GLOBAL CONVERGENCE OF FEEDFORWARD NETWORKS OF LEARNING AUTOMATA
V.V.PHANSALKAR AND M.A.L.THATHACHAR
DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF SCIENCE , BANGALORE 560 012

ABSTRACT: *A feedforward network composed of units of teams of parametrised learning automata is considered as a mode2 of a reinforcement learning system. The parameters of each learning automaton are updated using an algorithm consisting of a gradient following term and a random perturbation term. The algorithm is approximated by the Langevin equation and it is shown that it converges to the global maximum. The algorithm is decentralised and the units do not have any information exchange during updating. Simulation results on a pattern recognition problem show that reasonable rates of convergence can be obtained.*

1 INTRODUCTION:

Teams of Learning Automata have been used in various applications such as Pattern Recognition, Relaxation Labelling and Telephone Traffic Routing [NT89]. Many of these applications involve the optimisation, local or global, of an appropriate function. If local optima are required, standard algorithms such as L_R-I can be used. The only algorithms known to converge to global optima for teams of Learning Automata are the estimator algorithms [NT89]. In special cases, global algorithms can be designed, but these involve using some extra information about the problem which may not be readily available. Estimator algorithms have rapid rates of convergence and can be used to solve a large class of problems, but they need a large memory to store the estimates and this increases rapidly with the problem size. The number of estimates cannot be easily reduced.

In this paper, Learning Automata algorithms are presented for the reinforcement learning problem [NT89,W88] based on the Langevin equation [APZ85]. It is shown that these algorithms converge to the global optima. These algorithms are decentralised and also avoid the memory overheads of the estimator algorithms. The algorithm is based on the heat bath technique which consists of a gradient following part and a constant random walk to move out of local optima which are not the global optima.

In standard Learning Automata algorithms, the action probabilities are maintained as probability vectors [NT89]. For the algorithm presented here, the probabilities are calculated using a probability generating function and an internal state vector. This is to avoid the problem of adding noise to a probability vector which can destroy its property of all components remaining probabilities. Such a Learning Automaton is called a Parametrised Learning Automaton (PLA). A preliminary version of this paper was presented in [PT91].

2 NOTATION:

The complete system consists of a learning system, a

feedforward network in this case, interacting with an environment. The system functions as follows. At each instant, the environment presents a context vector to the learning system. Based on this vector and its internal state $\underline{u}$, the learning system chooses an action. The environment then emits a Scalar Reinforcement Signal (SRS) based on the context vector-action pair. (The SRS is assumed to be stochastic). Using this information, the learning system updates its internal state. A higher value of the SRS indicates better performance.

The environment is defined by the tuple $\langle C, A, R, D, P_c \rangle$. C is the set of context vectors, A is the (finite) set of actions and R the set of values the SRS can take. P_c is a probability distribution over C and the arrival of context vectors is governed by P_c . C is a compact subset of R^d , for some d . D is the set of reinforcement expectations,

$$D = \{d(a, \underline{c}) : a \in A, \underline{c} \in C\} \quad (2.1)$$

$$d(a, \underline{c}) = E[r | \text{action} = a, \text{context} = \underline{c}] \quad (2.2)$$

The network consists of units which are teams of PLAs. A single PLA is defined by the tuple $\langle AT, R, g, \underline{u}, T \rangle$ where AT is the set of outputs, R the set of SRS values and $\underline{u}$ the internal state of the unit. g is a function which generates the action probabilities given the internal state. Thus, for any a in AT ,

$$g(a, \underline{u}) = \text{Prob}\{\text{action} = a | \text{state} = \underline{u}\} \quad (2.3)$$

A single unit is made of a team of PLAs. Let the i^{th} PLA be defined by the tuple $\langle AT_i, R, g_i, \underline{u}_i, T_i \rangle$ where the subscripts stand for the i^{th} PLA. X is the set of all vectors which can be input to the unit and Y is its set of outputs. Let the action of the i^{th} PLA of the unit be a_i . a is the vector composed of the actions of all the PLAs of the unit and x is the context vector input to the unit. f is the output function of the unit and the output of the unit is y , where

$$y = f(a, x) \quad (2.4)$$

Let the number of units in the network be M . Without loss of generality, let each unit be composed of N PLA and each PLA have m actions. The results obtained hold without changes if N varies with each unit and m with each PLA. Denote by $\underline{u}_i$ the internal state vector of the i^{th} PLA of the network and g_i its probability generating function. The unit the PLA belongs to is not needed explicitly for the analysis. The internal state of the network is $\underline{u}$, composed of the internal state vectors of all the PLAs.

The ideal goal of the network is to learn the optimal action mapping. This maps each context vector into its optimal action, that is the action $a(\underline{c})$ which maximises $d(a, \underline{c})$ for each $\underline{c}$. This may not always be possible, since the structure chosen may not allow the function to be learnt. If the SRS is denoted by r , a natural criterion to maximize would be $E[r | \underline{u}]$. If there exists a

u^* which implements the optimal action mapping, that u^* also maximizes $E\{r|\underline{u}\}$. In this paper, algorithms to globally optimise $E\{r|\underline{u}\}$ are developed. Let the action chosen by the i^{th} PLA at instant k be $\alpha_i(k)$. Then,

$$\text{Prob}\{\alpha_i(k) = j\} = g_i(j, \underline{u}_i(k)) ; 1 \leq j \leq m \quad (2.5)$$

At each instant, the network is presented with a context vector. from the environment according to P_c . Each unit is assumed to have a finite number of outputs. The output of a unit is a function of the actions chosen by the PLAs which make up the unit and the context vector input to the unit, which can be made up of outputs of units in layers previous to the unit and the context vector input to the network. The output of the network is a vector made up of outputs of some prefixed units of the network. The goal of the system is to solve the following problem.

$$\text{maximise } f(\underline{u}) = E\{r|\underline{u}\} \quad (2.6)$$

where r is the SRS. In many cases, the maximum of f is attained when at least one of the components of $\underline{u}$ becomes unbounded which can lead to the algorithm exhibiting unbounded behaviour. To avoid this, the optimisation problem is reposed as

$$\text{maximise } f(\underline{u}) = E\{r|\underline{u}\} ; \text{ subject to } |u_{ij}| \leq L \text{ for all } i, j \quad (2.7)$$

where L is a positive constant and u_{ij} the j^{th} component of $\underline{u}_i$.

The solutions of (2.7) lie within a bounded set and therefore, if the algorithm does converge to solutions of (2.7), it will exhibit bounded behaviour. Also, depending on the choice of the g_i 's and L , the probabilities of the PLAs can be made to take on (almost) all possible values thus allowing the optimal action to be learnt. A suitable choice of g_i and $\underline{u}_i$ would be

$$g_i(j, \underline{u}_i) = \exp(u_{ij}) / \sum_{k=1}^m \exp(u_{ik}) \quad (2.8)$$

where $\underline{u}_i$ is a m -dimensional real vector. L is chosen large enough so that the probability of the j^{th} action of the i^{th} PLA is closer to unity than a prespecified value when $u_{ij} = L$ and $u_{ik} = -L$, $k \neq j$. Thus, any action of the PLA can have its probability close to unity.

3 ALGORITHM AND ANALYSIS:

The algorithm used for updating u_{ij} is

$$u_{ij}(k+1) = u_{ij}(k) + b r(k) \{\delta \ln g_i / \delta u_{ij}\} + b h'(u_{ij}(k)) + \sqrt{b} s_{ij}(k) \quad (3.1)$$

where g_i is evaluated at $(\alpha_i(k), \underline{u}_i(k))$ and

i) $h(\cdot)$ is a real valued function defined as

$$h(x) = \begin{cases} -K(x-L)^{2n} & x \geq L \\ 0 & |x| \leq L \\ -K(x+L)^{2n} & x \leq -L \end{cases} \quad (3.2)$$

K is a positive constant, n a positive integer and h' the first derivative of h .

ii) $\{s_{ij}(k): 1 \leq i \leq M, 1 \leq j \leq m, k \geq 0\}$ is a sequence of bounded, i.i.d. zero mean random variables with variance σ^2 , $\sigma > 0$.

iii) $b > 0$ is the learning parameter. It can be different for different PLAs, but is assumed to be the same for simplicity.

In (3.1) the second term on the RHS is a gradient following term. The third term is to ensure that the algorithm remains bounded and the final term is a random term to allow the algorithm to exit out of local optima that are not global optima. This algorithm is based on the REINFORCE algorithm of [W88]. The following procedure is used to show that algorithm (3.1) does converge to the solution of the optimisation problem (2.7). (3.1) is first approximated by a Stochastic Differential Equation (SDE) using weak convergence techniques [K84]. The invariant probability measure associated with the SDE is then shown to concentrate on the global optima of (2.7). For each $b > 0$, $\underline{u}(k)$ is a Markov process. Denote this dependence on b by $\underline{u}^b(k)$. The following continuous time interpolation of $\underline{u}^b(k)$ is needed to obtain the approximating SDE.

$$\underline{u}^b(t) = \underline{u}^b(k) \text{ for } t \in [kb, (k+1)b) \quad (3.3)$$

Theorem 3.1: The sequence $\{\underline{u}^b(.): b > 0\}$ converges weakly, as $b \rightarrow 0$, to $\underline{z}(.)$, where $\underline{z}(.)$ satisfies the SDE

$$d\underline{z} = V(\underline{z}) dt + \sigma d\underline{w} ; \underline{z}(0) = \underline{u}(0) \quad (3.4)$$

where $\underline{w}$ is Standard Brownian Motion and

$$V(\underline{u}) = f(\underline{u}) + \sum_{i,j} h(u_{ij}) \quad (3.5)$$

This convergence is valid under the assumption that

i. $(\delta g_i / \delta u_{ij})$ is continuous for all i, j .

ii. g_i is bounded away from zero on every compact $\underline{u}_i$ -set.

Comment: The proof of this theorem follows from theorem 5.7 of [K84] and an application of Donsker's Theorem [K84]. The assumptions on the g_i 's are satisfied by many kinds of functions, for example, the function described in (2.8). Thus, the conditions imposed are not unduly restrictive.

(3.4) is the Langevin equation. It is known that the invariant probability measure of this SDE concentrates on the global maxima of $V(.)$ as $\sigma \rightarrow 0$ [APZ85]. The goal of the

algorithm is to solve (2.7), not maximise $V(\cdot)$. On the set $FS = \{u: |u_{ij}| \leq L \text{ for all } i,j\}$, f and V coincide. Outside FS , $V < f$ since $\sum h(u_{ij}) < 0$. Thus, if the global maximum of V is in the set FS , (2.7) is exactly solved. If the global maximum of V is outside FS , it is easily seen that the value of f obtained is greater than the value of V at any point in FS . Since the constraints were introduced to avoid unboundedness problems, the constraints of (2.7) need not be exactly satisfied. The algorithm guarantees a performance at least as good as any point in FS . It can be shown that all points where V is globally maximised lie in a bounded set and so the algorithm exhibits bounded behaviour.

4 SIMULATIONS:

In this section a two class pattern recognition example is described and simulation results presented. The context vectors arrive uniformly from the set $[0,1] \times [0,1]$. The two classes are 0 and 1. Region A is defined by

$$[2x_1 - x_2 > 0] \text{ AND } [-x_1 + 2x_2 > 0] \quad (4.1)$$

The optimal action is 1 in region A and is 0 in A^c . In A,

$$\text{Prob}\{x \in \text{class 1}\} = 1 - \text{Prob}\{x \in \text{class 0}\} = 0.9 \quad (4.2a)$$

and in A^c

$$\text{Prob}\{x \in \text{class 1}\} = 1 - \text{Prob}\{x \in \text{class 0}\} = 0.1 \quad (4.2b)$$

x is the context vector and

$$x = (x_1, x_2)^t \quad (4.3)$$

The network is made up of two first layer units U_1, U_2 and one fixed second layer unit. The second layer unit performs the AND operation on the first layer unit outputs. Each first layer has two PLA, each with four actions. Table 1 gives details about these units. A_{ij} is the j^{th} automaton of the 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. This is because the optimal discriminant function can be learnt in two ways as the action values are symmetric. Let y_i be the output of U_i and α_{ij} the action chosen by A_{ij} . Then,

$$y_i = \text{us}[\alpha_{i1}(k)x_1(k) + \alpha_{i2}(k)x_2(k)] \quad (4.4)$$

where us is the unit step function. The output of the network is z , $z = [y_1 \text{ AND } y_2]$. The value of the learning parameter was fixed at 0.05 for all the automata and $L = 3$, $K = 1$ and $n = 1$. The g_i 's are defined by (2.7) with $m = 4$.

The initial conditions were biased in favour of the set of indices $(3,3,3,3)$ which are a local maximum of the system. The initial conditions were set such that these indices had a probability slightly greater than 0.98 and the remaining was

equally divided between the rest. σ was initially set at 10.0 and reduced as

$$\sigma(k+1) = 0.999 \sigma(k) , 0 \leq k \leq 4,000 \quad (4.5)$$

and kept constant. for $k \geq 4,000$. This is a permissible variation as σ is kept constant after a finite k . Twenty runs were conducted. 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 the optimal action of every automaton was 33,425. The number of samples generated was 500 and a vector from this set was chosen randomly at each instant. Thus, a large number of samples are not needed. A single run of 50,000 steps took about 36 seconds of CPU time on a VAX 8810. With similar initial conditions, a network composed of learning automata units using the L_{R-I} did not converge to the global maximum.

5 CONCLUSIONS:

It is shown that the Learning Automaton can be modified such that a feedforward network composed of units which are teams of these modified Learning Automata exhibit. global convergence properties. The results are asymptotic in the learning parameter b and time step k . Simulations show that convergence is obtained in a reasonable number of steps. Since the algorithm is totally decentralised, the units need not have any information exchange during updating.

6 REFERENCES:

UNIT	AUTOMATON	ACTIONS				INDEX OF OPTIMAL ACTION	
		1	2	3	4		
U_1	A_{11}	-2	-1	1	2	4	2
	A_{12}	-2	-1	1	2	2	4
U_2	A_{21}	-2	-1	1	2	2	4
	A_{22}	-2	-1	1	2	4	2

Table 1