Expectation Driven Learning with an Associative Memory

by G.Lukes, B.Thompson and P.Werbos
National Science Foundation and SYSCON[1]
Washington, D.C. 20550

Introduction

Supervised learning requires an explicit target paired with each input. When explicit targets are not available, internally generated targets are required. Expectation Driven Learning (EDL) generates internal expectations that provide these targets for any supervised learning method. A system which incorporates EDL must be provided intrinsic measures of value. Expectations are of these measures of value and provide targets for learning after each action (i.e. real time learning). In our experiments, the automaton (which moves on a two-dimensional grid) has expectations of the minimum future cost of actions leading to a goal state; learning occurs when expectations in the associative memory are modified. In these experiments, we vary the degree of generalization in the associative memory and note the effect on learning. EDL is well suited to problems where a model of the environment may not be available and must be acquired or refined through experience.

Overview of the Method and Prior Literature

Expectation driven approaches to reinforcement learning and utility maximization over time have received attention from many authors. Werbos (1989b) has argued that this problem is central to understanding human intelligence as well as a number of important engineering applications. For example, Jordan (1989) and Kawato (1989) have shown how trajectory-following problems in robotics can be solved by maximizing a complex utility function which includes terms to represent distance from the desired trajectory, the smoothness of the motion, and so on.

To maximize utility across future time, one needs some way to account for the link between present actions and future results. Neural network literature describes two ways of doing this. One way is to obtain an exact model of the external environment and use backpropagation through time to account for that link. This method has been used in the Nguyen and Widrow (1989) truck backer-upper, in Kawato's and Jordan's robots, and in the Werbos (1989d) model of the natural gas industry. The other approach is the adaptive critic approach.

The term "adaptive critic" was coined by Barto, Sutton and Anderson (1983), but it aptly describes a larger family of methods. An adaptive critic method is any method which includes the adaptation of a "critic" network -- a network which produces a kind of global evaluation of how well the network is doing. By this definition, Klopf (1982), Grossberg (1988), Werbos (1977, 1989a and b), Sutton (1988), Williams (1988), and Holland (1975) may all be seen as adaptive critic methods. (See Grossberg (1982, 1988) and Werbos (1989a) for discussions of the requirements of large scale problems. Lakoff (1987) discusses the induction of higher order categories.)

Anderson (1987) explores the use of backpropagation with adaptive critics. Werbos (1989b), in defining Heuristic Dynamic Programming (HDP), allows for any supervised learning network but does not give actual simulations. Booker (1988) demonstrates a classifier based system. In Sutton (1988), temporal difference methods provide results for a similar class of problems. The remainder of this paper will discuss the adaptive critic, the associative memory we use as a critic network, and the experimental results, in that order.

The Adaptive Critic and Dynamic Programming

In dynamic programming, the user supplies a utility function[2], $U(\underline{X})$, which is a function of the vector $\underline{X}$ ($\underline{X}$ describes the state of the external environment). The user then solves an equation, the Bellman equation, which yields another function $J^*(\underline{X})$. This second function, J^*, has the following property: by maximizing J^* in the immediate future (i.e. t+1), you automatically pick the strategy of action which maximizes the sum of U over the

[1] The views expressed here are personal views of the authors, not those of NSF, nor SYSCON.

[2] In our notation, vectors are underlined, scalars are not. Thus $U(\underline{X})$ is a function, U, of a vector, $\underline{X}$, that returns a scalar.

long-term future.

Heuristic Dynamic Programming (HDP) was first formulated as a neural-network approximation to dynamic programming in Werbos (1977). We have implemented Action Dependent Heuristic Dynamic Programming (ADHDP), where the costs are on the actions (or transitions), not on the states (as in standard HDP). ADHDP is particularly good for dynamic systems where costs are incurred by taking actions, instead of being intrinsic to a state. We start from the following simplification of Howard's (1960) version of the Bellman equation for an absorbing Markov chain, similar to the simplification found in Werbos (1989a):

$$J^*(\underline{X}(t)) = \min_{\underline{u}(t)} E\{ U(\underline{X}(t),\underline{u}(t)) + J^*(\underline{X}(t+1)) \} \tag{1}$$

where $\underline{u}(t)$ is a vector representing the choice of actions at time t, where E{} denotes the expected value, and where $\underline{X}(t+1)$ depends -- of course -- on the choice of action $\underline{u}(t)$. In our automaton, the action $\underline{u}(t)$ has a direct cost, U; thus we seek to <u>minimize</u> the sum of U over time. Note that U depends on both $\underline{X}(t)$ and $\underline{u}(t)$. Both $\underline{X}(t+1)$ and U may also be affected by random noise.

For our purposes, it is more convenient to use a critic that estimates a different function, J', defined as follows:

$$J'(\underline{X}(t), \underline{u}(t)) = E\{ U(\underline{X}(t), \underline{u}(t)) + J^*(\underline{X}(t+1)) \} \tag{2}$$

Combining this with into equation 1, we arrive at the following recursive equation:

$$J'(\underline{X}(t), \underline{u}(t)) = E\{ U(\underline{X}(t), \underline{u}(t)) + \min_{\underline{u}(t+1)} J'(\underline{X}(t+1), \underline{u}(t+1)) \} \tag{3}$$

ADHDP, as implemented for this automaton, is defined as adapting a critic network whose output is an approximation of J' (equation 3). In the automaton, the choice of actions $\underline{u}(t)$ was limited; we worked with only eight possible moves (like a King's moves in chess). Therefore, we chose to represent $J'(\underline{X}(t),\underline{u}(t))$ by the <u>vector</u> $\underline{J}'(\underline{X}(t))$, where $\underline{J}'$ has eight components corresponding to the eight possible moves.

As the system explores the state space, information required to make more accurate estimates of equation 3 will accumulate. Sutton (1988) and Werbos (1989c) have proven theorems about the consistency and convergence of very similar adaptive critic networks.

<u>The Associative Memory</u>

We have used an prototype based associative memory for our critic network, which was inspired by the work of Kanerva (1988) and of Albus (1981), though analogies to Kohonen (1988) and others are possible. Like a production rule, each prototype θ has a left hand side ($\underline{L}_\theta$), that corresponds to the conditional, and a right hand side ($\underline{R}_\theta$) that corresponds to the result. The two primary operations with the associative memory are reading from the memory and writing to, or updating, the memory.

We read from the memory by presenting an input pattern ($\underline{L}$) to the left hand side. The prototypes that will participate in the retrieval are selected by equation 4 (there are N^L elements in $\underline{L}$). This equation determines the strength of each prototype with respect to the current input pattern. The result, $\underline{R}$, of a read operation is just the weighted sum of the right hand sides (equation 5). The constant C is the threshold that selects which prototypes will be used in storing or retrieving a pattern.

We update the associative memory by presenting both an input pattern ($\underline{L}$) and a target pattern ($\underline{R}^T$). First we do a read (as above) to determine $\underline{R}$. Then we calculate the new right hand side for each prototype by equation 6.

$$\Phi_\theta = \begin{cases} 1 & \{ i; 1 \le i \le N^L \} \qquad | L_i - L_{\theta i} | \le C \\ 0 & \text{otherwise} \end{cases} \tag{4 - strength}$$

$$\underline{R} = \Sigma\, \underline{R}_\theta \Phi_\theta \qquad (5 - \text{retrieval})$$

$$\underline{R}_\theta = \underline{R}_\theta + \frac{(\underline{R}^T - \underline{R})\, \Phi_\theta}{\Sigma\, \Phi_\theta} \qquad (6 - \text{update})$$

The associative memory is structured as follows:

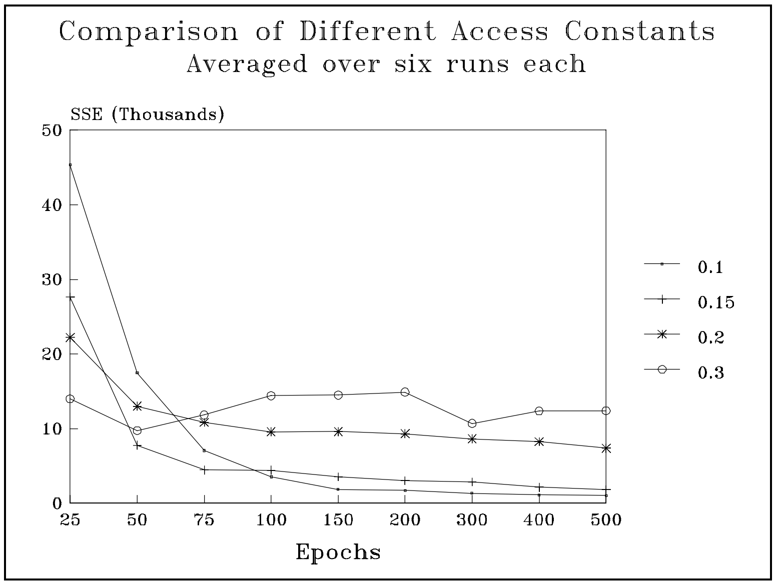
$$\underline{X}(t) \text{ -> } \underline{U}(t),\ \underline{J}(t+1) \qquad (7)$$

with $\underline{X}(t)$ on the left hand side, and $\underline{U}(t)$ and $\underline{J}(t+1)$ on the right hand side. Please note that for our experiments $\underline{X}$ is a vector of two components, <x,y>, that locate the automaton on the two dimensional grid. $\underline{J}$ and $\underline{U}$ are both vectors of eight components, (e.g. <j1,j2,j3,j4,j5,j6,j7,j8>), each element of which corresponds to one of the eight different moves. When the associative memory is initialized, the left hand side is a uniform distribution of the possible values of $\underline{X} \pm C$ (to account for edge effects). $\underline{J}$ and $\underline{U}$ are initialized to small, non-negative, random values.

Experimental Results

In our experiments, the associative memory has 800 prototypes. We vary the access constant, C, (see equation 4) to control the degree of generalization within the associative memory. C was set to 0.1, 0.15, 0.2, and 0.3 - corresponding respectively to an expected response from 2.8%, 5.3%, 8.2% and 14.1% of the prototypes during a read or write operation. The automaton environment is a 10 X 10 two dimensional grid. An epoch is defined as a walk from a randomly selected cell on the edge of the grid to the goal state at the center of the grid. A minimum of five steps is required for each epoch. Each step can be in one of eight directions and incurs a local cost equal to ten times the euclidean distance between the two states. Actions off the edge of the grid are legal, and take the automation to the opposite edge.

The automaton tries to discover the lowest cost path to the goal from each edge state. It does this by learning accurate estimates for the minimum sum of future costs (J') associated with alternative actions from each state. The automaton is given no model of: direction; the relative distance between two states; the relative differences or similarities between alternative actions; and the relative position of the goal from its current state. The automation is also unable to store the sequence of states it has visited. It is, therefore, unable to recall past sequences it has tried, identify a previously visited state, or recall the past history of a specific state. The automaton is only able to read from the current state. It uses the results of that read to select one of eight moves and update the estimate J' for the previous state. While the problem is simple for a human with a well developed model of direction and distance, it presents a very difficult machine learning problem; the automaton must induce a model based only on local cost information in forward time and primary reinforcement.



The results (see graph) illustrate the advantage of greater distribution across prototypes (generality) during the initial phase of learning, and the advantage of less distribution (specificity) during the final phase of learning. In addition, the performance of the more distributed memories was degraded by an inability to escape local optima.

Given the associative memory and action selection rule, local optima are formed whenever the estimate of J'(X(t),u(t)) for the optimal action(s) is falsely higher than the correct estimate of J'(X(t),u(t)) for all nonoptimal actions. In this case, developing more accurate estimates of the nonoptimal actions for a state will not lead to the exploration of an optimal action. Bad estimates will then be filtered back to states on paths which must pass through these suboptimal states on their way to the goal.

Because of the radial nature of the problem, local optima were particularly hard to avoid near the goal. Neighboring cells toward the edge of the grid have a far greater similarity in J' values for the same action than do neighboring cells at the center (around the goal). Generalizing across increasingly different distributions of J' makes the memory more susceptible to being trapped in local optima. Current research is exploring a variety of mechanisms for dynamically altering the access constant and modifying the distribution of prototypes over the state space to overcome these problems.

References

Albus, J. S. (1981). *Brains, Behavior, and Robotics*. Peterborough, NH: Byte Books.

Anderson, C.W. (1987). Strategy Learning with multilayer connectionist representations. In *Proceedings* of the Fourth International Workshop on Machine Learning., p.103-114. Irvine, CA: Morgan Kaufmann.

Barto, A.G., R.S. Sutton, and C.W. Anderson. (1983). Neuron-like adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics* 13: 834-846.

Booker, L.B. (1988). Classifier Systems that Learn Internal World Models. In *Machine Learning* 3: 161-192.

Grossberg, S. (1982). How the Brain Builds a Cognitive Code. *Studies of Mind and Brain*. Drodrecht, Holland: Reidel Press.

Grossberg, S. (1988). Some Psychophysiological and Pharmacological Correlates of a Developmental, Cognitive, and Motivational Theory. *The Adaptive Brain*, part one. Netherlands: Elseview Science Publishers B.V.

Holland, J.H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor: The University of Michigan Press.

Howard, R. (1960). *Dynamic Programming and Markov Processes*. Cambridge: The MIT Press.

Jordan, M.I. (1989) Generic Constraints on Under Specified Target Trajectories. In *Proceedings* of the International Joint Conference on Neural Networks (IEEE, June).

Kanerva, P. (1988). *Sparse Distributed Memory*. Cambridge: MIT Press.

Kawato, M. (1989). Computational Schemes and Neural Network Models for formation of multijoint arm trajectory. In MSW, *op. cit.*

Kohonen, T. (1988). *Self-organization and associative memory*. New York: Springer-Verlag.

Klopf, A. H. (1982). *The Hedonistic Neuron: A Theory of Memory, Learning, and Intelligence*. Washington, DC: Hemisphere.

Lakoff, G. (1987). *Women, Fire, and Dangerous Things: What Categories Reveal about the Mind*. Chicago: Chicago Press.

Nguyen, D., Widrow, B. (1989). Truck backer-upper: an example of self-learning in neural networks. In MSW, *op. cit.*

Sutton, R.S. (1988). Learning to predict by the methods of temporal differences. In *Machine Learning* 3: 9-44.

Werbos (1977). Advanced Forecasting Methods for Global Crisis Warning and Models of Intelligence. *General Systems Yearbook*. (Appendix B).

Werbos (1989a). A Menu of Designs for Reinforcement Learning Over Time. In *Neural Networks for Robotics and Control*. Editors: Miller, W. T., Sutton, R. S., Werbos. Cambridge: MIT Press.

Werbos (1989b). Backpropagation and neurocontrol: a review and prospectus. In *Proceedings* of the International Joint Conference on Neural Networks (IEEE,June).

Werbos (1989c). The consistency of HDP applied to a simple reinforcement learning problem. In *Neural*

Networks, forthcoming.

Werbos (1989d). Maximizing long-term gas industry profits in two minutes in Lotus using neural network methods. *IEEE Transactions Systems Man and Cybernetics*. March/April.

Williams, R.J. (1988). *Toward a theory of reinforcement-learning connectionist systems*. Tech. Rep. NU-CCS-88-3, College of Computer Science, Northeastern University, Boston, MA.