International Journal of the Korean Society of Precision Engineering, Vol. 3, No. 1. January 2002.

Barycentric Approximator for Reinforcement Learning Control

Whang Cho

Department of control and instrumentation, Kwangwoon University, Seoul, South Korea

ABSTRACT

Recently, various experiments to apply reinforcement learning method to the self-learning intelligent control of continuous dynamic system have been reported in the machine learning related research community. The reports have produced mixed results of some successes and some failures, and show that the success of reinforcement learning method in application to the intelligent control of continuous control systems depends on the ability to combine proper function approximation method with temporal difference methods such as Q-learning and value iteration. One of the difficulties in using function approximation method in connection with temporal difference method is the absence of guarantee for the convergence of the algorithm. This paper provides a proof of convergence of a particular function approximation method based on "barycentric interpolator" which is known to be computationally more efficient than multilinear interpolation.

Keywords: Reinforcement Learning, Q-learning, Barycentric Interpolation, Multilinear Interpolation

1. Introduction

Reinforcement learning algorithms generates functions that map states to actions based on state value or state action value functions. When dealing with continuous dynamic system, these functions must be approximated by using proper approximator scheme based on discretized state and action space.

Several researchers have recently reported that some classes of function approximators work better with temporal difference methods than others. For example, some experimental evidences that linear functions of coarse codes, such as CMACs, can converge reliably for some problems are provided in [1]. It is suggested in [1] that online exploration of a Markov decision process can help to concentrate the representational power of a function approximator in the important regions of the state space. Other than CMAC two frequently used function approximators are fine grid method and neural net approach.

The fine grid method may be used when system's order is low, i.e., of one or two dimensions. Above two dimensions, fine girds are too expensive. Furthermore the value functions can possess discontinuities, which can lead to suboptimalities even with very fine discretization in two dimensions.

Neural nets have been used in conjunction with temporal difference method[2] and Q-learning [3] in very high dimensional spaces[4-6]. While promising, it is not always clear that they produce the accurate value fucntions that might be needed for fine near optimal control of dynamic systems, and the most commonly used methods of applying value iteration or policy iteration with value function approximation with a neural-net often diverge, i.e., unstable[7].

This paper provides a proof of convergence of a temporal difference algorithm with barycentric value function interpolator. The algorithm will converge either when the decision process is discounted or undiscounted. Sufficient conditions for convergence to the exact value function are also given, and the bound for maximum error between the estimated and true value functions is estimated.

2. Definitions and preliminary results

Some definitions and preliminary results are introduced briefly in this section to facilitate the exposition of the main result of this paper.

Definition: An *affine combination* of the points $\{\eta_j \in R^n\}$, $0 \le j \le k$, is a point of the form

$$\eta = \sum_{0\le j\le k} \alpha_j \eta_j \tag{1}$$

where $\alpha_j \in R$ and $\sum_{0\le j\le k} \alpha_j = 1$.

This definition is mathematically legitimate because it can be recast as the sum of a point and a linear combination of vectors.

Definition: A set of points $\{\eta_j \in R^n\}$, $0 \le j \le k$, is called *affinely independent* when the vectors $\{\eta_j - \eta_k\}$ are linearly independent for $0 \le j < k$.

Definition: An *affine subspace* is the set of points that can be expressed as an affine combination of points from an affinely independent set.

Definition: A *convex combination* of points $\{\eta_j \in R^n\}$, $0 \le j \le k$, is a point of the form

$$\eta = \sum_{0\le j\le k} \alpha_j \eta_j \tag{2}$$

where $\alpha_j > 0$ for all j and $\sum_{0\le j\le k} \alpha_j = 1$.]

Definition: The *convex hull* of a set is the closure of the set under convex combination.

Definition: The convex hull of points from an affinely independent set U is a *simplex.*

Definition: Any state $x \in R^n$ inside some simplex made of points $\{\eta_j \in R^n\}$ for $0 \le j \le k$ is called *barycenter* of the points and can be expressed as a convex combination:

$$x = \sum_{0\le j\le k} p(x \mid \eta_j)\eta_j \tag{3}$$

where $\sum_{0\le j\le k} p(x \mid \eta_j)$ is called *barycentric coordinate* of x.

Definition: Let $V(\eta_j)$ be the value of a scalar function V at the point η_j. Then the *barycentric interpolator* of V at x is defined by

$$V(x) = \sum_{0\le j\le k} p(x \mid \eta_j)V(\eta_j) \tag{4}$$

Note that in the general case, for any given x, the choice of a simplex which contains x is not unique, and once a simplex is selected the choice of barycentric coordinate $p(x \mid \eta_j)$ is not unique if $k > n+1$. Note also that any n-dimensional hypercube can be broken into n! simplexes according to the Coxeter-Freudenthal-Kuhn triangulation[8].

Definition: A function f from a vector space S onto itself is a *contraction* if for any two points a and b in S

$$\| f(a) - f(b) \| \le \alpha \| a - b \| \tag{5}$$

where $0 \le \alpha < 1$ is called contraction factor. f is called *nonexpansion* if for any two points a and b in S

$$\| f(a) - f(b) \| \le \| a - b \| \tag{6}$$

Definition: A point x in a vector space S is called a *fixed point* of the function f if $f(x) = x$.

Theorem 2.1 (Contraction Mapping): Let S be a vector space with norm $\|\cdot\|$. Suppose f is a contraction on S with contraction factor α. Then f has exactly one fixed point x^* in S. And for any initial point x_0, the sequence x_0, $f(x_0)$, $f(f(x_0))$, ... converges to x^*.

Let S and A denote the state space and action space, respective, of a Markov decision process. At any given time t, the state of the system is denoted by $x_t \in S$ and learning agent chooses an action $a_t \in A$. Then system transition occurs according to the transition function τ acting on x_t and a_t and produces a next state $x_{t+1} \in S$. S_0 denotes a distribution on S which gives the probability of being in each state at time 0. The cost function c measures how will the agent is doing. At each time step t, the agent incurs a cost $c(x_t, a_t)$. The learning agent must act to minimize the expected discounted cost $E\{\sum \gamma^t c(x_t, a_t)\}$ where $\gamma \in [0,1]$ is called the discount factor. The optimal value function is denoted by $V^*(x)$, which represents the minimal possible expected discounted cost starting from state x. Markov decision process is called deterministic if the functions $c(x_t, a_t)$ and $\tau(x_t, a_t)$ are deterministic for all x_t and a_t.

There are several ways by which the existence of $V^*(x)$ is guaranteed. In a finite discounted Markov decision process, it is sufficient to require that the cost function $c(x_t, a_t)$ have bounded mean and variance for all x and a. For a nondiscounted Makov decision process to be of interest, it should have some set of states G which is absorbing and cost-free: that is, if the system is in G at time t, it should stay in G thereafter and $c(x_t, a_t) = 0$ for all $x \in G$ and $a \in A$. Without loss of generality, all absorbing states are assumed to be lumped to a single state indexed by 1. An action selection strategy or policy is called proper if, no matter what state it start from, following the strategy ensures that $P(x_t = 1) \to 1$ as $t \to \infty$. A finite nondiscounted Markov decision process have a well-defined optimal value function if the cost function has bounded mean and variance, there exists a proper strategy, and there does not exist a strategy which has expected cost equal to $-\infty$ from some initial state. From now on, it is assumed that all Markov decision process to be considered have well-defined V^* and that S_0 puts a nonzero probability on every state in S.

Given two Markov decision process M_1 and M_2 which share the same state space S but have two different action spaces A_1 and A_2, respectively, a new Markov decision process M_{12} can be defined by the composition of M_1 and M_2 such that at each time step the learning agent select one action from A_1 and one from A_2. The transition and cost function of the composite process M_{12} are defined by $\tau_{12}(x,(a_1,a_2)) = \tau_2(\tau_1(x,a_1),a_2)$ and $c_{12}(x,(a_1,a_2)) = c_1(x,a_1) + \gamma_1 c_2(\tau_1(x,a_1),a_2)$.

A policy π is defined to be a function $\pi : S \to A$. A learning agent may follow policy π by choosing action $\pi(x)$ wherever it is in state x. It is well known [9] that every Markov decision process with a well-defined V^* has at least one optimal policy π^*. Optimal policy satisfies the following Bellman's optimality equation

$$V^*(x) = E\left\{c(x,\pi^*(x)) + \gamma V^*(\tau(x,\pi^*(x)))\right\} \quad (7)$$

and every policy which satisfies Bellman's optimality equation is optimal.

There are two classes of learning problems for Markov decision processes: online and offline. While in offline case, learning agent is allowed to access the whole process, including the cost and transition functions, in online case only S and A are given to the agent and actual data must be obtained by interacting with the system. Online problem can be transformed into a offline one by observing one or more trajectories, estimating the cost and transition functions, and then pretending that those estimates are true. Similarly, the offline problem can be transformed into an online one by pretending that cost and transition functions are not known.

For a finite Markov decision process, V^* for any $x \in S$ can be found by repeated application of the following dynamic programming backup operation, called value iteration, with any initial guess

$$V(x) \leftarrow \min_{a \in A} E\left\{c(x,a) + \gamma V(\tau(x,a))\right\} \quad (8)$$

This operation can be generalized to allow parallel updating where instead of merely changing the estimate for one state at a time, the new value for every state is computed before altering any of the estimates. The following two theorems imply the convergence of this new operation called parallel value iteration[9].

Theorem 2.2 (value contraction): The parallel value iteration operator for a discounted Markov decision process is s contraction in max norm, with contraction factor equal to the discount factor. If all policies in a nondisounted Markov decision process are proper, then the parallel value iteration operator for that process is a contraction in some weighted max norms. The fixed point of each of these operators is the optimal value function for the Markov decision process.

Theorem 2.3: Let a nondiscounted Markov decision process have at least one proper policy, and let all improper policies have expected cost equal to $-\infty$ for at least one initial state. Then the parallel value iteration operator for that process converges from any initial guess to the optimal value function for that process.

3. Results for discounted Markov decision process

Suppose that T is the parallel value backup operator for a Markov decision process M. The basic value iteration algorithm sets V_{i+1} to be $T(V_i)$ at $i+1$ th iteration starting from some initial guess V_0 and the iteration continues until either time is run out or V_{i+1} is considered to be sufficiently close to V^*. In the algorithm V_i is usually represented by an array of real numbers of dimension equal to the numbers of states in S.

Now suppose that V_i is to be represented, not by a lookup table, but by some other more compact data structure such as a neural net. Then two difficulties are immediately confronted. First, computing $T(V_i)$ generally requires to examine almost every x in S and if M has large enough space that it is beyond using lookup table, computing V_i is probably not affordable. Second, even if V_i can be represented by a neural net, there is no guarantee that $T(V_i)$ can be represented by a neural net.

To address both of these difficulties, let's assume that there exists a sample X_0 of states from M. X_0 should be compact enough that each element of it can be examined repeatedly. On the other hand it should be large enough and representative enough that the states in X_0 can well characterizes the behavior of M. Now let' s define an approximate value iteration algorithm. Rather than setting V_{i+1} to $T(V_i)$, $(T(V_i))(x)$ are computed only for $x \in X_0$ and then a neural net (or other approximator) is fit to these training values to produce the function V_{i+1}.

In order to reason about approximate value iteration, function approximation methods themselves will be considered as operators on the space of value functions.

Definition: Suppose a function from a space S to a vector space R is to be approximated. Fix a sample vector X_0 of points from S, and fix a function approximator scheme A. For each possible vector Y of target values in R, A will produce a function $\hat{f}$ from S to R. Define $\hat{Y}$ to be a vector of fitted values; that is, the ith element of $\hat{Y}$ will be $\hat{f}$ applied to the ith element of X_0. Now define M_A, the mapping associated with A, to be the function which takes each possible Y to its corresponding $\hat{Y}$.

Now we can apply the powerful theorems about contraction mappings to the function approximation methods. In fact, it will turn out that if M_A is a nonexpansion in an appropriate norm, the combination of A with value iteration is stable. In other words, under the usual assumptions, the value iteration will converge to some approximation of the value function. The rest of this section states the required property more formally, then prove that barycentric interpolator have this property.

Definition: Two operators M_F and T on the space S are *compatible* if repeated application of $M_F \mathbf{o} T$ is guaranteed to converge to some $x^* \in S$ from any initial guess $x_0 \in S$.

Note that compatibility is symmetric: if the sequence of operators $M_F, T, M_F, T, \ldots$ converges from any initial guess $x_0 \in S$, then so the sequence $T, M_F, T, M_F, \ldots$ does.

Theorem 3.1 Let T_M be the parallel value backup operator for some Markov decision process M with state space S, action space A, and discount factor $\gamma < 1$. Let $X = S \times A$. Let F be a function approximator (for functions from X to R) with mapping $M_F \in \Box^{|X|} \to \Box^{|X|}$. Suppose M_F is a nonexpansion in max norm. Then M_F is compatible, and $M_F \circ T_M$ has contraction factor γ.

proof: By the value contraction theorem, T_M is a contraction in max norm with contraction factor γ. By assumption M_F is a nonexpansion in max norm. Therefore $M_F \mathbf{o} T_M$ is a contraction in max norm by the factor γ.

Corollary 3.1 The approximate value iteration algorithm based on F converges in max norm at the rate γ when applied to M.

It remains to show that barycentric interpolator A defined in Eq. (4) is compatible with value iteration. The barycentric coordinates involved in calculating the fitted value $\hat{Y}_i$ may depend on the sample vector X_0, but may not depend on the target values Y. More precisely, for a fixed X_0, if Y has n elements, there must exist n^2 nonnegative barycentric coordinates β_{ij} such that

$$\hat{Y}_i = \sum_{j=1}^{n} \beta_{ij} Y_j \tag{9}$$

and

$$\sum_{j=1}^{n} \beta_{ij} = 1 \tag{10}$$

Theorem 3.2 The mapping M_F associated with barycentric interpolator F is a nonexpansion in max norm and is therefore compatible with the parallel value backup operator for any discounted Markov decision process.

proof: Fix all the β_{ij} for F as in the above definition. Let Y and Z be any two vectors of target values. Consider a particular component i. Then,

$$\begin{aligned} &\left|[M_F(Y)-M_F(Z)]_i\right| \\ &= \left|\sum_{j=1}^{n}\beta_{ij}Y_j - \sum_{j=1}^{n}\beta_{ij}Z_j\right| = \left|\sum_{j=1}^{n}\beta_{ij}\left(Y_j - Z_j\right)\right| \\ &\le \max_j\left|Y_j - Z_j\right|\left|\sum_{j=1}^{n}\beta_{ij}\right| = \max_j\left|Y_j - Z_j\right| \\ &= \|Y-Z\| \end{aligned} \tag{11}$$

This implies that every element of $[M_F(Y)-M_F(Z)]$ is no larger than $\|Y-Z\|$. Therefore, the max norm of $[M_F(Y)-M_F(Z)]$ is no larger than $\|Y-Z\|$. In other words, M_F is a nonexpansion in max norm.

4. Results for nondiscounted Markov decision process

Consider now a nondiscounted Markov decision process M. Suppose for the moment that all policies for M are proper. Then the value contraction theorem states that the parallel backup operator T_M for this process is a contraction in some weighted max norm $\|\cdot\|_{w}$. The previous section proved that if the A is a barycentric interpolator, then M_A is a nonexpansion in unweighted max norm. If it can be proved that M_A were also a nonexpansion in $\|\cdot\|_{w}$, it can be said that M_A is compatible with T_M.

Fortunately, barycentric interpolator is compatible with nondiscounted Markov decision process. The proof relies on the property of barycentric interpolator. A barycentric interpolator can be viewed as a Markov process, so that state x has a transition to state y whenever $\beta_{xy} > 0$, i.e., whenever the fitted $V(x)$ depends on the target $V(y)$. Let's view the barycentric interpolator as a Makov process and compose this process with our original Markov decision process to derive a new Markov decision process.

Theorem 4.1 For a barycentric interpolator A with mapping M_A, and for any Markov decision process M (either discounted or nondiscounted) with parallel value backup operator T_M, the function $T_M \circ M_A$ is the parallel backup operator for a new Markov decision process $\tilde{M}$.

proof: Define new Markov decision process $\tilde{M}$ as follows: It will have the same state and action space as M, and it will also have the same discount factor and initial distribution. It is assumed without loss of generality that state 1 of M is cost-free and absorbing: if not, states of M can be renumbered starting at 2, adding a new state 1 which satisfies this property, and making all its incoming transition probabilities zero. Suppose that, in M, action a in state x takes us to state y with probability p_{axy}, and that A replaces $V(y)$ by $\sum_z \beta_{yz} V(z)$. Then the transition probabilities in $\tilde{M}$ for state x and action a may be defined to be

$$\tilde{p}_{axz} = \sum_y p_{axy}\beta_{yz} \quad (z \neq 1) \tag{12}$$

$$\tilde{p}_{ax1} = \sum_y p_{axy}\beta_{y1} \tag{13}$$

These transition probabilities are well defined. Since the barycentric interpolator A has the property that

$$\begin{aligned} &\sum\nolimits_z \beta_{yz} = 1, \\ &\sum_z \tilde{p}_{axz} = \sum_{z\neq 1}\sum_y p_{axy}\beta_{yz} + \sum_y p_{axy}\beta_{y1} \\ &= \sum_y p_{axy}\sum_z \beta_{yz} = \sum_y p_{axy} = 1 \end{aligned} \tag{14}$$

Now suppose that, in M, performing action a in state x yields expected cost c_{xa}. Then performing action a in state x in $\tilde{M}$ yields expected cost

$$\tilde{c}_{xa} = c_{xa} + \gamma\sum_{y'} p_{axy'}\beta_{y'z} \tag{15}$$

Now the parallel value backup operator $T_{M'}$ for M' is

$$
\begin{aligned}
V(x) &= \min_a E\left\{c'(x,a)+\gamma V(\delta'(x,a))\right\} \\
&= \min_a \sum_z p'_{axz}\left\{c'(x,a)+\gamma V(z)\right\} \\
&= \min_a\Big[\sum_{z=1}\left\{\sum_y p_{axy}\beta_{yz}\right\}\left\{\tilde{c}_{xa}+\gamma V(z)\right\} \\
&\quad +\sum_y p_{axy}\beta_{y1}\left\{\tilde{c}_{xa}+\gamma V(1)\right\}\Big] \\
&= \min_a \sum_y p_{axy}\left[\sum_{z\neq 1}\beta_{yz}\left\{\tilde{c}_{xa}+\gamma V(z)\right\}+\beta_{y1}\tilde{c}_{xa}\right] \qquad (16)\\
&= \min_a \sum_y p_{axy}\left\{\tilde{c}_{xa}+\gamma\sum_{z\neq 1}\beta_{yz}V(z)\right\} \\
&= \min_a \left\{\tilde{c}_{xa}+\gamma\sum_y p_{axy}\sum_{z\neq 1}\beta_{yz}V(z)\right\} \\
&= \min_a \left\{c_{xa}+\gamma\sum_{y'} p_{axy'}\beta_{y'z}+\gamma\sum_y p_{axy}\sum_{z\neq 1}\beta_{yz}V(z)\right\}
\end{aligned}
$$

On the other hand, the parallel value backup operator for $\boldsymbol{M}$ is

$$
\begin{aligned}
V(x) &= \min_a E\left\{c(x,a)+\gamma V(\delta(x,a))\right\} \\
&= \min_a \sum_y p_{axy}\left\{c_{xa}+\gamma V(y)\right\}
\end{aligned} \qquad (17)
$$

Replacing $V(y)$ by its approximation under $\mathbf{A}$, the operator becomes $T_M \circ M_A$.

$$
\begin{aligned}
V(x) &= \min_a \sum_y p_{axy}\left\{c_{xa}+\gamma\sum_z \beta_{yz}V(z)\right\} \\
&= \min_a\left[c_{xa}+\gamma\sum_{y'} p_{axy'}\beta_{y'z}+\gamma\sum_y p_{axy}\sum_{z\neq 1}\beta_{yz}V(z)\right]
\end{aligned} \qquad (18)
$$

which is exactly the same as $T_{M'}$ given in Eq. (16).

Given an initial estimate V_0 of the value function, approximate value iteration begins by computing $M_A(V_0)$, the representation of V_0 by using barycentric interpolator $\mathbf{A}$. Then it alternately applies T_M and M_A to produce the series of functions V_0, $M_A(V_0)$, $M_A(M_A(V_0))$,.... In an actual implementation, only the function $M_A(\cdot)$ would be represented explicitly while the function $T_M(\cdot)$ would just be sampled at the points X_0. On the other hand, exact value iteration on $\tilde{M}$ produces the series of functions V_0, $T_M \circ M_A(V_0)$, $T_M \circ M_A(T_M \circ M_A(V_0))$, ... This series obviously contains exactly the same information as the previous one. The only difference between the two algorithms is that approximate value iteration would stop at one of the function $M_A(\cdot)$, while iteration on $\tilde{M}$ would stop at one of the functions $T_M(\cdot)$.

5. The online problem and Q-learning

The results of the previous sections may carry over directly to a gradual version of the parallel value backup operator

$$V(x) \leftarrow \alpha_x \min_{a\in A} E\left\{c(x,a)+\gamma V(\tau(x,a))\right\} \qquad (19)$$

in which, rather than replacing $V(x)$ by its computed update on each step, a weighted average of the old and new values of $V(x)$ can be taken. The weights α_x may differ for each x, and may change from iteration to iteration. In other words, a derived Markov decision process $\tilde{M}$ can be constructed and gradual value iteration can be performed on it, and gradual value iteration on $\tilde{M}$ is still the same as gradual approximate value iteration on $\boldsymbol{M}$.

The results also apply nearly directly to dynamic programming with Watkins's Q-learning algorithm. The Q^* is defined by

$$Q^*(x,a) \,\square\, E\left\{c(x,a)+\gamma V^*(\delta(x,a))\right\} \qquad (20)$$

and Q-learning operator is defined by

$$Q(x,a) \leftarrow \alpha_{xa} E\left\{c(x,a)+\gamma\min_{a'} Q(\delta(x,a),a')\right\} \qquad (21)$$

where the learning rate α_{xa} may be random variable, and may depend for each step not only on x and a but also on the entire past history of the agent's interactions with the Markov decision process, up to but not including the current values of the random variables $c(x,a)$ and $\delta(x,a)$. That is, the behavior or the exact algorithm on the derived Markov decision process can still be defined so that the behavior of the approximate algorithm on the original Markov decision processs is the same as the behavior of the exact algorithm on the derived Markov decision process. The derived Markov decision process is, however, slightly different from the derived Markov decision process for value iteration.

The previous discussion imply that, if some transitions can be sampled at will from the derived Markov decision process, gradual Q-learning can be applied to these transitions and a policy can be learned. The convergence of this algorithm would be guaranteed, as long as the weights α_{xa} were chosen appropriately, by any sufficiently general convergence proof for Q-learning [10,11]

Unfortunately, Q-learning is designed to work for online problems, where the cost or transition functions are unknown and transitions are only sampled from our current state. The power of the approximate value iteration method, on the other hand, comes from the fact that the transitions from a certain small set of states are of interest. So, while the derived Markov decision process for Q-learning will still have only a few relevant states, it is not in general possible to observe many transitions from these states, and so the approximate Q-learing iteration will take a long time to converge.

There are two ways in which the approximate Q-learning algorithm might still be useful. The first is the case where a problem defined somewhere between online and offline is encountered: any transition can be sampled at will, but the cost or transitions functions are not known a priori or the necessary expectations can't be computed. In such a problem the value iteration can't be utilized, since it is difficult to compute an unbiased estimate of the value iteration update, so the approximate Q-learning algorithm is helpful.

The second case where the approximate Q-learning is useful is the case when possible lack of convergence still be acceptable. Suppose our function approximator pays attention to the states in the set X_0. If it is pretended that every transition observed from a state $x \notin X_0$ is actually a transition from the nearest state $x' \in X_0$, then enough data to compute the behavior of the derived Markov decision process on X_0 may be obtained. Unfortunately, following this approximation is equivalent to introducing hidden state into the derived Markov decision process. So the risk of possible divergence may be confronted.

6. Experiments

This section provides some experiments done with

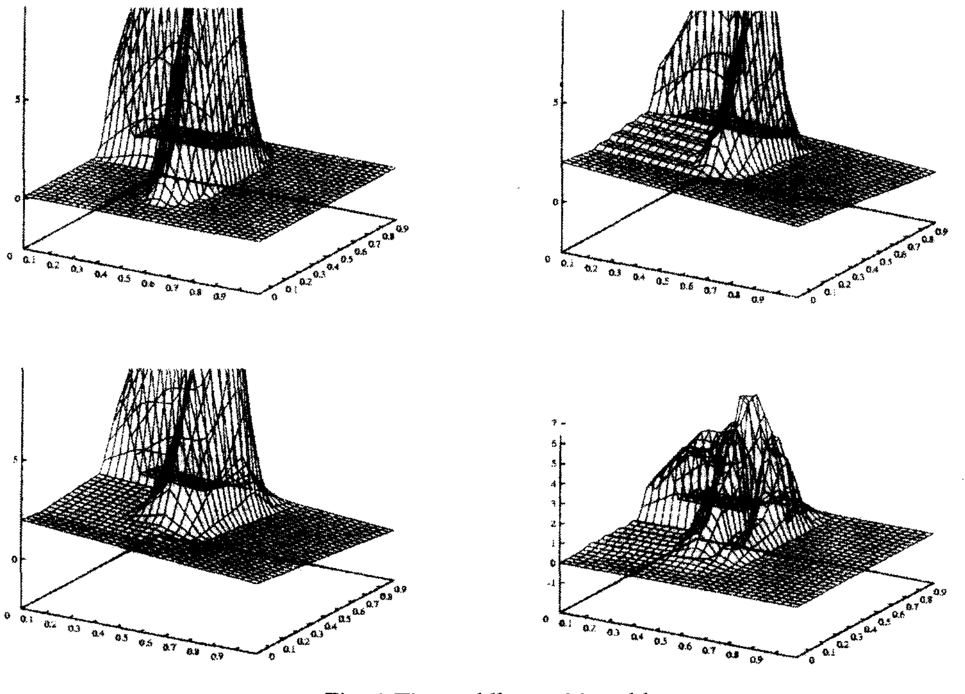
Fig. 1 The puddle world problem

Markov decision problem from [12].

6.1 Puddle world

In this world, the state space is the unit square, and the goal is the upper right corner. The agent has four actions, which move it up, left, right, or down by 0.1 per step. The cost of each action depends on the current state: for most states, it is the distance moved, but for states within the two "puddles," the cost is higher. See Figure 1. For a function approximator, a barycentric interpolator is used and defined as follows: to find the predi at a point (x, y), fisrt find the points of the simplex which contains (x, y). Find the barycenter of the point (x, y) by calculating the barycenter coordinate of the point. Figure 1 shows the cost function for one of the actions, the optimal value function computed on a $\mathbf{100 \times 100}$ grid, an estimate of the optimal value function computed with bilinear interpolation on the corners of a 7×7 (i.e., on 64 sample point), and the difference between the two estimates. Since the optimal value function is nearly piecewise linear outside the

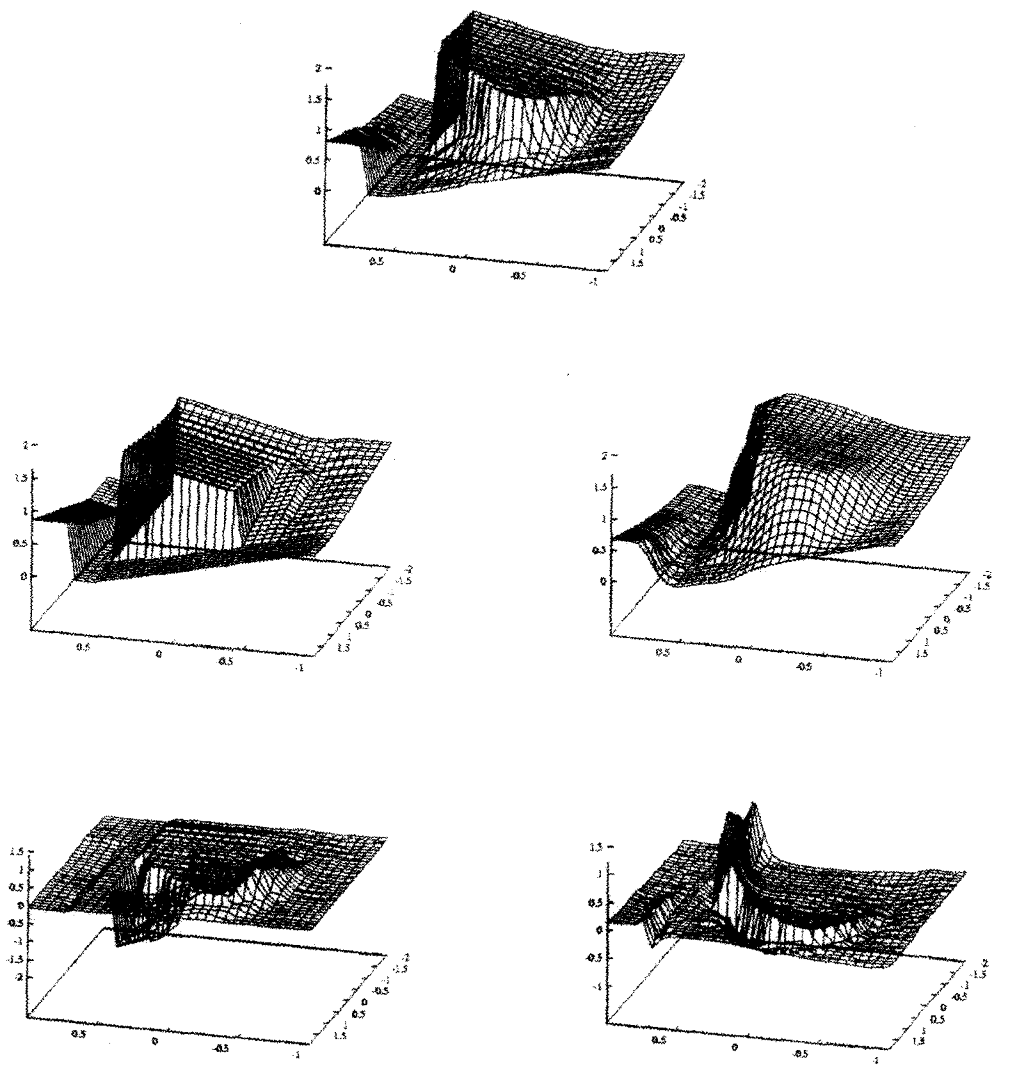
Fig. 2 The car on the hill problem

much better outside the puddles: the root mean squared difference between the two approximations is 2.27 within one step of the puddles, and 0.057 elsewhere. The lowest resolution grid that beats multilinear interpolation's performance away from the puddle is $\mathbf{20 \times 20}$. But even a $\mathbf{5 \times 5}$ grid can beat its performance near the puddles.

In the Figure 1, the cost of moving up, the optimal value function as seen by a 100×100 grid, the optimal value function as seen by bilinear interpolation on the corners of a 7×7 grid, and the difference between the two value functions are shown, respectively, from the top left and clockwise sense.

6.2 Car on a hill

In this experiments, the agent must drive a car up to the top of a steep hill. Unfortunately, the car's motor is weak: it can't climb the hill from a standing start. So, the agent must back the car up and get a running start. The state space is $[\mathbf{-1,1}]\times[\mathbf{-2,2}]$, which represents the position and velocity of the car; there are two actions, forward and backward. The cost function measures the time spent until car reaches the goal.

There are several interesting features to this problem. First, the value function contains a discontinuity despite the continuous cost and transition functions: there is a sharp transition between states where the agent has just enough speed to get up the hill and those where it must back up and try again. Since most function approximators have trouble representing discontinuities, it will be instructive to examine the state space near the goal through which all optimal trajectories must pass. So, excessive smoothing will cause errors over large regions of the state space. Finally, the physical simulation uses a fairly small time step, 0.03 seconds, so it is necessary to use fine resolution for the function approximator just to make sure that a barrier is not introduced.

The results of the experiments appear in figure 2. For a reference model, a $\mathbf{128 \times 128}$ grid is fitted using barycentric interpolater. While this model has 16384 parameters, it is still less than perfect: the right end of the discontinuity is somewhat rough. Two smaller grids are also fitted, one $\mathbf{64 \times 64}$ and one $\mathbf{32 \times 32}$. As the difference plots show, most of the error in the smaller models is concentrated around the discontinuity in the value function. Near the discontinuity, the barycentric approximator performs better than the multilinear models. But away from the discontinuity, the multilinear model outperforms. The $\mathbf{32 \times 32}$ multilinear model also beats the corresponding barycentric approximator model at the right end of the discontinuity: the car is moving slowly enough here that the barycentric approximator thinks that one of the actions keeps the car in exactly the same place. The multilinear model, on the other hand, since it smooths more, doesn't introduce as much drag as the barycentric approximator model does and so doesn't have this problem. The root mean squared error of the $\mathbf{64 \times 64}$ grid (not shown) from the reference model is 0.019 seconds, and of the $\mathbf{32 \times 32}$ grid is 0.336 seconds. All of the above models are fairly large: the smallest one requires to evaluate 2000 transitions for every value backup. Figure 3 shows what happens when a smaller model is attempted to fit. The $\mathbf{12 \times 12}$ grid is shown after 60 iterations, which is in the process of diverging because the transitions are too short to reach the goal f rom adjacent grid cells.

The result of the approximation to the value function

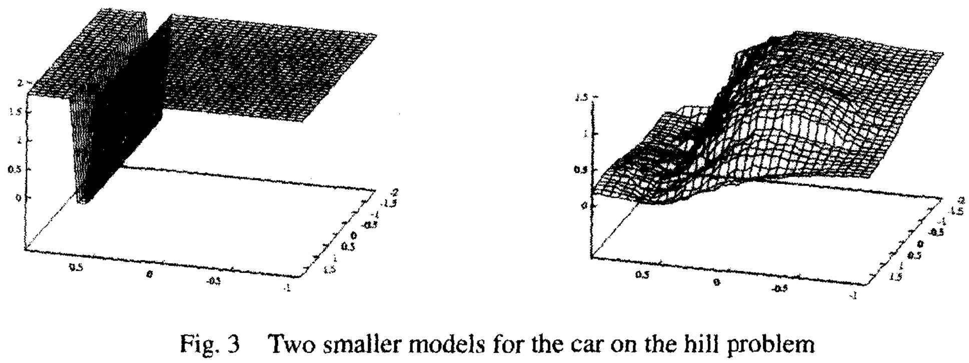

Fig. 3 Two smaller models for the car on the hill problem

for the car on the hill problem is shown in Figure 2. From the top in clockwise sense, the reference model, a 32 × 32 multilinear grid model, a 32 × 32 grid barycentric approximator model, the error of the 32 × 32 multilinear grid model, and the error of the 32 × 32 grid barycentric approximator model are depicted. In each plot, the horizontal axes represent the agent's position and velocity, and the vertical axis represents the estimated time remaining until reaching the summit at $x = 0.6$.

Figure 3 shows two smaller models for the car on the hill problem: a divergent 12 × 12 grid, and a convergent nearest neighbor model on the same 144 sample points.

7. Conclusions

We have proved the convergence of approximate temporal difference methods based on barycentric approximator, and shown experimentally that these methods can solve Markov decision process more efficiently than multilinear approximator of comparable accuracy. Unfortunately, many popular function approximators, such as neural net, linear regression, and CMACs, do not behaves like barycentric approximators or multilinear approximator. The chief reason for divergence is exaggeration: the more a method can exaggerate small changes in its target function, the more often it diverges under temporal differencing. Hence, further research on the convergent behavior of the various exaggerating approximator needs to be done.

Acknowledgements

This paper was accomplished with Research Fund provided by Korea Research Foundation, Support for Faculty Research Abroad.

References

1. J. C. Santamaria, R. S. Sutton, and A. Ram, "Experiments with Reinforcement Learning in Problems with Continuous State and Action Spaces," COINS Technical Report 96-088, Dec. 1996.
2. R. S. Sutton, "Learning to predict by the methods of temporal differences," Machine Learning, 3(1):9-44, 1988.
3. C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D thesis, King's college, Cambridge, England, 1989.
4. G. Tesauro, "Neurogammon: a neural network backgammon program," IN IJCNN Proceedings III pages 33-39, 1990.
5. A. G. Barto, and R. S. Sutton, "Reinforcement Learning: An Introduction," The MIT Press, Cambridge, Massachusetts, 1998.
6. R. H. Crites, and A. G. Barto, " Improving elevator performance using reinforcement learning," Advances in Neural Information Processing Systems: Proceedings of the 1995 Conference, pp. 1017-1023. MIT Press, Cambridge, MA.
7. J. A. Boyan, and A. W. Moore, "Generalization in reinforcement learning: safely approximating the value function," Advances in Neural Information Processing Systems, volume 7. Morgan Kaufmann, 1995.
8. D. W. Moore, "Simplical Mesh Generation with Applications," PhD. Thesis. Report no. 92-1322, Cornell University, 1992.
9. D. T. Bertsekas, and J. N. Tsitsiklis, "Parallel and Distributed Computation: Numerical Methods," Prentice Hall, 1989.
10. T. Jaakkola, M. I. Jordan, and S. P. Singh, "On the convergence of stochastic iterative dynamic programming algorithms," Neural computation, 6(6):1185-1201,1994.
11. J. N. Tsitsiklis, "Asynchronous stochastic approximation and Q-learning," Machine Learning, 16(3):185-202, 1994.
12. J. A. Boyan, and A. W. Moore, "Generalization in reinforcement learning: safely approximating the value function," Advances in Neural Information Processing Systems, volume 7, Morgan Kaufmann, 1995.
13. A. W. Moore, "Variable resolution dynamic programming: efficiently learning action maps in multivariate real-valued state-spaces," Machine Learning: Proceedings of the eighth international workshop, Morgan Kaufmann, 1991.