Associative Learning of Boolean Functions

SNEHASIS MUKHOPADHYAY AND M. A. L. THATHACHAR, SENIOR MEMBER IEEE

Abstract — **A cooperative game playing learning automata model is presented for learning a complex nonlinear associative task, namely learning of Boolean functions. The unknown Boolean function is expressed in terms of minterms, and a team of automata is used to learn the minterms present in the expansion. Only noisy outputs of the Boolean function are assumed to be available for the team of automata that *use* a variation of the fast converging estimator learning algorithm called the pursuit algorithm. A divide and conquer approach is proposed to overcome the storage and computational problems of the pursuit algorithm. Extensive simulation experiments have been carried out for six-input Boolean tasks. The main advantages offered by the model are generality, proof of convergence, and fast learning.**

I. Introduction

An ASSOCIATIVE learning task is one that requires the learning element to establish a connection between input and output. Let $X = \{x_1, \cdots, x_r\}$ be the finite set of inputs available to the learning element. For each input $x_r \in X$, the output y can belong to a set $Y = \{y_l, \cdots, y_n\}$. For an input $x \in X$ and output $y^x \in Y$, the connection (or association) between the input and output is assumed to be probabilistic and determined by a reward probability $d(x, y^x)$. The reward probability is a measure of the uncertainties involved in sensing inputs and outputs. The strongest association between x and an output y_m^x corresponds to the highest reward probability, i.e.,

$$d(x, y_m^x) = \max_{y^x \in Y} \{d(x, y^x)\}$$

The task of the learning element is to find out the action (or output) y_m^x for each $x \in X$.

Learning of a Boolean function is an important associative learning task. Here each input is a pattern vector consisting of signals ($\in \{0,1\}$) appearing on the input lines. The number of elements in the input set X is 2^m where m is the number of input lines. The output set Y is a binary set $\{0,1\}$. For each input pattern vector, the correct output is 0 or 1 depending upon which of them correspond to the highest reward probability. The objective of the learning task is to find the association between inputs and outputs, i.e., for each input $\underline{x} \in X$, it is required to find $y^{\underline{x}} \in \{0,1\}$ which maximizes the reward probability $d(\underline{x}, y^{\underline{x}})$.

The learning of associative Boolean tasks is significant in the context of neural models in cybernetics where the signals can be considered to be binary, and also in fault-tolerant computing.

We shall try to learn complex Boolean associative tasks through a popular learning model, namely, game playing team of learning automata.

Learning automata models represent a useful class of learning systems. A learning automaton interacts with a random environment in a feedback loop to improve its performance in some sense. In a particular iteration, it randomly selects an action out of a finite action-set according to a probability distribution over the action set. The environment randomly assigns reward or penalty for this action by sampling a distribution function corresponding to that action. The learning automaton updates its action probabilities depending upon the response from the environment according to a particular updating (reinforcement) scheme. Properties of learning automata have been extensively studied and applied to different learning tasks by different researchers **[1], [2], [3]**.

One important group behavior of learning automata is in the cooperative game. A team of automata involved in a cooperative game tries to maximize some common objective function through mutual cooperation and information exchange. In a particular play of the game, all members of the team receive identical payoffs from the common environment.

Many practical problems can be naturally modeled as cooperative games of learning automata. Some examples of such problems are learning optimal discriminant functions in pattern recognition **[4]** and relaxation labeling algorithms in image processing and computer vision [5]. We shall try to model the problem of learning Boolean tasks as a cooperative game of learning automata in a similar manner.

Barto **[6]** has proposed learning models consisting of networks of random threshold elements (called $A_{,,}$ elements) to learn some specific Boolean tasks. There are two apparent limitations in using Barto's models. Firstly, there is no proof of convergence for a network of $A_{,,}$ elements, though the learning capabilities of a single $A_{,,}$ element when faced with an associative task are summarized by a theorem proved by Barto and Anandan [7]. Secondly, it is

Manuscript received November 27, 1987: revised March 18, 1989.

S. Mukhopadyay was with the Department of Electrical Engineering, Indian Institute of Science, Bangalore, India. He is now with the Department of Electrical Engineering, Yale University, New Haven, CT 06520. M. A. L. Thathachar was with the Department of Electrical Engineering, Indian Institute of Science, Bangalore, India. He is now on leave at Department of Electrical and Computer Engineering, Concordia University, Montreal, Quebec, Canada.

IEEE Log Number 8929228.

0018-9472/89/0900-1008$01.00 ©1989 IEEE

not obvious how the model can be generalized to learn an arbitrary Boolean task.

The game playing automata model presented here overcomes these limitations. We also present simulation results for a variety of six-input multiplexer tasks. As will be seen from the simulation results, this model learns a task much faster than Barto's model.

II. Learning Automata Models

The model is based on expressing any Boolean function in terms of minterms [8], [9]. It is assumed that we can present all possible input patterns to the learning system during the process of learning.

Suppose we consider a Boolearn function f of n variables (inputs) $x_1, x_2, \cdots, x_n$.

A minterm is expressed as a product of n variables where each variable involves a district input variable either in an uncomplemented or complemented form.

The total number of minterms possible is given by,

$$m = 2^n \tag{1}$$

Suppose T_i denotes the ith minterm. Then f can be expressed as

$$f = \sum_{i=1}^{m} K_i T_i \tag{2}$$

where K_is are coefficients of the minterms and can have values 0 or 1, depending on whether the minterm is actually present or not. Thus if we learn the parameters $K_1, K_2, \cdots, K_m$, through repeated trials, the output function is learnt.

In the general game playing automata model, we employ one automaton to learn each of the parameters $K_1, K_2, \cdots, K_m$. Hence the number of automata will be 2^n. Each automaton will have two possible actions (corresponding to coefficient values 0 or 1). Note that the total number of automata increases exponentially with the number of input variables.

Each automaton chooses a value of the corresponding parameter in accordance with the action probabilities over the set $\{0,1\}$ of the possible values of the parameter. The set of values of all the parameters chosen by the team determines a specific value of the output according to the mapping f. A random environment compares this output with the correct output and assigns reward (corresponding to a payoff equal to unity) or penalty (corresponding to zero payoff) to the entire team according to an unknown random distribution. In practice this means that each set of actions by the team for a particular input pattern $\underline{x}$ generates noisy output $y^{\underline{x}}$. If $y^{\underline{x}}$ coincides with the desired output, a reward is given to the team and penalty otherwise. This amounts to an environment assigning reward with probability $d(\underline{x}, y^{\underline{x}})$ that is higher when $f(\underline{x})$ matches with the desired function output.

Hence the problem of learning a Boolean function can be posed as a cooperative game played by the previous team. It can be easily seen that only one set of values of the parameters, the set being denoted by $\{a_m^1, \ldots, a_m^N\}$ will be optimal resulting in maximum expected reward for a given Boolean problem. Following a learning algorithm (one such is given in the next section), the team is expected to converge to this optimal set that completely defines the particular Boolean function to be learned.

A. Number of Automata Required for Learning a Multiplexer Task

If we concentrate only on multiplexer tasks, the number of parameters (automata) can be reduced from exponential to polynomial. This is done as follows.

Let $x_1, \cdots, x_K$ be the address lines among the n-input lines. for full utilization of the address space of the multiplexer,

$$n = K + 2^K. \tag{3}$$

Now the output from a multiplexer can be expressed as

$$f = \bar{x}_1\bar{x}_2\cdots\bar{x}_{K-1}\bar{x}_K x_{K+1} + \bar{x}_1\bar{x}_2\cdots\bar{x}_{K-1}x_K x_{K+2} + \ldots + x_1 x_2 \ldots x_{K-1} x_K x_n. \tag{4}$$

Expressing $\bar{x}_i = (1 - x_i)$, (4) can be written as

$$\begin{aligned} f = {} & C_{11}x_{K+1} + C_{12}x_{K+2} + \ldots \\ & + C_{211}x_1x_{K+1} + C_{212}x_1x_{K+2} + \ldots \\ & + C_{221}x_2x_{K+1} + C_{222}x_2x_{K+2} + \cdots \end{aligned}$$

where $C_{q i_1 i_2 \ldots i_q}$ is the coefficient of the term containing the product of q input variables; $i_1 i_2 \cdots i_{q-1}$ give the indices of the first $(q-1)$ variables and $(K + i_q)$ gives the index of the last input variable.

In other words the output may consist of terms involving

1) only one of the variables from $x_{K+1}, \cdots, x_n$ (number of such terms $= n - K = r$ (say));
2) one address variable from $x_1, \cdots, x_K$ and one input variable from $x_{K+1}, \cdots, x_n$ (number of such terms $= K_{C_1} r$) (note that K_{C_s} represents the number of combinations possible for selecting s items out of K);

.

K addresses variables from $x_1, \cdots, x_n$ (number of such terms $= K_{C_K} r$). Hence the total number of parameters required

$$\begin{aligned} &= r(K_{C_0} + K_{C_1} + \ldots + K_{C_K}) \\ &= r \cdot 2^K \\ &= r \cdot r = r^2 \end{aligned}$$

Therefore the number of parameters (automata) is a polynomial (specifically, square) of the number of nonaddress input lines.

Note that for this model for multiplexers, each parameter can have one of three possible values $\{-1, 0, 1\}$.

III. PURSUIT ALGORITHM FOR A TEAM OF COOPERATIVE GAME PLAYING AUTOMATA

The most important aspect of learning automata theory is the design of efficient learning schemes (also called reinforcement schemes). Recently Thathachar and Sastry [5], [10] have proposed a fast converging reinforcement scheme, called Estimator scheme, which uses estimates of reward probabilities of the actions to converge to the optimal action with a high probability. The extension of this algorithm to a cooperative game playing automata team case is also available in [4], [5].

More recently the same authors have proposed a variation of the estimator scheme that is much simplified in expression but retains all the major characteristics of an estimator algorithm. This they termed as Pursuit Algorithm [11].

Following the analysis for cooperative game playing estimator algorithm that has been extensively done in [4], [5], we propose a game playing pursuit algorithm in the following way.

1) Let $A^1, A^2, \cdots, A^N$ be the N-automata that represent the N-players.
2) Let ith player have r_i strategies (actions), $i = 1, \cdots, N$.
3) $\{a_1^i, a_2^i, \cdots, a_{r_i}^i\}$ is the set of actions or strategies of A^i.
4) $[p_1^i(k), \cdots, p_{r_i}^i(k)]^T = p^i(k)$ is the action probability vector of A^i at k ($k = 0, 1, 2, \ldots$ represents discrete time instants).
5) $a^i(k)$ is the action selected by A^i at k.
6) $B(k)$ is the payoff at k. The payoff is common to all the players and takes values 0 or 1.
7) Let D be an N-dimensional matrix with elements,
$$d_{i_1 i_2 \cdots i_N} = Prob\left[B(k) = 1 | a^n(k) = a_{i_n}^n, n = 1, \cdots, N\right]$$
8) Let $d_{m_1 m_2 \cdots m_N} = \max_{i_1, \cdots, i_N}\{d_{i_1 i_2 \cdots i_N}\}$.
9) Each A^i maintains an estimate of D in $\hat{D}(k)$ as indicated in (6) in the algorithm next.
10) Let $\underline{E}^n = [E_1^n, \cdots, E_{r_n}^n]^T, n = 1, \cdots, N$, ($T$ denotes transpose) such that
$$E_j^n = i_s, \max_{\substack{i_s, 1 \leqslant s \leqslant N \\ s \neq n}} \{d_{i_1} \cdots i_{n-1} j i_{n+1} \cdots i_N\}$$
and let
$$\hat{E}_j^n(k) = i_s, \max_{\substack{i_s, 1 \leqslant s \leqslant N \\ s \neq n}} \{\hat{d}_{i_1}(k) \cdots i_{n-1} j i_{n+1} \cdots i_N\}$$
be an estimate of E_j^n at k.
11) Let H_n be the random index such that,
$$\hat{E}_{H_n}^n(k) = \max_i \{\hat{E}_i^n(k)\}$$

A. Algorithm

Let $a^n(k) = a_{i_n}^n, n = 1, \cdots, N$. Then updating of the probability vector over the action set of $A^n, n = 1, \cdots, N$ is as follows:

$$p^n(k+1) = p^n(k) + \lambda\left[e_{H_n} - \underline{p}^n(k)\right]. \tag{5}$$

Here $\underline{e}_{H_n}$ = unit vector with H_nth component 1 and $0 < \lambda < 1$. Also, the estimates are updated as follows:

$$R_{i_1 \cdots i_N}(k+1) = R_{i_1 \cdots i_N}(k) + B(k)$$
$$\hat{d}_{j_1 \cdots j_N}(k+1) = \frac{R_{j_1 \cdots j_N}(k+1)}{Z_{j_1 \cdots j_N}(k+1)} \tag{6}$$
$$j_n \in \{1, \cdots, r_n\}, 1 \leqslant n \leqslant N$$

$$\hat{E}_{i_-}^n(k+1) = \max_{\substack{i_s, 1 \leqslant s \leqslant N, \\ s \neq n}} \left\{\hat{d}_{i_1} \cdots i_{n-1} i_n i_{n+1} \cdots i_N(k+1)\right\}. \tag{7}$$

Here $Z_{i_1 \ldots i_N}(k)$ represents the count of number of times action set $\{a_{i_1}^1, \cdots, a_{i_N}^N\}$ has been selected by the team up to time instant k. $R_{i_1 \cdots i_N}(k)$ is the cumulative reward obtained by the team for selecting the action set $\{a_{i_1}^1, \cdots, a_{i_N}^N\}$ up to instant k. Thus $\hat{d}_{i_1 \cdots i_N}(k)$ is an estimate of the reward probability for the action set $\{a_{i_1}^1, \cdots, a_{i_N}^N\}$ obtained by the team on the basis of experience up to and including time instant k.

The convergence result for the algorithm can be obtained following a line similar to that for cooperative game playing estimator algorithm [4], [5], and can be summarized by the following theorem.

B. Theorem 1

In every stationary random environment, a team of cooperative game playing learning automata using the pursuit algorithm is $\in$-optimal.

That is, given any $\in > 0, \delta > 0$, there exist $\lambda^* > 0$ and $K_o < \infty$ such that

$$\text{Prob}\left[|p_m^n(k) - 1| < \in\right] > 1 - \delta$$

for all $k > K_o, 0 < \lambda < \lambda^*$ and $n = 1, \cdots, N$.

C. Proof

The proof of the theorem is given in the Appendix.

D. Comment

This means that the probability of selection of the best (optimal) action by each member of the team, can be made as close to 1 as desired by properly choosing λ and k.

In an actual learning exercise, the parameter λ is to be chosen carefully. Too small a value of λ will ensure convergence with higher probability, but will make the learning too slow to be useful. On the other hand large λ may help in increasing the speed of learning, but may also lead to false convergence. There is, at present, no theoretical basis for selecting a value for λ in general. In most of the simulation experiments, various λ values are tried out and a compromise value is worked out.

IV. SOLUTION OF HIGH DIMENSIONAL PROBLEMS

Straightforward application of the pursuit game playing model developed in the previous section for learning Boolean functions of relatively high dimensionality (e.g.,

six or more input Boolean function) is ruled out as can be seen from the reasoning given below.

The problem is a basic limitation (weakness) of pursuit (or any other estimator) algorithm. Since in an estimator algorithm we have to keep estimates of reward probabilities for each action-tuple, there is a severe storage and computation requirement as the number of players increases even though the number of actions per player is small. Actually the storage and computation requirement increases exponentially with number of players. The success of the estimator algorithms depends upon the accuracy of the estimates and each action set must be tried out at least once (maybe several times) before we can have confidence in the accuracy of the estimates. Hence when the dimension of the action-space increases, straightforward application of estimator algorithms is unattractive, both computationwise and storagewise.

As an illustration, let us try to apply the previously developed model to the problem of learning a Boolean task with six inputs. There will be $2^6 = 64$ players corresponding to as many minterms. Each player will have two actions corresponding to two possible values, $(0,1)$ of the coefficient of each minterm. Hence the dimension of the action space (D and $\hat{D}$ matrices) will be $2^{64} \cong 10^{19}$ which is too large to be manageable.

The way out may be decentralized learning using the linear reward inaction (L_{RI}) and related algorithms. But again the problem would be slowness and also optimality cannot be guaranteed.

The specific solution to this problem to be considered here is to break up the Boolean task of larger number of inputs to several smaller problems with smaller number of inputs. The idea is borrowed from [8].

Let us illustrate the concepts with a three-input problem. The general minterm expression for such a problem is given by

$$y = a_1\bar{x}_1\bar{x}_2\bar{x}_3 + a_2\bar{x}_1\bar{x}_2x_3 + a_3\bar{x}_1x_2\bar{x}_3 + a_4\bar{x}_1x_2x_3 + a_5x_1\bar{x}_2\bar{x}_3 + a_6x_1\bar{x}_2x_3 + a_7x_1x_2\bar{x}_3 + a_8x_1x_2x_3, \quad (8)$$

which can also be written as

$$y = \bar{x}_1\bar{x}_2(a_1\bar{x}_3 + a_2x_3) + \bar{x}_1x_2(a_3\bar{x}_3 + a_4x_3) + x_1\bar{x}_2(a_5\bar{x}_3 + a_6x_3) + x_1x_2(a_7\bar{x}_3 + a_8x_3) \quad (9)$$

or,

$$y = \bar{x}_1(a_1\bar{x}_2\bar{x}_3 + a_2\bar{x}_2x_3 + a_3x_2\bar{x}_3 + a_4x_2x_3) + x_1(a_5\bar{x}_2\bar{x}_3 + a_6\bar{x}_2x_3 + a_7x_2\bar{x}_3 + a_8x_2x_3). \quad (10)$$

We can attempt to learn the function in any of the three forms just given. If we employ the form given in (9), we will use the two input lines x_1 and x_2 for selecting different groups. There will be four such groups corresponding to four different possible patterns of x_1 and x_2 lines. Each group will use only x_3 to learn the function, and will consist of two automata each, corresponding to the coefficients of $\bar{x}_3$ and x_3 respectively. For example the group corresponding to $x_1 = 0$ and $x_2 = 0$ will be used to learn the parameters a_1 and a_2 in (9). If the four different possible groups are denoted by G_0, G_1, G_2, and G_3, then any input pattern with $x_1 = 0$ and $x_2 = 0$ will select G_0 whose players will learn the parameters a_1 and a_2. Since with each input pattern, one and only one group is selected, the members in each group need to keep estimates of the reward probabilities of the action-tuples consisting of actions of the members in their group only. In other words each group will learn its assigned task independently of the others.

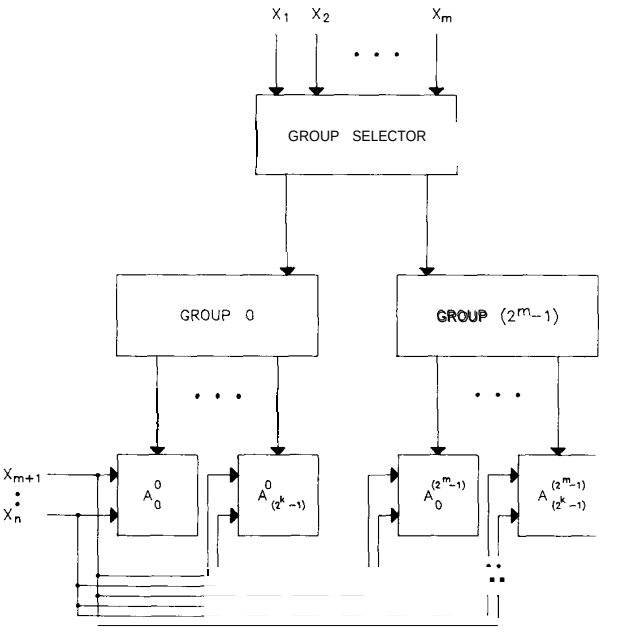


Fig. 1. **Illustration of group selector concept.**

Using (9), the size of the D matrix for each group is $2^2 = 4$. There will be four such groups, so the total storage and computation requirement for keeping estimates of the reward probabilities will be $0(2^2.4) = 0(16)$. Previously with eight automata in a single group we required $0(2^8)$ memory and computation time. Also note that by this process of breaking up, the total number of automata required is unchanged.

The breaking up can be done in various possible ways. For a three-input problem, we can have $0, 1, 2$, or 3 lines for selecting the groups. In actual practice any number of lines can be used as selectors provided the total storage and computation requirement is within manageable limits.

For a general n-input problem we can use the first m-lines (note that the exact identities of the m-lines out of n input lines are unimportant) for selection of groups, and the remaining $K = n - m$ lines for learning the partitions of the function corresponding to particular patterns on the first n-lines. The exact way of partitioning the input space through the choices of m and n can be done in any convenient manner.

The model can be schematically represented by Fig. 1.

TABLE 1

Task No.	Number of Runs Tried	Range of Number of Iterations Required for Convergence		Average Number of Iterations for Convergence
		Lowest	Highest	
1	10	10659	15398	12628
2	10	10748	15398	12641
3	10	10403	12937	11917

V. Simulation Results

For illustration of the ideas developed in previous sections, we concentrate on the six-input Boolean function learning problem.

Two possible partitions of the input lines between groups are considered. Out of six-input lines, initially four are used as group selector inputs. Subsequently, all six are used for this purpose. The first model will be referred to as 16×4 model (16 groups with four automata in each group). Similarly the second one will be referred to as 64×1 model.

The three tasks are defined as follows.

***Tusk** I*

Inputs x_6 and x_1 are used as address lines according to the following.

Pattern On x_6	Pattern On x_1	Selects	Line
0	0		x_2
0	1		x_3
1	0		x_4
1	1		x_5

Task 2

Pattern On x_4	Pattern On x_2	Selects	Line
0	0		x_1
0	1		x_3
1	0		x_5
1	1		x_6

Tusk 3

Pattern On x_5	Pattern On x_1	Selects	Line
0	0		x_4
0	1		x_3
1	0		x_2
1	1		x_6

TABLE 2
COMPARISON OF 16×4 AND 64×1 MODELS—TASK 1

A	Runs Tried	Average Number of Iterations Required for Convergence		Average Value of J	
		16×4	64×1	16×4	64×1
0.01	10	12628	28748	63.9	64.0
0.10	10	4324	4314	63.0	64.0

Initially a 16×4 model with x_6, x_5, x_4, and x_3 as inputs to the group selector is tried for each of the tasks 1, 2, and 3 with a fixed learning parameter value ($\lambda = 0.01$) to illustrate the generality of the model.

Then its performance is compared with the 64×1 model for learning Task 1 with various learning parameters.

The results of the simulation experiments are summarized in the following tables. J is the number of input patterns for which the correct. output is obtained from the system after convergence. Convergence of the team is assumed to have occurred when an action probability of each automaton attains a value of at least 0.9. J is used as a performance index with maximum possible value of 64, since there are 64 different input patterns.

Reward probabilities have been chosen as follows:

$$d(\underline{x}, y^{\underline{x}}) = 0.9$$

if $y^{\underline{x}}$ coincides with the desired function,

$$\text{output} = 0.1, \qquad \text{otherwise.}$$

VI. Conclusion

From the extensive simulations of the six-input multiplexer problems presented here, it is clear that cooperative teams of learning automata present attractive models for learning complex nonlinear associative tasks like the Boolean task.

Out of the 30 runs tried for three different tasks with $h = 0.01$, only one run resulted in convergence that corresponds to incorrect output for one out of 64 possible input patterns and correct output for the remaining 63. In all other runs the convergences were optimal in the sense that they resulted in correct output for all possible input patterns. This illustrates the stability of the model.

A quick glance at Table II gives one the idea that the 16×4 model is much faster in learning than the 64×1 model for smaller values of λ. But for higher values of λ, they behave almost identically. However it must be pointed out that'these impressions are based on simulation results alone. So far no way has been found for estimating the

speed of convergence for a team of automata, nor do we have any general results for studying the variation of speed of convergence with λ values.

The first, and in our opinion, the most important advantage of learning automata models is, as mentioned before, the generality of the model. For example though simulations were performed for multiplexer tasks, it can be easily seen that the same models are valid for any other six-input Boolean function.

Secondly the rigorous mathematical theory behind learning automata models guarantees convergence with very good confidence level without over-dependence upon simulation results.

The third advantage that we claim with learning automata models is the high speed of learning. Recently with the advent of estimator algorithms in learning automata, the speed of learning has improved very considerably. However these claims are based only on simulation results.

We can compare our models with the A_{∞} network model proposed by Barto [6]. He proposed empirical models to learn some specific Boolean tasks. Mathematical convergence proof for such models does not exist. On an average **133 149** trials were required to learn a six-input multiplexer task.

Our model is a general one and is based on sound convergence concepts. The number of iterations required for learning a six-input multiplexer task with high probability (with $\lambda = 0.01$) was 12628 for 16×4 model, which is more than ten times faster than Barto's model. We can increase the speed of learning further by proper choices of λ and learning model.

All these results point out the usefulness of learning automata models for learning complex Boolean tasks.

Appendix

The proof of Theorem **1** proceeds in two stages. First it will be shown that with sufficiently small λ, all action combinations are chosen enough number of times so that $\hat{d}_{m_1 \cdots m_n}(k)$ will remain the maximum element of $\hat{\underline{D}}(k)$ after a finite time. Then it is shown that this implies convergence of $p^n_{m_n}(k), n = 1 \cdots N$, as desired.

Lemma 1: For given constants $\delta > 0$ and $M < \infty$, there exists $\lambda^* > 0$ and $N_o < \infty$ such that under the pursuit algorithm, for all $\lambda \in (0, \lambda^*)$, prob [all N-tuples of actions are chosen by the team at least M times each before instant $k] > 1 - \delta$ for all $k \geqslant N_o$.

Proof: For each realization of the process $\{\underline{p}^1(k), \cdots, \underline{p}^N(k)\}$ we define a set of random variables

$$x(i_1, \cdots, i_N, k) = 1 \text{ if } a^n(k) = a^n_{i_n}(k), n = 1, \cdots, N,$$

$$= 0 \text{ otherwise.}$$

Let $\tilde{\underline{p}}^i(k)$ denote any specific realization of the process $\{\underline{p}^i(k), k \geqslant 0\}$. Now,

$$Ex(i_1, \cdots, i_N, k) = \prod_{n=1}^{N} \tilde{p}^n_{i_n}(k)$$

$$E[x(i_1, \cdots, i_N, k)]^2 = \prod_{n=1}^{N} \tilde{p}^n_{i_n}(k).$$

Hence,

$$\begin{aligned}\operatorname{Var} x(i_1, \cdots, i_N, k) &= E[x(i_1, \cdots, i_N, k)]^2 \\ &\quad - [Ex(i_1, \cdots, i_N, k)]^2 \\ &= \prod_{n=1}^{N} \tilde{p}^n_{i_n}(k)\left[1 - \prod_{n=1}^{N} \tilde{p}^n_{i_n}(k)\right].\end{aligned}$$

Now the random variables $\{x(i_1, \cdots, i_N, k), k \geqslant 0\}$ are uncorrelated because

$$\begin{aligned}&E[x(i_1, \cdots, i_N, k_1) \cdot x(i_1, \cdots, i_N, k_2)] \\ &= \prod_{n=1}^{N} \left(\tilde{p}^n_{i_n}(k_1) \cdot \tilde{p}^n_{i_n}(k_2)\right) \\ &= E[x(i_1, \cdots, i_N, k_1)] \cdot E[x(i_1, \cdots, i_N, k_2)].\end{aligned}$$

Let us define another random variable $Y(i_1, \ldots, i_N, k)$ that gives the number of times the n-tuple of actions $(i_1, \cdots, i_N)$ is chosen up to time k. Thus,

$$Y(i_1, \cdots, i_N, k) = \sum_{s=1}^{k} x(i_1, \cdots, i_N, s).$$

Since from (5) it follows that

$$\tilde{p}^n_{i_n}(k+1) \geqslant \tilde{p}^n_{i_n}(k)(1-\lambda), \text{ and}$$

$$\tilde{p}^n_{i_n}(k) \geqslant \tilde{p}^n_{i_n}(o)(1-\lambda)^k.$$

Now,

$$\begin{aligned}E[Y(i_1, \cdots, i_N, k)] &= \sum_{s=1}^{k} Ex(i_1, \cdots, i_N, s) \\ &= \sum_{s=1}^{k} \prod_{n=1}^{N} \tilde{p}^n_{i_n}(s) \\ &\geqslant \sum_{s=1}^{k} \prod_{n=1}^{N} \tilde{p}^n_{i_n}(o)(1-\lambda)^s \\ &= \left[\prod_{n=1}^{N} \tilde{p}^n_{i_n}(o)\right](1-\lambda)^N \frac{1-(1-\lambda)^{kN}}{1-(1-\lambda)^N}.\end{aligned}$$

Now,

$$\lim_{\lambda \to 0} (1-\lambda)^N \frac{1-(1-\lambda)^{kN}}{1-(1-\lambda)^N} = k \text{ by L'Hospital's rule.}$$

Again,

$$\lim_{\lambda \to 1} (1-\lambda)^N \frac{1-(1-\lambda)^{kN}}{1-(1-\lambda)^N} = 0.$$

Hence there exists a finite $\lambda(i_1, \cdots, i_N, k)$ such that for

$A < \lambda(i_1, \cdots, i_N, k)$

$$EY(i_1, \cdots, i_N, k) \geqslant \left[\prod_{n=1}^{N} \tilde{p}_{i_n}^n(o) \right] (k-1). \quad (11)$$

Again since $\{x(i_1, \cdots, i_N, k), k \geqslant \}$ are uncorrelated

$$\begin{aligned} \operatorname{Var} Y(i_1, \cdots, i_N, k) &= \operatorname{Var} \sum_{s=1}^{k} x(i_1, \cdots, i_N, s) \\ &= \sum_{s=1}^{k} \prod_{n=1}^{N} \left[\tilde{p}_{i_n}^n(s) \left[1 - \tilde{p}_{i_n}^n(s) \right] \right] \\ &\leqslant \sum_{s=1}^{k} \prod_{n=1}^{N} \tilde{p}_{i_n}^n(s) \\ &= E\left[Y(i_1, \cdots, i_N, k) \right]. \end{aligned} \quad (12)$$

Using Chebyshev inequality,

$$\begin{aligned} \operatorname{prob}[Y > M] &= 1 - \operatorname{prob}[Y \leqslant M] \\ &\geqslant 1 - \operatorname{prob}[|Y - EY| \geqslant EY - M] \\ &> 1 - \frac{\operatorname{Var} Y}{(EY - M)^2} \\ &> 1 - \frac{EY}{(EY - M)^2}. \end{aligned} \quad (13)$$

From **(11)**, $E[Y(i_1, \cdots, i_N, k)]$ increases linearly with k. Since M is fixed, there exists $k_o(i_1, \cdots, i_N) < \infty$ and $\lambda(i_1, \cdots, i_N, k)$ such that

$$\operatorname{prob}[Y > M] \geqslant 1 - \delta$$

for $k > k_o(i_1, \cdots, i_N)$ and $A < \lambda(i_1, \cdots, i_N)$.

Now, considering all N-tuples of actions, there exists $A^* > 0$ and $k_o < \infty$ such that

$$k_o = \max_{i_1, \cdots, i_N} \{k_o(i_1, \cdots, i_N)\}$$

$$A^* = \min_{i_1, \cdots, i_N} \{\lambda(i_1, \cdots, i_N)\}.$$

Hence,

$$\operatorname{prob}[Y(i_1, \cdots, i_N, k) > M] \geqslant 1 - \delta \quad (14)$$

$$\forall \lambda \in (0, \lambda^*), i = 1, \cdots, N \text{ and } k > k_o.$$

This completes proof of the lemma.

Lemma 2

Suppose there exist indices $q_n, n = 1, \cdots, N$ and a time instant $N_o < \infty$ such that $\hat{E}_{q_n}^n(k) > \hat{E}_{j_n}^n(k)$ for all $j, \neq q_n$ and $k > k_o$, then $p_{q_n}^n(k) \to 1$ w.p.1 as $k \to \infty$ for $n = 1, \cdots, N$. Consequently $\operatorname{prob}[|p_{q_n}(k) - 1| < \in] \geqslant 1 - S$ for all $k \geqslant N_o$.

Proof: Let us define

$$\Delta p_{q_n}^n(k) = E\left[p_{q_n}^n(k+1) - p_{q_n}^n(k) | Q(k) \right]$$

where $Q(k)$ is the state of the system consisting of $\{p(k), \hat{D}(k)\}$. Using (5) and assumptions of the lemma.

$$\Delta p_{q_n}^n(k) = \lambda\left(1 - p_{q_n}^n(k)\right) \geqslant 0 \text{ for all } k \geqslant N_o \quad (15)$$

Hence by Martingale convergence theorem and noting $A \neq 0$, it follows:

$$p_{q_n}^n(k) \to 1 \text{ w.p.1 as } k \to \infty \text{ for } n = 1, \cdots, N. \quad (16)$$

Hence the lemma.

Proof of Theorem **1**: Since $(a_{m_1}^1 \ldots a_{m_N}^N)$ is the optimal action N-tuple for the team and since by assumption this is unique, there exists one and only one maximum of $\underline{D}$ matrix given by $d_{m_1 \ldots m_N}$.

Since we are estimating $\underline{D}$ matrix by $\hat{\underline{D}}(k)$ given by (6), it is obvious $\hat{\underline{D}}(k)$ will converge to $\underline{D}$ if each of the N-tuples of actions is attempted infinitely often.

Also from (7) it follows $\hat{\underline{E}}^n$ converges to $\underline{E}^n$ if $\hat{\underline{D}} \to \underline{D}$. Now $E_{m_n}^n = \max_i \{E_i^n\}$.

Hence $A_{m_n} = E_{m_n}^n - \max_{i \neq m_n} \{E_i^n\} > 0$.

For each of the automata, the game playing algorithm is identical with a single pursuit algorithm with, expected reward vector E^n and estimated reward vector $\hat{\underline{E}}^n$.

Hence the law of large numbers when applied to the nth automaton gives the following result: given $\delta > 0$ there exists $N_{i_n}^n < \infty$ such that

$$\operatorname{prob}\left[|\hat{E}_{i_n}^n(k) - E_{i_n}^n| < \frac{A_n}{2} \right] > 1 - \delta. \quad (17)$$

for all k such that the action $a_{i_n}^n$ is attempted more than $N_{i_n}^n$ times (say, $\eta_{i_n}^n$ times).

By the definition of A_{m_n} for all $i_n \neq m$,

$$\operatorname{prob}\left[\hat{E}_{m_n}^n(k) > \hat{E}_{i_n}^n(k) \right] > 1 - \delta, \quad \text{for all } k.$$

Hence combining this result with Lemma **1** and Lemma 2, it follows that given an $\in > 0$, there exists $N_o < \infty$ and $A^* > 0$ such that

$$\operatorname{prob}\left[|p_{m_n}^n(k) - 1| < \in \right] \geqslant P_1 P_2 P_3 \geqslant (1 - \delta)^3$$

$\forall k > N_o, A \in (0, A^*)$ and all $n = 1, 2, \ldots N$ where,

$$P_1 = \operatorname{prob}\left[|p_{m_n}^n(k) - 1| < \in | \hat{E}_{m_n}(k) > \hat{E}_{i_n}(k), \right.$$

$$\left. i_n \neq m_n, \min\{\eta_{i_n}^n(k)\} > M^n \right]$$

$$P_2 = \operatorname{prob}\left[\hat{E}_{m_n}(k) > \hat{E}_{i_n}(k), i_n \neq m_n | \min\{\eta_{i_n}^n(k)\} > M^n \right]$$

$$P_3 = \operatorname{prob}\left[\min_{i_n} \{\eta_{i_n}^n(k)\} > M^n \right]$$

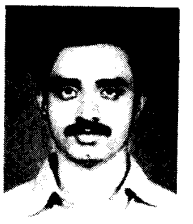
This proves Theorem **1**

REFERENCES

[1] V. I. Varshavskii and I. P. Vorontsova, "On the behaviour of stochastic automata with variable structure," *Automat. Remote Contr.*, vol. 24, pp. 321–333, Oct. 1963.

[2] K. S. Fu, "Learning control systems-review and outlook," *IEEE Trans. Automat. Contr.*, vol. AC-15, pp. 210–221, Apr. 1970.

[3] K. S. Narendra and M. A. L. Thathachar, "Learning automata-A survey," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-4, pp. 323–334, July/Aug. 1974.
[4] M. A. L. Thathachar and P. S. Sastry, "Learning optimal discriminant functions through a cooperative game of automata," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-17, No. 1, pp. 73–85, Jan./Feb. 1987.
[5] P. S. Sastry, "Systems of learning automata—Estimator algorithms and applications," Ph.D. Thesis, Department of Electrical Engineering, Indian Inst. of Science, Bangalore, India, June 1985.
[6] A. G. Barto, "Learning by statistical cooperation of self-interested neuron-like computing elements." Univ. of Massachusetts. Amherst. COINS Tech. Rep. 81-11, Apr. 1985, pp. 360–375.
[7] A. G. Barto and P. Anandan, "Pattern recognizing stochastic learning automata," *IEEE Trans. Syst. Man Cybern.*, vol. SMC-15, No. 3, May/June 1985.
[8] Z. Kohavi, *Switching and Finite Automata Theory.* New York: McGraw-Hill, 1970.
[9] H. Taub and D. Schilling, *Digital Integrated Circuits.* New York: McGraw-Hill/Kogakusha, 1981.
[10] M. A. L. Thathachar and P. S. Sastry, "A new approach to the design of reinforcement schemes for learning automata," *IEEE Truns. Syst. Man Cybern.*, vol. SMC-15, pp. 168–175, Jan./Feb. 1985.
[11] M. A. L. Thathachar and P. S. Sastry, "Estimator algorithms for learning automata," Platinum Jubilee Conf. Syst. and Signal Processing, Department of Electrical Engineering, Indian Inst. of Science, Bangalore, India, Dec. 11-13, 1986.



Snehasis Mukhopadhyay was born near Calcutta, India in 1964. He received the B.E. degree in electronics and telecommunications engineering from Jadavpur University, Calcutta, India in 1985 and the M.E. degree in systems science and automation from Indian Institute of Science, Bangalore, India in 1987. Currently he is a graduate student pursuing the Ph.D. degree in electrical engineering at Yale University, New Haven, CT.

His research interests are in learning systems including neural networks and learning automata, possible applications of learning systems in control problems, learning for path planning in robotics and application of learning algorithms in computer vision.

M. A. L. Thathachar (SM'79) was born in 1939 in Mysore City, India. He received the B.E. degree in electrical engineering from the University of Mysore in 1959 and the M.E. and Ph.D. degrees from the Indian Institute of Science, Bangalore, in 1961 and 1968, respectively.

From 1961 to 1964 he was a faculty member of the Indian Institute of Technology, Madras. Since 1964, he has been with the Department of Electrical Engineering, Indian Institute of Science, where he is currently Professor. He has held visiting positions at Yale University, New Haven, CT, and at Concordia University, Montreal, PQ, Canada. His research interests are in learning systems, adaptive control, and artificial intelligence.

Dr. Thathachar is a Fellow of the Indian Academy of Sciences and of the Indian National Science Academy.