Convergence of Teams and Hierarchies of Learning Automata in Connectionist Systems

M. A. L. Thathachar, *Fellow, IEEE*, and V. V. Phansalkar

***Abstract*— Learning algorithms for feedforward connectionist systems in a reinforcement learning environment are developed and analyzed in this paper. The connectionist system is made of units of groups of learning automata. The learning algorithm used is the L_{R-I} and the asymptotic behavior of this algorithm is approximated by an Ordinary Differential Equation (ODE) for low values of the learning parameter. This is done using weak convergence techniques. The reinforcement learning model is used to pose the goal of the system as a constrained optimization problem. It is shown that the ODE, and hence the algorithm exhibits local convergence properties, converging to local solutions of the related optimization problem. The three layer pattern recognition network is used as an example to show that the system does behave as predicted and reasonable rates of convergence are obtained. Simulations also show that the algorithm is robust to noise.**

I. Introduction

In this paper, we consider connectionist systems which are based on the *Learning Automaton* model [1]. A general connectionist system consists of a (large) number of interconnected units. Various models can be used for the units. The Learning Automaton has several features which make it attractive as a basic computational unit in a connectionist system. One is the simple form of computation required for its operation. Another is the fact that as an inherently stochastic unit, it is robust to noise. Also, there are a range of techniques available to analyze such systems.

The learning automaton illustrates the concept of reinforcement learning. The main feature of this concept is the availability of an evaluatory signal from the environment which is indicative of the merit of the selected action. The learning automaton updates its internal state using the evaluatory signal and then selects a new action. A proper updating enables the automaton to improve its performance in some sense over a period of time. In many learning problems, there is additional information about the problem to be solved in the form of a *pattern* or *context vector*. The main drawback in using the original model of the Learning Automaton in connectionist systems is its inability to accept such a vector.

There are two ways in which the Learning Automaton can be incorporated in a connectionist system. One is to keep the structure of the learning automaton intact and use groups of learning automata as the basic computing units. The second method is to generalize the structure to allow for context vector inputs. We deal with the first method in this paper. This method has the advantage that the computational simplicity of the basic Learning Automaton is preserved. It is possible to build a system which consists of a single Learning Automaton, but such a system would have an unacceptably slow learning rate due to the large number of actions. To circumvent this problem, systems with a number of Learning Automata formed into a team or network of teams are considered. Such systems can be used, for instance, wherever feedforward networks are used. These reinforcement learning systems are gaining relevance in diverse areas such as function optimization, learning control, pattern recognition and robotics.

The Linear Reward-Inaction (L_{R-I}) algorithm [1] is used by the units described here. Weak convergence results are used to obtain an Ordinary Differential Equation (ODE) whose asymptotic behavior approximates the asymptotic behavior of the algorithm, for low values of the learning parameter. This ODE is analyzed along with the optimization problem to show that the algorithm does converge to a local optimum of the optimization problem. Since this is a local optimization result, it is expected that the system functions as needed only when the initial conditions are proper. Using examples from pattern recognition it is seen that the correct solution is learned when the initial conditions are properly chosen. Algorithms for Learning Automata exhibiting global convergence properties have been described in [2].

The rest of the paper is organized as follows. Section II develops the algorithm and the notation required for the analysis of the algorithm. Section III contains the analysis of the algorithm leading to convergence results. In Section IV the simulation results are presented. These consist of simulations done on pattern recognition problems using a three layer feedforward network. Section V contains the conclusions. Partial results of this paper appeared in [3].

II. Reinforcement Learning

We are concerned here with the *Associative Reinforcement* problem, in which an input-output mapping has to be learned. The basic setup to handle the Associative Reinforcement problem can be divided into two parts, the *Environment* and the *Learning System*. We consider here a Learning System made up of a feedforward network of units. The Associative Reinforcement system in this form is shown in Fig. 1. The functioning is as follows.

Manuscript received July 11, 1992; revised February 13, 1994 and December 10, 1994. This work was supported in part by by the Indo-U.S. Grant N-00014-J-1324.

The authors are with the Department of Electrical Engineering, Indian Institute of Science, Bangalore 560 012, India.

IEEE Log Number 9414028.

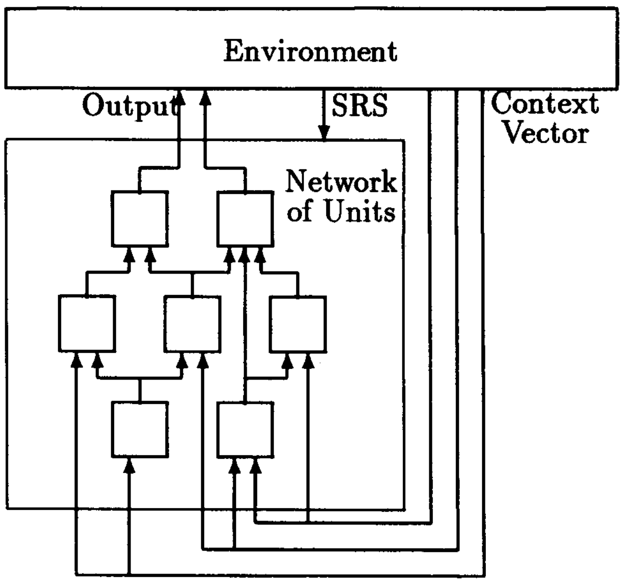


Fig. 1. Associative Reinforcement Learning System of a feedforward network of units interacting with environment.

- At each instant, the environment generates a *context vector* which is input to the learning system.
- Based on its internal state and the context vector, the learning system outputs an action.
- The environment then generates an evaluatory signal which indicates the appropriateness of the action to the particular context vector. This signal is known as the *Scalar Reinforcement Signal* (SRS) and a higher value of the SRS implies better performance of the system. Two of the main features of the SRS are that it is stochastic in nature with unknown distribution and its instantaneous value does not give any information about the relative performances of the other actions. The best action can be selected only after repeated trials of the system. Also, each context vector has its own optimal action, and this mapping of context vector into its optimal action is what should be ideally learned by the learning system. This is not always possible and what can be actually learned by the learning system is discussed later.
- The learning system then updates its internal state depending on the context vector—action—SRS triple so as to improve its performance with respect to an appropriate performance index.

The formal model of the environment is described in Subsection II-A. The learning system considered here is a feedforward connectionist network of units made up of Learning Automata and is described in Subsection II-B.

A. Environment

The context vector and SRS are obtained from an external source which is assumed to be independent of the learning system. This external source is the environment. The environment completely defines the problem which has to be solved by the learning system.

The signal which evaluates the action is the SRS. This is a real number and a higher value of the SRS indicates better performance of the system. The SRS is assumed to be stochastic in nature. Otherwise, at least when the number of actions is manageably finite, each action can be tried once and the action which returns the highest value of the SRS selected as the optimal action. The system receives, along with the SRS, a context vector which gives information about the state of the environment. Associated with each state of the environment is an optimal action. Thus the system has to learn a mapping from the set of states of the environment to the set of actions, associating with each state its optimal action. This mapping is known as the optimal action mapping.

The environment is defined by the tuple $\langle C, A, R, \mathbf{F}, P_e \rangle$ where,

1) C is the set of context vectors which give the learning system information about the state of the environment. C is usually a compact subset of $\mathbb{R}^n$, for some n.
2) A is the finite set of actions, $A = \{a_1, \ldots, a_m\}$.
3) R is the set of values the SRS can take. Here R is assumed to be the closed set $[0, 1]$, but can be any compact set of $\mathbb{R}$ in general.
4) $\mathbf{F} = \{F(a, c) : a \in A, c \in C\}$ is a set of probability distributions over R. $F(a, c)$ is the unknown distribution of the SRS when the state of the environment is c and the action is a.
5) P_e is an unknown probability distribution over C. The state of the environment at any instant is described by P_e . Thus, at any instant k, $k = 0, 1, 2 \ldots$, and for any Borel set $B \subset C$,

$$\text{Prob}\{c(k) \in B\} = P_e(B).$$

When the $F(a, c)$'s are independent of time, the environment is said to be stationary. For every $a \in A$ and $c \in C$ define

$$d(a, c) = E^{(a,c)}(r)$$

where $E^{(a,c)}$ denotes expectation with respect to the distribution $F(a, c)$. The optimal action for c is defined to be that action which gives the highest value of $d(a, c)$. There is an optimal action for each c. These optimal actions are, in general, different for different c. The optimal action mapping is denoted by OA where

$$d(OA(c), c) = \max_{a \in A} d(a, c). \qquad (1)$$

It is assumed that P_e is independent of time. More complicated models, in which the present state of the environment depends on past actions/context vectors, are not considered here. In many cases, what is needed is classification of a finite set of vectors, and for this purpose (1) can be assumed to hold.

B. Structure of the Learning System

The learning system consists of a feedforward network of units, each unit composed of a structure of learning automata. In the simplest case, each unit would have a set of weights as

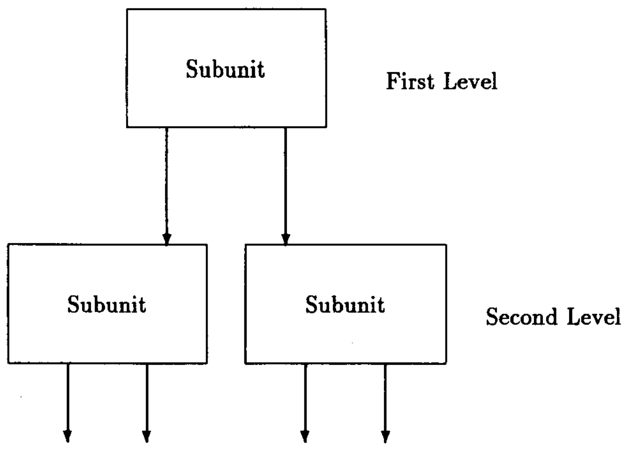


Fig. 2. A unit consisting of two levels of subunits.

its actions and the output would be generated as a function of the actions of the various automata in the unit and the context vector input to the unit.

A single unit in the network is described by the tuple $\langle X, Y, R, p, v, g, T \rangle$ where

1) X is the set of context vectors which are possible inputs to the unit.
2) Y is the set of outputs or actions of the unit.
3) R is the same as in the definition of the environment. It is the set of values the SRS can take. In this paper, R is assumed to be a subset of the closed interval [0, 1]. This assumption is made in learning automata literature and is needed to keep the probabilities in the closed interval [0, 1] after updating.
4) p is the vector composed of the action probability vectors of all the learning automata in the unit.
5) v is the vector made up of the actions of the learning automata. v and p have the same dimension.
6) g is a vector of functions.
7) T is the learning algorithm which updates p.

A single unit is composed of a tree hierarchy of subunits. Each subunit is made up of a team of learning automata. The functioning of the unit and subunit is as follows. There is one subunit in the first level of the hierarchy. This subunit chooses an action which activates exactly one subunit in the second level of the hierarchy. The action of this subunit activates precisely one subunit at the third level and so on. The output of the unit is the action of the final level subunit. Exactly one path in the hierarchy is activated. Figures 2 and 3 show the structures of a unit with two levels and a subunit with three automata respectively.

Each subunit is composed of a number of learning automata acting as a team. Each learning automaton in the subunit chooses an action according to its action probability vector. Let the subunit be capable of choosing from one of m actions. There are m functions, say $g_1, \ldots, g_m$ associated with this subunit. The g_is of all the subunits in the unit make up the vector function g associated with the unit. The ith action is

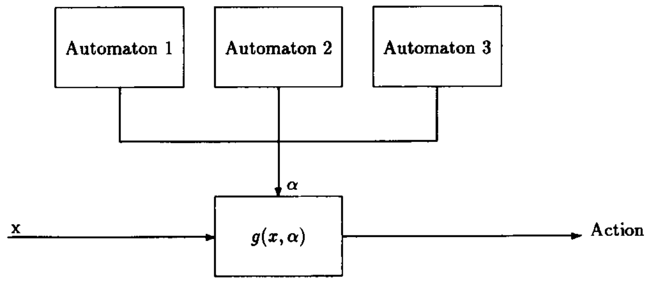


Fig. 3. A subunit consisting of three automata.

output by the subunit if

$$g_i(x, \alpha) = \max_{1 \le j \le m} g_j(x, \alpha)$$

where x is the context vector input to the unit of which the subunit is a part and α is the vector of actions chosen by the learning automata in the subunit.

In the case of a single unit interacting with the environment, the context vector input to the unit and the context vector from the environment are the same. However, in a network of units the context vector input to each unit can be composed of actions of other units and components of the context vector from the environment as in Fig. 1. The output of the network is a vector composed of the outputs of some prefixed units in the network. This vector is an action which affects the environment, that is, it belongs to the set A, A as defined in the definition of the environment. In the case of a single unit interacting with the environment, $X = C$ and $Y = A$.

The ideal goal of the system would be to learn the optimal action mapping as defined by (1). This may not be always possible due to the structure of the network. A practical solution is to maximize the expected value of the SRS as a function of the internal state of the network, that is p. This has to be done under the constraint that the action probability vector of each automaton remains a probability vector. Let the total number of automata in the network be N. p_i is the action probability vector of the ith automaton. Each automaton is assumed to have m actions. Thus, $p_i(k)$ is a m-dimensional vector,

$$p_i(k) = (p_{i1}(k), \ldots, p_{im}(k))^t.$$

The optimization problem can be written as

$$\left.\begin{array}{c} \text{maximize } f(p) = E[r \mid p] \\ \text{subject to } p_{ij} \ge 0 \quad 1 \le i \le N, 1 \le j \le m \\ \sum_{j=1}^{m} p_{ij} = 1 \quad 1 \le i \le N \end{array}\right\}. \tag{2}$$

The region in which p is allowed to take values is the *feasible region*, denoted by $\mathcal{F}$. Hence,

$$\mathcal{F} = \left\{ p \colon p_{ij} \ge 0, \sum_j p_{ij} = 1 \right\}. \tag{3}$$

Conditions for a point p to be a solution of the above optimization problem are given by the Kuhn-Tucker conditions [4], [5]. For the optimization problem (2) the First Order Necessary conditions are as follows.

Fact 1: Necessary conditions for p to be a local maximum of (2) are that there exist λ_{ij} and μ_i such that

$$\frac{\partial f}{\partial p_{ij}}(p) + \lambda_{ij} + \mu_i = 0 \tag{4}$$

$$\lambda_{ij} p_{ij} = 0 \tag{5}$$

$$\lambda_{ij} \geq 0 \tag{6}$$

$$p_{ij} \geq 0 \tag{7}$$

$$\sum_j p_{ij} = 1. \tag{8}$$

These are the First Order Necessary Kuhn Tucker (FONKT) conditions. These form the counterpart of the condition that the gradient must vanish in unconstrained optimization. Second order necessary and second order sufficient conditions also exist, but will not be used in this paper.

C. Learning Algorithm

The algorithm used by the system is described below. $p_i(k)$ is the action probability vector of the ith automaton at instant k. The ith automaton has $n(i)$ actions. Let v_i be the vector of actions of this automaton and $\alpha_i(k)$ the action chosen by the automaton at instant k. Then,

$$v_i = (v_{i1}, \ldots, v_{in(i)})^t$$
$$\text{Prob}\{\alpha_i(k) = v_{ij}\} = p_{ij}(k)$$

The updating algorithm is the L_{R-I} algorithm [1] and is as follows.

$$p_{ij}(k+1) = p_{ij}(k) + br(k)(1 - p_{ij}(k)), \quad \alpha_i(k) = v_{ij}$$
$$p_{ij}(k+1) = (1 - br(k))p_{ij}(k), \quad \alpha_i(k) \neq v_{ij} \tag{9}$$

for all automata i which belong to an activated subunit, that is, a subunit which is in the activated path. All those automata which do not belong to subunits in the activated path are not updated. $r(k)$ is the SRS from the environment in response to the output of the system at instant k. b is the learning parameter and $0 < b < 1$. This is fixed for every automaton but can vary from automaton to automaton without affecting the results obtained.

III. Analysis

In this section, the algorithm presented in the previous section is analyzed. It is first approximated by an ODE using weak convergence techniques and the resulting ODE is analyzed to show that the ODE (and hence the algorithm) converges to a solution of the optimization problem. To obtain the approximating ODE, a preliminary lemma is needed.

Definition 1: The drift Δp_{ij} of the jth component of p_i,the action probability vector of the ith automaton, is defined as

$$\Delta p_{ij} = E[p_{ij}(k+1) - p_{ij}(k) \mid p(k) = p].$$

Lemma 1: The drift Δp_{ij} for algorithm (9) is

$$\Delta p_{ij} = bp_{ij} \sum_{\xi} p_{i\xi} \left(\frac{\partial f}{\partial p_{ij}} - \frac{\partial f}{\partial p_{i\xi}} \right)$$

where $f(.)$ is as defined by (2).

Proof: The proof is quite lengthy and is therefore relegated to the Appendix. □

Remark 1: It is seen that Δp_{ij} is not directly a function of the time step k. Thus Δp_{ij} can be written as

$$\Delta p_{ij}(k) = bs_{ij}(p(k))$$
$$s_{ij}(p) = p_{ij} \sum_{\xi} p_{i\xi} \left(\frac{\partial f}{\partial p_{ij}} - \frac{\partial f}{\partial p_{i\xi}} \right). \tag{10}$$

For each $b > 0$, $\{p(k) : k \geq 0\}$ is a Markov process whose dynamics depend on b. Denote this dependence explicitly by writing $p^b(k)$. Define continuous time interpolations $P^b(t)$ of $p^b(k)$ by

$$P^b(t) = p^b(k) \quad \text{if} \quad t \in [kb, (k+1)b).$$

Algorithm (9) can be rewritten as

$$p^b(k+1) = p^b(k) + bG^b(p^b(k), \theta^b(k)) \tag{11}$$

where the superscript b denotes dependence on the learning parameter b. θ is composed of the SRS, the actions of the automata in the system, the outputs of all the units and the subunits and the context vector from the environment. The following theorem establishes the connection between the algorithm and the ODE.

Theorem 1: The sequence of interpolated processes $\{P^b(.) : b > 0\}$ converges weakly as $b \to 0$ to $z(.)$, where $z(.)$ satisfies the ODE

$$\frac{dz_{ij}}{dt} = s_{ij}(z), \quad z(0) = p(0). \tag{12}$$

Proof: The following conditions are satisfied by the algorithm (9), or equivalently (11).

1) $\{p(k), \theta(k-1) : k \geq 0\}$ is a Markov process.
2) The context vectors from the environment come from a compact subset of $\mathbb{R}^s$, for some s. The outputs of the units and subunits are from finite sets. The SRS takes values from the closed interval $[0, 1]$. Thus, $\theta(k)$ takes values from a compact metric space S.
3) The function $G(.,.)$ defined in (11) is bounded, continuous and independent of b.
4) Let B be a Borel subset of S, S as defined above. The one step transition function TF(.) is defined as

$$\text{TF}(\theta, 1, B \mid p)$$
$$= \text{Prob}\{\theta(k) \in B \mid \theta(k-1) = \theta, p(k) = p\}.$$

It is seen that TF(.) is independent of b, k and θ. Thus

$$\text{TF}(\theta, 1, B \mid p) \equiv \text{TF}(B \mid p).$$

5) $\text{TF}(\theta, 1, . \mid p)$ is its own unique invariant probability measure [6], since it is independent of θ. Denote this invariant probability measure by $M(p)$. As S is compact, the set of probability measures $\{M(p) : p \in \mathcal{F}\}$ is trivially tight.
6) $\int G(p, \theta')\text{TF}(\theta, 1, d\theta' \mid p)$ is independent of θ and continuous with respect to p.

Then by [7, Ch. 5, Theorem 2], the sequence $\{\mathbf{P}^b(.) : b > 0\}$ converges weakly, as $b \to 0$, to the solution of the ODE

$$\frac{dz}{dt} = s(z), \quad z(0) = p(0)$$

where

$$s(z) = E^z[G(z, \theta]$$

and E^z denotes expectation with respect to the invariant measure $M(z)$. Thus,

$$s(z) = \int G(z, \theta)\mathrm{TF}(d\theta \mid z)$$

where $s(.)$ is the vector whose components are the s_{ij}'s as defined by (10). □

From now on, the ODE will be represented in terms of p rather than z. This is to facilitate comparison between the ODE and the optimization problem. This is an abuse of notation as p is a discrete time variable. The same variable will be used to denote the continuous time variable for convenience.

The ODE (12) is not a gradient ascent as there are constraints on p. It can however be shown that $f(p(t))$ increases monotonically along the ODE paths.

Lemma 2: Along the paths of the ODE (12), $(df/dt)(q) \geq 0$. Moreover, $(df/dt)(q) = 0$ iff q is an equilibrium point of the ODE.

Proof: In this proof, k is used as a subscript and has nothing to do with the k used in algorithm (9) as a discrete time step. Using the chain rule

$$\begin{aligned}\frac{df}{dt}(q) &= \sum_i \sum_j \frac{\partial f}{\partial p_{ij}}(q) s_{ij}(q) \\ &= \sum_{i,j,k} q_{ij} q_{ik} \left[\frac{\partial f}{\partial p_{ij}}(q) - \frac{\partial f}{\partial p_{ik}}(q)\right] \frac{\partial f}{\partial p_{ij}}(q)\end{aligned}$$

by substituting for $s_{ij}(.)$ using (10). By noting that there are precisely two terms in which $q_{ij}q_{ik}$ is the multiplying factor, the above expression for (df/dt) can be simplified. For the remainder of the proof let $d_{ij} = (\partial f)/(\partial p_{ij})(q)$. Then,

$$\begin{aligned}\frac{df}{dt}(q) &= \sum_{i,j,k<j} q_{ij} q_{ik} [(d_{ij} - d_{ik}) d_{ij} + (d_{ik} - d_{ij}) d_{ik}] \\ &= \sum_{i,j,k<j} q_{ij} q_{ik} (d_{ij} - d_{ik})^2 \geq 0\end{aligned}$$

since the q_{ij}'s are all nonnegative, being probabilities. It is further seen that for $(df/dt)(q) = 0$, each term in the above summation should be zero. Thus $(df/dt)(q) = 0$ iff

$$q_{ij} q_{ik} (d_{ij} - d_{ik})^2 = 0 \quad \forall i, j, k.$$

This implies that $(df/dt)(q) = 0$ iff

$$q_{ij} q_{ik} (d_{ij} - d_{ik}) = 0 \quad \forall i, j, k \tag{13}$$

since $x^2 = 0$ is equivalent to $x = 0$. Summing the above equation over k, for fixed i and j,

$$\sum_k q_{ij} q_{ik} (d_{ij} - d_{ik}) = 0.$$

The LHS of the above equation is the expression for (dq_{ij}/dt). Thus, q is an equilibrium point of the ODE. It is trivially true that $(df/dt)(q) = 0$ if q is an equilibrium point of the ODE. Thus, $(df/dt)(q) = 0$ iff q is an equilibrium point of the ODE. □

Remark 2: It is seen that $f(p(t))$ is a strictly increasing function of t along the solutions of the ODE. This is a property like absolute expediency [1] but while absolute expediency holds at every step, the above lemma shows that a similar property holds for algorithm (9) with high probability for small enough learning parameter b. It has recently been shown that a different kind of system is absolutely expedient if the learning parameter b is small enough [8]. This system is not used in a network but is a game of units with a different payoff structure. No asymptotic analysis is carried out there but it should be possible to derive asymptotic results there using techniques similar to those presented here.

The next lemma shows the relationship between the equilibrium points of the ODE and the points satisfying the FONKT conditions for the optimization problem.

Lemma 3:

1) If q satisfies FONKT conditions of the optimization problem (2) given in Fact (1) then q is an equilibrium point of the ODE (12).
2) If q is an equilibrium point of the ODE (12) and does not satisfy the FONKT conditions, then q is unstable.

Proof:

1) Let $d_{ij} \equiv (\partial f/\partial p_{ij})(q)$. If $q_{ij} = 0$ or $q_{ik} = 0$ then

$$q_{ij} q_{ik} (d_{ij} - d_{ik}) = 0.$$

Let both q_{ij} and q_{ik} be strictly positive. Then the FONKT conditions imply $\lambda_{ij} q_{ij} = \lambda_{ik} q_{ik} = 0$. From this it is seen that $\lambda_{ij} = \lambda_{ik} = 0$. Then we get $d_{ij} + \mu_i = d_{ik} + \mu_i = 0$. From this we obtain $d_{ij} = d_{ik} = -\mu_i$. Thus, again we get $q_{ij} q_{ik} (d_{ij} - d_{ik}) = 0$. This condition therefore holds for all i, j and k when q satisfies the FONKT conditions. Summing over k, for any fixed i, j

$$q_{ij} \sum_k q_{ik} (d_{ij} - d_{ik}) = 0.$$

From (10), it is seen that the above condition is $s_{ij}(q) = 0$ which is the same as $(dq_{ij}/dt) = 0$. Thus, as the above argument is for any i and j, q is an equilibrium point of the ODE.

2) k is again used as a subscript and $d_{ij} = (\partial f/\partial p_{ij})(q)$. Since q is an equilibrium point of the ODE, using the proof of Lemma 2,

$$q_{ij} q_{ik} (d_{ij} - d_{ik}) = 0$$

for all i, j, k. The FONKT conditions are given by Fact (1). Since $q \in \mathcal{F}$, the feasible region, (7, 8) are automatically satisfied. By choosing λ_{ij} such that $\lambda_{ij} = 0$ whenever $q_{ij} > 0$, (6) is also satisfied. Let

$$A_i = \{j : q_{ij} > 0\}.$$

Since $q \in \mathcal{F}$, $A_i \neq \emptyset$. For all $j \in A_i$, $\lambda_{ij} = 0$ so as to satisfy (4). Thus, for all $j \in A_i$, (4) reduces to

$$d_{ij} + \mu_i = 0$$

That is,

$$d_{ij} = -\mu_i \quad \forall j \in A_i. \tag{14}$$

For all $j \notin A_i$, the λ_{ij}'s can be calculated according to

$$\lambda_{ij} = -\mu_i - d_{ij}. \tag{15}$$

There are two ways the FONKT conditions can be violated. One is if at least one of the λ_{ij}'s is strictly negative, violating (6). The second is if the d_{ij}'s differ on A_i, so that the μ_i cannot even be calculated. These two cases are considered separately.

Case 1: In this case, it is assumed that μ_i is well defined, but that there exists a $\lambda_{ij} < 0$. Without loss of generality, let $\lambda_{11} < 0$. It will be shown that q is unstable. Shift the equilibrium point q to the origin by the transformation

$$\epsilon = p - q.$$

Then,

$$\frac{d\epsilon_{11}}{dt} = \frac{dp_{11}}{dt} = p_{11} \sum_j p_{1j} \left\{ \frac{\partial f}{\partial p_{11}}(p) - \frac{\partial f}{\partial p_{1j}}(p) \right\}.$$

For all $j \in A_1$, $q_{1j} > 0$ and $q_{1j} = 0$ otherwise. Since λ_{11} has been defined by (15), it has been assumed that $1 \notin A_1$. Thus $q_{11} = 0$ and

$$\frac{d\epsilon_{11}}{dt} = \epsilon_{11} \sum_j p_{1j} \left\{ \frac{\partial f}{\partial p_{11}}(p) - \frac{\partial f}{\partial p_{1j}}(p) \right\}.$$

To prove instability we consider a linear approximation and hence only the constant contribution from the summation term of the above equation is needed. Thus, to a linear approximation, and noting that $q_{1j} = 0$ for all $j \notin A_1$,

$$\frac{d\epsilon_{11}}{dt} = \epsilon_{11} \sum_{j \in A_1} q_{1j}(d_{11} - d_{1j}).$$

Substituting for d_{11} using (15) and for d_{1j}, $j \in A_1$, using (14),

$$\begin{aligned}\frac{d\epsilon_{11}}{dt} &= \epsilon_{11} \sum_{j \in A_1} q_{1j}(-\mu_1 - \lambda_{11} + \mu_1) \\ &= -\lambda_{11}\epsilon_{11}.\end{aligned} \tag{16}$$

As λ_{11} has been assumed to be negative, $-\lambda_{11}$ is positive. Thus the linearized differential (16) for ϵ_{11} is unstable and hence q is unstable for the ODE (12).

Case 2: In this case, it is assumed that μ_1 cannot be defined uniquely. Thus, there exist i,j in A_1 such that $(\partial f/\partial p_{1i})(q) \neq (\partial f/\partial p_{1j})(q)$. But, since both q_{1i} and q_{1j} are greater than zero, (13) implies that $(\partial f/\partial p_{1j})(q)$ and $(\partial f/\partial p_{1i})(q)$ have to be equal for q to be an equilibrium point of the ODE. Thus, this case cannot arise when q is an equilibrium point. This completes the proof of the lemma. □

If q satisfies the FONKT conditions, the above lemma does not say anything about its stability or otherwise. For the algorithm to be useful, the equilibrium points which are stable should correspond to solutions of the optimization problem and vice-versa. It may not be possible to prove such a result in such generality, but partial results have been obtained [9]. These results also indicate that the correspondence between the equilibrium points which are stable and solutions of the optimization problem would be violated only in pathological cases. The partial results are fairly technical in nature and to simplify the analysis, the following assumption is made. This is a fairly strong assumption, but it has been made so that the main argument of the paper does not get obscured by technicalities. The interested reader is refereed to [9] for detailed partial results.

Assumption 1: Any maximum of the optimization problem (2) is an isolated maximum.

Lemma 4: The following are true under Assumption 1.

1) If q is a locally asymptotically stable equilibrium point of the ODE (12), it is an isolated maximum of the optimization problem (2).
2) If q is an isolated maximum of the optimization problem, it is a stable equilibrium point of the ODE.

Proof:

1) Let q be locally asymptotically stable. There exists a neighborhood Nb of q such that if $p(0) \in \text{Nb}$, $p(t) \to q$. Consider any $u \in \text{Nb} - q$. Let $p(0) = u$. As $p(t) \to q$, $(df/dt)(u) > 0$, else u would be an equilibrium point of the ODE and $p(t)$ would not converge to q. Thus, $f(u) < f(q)$, using Lemma 2. That is for all $u \in \text{Nb} - q$, $f(u) < f(q)$. Thus q is a strict local maximum. If there is any other maximum of f in $\text{Nb} - q$, that point would satisfy the FONKT conditions and would thus be an equilibrium point of the ODE (12), by Lemma 3. This is not possible as q is locally asymptotically stable with at least Nb as its attractive bowl. Thus there is no maximum of f in $\text{Nb} - q$ and therefore q is an isolated local maximum of f.
2) Let q be an isolated local maximum of the optimization problem. There is a neighborhood Nb of q such that $f(p) < f(q)$ for all $p \in \text{Nb} - q$. On Nb, define a Lyapunov function

$$V(p) = f(q) - f(p)$$

$V(q) = 0$ and $V > 0$ on $\text{Nb} - q$. Along the ODE (12)

$$\frac{dV}{dt} = -\frac{df}{dt} \leq 0$$

by Lemma 2. Thus by [10, Theorem 3.3, Ch. VIII], q is a (locally) stable attractor of the ODE. □

Remark 3: As $(df/dt) \geq 0$, it is easily seen by an application of LaSalle's Theorem [11] that the solutions of the ODE converge to the set $\mathcal{E}$ where

$$\mathcal{E} = \left\{ p : \frac{df}{dt} = 0 \right\}$$

which is exactly the set of equilibrium points of the ODE by Lemma 2. Thus, every solution of the ODE converges to the set $\mathcal{E}$. Combining this with Lemma 4, we can say that the ODE (and hence the algorithm) solves the optimization problem; that is, solutions of the ODE converge to local maxima of the optimization problem.

IV. Three Layer Network for Pattern Recognition

In this section, the three layer network for pattern recognition [12], [13] is considered as an application of the algorithm described in the previous sections. After showing how a pattern recognizer can be described in terms of a connectionist system of units considered so far, specific examples are taken up for simulation to verify the results of the previous section.

One of the standard ways of implementing pattern classifiers is by using discriminant functions [14]. Any surface which divides the feature space (assumed compact) can be approximated by a piecewise linear function [12], [13]. Any piecewise linear surface can be implemented using linear threshold units in a three layer network. In such a network, the first layer units learn hyperplanes. Units in the second layer perform the AND operation on the outputs of selected units in the first layer and thus learn convex sets with piecewise linear boundaries. The final layer performs an OR operation on the second layer outputs. The formal notation for the three layer network is developed below.

The first layer is composed of M units. The network is thus capable of learning a surface with M hyperplanes. The second layer has N units. Thus at most N separate convex pieces can be learned. The feature vector is $x(k)$ where

$$x(k) = (x_1(k), \ldots, x_m(k))^t.$$

Denote by U_i the ith first layer unit. $x_m(k) \equiv 1$. The number of features measured is $(m-1)$ and x_m is an extra component to facilitate thresholding. Thus, the space in which the classification is done is $\mathbb{R}^{m-1}$. Each U_i implements a hyperplane in $\mathbb{R}^{m-1}$ which can be represented by a m dimensional real vector, the mth component representing the threshold value. Therefore a first layer unit is composed of m automata and each automaton learns one of the components of the vector defining the hyperplane. Denote by A_{ij} the jth automaton of U_i. $p_{ij}(k)$ is the action probability vector of A_{ij} at instant k and $p_{ijs}(k)$ its sth component. v_{ij} is the vector of actions of A_{ij}. As each automaton has to learn a component of the weight vector, v_{ij} is composed of real scalars. This discretization of weights does not necessarily cause a loss in performance, as actions can be finely discretized to allow for the learning of the optimal discriminant function. Let $\alpha_{ij}(k)$ be the action of A_{ij} and y_i(k) the output of U_i, both at instant k. Then,

$$\text{Prob}\{\alpha_{ij}(k) = v_{ijs}\} = p_{ijs}(k)$$

$$y_i(k) = us\left(\sum_j \alpha_{ij}(k) x_j(k) \right).$$

where $us(.)$ is the unit step function.

Let V_i be the i^{th} second layer unit. It has connections with $n(i)$ first layer units. Thus, it can learn a convex set bounded by at most $n(i)$ hyperplanes. V_i is composed of a team of $n(i)$ learning automata and the jth automaton of V_i is B_{ij}. Each B_{ij} has two actions, 0 and 1. Let $z_{ij}(k)$ be the action of B_{ij}. q_{ij} is the probability associated with action 1 of B_{ij}. Then,

$$\text{Prob}\{z_{ij}(k) = 1\} = 1 - \text{Prob}\{z_{ij}(k) = 0\} = q_{ij}.$$

Let $a_i(k)$ be the output of V_i. $a_i(k)$ is the AND of all those inputs which are connected to V_i and are activated, that is $z_{ij}(k) = 1$. The output of the third layer unit is the OR of all the second layer outputs. There is no learning involved in the third layer. This is because two class pattern recognition problems are considered. In general, there would be more than one unit in the third layer and each unit would have to learn the connection weights from the second layer units. Denote the output of the network by z. $z = 1$ denotes class 1 and $z = 0$ class 2. $P(e)$ is the probability of misclassification,

$$\begin{aligned} P(e) = \{&\text{Prob(class1)Prob}(z = 0 \mid \text{class1}) \\ &+ \text{Prob(class0)Prob}(z = 1 \mid \text{class0})\}. \end{aligned}$$

The error signal is r, where

$$r = \begin{cases} 1 & \text{if } x \text{ is classified correctly} \\ 0 & \text{if } x \text{ is classified incorrectly} \end{cases}$$

r is used as the SRS to the network. The probability of misclassification, if the internal state is p and the feature vector is x, is

$$\begin{aligned} &P(e \mid p) \\ &= \{\Pr(\text{class 1})\Pr(z = 0 \mid p, x \in \text{ class 1}) \\ &\quad + \Pr(\text{class 0})\Pr(z = 1 \mid p, x \in \text{ class 0})\} \\ &= \{\Pr(\text{class 1})\Pr(r = 0 \mid p, x \in \text{ class 1}) \\ &\quad + \Pr(\text{class 0})\Pr(r = 0 \mid p, x \in \text{ class 0})\} \\ &= 1 - E[r \mid p]. \end{aligned}$$

Thus, $P(e \mid p) = 1 - E[r \mid p]$ and therefore minimizing the probability of error with the given setup is the same as maximizing $E[r \mid p]$.

The units in these networks fall into the category analyzed in this paper since each unit is a team of automata which is a simpler version of the units described in the previous sections. Thus, all the results of the previous sections hold for this network. Examples along with simulation results of the three layer networks are presented below. The examples considered are with overlapping samples and are hence not perfectly separable. The solutions given correspond to the minimum probability of misclassification.

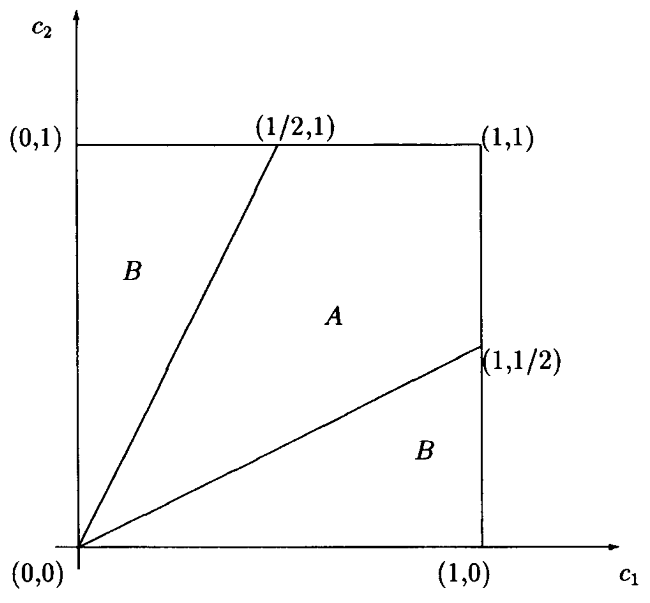


Fig. 4. Optimal regions for example 1.

Example 1: In this example, the feature vectors from the environment arrive uniformly from the set $[0,1] \times [0,1]$. Each feature vector has a non zero probability of belonging to either of two classes 1 and 2. In some regions of the feature space, the probability of belonging to class 1 is higher and vice versa in other regions. The problem of pattern recognition is thus one of overlapping classes and reduces to the classification of such regions. Using the terminology of Fig. 4, in region A

$$\text{Prob}\{x \in \text{class1}\} = 1 - \text{Prob}\{x \in \text{class2}\} = 0.9$$

and in region B

$$\text{Prob}\{x \in \text{class1}\} = 1 - \text{Prob}\{x \in \text{class2}\} = 0.1.$$

Thus the optimal class is 1 in region A and is 2 in region B. The discriminant function to be learned is

$$[2x_1 - x_2 > 0] \text{ AND } [-x_1 + 2x_2 > 0]$$

where $x = (x_1, x_2)^t$ is the feature vector. In this example, no extra component is added since it is assumed known that the threshold is zero.

The network is made up of two first layer units, U_1 and U_2. There is one fixed second layer unit. The second layer unit performs the AND operation on the first layer unit outputs. Each first layer unit has two automata, each with four actions. These are the discretized values of the weights used to represent the hyperplanes. Table I gives details about these units. A_{ij} is the jth automaton of unit U_i. It should be noted that there are two sets of indices, $(2,4,4,2)$ and $(4,2,2,4)$, at which the global optimum is attained.

The value of the learning parameter was fixed at 0.005 for all the automata. Twenty simulation runs were conducted with the initial conditions being equal, that is, a probability of 0.25 for each action. The network converged to one of the two sets of optimal actions in every run. The average number of steps to reach a probability of 0.98 for the optimal action of every automaton was 10,922. The number of samples generated was

TABLE I
DETAILS OF THE FIRST LAYER UNITS OF EXAMPLE 1

Unit	Automaton	Actions				Optimal Action	
		1	2	3	4	Set 1	Set 2
U_1	A_{11}	-2	-1	1	2	4	2
U_1	A_{12}	-2	-1	1	2	2	4
U_2	A_{21}	-2	-1	1	2	2	4
U_2	A_{22}	-2	-1	1	2	4	2

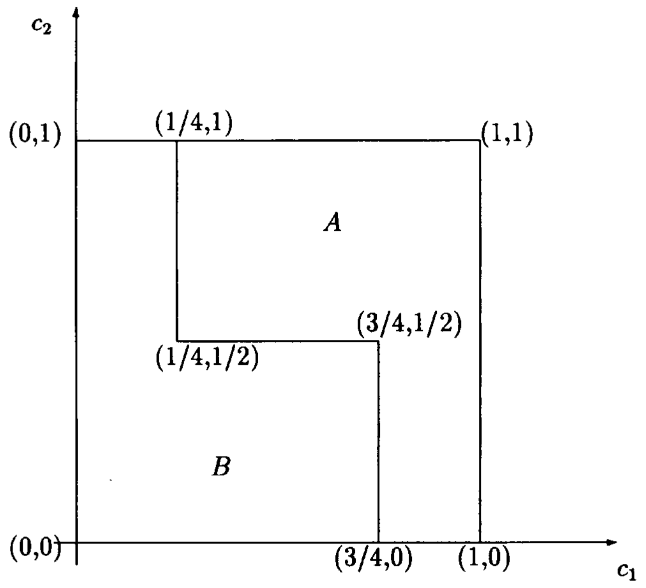


Fig. 5. Optimal regions for example 2.

500 and a vector from this set was chosen at random at each instant. Thus, it is not necessary to have a large number of samples. The algorithm can be used with a fixed number of classified samples. The average number of steps with a new sample being generated at each instant was 10,425.

Example 2: In this example, feature vectors from the environment arrive uniformly from the set $[0,1] \times [0,1]$. The discriminant function to be learned is shown in Fig. 5. In region A,

$$\text{Prob}\{x \in \text{class1}\} = 1 - \text{Prob}\{x \in \text{class2}\} = 0.8$$

and in region B

$$\text{Prob}\{x \in \text{class1}\} = 1 - \text{Prob}\{x \in \text{class2}\} = 0.2.$$

The discriminant function to be learned is

$$[x_1 - .75 > 0] \text{ OR } \{[x_1 - .25 > 0] \text{ AND } [x_2 - .5 > 0]\}$$

and the feature vector is $x = (x_1, x_2, x_3)^t$ where $x_3 \equiv 1$ is added for thresholding.

The network is composed of three first layer units U_1, U_2 and U_3. It has two second layer units V_1 and V_2. The structure is shown in Fig. 6. Both the second layer units get inputs from all the three first layer units.

Each first layer unit automaton has five actions which are the discretized values of the weights which are used to represent the hyperplanes. Each first layer unit has three automata since x has three components. Table II gives details of the first layer units and Table III details of the second layer units.

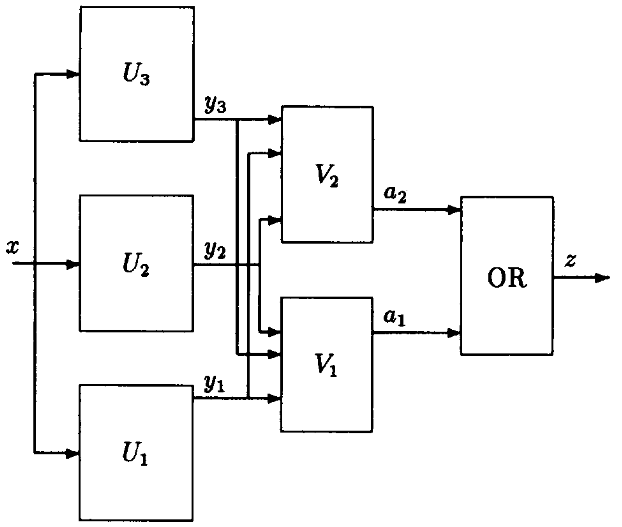


Fig. 6. Structure for example 2.

TABLE II
DETAILS OF THE FIRST LAYER UNITS OF EXAMPLE 2

Unit	Automaton	Actions					Optimal
		1	2	3	4	5	Action
U_1	A_{11}	-1	-0.5	0	0.5	1	5
U_1	A_{12}	-1	-0.5	0	0.5	1	3
U_1	A_{13}	-1	-0.5	-0.25	0	0.5	3
U_2	A_{21}	-1	-0.5	0	0.5	1	3
U_2	A_{22}	-1	-0.5	0	0.5	1	5
U_2	A_{23}	-1	-0.5	0	0.5	1	2
U_3	A_{31}	-1	-0.5	0	0.5	1	5
U_3	A_{32}	-1	-0.5	0	0.5	1	3
U_3	A_{33}	-1	-0.75	0	0.5	1	3

Thirty three simulation runs were conducted. For convenience, a new sample vector was generated at each instant. Convergence to the optimal set of actions was observed in thirty cases. The average to reach a probability of 0.99 for all the optimal actions was 62,600 steps. This excludes the three wrong convergences. Since the results in section 3 are local convergence results, an initial bias toward the optimal actions was required. For the first layer automata, the initial probability of the optimal action was set at 0.6 and for the second layer at 0.7. The learning parameter for the first layer was set at 0.005 and 0.001 for the second layer.

Example 3: In this example, a simplified version of the iris data [14] was considered. This is a three class four feature problem. The three classes are iris-setosa, iris-versicolor and iris-virginica. Of these, setosa is linearly separable from the other two. As two class pattern recognition is being considered in this section, setosa was ignored and the problem was reduced to classify versicolor and virginica.

The data consists of 50 samples from each class with the correct classification. Simulations were conducted using the L_{R-I} algorithm and the standard Backpropagation with Momentum (BPM) algorithm.

The network used for the L_{R-I} case consisted of 9 first layer units and 3 second layer units. Each first layer automaton had 9 actions which were $\{-4, -3, -2, -1, 0, 1, 2, 3, 4\}$. Uniform initial conditions were used.

TABLE III
DETAILS OF THE SECOND LAYER UNITS OF EXAMPLE 2

Unit	Automaton	Optimal Action
V_1	B_{11}	1
V_1	B_{12}	1
V_1	B_{13}	0
V_2	B_{21}	ANY
V_2	B_{22}	1
V_2	B_{23}	1

TABLE IV
SIMULATION RESULTS FOR IRIS DATA

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

* RMS error for BPM and probability of misclassification for L_{R-I}.

BPM was used with standard feedforward neural networks of 3 and 4 layers (excluding the input layer which just feeds the feature vector to various units). The 3 layer network had 9 and 3 units in the first two layers and one unit in the last layer. The 4 layer network had 8 units in the first three layers and one unit in the fourth layer. Initial conditions were generated randomly.

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. Thus with 40% noise each sample had a probability of 0.6 of correct classification and 0.4 of wrong classification as opposed to 1 and 0 in the noise free case.

The learning parameters used for the L_{R-I} network were 0.005 in the first layer and 0.002 in the second layer.

The momentum term for the BPM algorithm was set at 0.9 and various values of the learning parameter were considered and the best results are reported. The results are summarized in Table IV. The averages are over 10 runs each. *It should be noted that the error for the BPM is the RMS Error while that for the L_{R-I} is the probability of misclassification. They cannot be compared, but the performances would be about the same at the given values.*

Simulation results show that in the noise-free case BPM algorithm does converge faster. However, it fails to converge even when 20% noise is added. The learning automata model is slower in the noise free case, but continues to converge even with 40% noise and there is only a gradual degradation of performance in terms of speed and probability of correct classification. The superior performance of the connectionist system of learning automata in the noisy situation is evident.

Remark 4: Example 1 converges to the global maximum without any bias in favor of the global set of actions while

Example 2 requires a bias. This is because Example 2 is more complicated than Example 1. Both the examples have local maxima, but those in Example 1 are better separated than those in Example 2. Also, if the entire system were to be reduced to a single automaton, Example 1 would consist of $4^4 = 256$ actions while Example 2 would consist of $5^9 \times 2^3 \approx 15 \times 10^6$ actions. Since the results in this paper are local convergence results, initial conditions have to be sufficiently close to the global maximum for convergence to the global maximum. This is shown by Example 2 where the problem requires a bias for convergence. A simpler problem, such as Example 1, requires little or no bias.

In Example 3, it is seen that the standard BPM algorithm does not converge in the noisy case whereas the Learning Automata network does. In the no noise case, BPM does perform better, which is to be expected. Only a few simulations are reported and these are indicative of the results which can be expected. Also, the simulations depend on the structure of network used for simulation. This problem is not considered here.

V. Conclusion

In this paper, a model for connectionist systems based on teams of learning automata has been developed and analyzed. The model is fairly general and many different problems fit into this framework.

The three layer network for pattern recognition has been analyzed as an example of such a connectionist system and simulations show that these do have the local convergence properties as shown by analysis. It is also seen that the algorithm converges even if it has only a finite number of samples to work with. Since the proofs in this paper are local convergence results, global convergence cannot be expected in all cases. If, however, the problem is characterized by maxima which are well separated, global convergence is possible. Also, it is seen that this algorithm is robust to noise and works well in noisy cases where the BPM algorithm fails to converge.

Even though the learning automaton as a single unit does not have the capability to handle context vectors, it is seen that the problem can be circumvented using teams of learning automata. The simplicity of the scheme is also preserved. The results of this paper can be directly extended to the case where each unit is composed of a feedforward network of subunits instead of the tree hierarchy considered here.

Convergence is established by replacing the discrete stochastic algorithm by an ODE whose long term behavior approximates the algorithm for low values of learning parameter. This technique can be extended to the case where the actions are from a compact set instead of a finite set. It can also be applied to analyze many other discrete connectionist algorithms.

Appendix

Proof of Lemma 1:

Let p_{ij} be the probability of choosing an action by the i^{th} automaton of the network. This automaton is assumed to belong to a subunit denoted by SU. SU belongs to the unit denoted U. x is the context vector from the environment. Context vectors from the environment arrive according to the probability distribution P_e.

p is the vector composed of the action probability vectors of all the automata. x' is the context vector to unit U under consideration. To calculate the drift Δp_{ij}, condition first on the context vector x from the environment and then x', the context vector to unit U. The unit U can then be studied in isolation, since the effect of other units on U is only through x'. Since the system is a feedforward network, the units in the same and later layers do not affect the functioning of the unit U. $Q(.,p,x)$ is the probability distribution of x', conditioned on p and x. Q is independent of p_{ij}, since the unit U has no effect on x', once p and x are known. In fact, the conditioning need not be on the entire p, but only on those components of p which are the action probability vectors of automata belonging to units in layers before unit U. Conditioning on x and x'(all conditioning is at the instant k),

$$\Delta p_{ij} = \int\int E[p_{ij}(k+1) - p_{ij}(k) \mid p, x, x'] \times Q(dx' \mid p, x)P_e(dx).$$

In unit U, condition further on that path which activates the subunit SU. The path is considered only until SU is activated. Call this portion of the path π'. The other paths can be ignored, since $p_{ij}(k+1) = p_{ij}(k)$ if SU is not activated. Thus

$$\Delta p_{ij} = \int\int E[p_{ij}(k+1) - p_{ij}(k) \mid p, x, x', \pi'] \times \Pr[\pi' \mid p, x']Q(dx' \mid p, x)P_e(dx).$$

The probability of choosing π' is independent of x since x affects U only through x'. It is also independent of p_{ij} since π' is the path only until SU is activated. Condition further on the action of the automaton which has p_{ij} as one of its action probability vector components. The action probability vector of the automaton under consideration is p_i. v_{iq} is the action corresponding to the probability p_{iq}. Thus

$$\Delta p_{ij} = \int\int \sum_q p_{iq}\{E[p_{ij}(k+1) - p_{ij}(k) \mid p, x, x', \pi', v_{iq}] \times \Pr[\pi' \mid p, x']\}Q(dx' \mid p, x)P_e(dx).$$

As the L_{R-I} algorithm is used for updating, using (9) in the above equation,

$$\Delta p_{ij} = \int\int bp_{ij} \sum_q p_{iq}\{E[r \mid p, x, x', \pi', v_{ij}] - E[r \mid p, x, x', \pi', v_{iq}]\} \times \Pr[\pi' \mid p, x']Q(dx' \mid p, x)P_e(dx). \quad (17)$$

We need to calculate $(\partial f/\partial p_{ij})$ and compare it with (17). For this, condition $f(p)$ on x and then x',

$$f(p) = \int\int E[r \mid p, x, x']Q(dx' \mid p, x)P_e(dx).$$

In the unit U, consider all the paths π that can be activated. If the path which is activated does not contain SU, then

$E[r \mid p, x, x', \pi]$ is independent of p_{ij}. Noting this, and conditioning only until the subunit SU is activated, that is, conditioning only by π',

$$\begin{aligned} f(p) &= \eta + \int\int E[r \mid p, x, x', \pi']\Pr[\pi' \mid p, x'] \\ &\quad \times Q(dx' \mid p, x)P_e(dx) \\ &= \eta + \int\int \sum_q p_{iq} E[r \mid p, x, x', \pi', v_{iq}]\Pr[\pi' \mid p, x'] \\ &\quad \times Q(dx' \mid p, x)P_e(dx) \end{aligned} \tag{18}$$

where η is independent of p_{ij}. For all q, $E[r \mid p, x, x', \pi', v_{iq}]$ is independent of p_{ij} because the automaton has already chosen its action. As mentioned before, $Q(. \mid p, x)$ is independent of p_{ij}. Therefore, differentiating (18) with respect to p_{ij},

$$\begin{aligned} \frac{\partial f}{\partial p_{ij}} &= \int\int E[r \mid p, x, x', \pi', v_{ij}]\Pr[\pi' \mid p, x'] \\ &\quad \times Q(dx'p, x)P_e(dx). \end{aligned} \tag{19}$$

Comparing (17) and (19), we have

$$\Delta p_{ij} = bp_{ij} \sum_q p_{iq} \left(\frac{\partial f}{\partial p_{ij}} - \frac{\partial f}{\partial p_{iq}} \right)$$

completing the proof of Lemma 1.

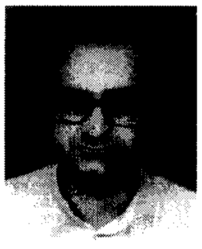
REFERENCES

[1] K. S. Narendra and M. A. L. Thathachar, *Learning Automata: An Introduction*. Englewood Cliffs, NJ: Prentice-Hall, 1989.
[2] M. A. L. Thathachar and V. V. Phansalkar, "Learning the global maximum with parameterised learning automata," to appear in *IEEE Trans. Neural Networks*.
[3] ——, "A feedforward network of learning automata for pattern classification," in *Proc. Int. Joint Conf. Neural Networks*, Singapore, Nov. 1991.
[4] G. P. McCormick, "Second order conditions for constrained minima," *SIAM J. Appl. Math.*, vol. 15, pp. 641–652, May 1967.
[5] W. Zangwill, *Nonlinear Programming: A Unified Approach*. Englewood Cliffs, NJ: Prentice-Hall, 1969.
[6] Y. V. Prohorov and Y. A. Rozanov, *Probability Theory*. Berlin: Springer-Verlag, 1969.
[7] H. J. Kushner, *Approximation and Weak Convergence Methods for Random Processes*. Cambridge, MA: MIT Press, 1984.
[8] K. S. Narendra and K. Parthasarathy, "Learning automata approach to hierarchical multiobjective analysis," *IEEE Trans. Syst. Man Cyber.*, vol. 21, no. 1, pp. 263–273, 1991.
[9] V. V. Phansalkar, "Learning automata algorithms for connectionist systems—Local and global convergence," Ph.D. Thesis, Dept. of Electrical Engineering, Indian Institute of Science, 1991.
[10] N. P. Bhatia and G. P. Szego, *Stability Theory of Dynamical Systems*. Berlin: Springer-Verlag, 1970.
[11] K. S. Narendra and A. Annaswamy, *Stable Adaptive Systems*. Englewood Cliffs: Prentice Hall, 1989.
[12] W. S. Meisel, *Computer Oriented Approaches to Pattern Recognition*. New York: Academic Press, 1972.
[13] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Mag.*, pp. 4–22, Apr. 1987.
[14] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.

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

He was a member of the faculty of the Indian Institute of Technology, Madras from 1961 to 1964. Since 1964, he has been with the Indian Institute of Science, where he is currently Professor in the Department of Electrical Engineering and the Chairman of the Division of Electrical Sciences. He has held visiting faculty positions at Yale University, New Haven, CT, Concordia University, Montréal, Québec, Canada, and Michigan State University, East Lansing. His current research interests include learning automata, neural networks, and fuzzy systems.

Dr. Thathachar is a fellow of the Indian National Science Academy, the Indian Academy of Sciences, and the Indian National Academy of Engineering.



V. V. Phansalkar obtained the Bachelor's degree in electrical engineering and the Master's in systems and control, both from IIT Bombay, in 1983 and 1985, respectively. He obtained the Ph.D. in electrical engineering from IISc Bangalore in 1992.

He was a project associate in the Department of electrical engineering, IISc from March 1992 to June 1994, and a Visiting Scientist at Citibank, Bombay, in July and August 1994. He is presently a member of the faculty at the V.E.S. Institute of Technology, Bombay. His current research interests include neural networks, learning automata, and applications of probability.