cooperation of self-interested neuron-like computing elements. COINS Tech. Report. 81–11, Univ. of Massachusetts, Amherst, **MA**

Barto A G, Anandan P 1985 Pattern-recognizing stochastic learning automata. *IEEE Trans. Syst., Man Cybern.* 15: 360–374

Baum E, Haussler D L 1989 What sized net gives valid generalisation. *Neural Comput.* 1: 151–160

Blum J R 1954 Multidimensional stochastic approximation methods. *Ann. Math. Stat. 25:* 737–744

Blumer A, Ehrenfeucht A, Haussler D L, Warmuth M K 1989 Learnability and the Vapnik-Chervonenkis dimension. **J.** *Assoc. Comput. Mach.* 36: 929–965

Borkar V S 1998 Stochastic approximation algorithms: Overview and recent trends. *Sadhana* 24:

Bush R R, Mosteller F 1958 *Stochastic modelsfor learning* (New York: John Wiley)

Chiang T, Hwang C, Sheu S 1987 Diffusion for global optimisation in R^n. *SIAM* **J.** *Control Optim.* 25: 737–753

Duda R O, Hart P E 1973 *Pattern classifcation and scene analysis* (New York: Wiley)

Haussler D L 1992 Decision theoretic generalization of the PAC model for neural net and learning applications. *In$ Comput.* 100: 78–150

Kashyap R L, Blaydon C C, Fu K S 1970 Stochastic approximation. *Adaptive, learning and pattern recognition systems: Theory and applications* (eds) J M Mendel, K S Fu (New York Academic Press)

Kiefer J, Wolfowitz J 1952 Stochastic estimation of a regression function. *Ann.* Math. Stat. 23: 462–466

Kushner H Y and Yin G G 1997 *Stochastic approximation algorithms and applications* (New York: Springer-Verlag)

Lippmann R P 1987 An introduction to computing with neural nets. *IEEE Acoust., Speech Signal Process. Mag.* April: 4–22

Minsky M L, Papert S A 1969 *Perceptrons* (Cambridge, **MA:** MIT Press)

Nagendra G D 1997 *PAC learning with noisy samples.* M E thesis, Dept. of Electrical Engineering, Indian Institute of Science, Bangalore

Narendra K S, Thathachar M A L 1989 *Learning automata: An introduction* (Englewood Cliffs, NJ: Prentice Hall)

Natarajan B K 1991 *Machine learning:* A *theoretical approach* (San Mateo, **CA:** Morgan Kaufmann)

Phansalkar V V 1991 *Learning automata algorithms for connectionist suystems – local and global convergence.* Ph D thesis, Dept. of Electrical Engineering, Indian Institute of Science, Bangalore

Rajaraman K, Sastry P S 1996 Finite time analysis of pursuit algorithm for learning automata. *IEEE Trans. Syst., Man Cybern.* 27: 590–599

Rajaraman K, Sastry P S 1997 A parallel stochastic algorithm for learning logic expressions under noise. *J. Indian Inst. Sci.* 77: 15-45

Rumelhart D E, Hinton G E, Williams R J 1986 Learning internal representations by error propagation. *Parallel distributed processing* (eds) D E Rumelhart, J McLlelland (Cambridge, **MA:** MIT Press) vol. 1

Santharam G 1994 *Distributed learning with connectionist models for optimisation and control.* Ph D thesis, Dept. of Electrical Engineering, Indian Institute of Science, Bangalore

Santharam G, Sastry P S, Thathachar M A L 1994 Continuous action set learning automata for stochastic optimization. **J.** *Franklin Inst.* 331: 607–628

Sastry P S, Rajaram K, Ranjan S R 1993 Learning optimal conjunctive concepts using stochastic automata. *IEEE Trans. Syst., Man Cybern.* 23: 1175–1184

Sastry P S, 'Phansalkar V V, Thathachar M A L 1994 Decentralised learning of Nash equilibria in multi-person stochastic games with incomplete information. *IEEE Trans. Syst., Man Cybern.* 24: 769–777

Shapiro I J, Narendera K S 1969 Use of stochastic automata for parameter self optimisation multi-model performance criteria. *IEEE Trans. Syst. Sci. Cybern.* 5: 352–360

Sklansky J, Wassel G N 1981 *Pattern classification and trainable machines* (New York: Springer-Verlag)

Thathachar M A L, Sastry P S 1985 A new approach to the design of reinforcement schemes for learning automata. *IEEE Trans. Syst,, Man Cybern.* 15: 168–175

Thathachar M A L, Sastry P S 1987 Learning optimal discriminant functions through a cooperative game of automata. *ZEEE Trans. Syst., Man Cybern.* 17: 73–85

Thathachar M A L, Sastry P S 1991 Learning automata in stochastic games with incomplete information. *Systems and signal processing* (eds) R N Madan, N Vishwanathan, R L Kashyap (New Delhi: Oxford and IBH) pp 417–434

Thathachar M A L, Phansalkar V V 1995a Learning the global maximum with parameterised learning automata. *IEEE Trans. Neural Networks* 6: 398–406

Thathachar M A L, Phansalkar V V 1995b Convergence of teams and hierarchies of learning automata in connectionist systems. *IEEE Trans. Syst., Man Cybern.* 25: 1459–1469

Thathachar M A L, Arvind M T 1998 Parallel algorithms for modules of learning automata. *ZEEE Trans. Syst., Man Cybern.* B28: 24–33

Vapnik V N 1982 *Estimation of dependences based on empirical data* (New York: Springer-Verlag)

Vapnik V N 1997 *Nature of statistical learning theory* (New York: Springer-Verlag)

Williams R J 1992 Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8: 229–256