Stable Function Approximation in Dynamic Programming

Geoffrey J. Gordon
Computer Science Department
Carnegie Mellon University
Pittsburgh PA 15213
`ggordon@cs.cmu.edu`

Abstract

The success of reinforcement learning in practical problems depends on the ability to combine function approximation with temporal difference methods such as value iteration. Experiments in this area have produced mixed results; there have been both notable successes and notable disappointments. Theory has been scarce, mostly due to the difficulty of reasoning about function approximators that generalize beyond the observed data. We provide a proof of convergence for a wide class of temporal difference methods involving function approximators such as k-nearest-neighbor, and show experimentally that these methods can be useful. The proof is based on a view of function approximators as expansion or contraction mappings. In addition, we present a novel view of fitted value iteration: an approximate algorithm for one environment turns out to be an exact algorithm for a different environment.

1 INTRODUCTION AND BACKGROUND

The problem of temporal credit assignment — deciding which of a series of actions is responsible for a delayed reward or penalty — is an integral part of machine learning. The methods of temporal differences are one approach to this problem. In order to learn how to select actions, they learn how easily the agent can achieve a reward from various states of its environment. Then, they weigh the immediate rewards for an action against its long-term consequences — a small immediate reward may be better than a large one, if the small reward allows the agent to reach a high-payoff state. If a temporal difference method discovers a low-cost path from one state to another, it will remember that the first state can't be much harder to get rewards from than the second. In this way, information propagates backwards from states in which an immediate reward is possible to those from which the agent can only achieve a delayed reward.

One of the first temporal difference methods was the Bellman-Ford single-destination shortest paths algorithm (Bellman, 1958, Ford and Fulkerson, 1962), which learns paths in a graph by repeatedly updating the estimated distance-to-goal for each node based on the distances for its neighbors. At around the same time, research on optimal control led to the solution of Markov processes and Markov decision processes (see below) by temporal difference methods (Bellman, 1961, Blackwell, 1965). More recently (Witten, 1977, Sutton, 1988, Watkins, 1989), researchers have attacked the problem of solving an unknown Markov process or Markov decision process by experimenting with it.

Many of the above methods have proofs of convergence (Bertsekas and Tsitsiklis, 1989, Watkins and Dayan, 1992, Dayan, 1992, Jaakkola *et al.*, 1994, Tsitsiklis, 1994). Unfortunately, most of these proofs assume that we represent our solution exactly and therefore expensively, so that solving a Markov decision problem with n states requires $O(n)$ storage. On the other hand, it is perfectly possible to perform temporal differencing on an approximate representation of the solution to a decision problem — Bellman discusses quantization and low-order polynomial interpolation in (Bellman, 1961), and decomposition by orthogonal functions in (Bellman and Dreyfus, 1959, Bellman *et al.*, 1963). These fitted temporal difference methods are not covered by the above convergence proofs. But, if they do converge, they can allow us to find numerical solutions to problems which would otherwise be too large to solve.

Researchers have experimented with a number of fitted temporal difference methods. Results have been mixed: there have been notable successes, including Samuels' (1959) checkers player, Tesauro's (1990) backgammon player, and Lin's (1992) robot navigation. But these algorithms are notoriously unstable;

Boyan and Moore (1995) list several embarrassingly simple situations where popular algorithms fail miserably. Some possible reasons for these failures are given in (Thrun and Schwartz, 1993, Sabes, 1993).

We will prove convergence for a significant class of fitted temporal difference algorithms, including algorithms based on k-nearest-neighbor, linear interpolation, some types of splines, and local weighted averaging. These algorithms will converge when applied either to discounted decision processes or to an important subset of nondiscounted decision processes. We will give sufficient conditions for convergence to the exact value function, and for discounted processes we will bound the maximum error between the estimated and true value functions.

2 DEFINITIONS AND BASIC THEOREMS

Our theorems in the following sections will be based on two views of function approximators. First, we will cast function approximators as expansion or contraction mappings; this distinction captures the essential difference between approximators that can exaggerate changes in their training values, like linear regression and neural nets, and those like k-nearest-neighbor that respond conservatively to changes in their inputs. Second, we will interpret some function approximators as Markov processes; this view will allow us to show that fitted temporal difference learning with these approximators is equivalent to exact temporal difference learning on a modified decision process. To aid the statement of these theorems, we will need several definitions. (See (Kolmogorov and Fomin, 1970) for more information on norms and contraction mappings; see (Bertsekas and Tsitsiklis, 1989) for a treatment of Markov decision processes.)

DEFINITION: Let S be a complete vector space with norm $\|\cdot\|$. A function f from S to itself is a *contraction mapping* if, for all points a and b in S,

$$\| f(a) - f(b) \| \leq \alpha \| a - b \|$$

Here α, the *contraction factor* or *modulus*, is any real number in $[0, 1)$. If we merely have

$$\| f(a) - f(b) \| \leq \| a - b \|$$

we call f a *nonexpansion*.

For example, the function $f(x) = 5 + \frac{x}{2}$ is a contraction with contraction factor $\frac{1}{2}$. It is also a contraction with factor $\frac{3}{4}$. The identity function is a nonexpansion. All contractions are nonexpansions.

DEFINITION: A point x is a *fixed point* of the function f if $f(x) = x$.

A function may have any number of fixed points. For example, the function x^2 on the real line has two fixed points, 0 and 1; any number is a fixed point of the identity function; and $x + 1$ has no fixed points.

Theorem 2.1 (Contraction Mapping) *Let S be a complete vector space with norm $\|\cdot\|$. Suppose f is a contraction mapping on S with contraction factor α. Then f has exactly one fixed point x^* in S. For any initial point x_0 in S, the sequence $x_0, f(x_0), f(f(x_0)), \ldots$ converges to x^*; the rate of convergence of the above sequence in the norm $\|\cdot\|$ is at least α.*

For example, $5 + \frac{x}{2}$ has a unique fixed point at $x = 10$. If we iterate this function starting from 0, we get the sequence $0, 5, 7.5, 8.75, \ldots, \rightarrow 10$.

Next we define a formalism which describes the experiences of an agent sensing its environment and acting to achieve its goals.

DEFINITION: A *Markov decision process* is a tuple $(S, A, \delta, c, \gamma, S_0)$. S is the *state space*; A is the *action space*. At any time t, the environment is in some state $x_t \in S$. The agent perceives x_t, and is allowed to choose an action $a_t \in A$. The *transition function*, δ (which may be probabilistic), then acts on x_t and a_t to produce a next state x_{t+1}, and the process repeats. S_0 is a distribution on S which gives the probability of being in each state at time 0. The *cost function*, c (which may be probabilistic, but must have finite mean and variance), measures how well the agent is doing: at each time step t, the agent incurs a cost $c(x_t, a_t)$. The agent must act to minimize the *expected discounted cost* $E(\sum_{t=0}^{\infty} \gamma^t c(x_t, a_t))$; $\gamma \in [0, 1]$ is called the *discount factor*.

We will write $V^*(x)$ for the *optimal value function*, the minimal possible expected discounted cost starting from state x. V^* will be well-defined if $\gamma < 1$. If $\gamma = 1$, V^* will be well-defined if the sequence $V^{*1}, V^{*2}, \ldots$ converges, where $V^{*n}(x)$ is the minimal possible n-step expected cost from state x. One way to ensure convergence is to require that there be a set of states G with all costs zero and no outgoing transitions so that, no matter what actions the agent chooses, $\lim_{t \rightarrow \infty} P(x_t \in G) = 1$. (We will call such a set *cost-free* and *properly absorbing*. Without loss of generality, we may collapse G to a single state and renumber so that $G = \{1\}$.)

We will say that an MDP is *deterministic* if the functions $c(x, a)$ and $\delta(x, a)$ are deterministic for all x and a, *i.e.*, if the current state and action uniquely determine the cost and the next state. An MDP is *finite* if its state and action spaces are finite; it is *discounted* if $\gamma < 1$. (We discuss only finite MDPs here, although many of the results carry over to continuous processes.) We will call an MDP a *Markov process* if $|A| = 1$ (*i.e.*, if the agent never has a choice). In a Markov process, we cannot influence the cost; our goal is merely to compute it.

Given two MDPs $M_1 = (S, A_1, \delta_1, c_1, \gamma_1, S_0)$ and $M_2 = (S, A_2, \delta_2, c_2, \gamma_2, S_0)$ which share the same state space, we can define a new MDP M_{12}, the *composition* of M_1 and M_2, by alternately following transitions from M_1 and M_2. More formally, let M_{12} be $(S, A_1 \times A_2, \delta_{12}, c_{12}, \gamma_1\gamma_2, S_0)$. At each step, the agent will select one action from A_1 and one from A_2; we define the composite transition function δ_{12} so that $\delta_{12}(x, (a_1, a_2)) = \delta_2(\delta_1(x, a_1), a_2)$. The cost of this composite action will be $c_1(x, a_1) + \gamma_1 c_2(\delta_1(x, a_1), a_2)$.

Define a *policy* to be a function $\pi : S \mapsto A$. An agent may follow policy π by choosing action $\pi(x)$ whenever it is in state x. It is well-known (Bellman, 1961, Bertsekas and Tsitsiklis, 1989) that every Markov decision process with a well-defined V^* has at least one *optimal* policy π^*; an agent which follows π^* will do at least as well as any other agent, including agents which choose actions according to non-policies. The policy π^* satisfies *Bellman's equation*

$$(\forall x \in S)\, V^*(x) = E(c(x, \pi^*(x)) + \gamma V^*(\delta(x, \pi^*(x))))$$

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

There are two broad classes of learning problems for Markov decision processes: *online* and *offline*. In both cases, we wish to compute an optimal policy for some MDP. In the offline case, we are allowed access to the whole MDP, including the cost and transition functions; in the online case, we are only given S and A, and then must discover what we can about the MDP by interacting with it. (In particular, in the online case, we are not free to try an action from any state; we are limited to acting in the current state.) We can transform an online problem into an offline one by observing some transitions, estimating the cost and transition functions, and then pretending that our estimates are the truth. (This approach is called the *certainty equivalent* method.) Similarly, we can transform an offline problem into an online one by pretending that we don't know the cost and transition functions. The remainder of the paper deals with offline problems; for a discussion of the difficulties with applying our theorems to online problems, and in particular when we can use function approximation with Watkins' (1989) Q-learning algorithm, see (Gordon, 1995).

In the offline case, the optimal value function tells us the optimal policies: we may set $\pi^*(x)$ to be any a which maximizes $E(c(x, a) + \gamma V^*(\delta(x, a)))$. (In the online case, V^* is not sufficient, since we can't compute the above expectation.) For a finite MDP, we can find V^* by dynamic programming. With appropriate assumptions, repeated application of the dynamic programming backup operator T which for every state x simultaneously sets

$$V(x) \leftarrow \min_{a \in A} E(c(x, a) + \gamma V(\delta(x, a)))$$

is guaranteed to converge to V^* from any initial guess. (For a nondiscounted problem, we define the backup operator to set $V(1) \leftarrow 0$ as a special case.) This dynamic programming algorithm is called *parallel value iteration*. The following theorem implies the convergence of parallel value iteration.

Theorem 2.2 (value contraction) *The value iteration operator for a discounted Markov decision process is a contraction in max norm, with contraction factor equal to the discount. If a nondiscounted Markov decision process contains a cost-free properly absorbing state, then the value iteration operator for that process is a contraction in some weighted max norm. In both cases, the fixed point of the operator is the optimal value function for the MDP.*

3 MAIN RESULTS: DISCOUNTED PROCESSES

In this section, we will consider only discounted Markov decision processes. The following section generalizes the results to nondiscounted processes.

Suppose that T_M is the parallel value backup operator for a Markov decision process M. In the basic value iteration algorithm, we start off by setting V_0 to some initial guess at M's value function. Then we repeatedly set V_{i+1} to be $T_M(V_i)$ until we either run out of time or decide that some V_n is a sufficiently accurate approximation to V^*. Normally we would represent each V_i as an array of real numbers indexed by the states of M; this data structure allows us to represent any possible value function exactly.

Now suppose that we wish to represent V_i, not by a lookup table, but by some other more compact data structure such as a neural net. We immediately run into two difficulties. First, computing $T_M(V_i)$ generally requires that we examine $V_i(x)$ for nearly every x in M's state space; and if M has enough states that we can't afford a lookup table, we probably can't afford to compute V_i that many times either. Second, even if we can represent V_i exactly, there is no guarantee that we can also represent $T_M(V_i)$.

To address these difficulties, we will assume that we have a sample X_0 of states from M. X_0 should be small enough that we can examine each element repeatedly; but it should be representative enough that we can learn something about M by examining only states in X_0. Now we can define a fitted value iteration algorithm. Rather than setting V_{i+1} to $T_M(V_i)$, we will first compute $(T_M(V_i))(x)$ only for $x \in X_0$; then we will fit our neural net (or other approximator) to these training values and call the resulting function V_{i+1}.

In order to reason about fitted value iteration, we will consider function approximators themselves as operators on the space of value functions. In the following definition, it is important to distinguish the target function f and the learned function $\hat{f}$ from the mapping M_A: the former are real-valued functions, while the latter is a function from functions to functions. It is also important to remember that X_0 is fixed and f is deterministic, so there is no element of randomness in selecting A's training data. Therefore, M_A is a deterministic function.

DEFINITION: Suppose we wish to approximate a function from a set S to a real interval R. Fix a function approximation scheme A (which may depend on the sample $X_0 \subseteq S$). For each possible target function $f \in S \mapsto R$, A will produce a fitted function $\hat{f}$. Define M_A, the *mapping associated with* A, to be the function which takes each possible f to its corresponding $\hat{f}$.

In this framework, fitted value iteration works as follows. Given an initial estimate V_0 of the value function, we begin by computing $M_A(V_0)$, the representation of V_0 in A. Then we alternately apply T_M and M_A to produce the series of functions $V_0, M_A(V_0), T_M(M_A(V_0)), M_A(T_M(M_A(V_0))), \ldots$. (In an actual implementation, only the functions $M_A(\ldots)$ would be represented explicitly; the functions $T_M(\ldots)$ would just be sampled at the points X_0.) Finally, when we satisfy some termination condition, we return one of the functions $M_A(\ldots)$.

The characteristics of the mapping M_A determine how it behaves when combined with value iteration. Figure 1 illustrates one particularly important property. As the figure shows, linear regression can exaggerate the difference between two target functions f and g: a small difference between the target values $f(x)$ and $g(x)$ can lead to a larger difference between the fitted values $\hat{f}(x)$ and $\hat{g}(x)$. Many function approximators, such as neural nets and local weighted regression, can exaggerate this way; others, such as k-nearest-neighbor, can not.

This sort of exaggeration can cause instability in a fitted value iteration algorithm. By contrast, we will show that approximators which never exaggerate can always be combined safely with value iteration. These approximators are exactly the ones whose mappings are nonexpansions in max norm: by definition, if M_A is a nonexpansion in max norm, then for any x we have $|\hat{f}(x) - \hat{g}(x)| \leq |f(x) - g(x)|$. (Note that we do not require that $f(x)$ and $\hat{f}(x)$ be particularly close to each other, nor that $\hat{f}(x)$ and $\hat{f}(y)$ be as close to each other as $f(x)$ and $f(y)$.)

The above discussion is summarized in the following theorem (Gordon, 1995):

Theorem 3.1 *Let T_M be the parallel value backup operator for some Markov decision process M with discount $\gamma < 1$. Let A be a function approximator with mapping M_A. Suppose M_A is a nonexpansion in max norm. Then $M_A \circ T_M$ has contraction factor γ; so the fitted value iteration algorithm based on A converges in max norm at the rate γ when applied to M.*

This theorem is in a sense the best possible: if there are two value functions x and y so that

$$\| M_A(x) - M_A(y) \| > \| x - y \|$$

then there exists a Markov process M with backup operator T_M so that $T_M \circ M_A$ does not have a unique fixed point.

It remains to show which function approximators can exaggerate and which can not. Unfortunately, many common approximators can. For example, as figure 1 demonstrates, linear regression can be an expansion in max norm; and Boyan and Moore (Boyan and Moore, 1995) show that fitted value iteration with linear regression can diverge. Other methods which may diverge include standard feedforward neural nets and local weighted regression (Boyan and Moore, 1995).

On the other hand, many approximation methods are nonexpansions, including local weighted averaging, k-nearest-neighbor, Bèzier patches, linear interpolation, bilinear interpolation on a square (or cubical, *etc.*) mesh, as well as simpler methods like grids and other state aggregation. These methods are all *averagers* (Gordon, 1995):

DEFINITION: A real-valued function approximation scheme is an averager if every fitted value is the weighted average of zero or more target values and possibly some predetermined constants. The weights involved in calculating the fitted value $\hat{f}(x)$ may depend on the sample vector X_0, but may not depend on the target values $f(y)$ for any y. More precisely, for a fixed X_0, if S has n elements, there must exist n real numbers k_i, n^2 nonnegative real numbers β_{ij}, and n nonnegative real numbers β_i, so that for each i we have $\beta_i + \sum_j \beta_{ij} = 1$ and $\hat{f}(x_i) = \beta_i k_i + \sum_j \beta_{ij} f(x_j)$.

Most of the β_{ij} will generally be zero. In particular, β_{ij} should be zero if $j \notin X_0$. Averagers satisfy the following theorem (Gordon, 1995):

Theorem 3.2 *The mapping M_A associated with any averager A is a nonexpansion in max norm; so the fitted value iteration algorithm based on A converges when applied to any discounted MDP.*

4 NONDISCOUNTED PROCESSES

If $\gamma = 1$, Theorem 3.1 no longer applies: $T_M \circ M_A$ is merely a nonexpansion in max norm, and so is no longer guaranteed to converge. Fortunately, there are averagers which we may use with nondiscounted MDPs. The proof relies on an intriguing property of

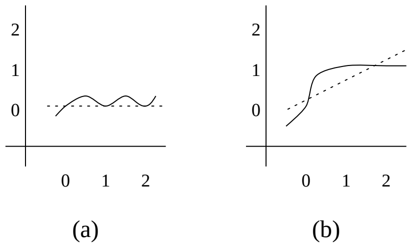


Figure 1: Linear regression on the sample $X_0 = \{0, 1, 2\}$. (a) A target function f (solid line) and its corresponding fitted function $\hat{f}$ (dotted line). (b) Another target function, g, and its fitted function $\hat{g}$. Regression exaggerates the difference between the target functions: the largest difference between f and g is 1 (at $x = 1, 2$), but the difference between $\hat{f}$ and $\hat{g}$ at $x = 2$ is $\frac{7}{6}$.

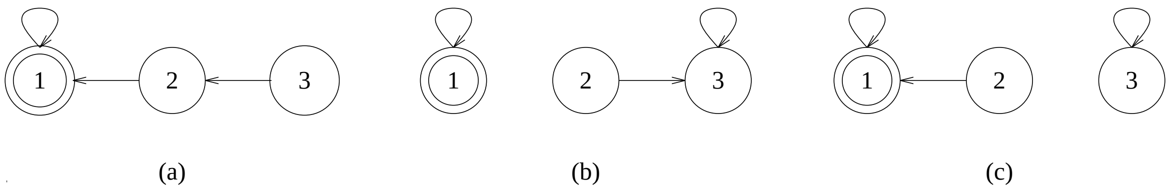


Figure 2: A nondiscounted deterministic Markov process and an averager. The process is shown in (a); the goal is state 1, and all arc costs are 1 except at the goal. In (b) we see the averager, represented as a Markov process: states 1 and 3 are unchanged, while $V(2)$ is replaced by $V(3)$. The derived Markov process is shown in (c); state 3 has been disconnected, so its value estimate will diverge.

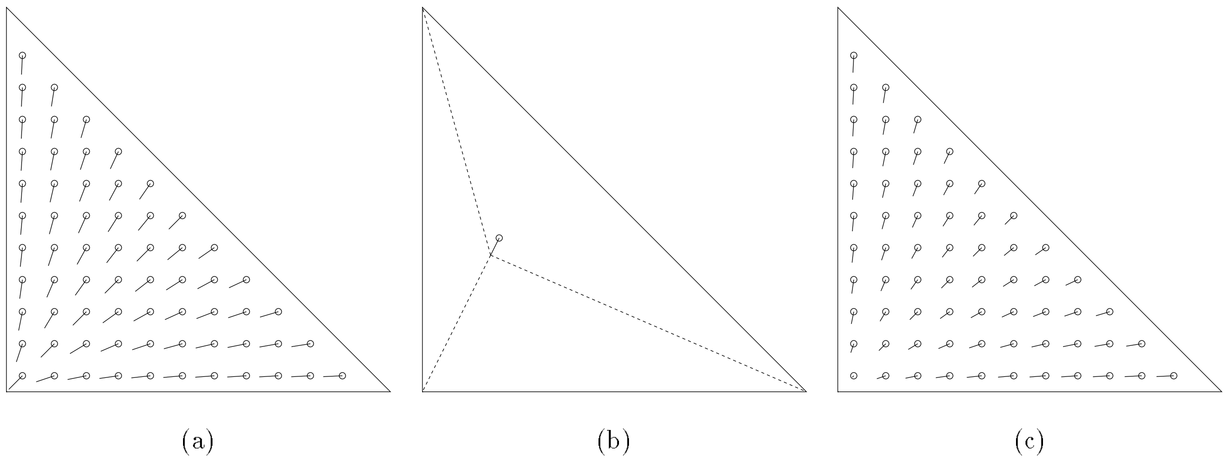


Figure 3: Constructing the derived Markov process. (a) A deterministic process: the state space is the unit triangle, and on every step the agent moves a constant distance towards the origin. The value of each state is its distance from the origin, so V^* is nonlinear. (b) A representative transition from the derived process. For our averager, we used linear interpolation on the corners of the triangle; as before, the agent moves towards the goal, but then the averager moves it randomly to one of the corners. On average, this scattering moves the agent back away from the goal, so steps in the derived process don't get the agent as far. The value function for the derived process is $x + y$. (c) The expected progress the agent makes on each step.

averagers: we can view any averager 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 $V(y)$; presumably, this happens when the averager considers states x and y somehow similar). Figure 2(b) shows one example of a simple averager viewed as a Markov process; this averager has $\beta_{11} = \beta_{23} = \beta_{33} = 1$ and all other coefficients zero.

If we view an averager as a Markov process, and compose this process with our original MDP, we will derive a new MDP. The derived MDP is the same as the original one except that after every step the agent gets randomly scattered (with probabilities depending on the βs) from its current state to some nearby state. That is, if a transition leads from x to y in the original MDP, and if the averager considers state z similar to y, then the same transition in the derived MDP has a chance of moving the agent from x to z. Figure 2 shows a simple example of the derived MDP; a slightly more complicated example is in figure 3. As the following theorem shows (see (Gordon, 1995) for a proof), exact value iteration on the derived MDP is the same as fitted value iteration on the original MDP.

Theorem 4.1 (Derived MDP) *For any averager A with mapping M_A, and for any MDP M (either discounted or nondiscounted) with parallel value backup operator T_M, the function $T_M \circ M_A$ is the parallel value backup operator for a new Markov decision process M'.*

In general, the backup operator for the derived MDP may not be a contraction in any norm. Figure 2 shows an example where this backup operator diverges, since the derived MDP has a state with infinite cost. However, we can often guarantee that the derived MDP is well-behaved. For example, if M is discounted, or if A uses weight decay (*i.e.*, if $\beta_y > 0$ for all y), then $T_M \circ M_A$ will be a max norm contraction; and if A is *self-weighted* for M (Gordon, 1995), $T_M \circ M_A$ will be a contraction in some weighted max norm.

5 CONVERGING TO WHAT?

Until now, we have only considered the convergence or divergence of fitted dynamic programming algorithms. Of course we would like not only convergence, but convergence to a reasonable approximation of the value function.

Suppose that M is an MDP with value function V^*, and let A be an averager. What if V^* is also a fixed point of M_A? Then V^* is a fixed point of $T_M \circ M_A$; so if we can show that $T_M \circ M_A$ converges to a unique answer, we will know that it converges to the right answer. For example, if M is discounted, or if it has $E(c(x,a)) > 0$ for all $x \neq 1$, then $T_M \circ M_A$ will converge to V^*.

If we are trying to solve a nondiscounted MDP and V^* differs slightly from the nearest fixed point of M_A, arbitrarily large errors are possible. If we are trying to solve a discounted MDP, on the other hand, we can prove a much stronger result: if we only know that the optimal value function is near a fixed point of our averager, we can guarantee an error bound for our learned value function (Gordon, 1995). (A bound immediately follows (see *e.g.* (Singh and Yee, 1994)) for the loss incurred by following the corresponding greedy policy.)

Theorem 5.1 *Let V^* be the optimal value function for a finite Markov decision process M with discount factor γ. Let T_M be the parallel value backup operator for M. Let M_A be a nonexpansion. Let V^A be any fixed point of M_A. Suppose $\| V^A - V^* \| = \epsilon$, where $\| \cdot \|$ denotes max norm. Then iteration of $T_M \circ M_A$ converges to a value function V_0 so that*

$$\begin{aligned} \| V^* - V_0 \| &\leq \frac{2\gamma\epsilon}{1-\gamma} \\ \| V^* - M_A(V_0) \| &\leq 2\epsilon + \frac{2\gamma\epsilon}{1-\gamma} \end{aligned}$$

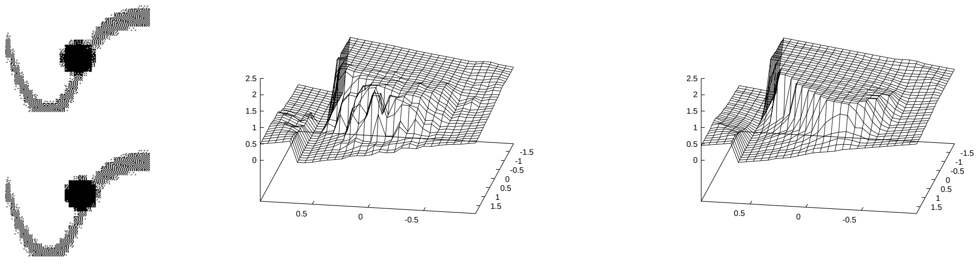
Others have derived similar bounds for smaller classes of function approximators. For a bound on the error introduced by approximating a continuous MDP with a grid, see (Chow and Tsitsiklis, 1989). For a bound on the error introduced by state aggregation, and another bound for a class of linear architectures including narrow localized basis functions and interpolation, see (Tsitsiklis and Van Roy, 1994).

The sort of error bound which we have proved is particularly useful for approximators such as linear interpolation and grids which have many fixed points. Because it depends on the maximum difference between V^* and V^A, the bound is not very useful if V^* may have large discontinuities at unknown locations: if V^* has a discontinuity of height d, then any averager which can't mimic the location of this discontinuity exactly will have no representable functions (and therefore no fixed points) within $\frac{d}{2}$ of V^*.

6 EXPERIMENTS: HILL-CAR THE HARD WAY

In the hill-car world (Moore, 1991, Boyan and Moore, 1995, Gordon, 1995), the agent must drive a car up to the top of a steep hill. At any time, it may choose between two actions, forward and reverse. Unfortunately, the car's motor is weak, and can't climb the hill from a standing start. So, the agent must back the car up and get a running start.

In the standard formulation of this world, the state space is $[-1,1] \times [-2,2]$, which represents the position and velocity of the car. This state space is small enough that value iteration on a reasonably-sized grid



(a) (b) (c)

Figure 4: The hill-car world.

(1000 to 40000 cells, depending on the desired accuracy) can find the optimal value function. To test fitted value iteration, we expanded the state space a thousandfold: instead of position and velocity, we represented each state with two 32×32 grayscale pictures like the ones in figure 4(a), making the new state space $[0,1]^{2048}$. The top picture shows the car's current position; the bottom one shows where it would be in $.03s$ if it took no action. A simple grid on this expanded state space is unthinkable: even with just 2 partitions per pixel, the grid would have 2^{2048} cells.

To approximate the value function, we took a random sample of 5000 legal pictures and ran fitted value iteration with local weighted averaging. In local weighted averaging, the fitted value at state x is an average of the target values at nearby sampled states x', weighted by a Gaussian kernel centered at x. We used a symmetric kernel with height 1 at the center and height $\frac{1}{e}$ when the Euclidean distance from x' to x was about 22. (We arrived at this kernel width by a coarse search: it is the narrowest kernel width we tested for which the derived MDP was usually connected.) We repeated the experiment three times and selected the run with the median RMS error.

The resulting value function is shown in figure 4(b); its RMS error from the exact value function (figure 4(c)) is $0.155s$. By comparison, a 70×71 grid *on the original, two-dimensional problem* has RMSE $0.186s$.

7 CONCLUSIONS AND FURTHER RESEARCH

We have proved convergence for a wide class of fitted temporal difference methods, and shown experimentally that these methods can solve Markov decision processes more efficiently than grids of comparable accuracy.

Unfortunately, many popular function approximators, such as neural nets and linear regression, do not fall into this class (and in fact can diverge). 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.

There is another important difference between averagers and methods like neural nets. This difference is the ability to allocate structure dynamically: an averager cannot decide to concentrate its resources on one region of the state space, whether or not this decision is justified. This ability is important, and it can be grafted on to averagers (for example, adaptive sampling for k-nearest-neighbor, or adaptive meshes for grids or interpolation). The resulting function approximator is no longer an averager, and so is not covered by this paper's proofs. Still, methods of this sort have been shown to converge in practice (Moore, 1994, Moore, 1991), so there is hope that a proof is possible.

Acknowledgements

I would like to thank Rich Sutton, Andrew Moore, Justin Boyan, and Tom Mitchell for their helpful conversations with me. Without their constantly poking holes in my misconceptions, this paper would never have been written. Thanks also to Michael Littman, Mark Ollis, and Ken Lang for their comments on drafts. This material is based on work supported under a National Science Foundation Graduate Research Fellowship, by NSF grant number BES-9402439, and by ARPA grant number F33615-93-1-1330. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the author and do not necessarily reflect the views of the National Science Foundation, ARPA, or the United States government.

References

R. Bellman and S. Dreyfus. Functional approximations and dynamic programming. *Mathematical Tables and Aids to Computation*, 13:247–251, 1959.

R. Bellman, R. Kalaba, and B. Kotkin. Polynomial approximation — a new computational technique in dynamic programming: allocation processes. *Mathematics of Computation*, 17:155–161, 1963.

R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

R. Bellman. *Adaptive Control Processes.* Princeton University Press, 1961.

D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Prentice Hall, 1989.

D. Blackwell. Discounted dynamic programming. *Annals of Mathematical Statistics*, 36:226–235, 1965.

J. A. Boyan and A. W. Moore. Generalization in reinforcement learning: safely approximating the value function. In G. Tesauro and D. Touretzky, editors, *Advances in Neural Information Processing Systems*, volume 7. Morgan Kaufmann, 1995.

C.-S. Chow and J. N. Tsitsiklis. An optimal multigrid algorithm for discrete-time stochastic control. Technical Report P-135, Center for Intelligent Control Systems, 1989.

P. Dayan. The convergence of TD(λ) for general lambda. *Machine Learning*, 8(3–4):341–362, 1992.

L. R. Ford, Jr. and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.

G. J. Gordon. Stable function approximation in dynamic programming. Technical Report CS-95-103, CMU, 1995.

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.

A. N. Kolmogorov and S. V. Fomin. *Introductory Real Analysis.* Prentice Hall, 1970. Revised English edition translated and edited by A. N. Silverman.

L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning, and teaching. *Machine Learning*, 8(3–4):293–322, 1992.

A. W. Moore. Variable resolution dynamic programming: efficiently learning action maps in multivariate real-valued state-spaces. In L. Birnbaum and G. Collins, editors, *Machine Learning: Proceedings of the eighth international workshop.* Morgan Kaufmann, 1991.

A. W. Moore. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state spaces. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems*, volume 6. Morgan Kaufmann, 1994.

P. Sabes. Approximating Q-values with basis function representations. In *Proceedings of the Fourth Connectionist Models Summer School*, Hillsdale, NJ, 1993. Lawrence Erlbaum.

A. L. Samuels. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

S. P. Singh and R. C. Yee. Technical note: an upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16(3):227–233, 1994.

R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

G. Tesauro. Neurogammon: a neural network backgammon program. In *IJCNN Proceedings III*, pages 33–39, 1990.

S. Thrun and A. Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the Fourth Connectionist Models Summer School*, Hillsdale, NJ, 1993. Lawrence Erlbaum.

J. N. Tsitsiklis and B. Van Roy. Feature-based methods for large-scale dynamic programming. Technical Report P-2277, Laboratory for Information and Decision Systems, 1994.

J. N. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16(3):185–202, 1994.

C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3–4):279–292, 1992.

C. J. C. H. Watkins. *Learning from Delayed Rewards.* PhD thesis, King's College, Cambridge, England, 1989.

I. H. Witten. An adaptive optimal controller for discrete-time Markov environments. *Information and Control*, 34:286–295, 1977.