Stable Function Approximation in Dynamic Programming

Geoffrey J. Gordon

January 1995
CMU-CS-95-103

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

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 approximate value iteration: an approximate algorithm for one environment turns out to be an exact algorithm for a different environment.

The author's current e-mail address is ggordon@cs.cmu.edu. This material is based on work supported under a National Science Foundation Graduate Research Fellowship, and by NSF grant number BES-9402439. 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 or the United States Government.

Keywords: machine learning, dynamic programming, temporal differences, value iteration, function approximation

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 examples of a temporal difference method was the Bellman-Ford single-destination shortest paths algorithm [Bel58, FF62], which learns paths through 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 [Bel61, Bla65]. More recently [Wit77, Sut88, Wat89], 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 [BT89, WD92, Day92, JJS94, Tsi94]. 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 [Bel61], and approximation by orthogonal functions in [BD59, BKK63]. These approximate 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 approximate temporal difference methods. Results have been mixed: there have been notable successes, including Samuels' checkers player [Sam59], Tesauro's backgammon player [Tes90], and Lin's robot navigation [Lin92]. But these algorithms are notoriously unstable; Boyan and Moore [BM95] list several embarrassingly simple situations where popular approximate algorithms fail miserably. Some possible reasons for these failures are given in [TS93, Sab93].

Several researchers have recently conjectured that some classes of function approximators work more reliably with temporal difference methods than others. For example, Sutton [SS94] has provided experimental evidence that linear functions of coarse codes, such as CMACs, can converge reliably for some problems. He has also suggested 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.

We will prove convergence for a significant class of approximate 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 show that approximate temporal difference learning with some function approximators is equivalent to exact temporal difference learning for a slightly different problem. To aid the statement of these theorems, we will need several definitions.

DEFINITION: Consider a vector space S and a norm $\| \cdot \|$ on S. If S is closed under limits in $\| \cdot \|$, then S is a *complete vector space*. Unless otherwise noted, all vector spaces will be assumed to be complete.

Examples of complete vector spaces include the real numbers under absolute value and the n-vectors of real numbers under Manhattan (L^1), Euclidean (L^2), and max (L^∞) norms. The max norm, defined as

$$\| \mathbf{a} \|_\infty \equiv \max_i | a_i |$$

and the weighted max norm with weight vector W, defined as

$$\| \mathbf{a} \|_W \equiv \max_i \frac{1}{W_i} | a_i |$$

are particularly important for reasoning about Markov decision problems.

DEFINITION: A function f from a vector space S to itself is a *contraction mapping* if, for all points a and b in S, $\| f(a) - f(b) \| \leq \gamma \| a - b \|$. Here γ, 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 $\frac{1}{2}$. The identity function is a nonexpansion. All contractions are nonexpansions.

Lemma 2.1 *Let C and D be contractions with contraction factors γ and δ on some vector space S. Let M and N be nonexpansions on S. Then*

- *$C \circ N$ and $N \circ C$ are each contractions with contraction factor γ*
- *$C \circ D$ is a contraction with factor $\gamma\delta$*
- *$M \circ N$ is a nonexpansion.*

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 vector space with norm $\| \cdot \|$. Suppose f is a contraction mapping on S with contraction factor γ. 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 γ.*

For example, the function $5 + \frac{x}{2}$ has exactly one fixed point on the real line, namely $x = 10$.

If S is a finite-dimensional vector space, then convergence in any norm implies convergence in all norms.

DEFINITION: A *Markov decision process* is a tuple $(S, A, \delta, c, \gamma, S_0)$. MDPs are a formalism for describing the experiences of an agent interacting with its environment. The set S is the *state space*; A is the *action space*. At any given 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), 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. We introduce conditions below under which V^* is unique and well-defined.

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 will call an MDP a *Markov process*

if $|A| = 1$. In a Markov process, we cannot influence the expected discounted cost; our goal is merely to compute it.

There are several ways that we can ensure that V^* exists. In a finite discounted MDP, it is sufficient to require that the cost function $c(x, a)$ have bounded mean and variance for all x and a. For a nondiscounted MDP, even if c is bounded, cycles may cause the expected total cost for some states to be infinite. So, we are usually interested in the case where some set of states G is *absorbing* and *cost-free*: that is, if we are in G at time t, we will be in G at time $t + 1$, and $c(x, a) = 0$ for any $x \in G$ and $a \in A$. Without loss of generality, we may lump all such states together and replace them by a single state. Suppose that state 1 of an MDP is absorbing. Call an action selection strategy *proper* if, no matter what state we start in, following the strategy ensures that $P(x_t = 1) \to 1$ as $t \to \infty$. A finite nondiscounted MDP will have a well-defined optimal value function as long as

- 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, we will assume that all MDPs that we consider have a well-defined V^*. We will also assume that S_0 puts a nonzero probability on every state in S. This allows us to avoid worrying about inaccessible states.

If we have 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 $M_{12} = (S, A_1 \times A_2, \delta_{12}, c_{12}, \gamma_1\gamma_2, S_0)$. 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 the composite action will be $c_{12}(x, (a_1, a_2)) = c_1(x, a_1) + \gamma_1 c_2(\delta_1(x, a_1), a_2)$.

A *trajectory* is a sequence of tuples $(x_0, a_0, c_0), (x_1, a_1, c_1), \ldots$; trajectories describe the experiences of an agent acting in an MDP. If the MDP is absorbing, there will be a point t so that $c_t = c_{t+1} = \ldots = 0$; we will usually omit the portion of the trajectory after t.

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 possible to generalize the above definition to include randomized strategies and strategies which change over time; but the extra generality is unnecessary. It is well-known [Bel61, BT89] 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^* will satisfy *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 one or more trajectories, 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. Most of the remainder of the paper deals with offline problems, but we mention online problems again in section 5.

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

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

to every state is guaranteed to converge to V^* from any initial guess [BT89]. (In the case of a nondiscounted problem with cost-free absorbing state g, we define the backup operator to set $V(g) \leftarrow 0$ as a special case.) This dynamic programming algorithm is called *value iteration*. If we need to solve an infinite MDP that satisfies certain continuity conditions, we may first approximate it by a finite MDP as described in [CT89], then solve the finite MDP by value iteration. For this reason, the remainder of the paper will focus on finite (although possibly very large) Markov decision processes.

We can generalize the above single-state version of the value iteration backup operator to allow parallel updating: instead of merely changing our estimate for one state at a time, we compute the new value for every state before altering any of the estimates. The result of this change is the *parallel value iteration* operator. The following two theorems imply the convergence of parallel value iteration. See [BT89] for proofs.

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

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 Main results: a simple case

In this section, we will consider only discounted Markov decision processes. The following sections generalize the results to other interesting cases.

Suppose that T 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 $T(V_i)$ 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 $T(V_i)$ 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 with a neural net, there is no guarantee that we can also represent $T(V_i)$.

To address both of 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 large enough and representative enough that we can learn something about M by examining only the states in X_0. Now we can define an approximate value iteration algorithm. Rather than setting V_{i+1} to $T(V_i)$, we will first compute $(T(V_i))(x)$ 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 approximate value iteration, we will consider function approximation methods themselves as operators on the space of value functions: given any target value function, the approximator will produce a fitted value function, as shown in figure 1. In the figure, the sample X_0 is the first five natural numbers, and the representable functions are the cubic splines with knots in X_0.

The characteristics of the function approximation operator determine how it behaves when combined with value iteration. One particularly important property is illustrated in figure 2. As the figure shows, linear regression can exaggerate the difference between two target value functions V_1 and V_2: a small difference between the targets $V_1(x)$ and $V_2(x)$ can lead to a larger difference between the fitted values $\hat{V}_1(x)$ and $\hat{V}_2(x)$. Many function approximators, such as neural nets and local weighted regression, can exaggerate this way; others, such as k-nearest-neighbor, can not. We will show later that this sort of exaggeration can cause instability in an approximate value iteration algorithm.

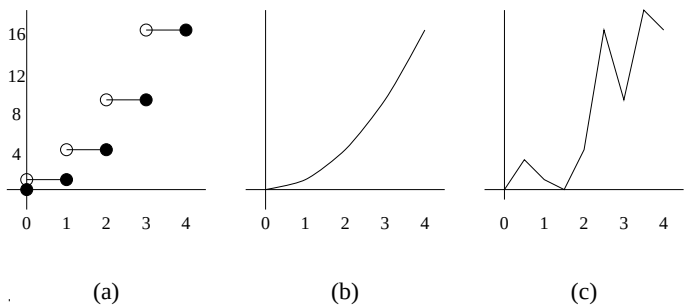

Figure 1: Function approximation methods as mappings. In (a) we see the value function for a simple random walk on the positive real line. (On each transition, the agent has an equal probability of moving left or right by one step. State 0 is absorbing; transitions from other states have cost 1.) Applying a function approximator (in this case, fitting a spline with knots at the first five natural numbers) maps the value function in (a) to the value function in (b). Since the function approximator discards some information, its mapping can't be 1-to-1: in (c) we see a different value function which the approximator also maps to (b).

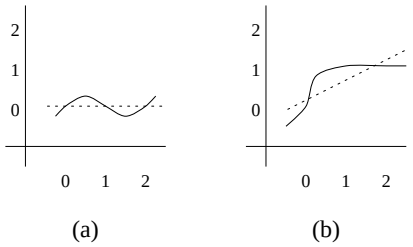

Figure 2: The mapping associated with linear regression when samples are taken at the points $x = 0, 1, 2$. In (a) we see a target value function (solid line) and its corresponding fitted value function (dotted line). In (b) we see another target function and another fitted function. The first target function has values $y = 0, 0, 0$ at the sample points; the second has values $y = 0, 1, 1$. Regression exaggerates the difference between the two functions: the largest difference between the two target functions at a sample point is 1 (at $x = 1$ and $x = 2$), but the largest difference between the two fitted functions at a sample point is $\frac{7}{6}$ (at $x = 2$).

DEFINITION: Suppose we wish to approximate a function from a space S to a vector space R. Fix a sample vector X_0 of points from S, and fix a function approximation scheme A. Now 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 the vector of fitted values; that is, the i-th element of $\hat{Y}$ will be $\hat{f}$ applied to the i-th 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 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. (That is, under the usual assumptions, value iteration will converge to some approximation of the value function.) The rest of this section states the required property more formally, then proves that some common function approximators have this property.

DEFINITION: Two operators M_F and T on the space S are *compatible* if repeated application of $M_F \circ T$ 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 $M_F, T, M_F, T, M_F, \ldots$ causes convergence from any initial guess x_0, then it also causes convergence from $T(x_0)$; so the sequence $T, M_F, T, M_F, \ldots$ also causes convergence.

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 $\gamma < 1$. Let $X = S \times A$. Let F be a be a function approximator (for functions from X to $\mathbb{R}$) with mapping $M_F \in \mathbb{R}^{|X|} \mapsto \mathbb{R}^{|X|}$. Suppose M_F is a nonexpansion in max norm. Then M_F is compatible with T_M, and $M_F \circ T_M$ has contraction factor γ.*

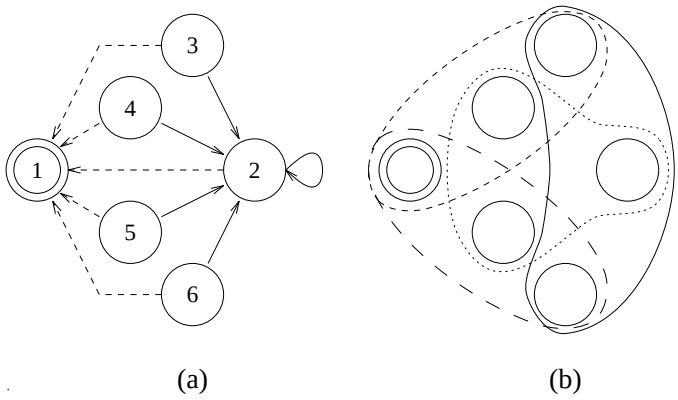

Figure 3: A Markov process and a CMAC which are incompatible. Part (a) shows the process. Its goal is state 1. On each step, with probability 95%, the process follows a solid arrow, and with probability 5%, it follows a dashed arrow. All arc costs are zero. Part (b) shows the CMAC, which has 4 receptive fields each covering 3 nodes. If the CMAC starts out with all predictions equal to 1, approximate value iteration produces the series of target values $1, \frac{77}{60}, (\frac{77}{60})^2, \ldots$ for state 2.

PROOF: By the value contraction theorem, T_M is a contraction in max norm with factor γ. By assumption, M_F is a nonexpansion in max norm. Therefore $M_F \circ T_M$ 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 which function approximators are compatible with value iteration. Many common approximators are incompatible with standard parallel value backup operators. For example, as figure 2 demonstrates, linear regression can be an expansion in max norm; and Boyan and Moore [BM95] show that the combination of value iteration with linear regression can diverge. Other incompatible methods include standard feedforward neural nets, some forms of spline fitting, and local weighted regression [BM95].

A particularly interesting case is the CMAC. Sutton [SS94] has recently reported success in combining value iteration with a CMAC, and has suggested that function approximators similar to the CMAC are likely to allow temporal differencing to converge in the online case. While we have no evidence to support or refute this suggestion, it is worth mentioning that convergence is not guaranteed in our offline framework, as the counterexample in figure 3 shows.

In this example, the optimal value function is uniformly zero, since all arc costs are zero. The exact value backup operator assigns 0 to $V(1)$ (since state 1 is the goal) and $.05V(1) + .95V(2)$ to $V(i)$ for $i \neq 1$ (since all states except 1 have a 5% chance of transitioning to state 1 and a 95% chance of transitioning to state 2). The approximate backup operator computes these same 6 numbers as training data for the CMAC; but since the CMAC can't represent this function exactly, our output is the closest representable function in the sense of least squared error. If the exact operator produces $k\mathbf{v}$ where $\mathbf{v} = (0, 1, 1, 1, 1, 1)^{\mathrm{T}}$, then the closest representable function will be $k\mathbf{w}$ where $\mathbf{w} = (\frac{1}{3}, \frac{4}{3}, \frac{5}{6}, \frac{5}{6}, \frac{5}{6}, \frac{5}{6})^{\mathrm{T}}$. If we repeat the process from this new value function, the exact backup operator will now produce $\frac{77k}{60}\mathbf{v}$, and the CMAC will produce $\frac{77k}{60}\mathbf{w}$. Further iteration causes divergence at the rate $\frac{77}{60}$.

The CMAC still diverges even if we choose a small learning rate: with learning rate α, the rate of divergence is $1 + \alpha\frac{17}{60}$. However, if we train the CMAC based on actual trajectories in the Markov process, as Sutton suggests, we no longer diverge: transitions out of state 2, which lower the coefficients of our CMAC substantially, are as frequent as transitions into state 2, which raise the coefficients somewhat. In fact, since this example is a Markov process rather than a Markov decision process, convergence of the online algorithm is guaranteed by a theorem in [Day92].

We will prove that a broad class of approximation methods is compatible with value iteration. This class includes kernel averaging, k-nearest-neighbor, weighted k-nearest-neighbor, Bèzier patches, linear interpolation on a triangular (or tetrahedral, *etc.*) mesh, bilinear interpolation on a square (or cubical, *etc.*)

mesh, and many others. (See the Experiments section for a definition of bilinear interpolation. Note that the square mesh is important: on non-rectangular meshes, bilinear interpolation will sometimes need to extrapolate.)

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 $\hat{Y}_i$ 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 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 $\hat{Y}_i = \beta_i k_i + \sum_j \beta_{ij} Y_j$.

It should be obvious that all of the methods mentioned before the definition are averagers: in all cases, the fitted value at any given coordinate is a weighted average of target values, and the weights are determined by distances in X values, and so are unaffected by the target Y values.

Theorem 3.2 *The mapping M_F associated with any averaging method F is a nonexpansion in max norm, and is therefore compatible with the parallel value backup operator for any discounted MDP.*

PROOF: Fix the βs and ks for F as in the above definition. Let Y and Z be two vectors of target values. Consider a particular coordinate i. Then, letting $\| \cdot \|$ denote max norm,

$$\begin{aligned} \left| [M_F(Y) - M_F(Z)]_i \right| &= \left| (\beta_i k_i + \sum_j \beta_{ij} Y_j) - (\beta_i k_i + \sum_j \beta_{ij} Z_j) \right| \\ &= \left| \sum_j \beta_{ij} (Y_j - Z_j) \right| \\ &\leq \max_j (Y_j - Z_j) \\ &= \| Y - Z \| \end{aligned}$$

That is, 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 Nondiscounted processes

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 value backup operator T_M for this process is a contraction in some weighted max norm $\| \cdot \|_W$. The previous section proved that if the approximation method A is an averager, then M_A is a nonexpansion in (unweighted) max norm. If we could prove that M_A were also a nonexpansion in $\| \cdot \|_W$, we would have M_A compatible with T_M. Unfortunately, M_A may be an expansion in $\| \cdot \|_W$; in fact parts (a) and (b) of figure 4 show a simple example of a nondiscounted MDP and an averager which are incompatible. (Part (c) is a simple proof that the MDP and the averager are incompatible; see below for an explanation.)

Fortunately, there are averagers which are compatible with nondiscounted MDPs. The proof relies on an intriguing property of 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 $V(y)$. Part (b) of figure 4 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. Part (c) of figure 4 shows a simple example; a slightly more complicated example is in figure 5. As the following theorem shows, exact value iteration on this derived MDP is the same as approximate 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'.*

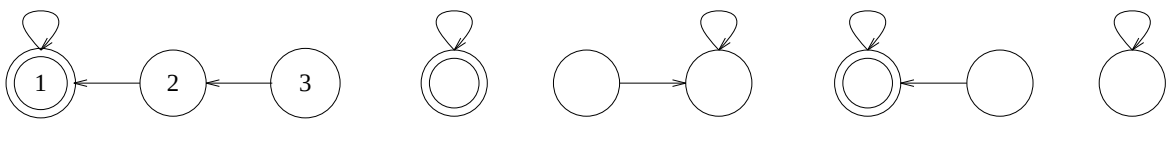

Figure 4: A nondiscounted deterministic Markov process, and an averager with which it is incompatible. The process is shown in (a); the goal is state 1, and all arc costs except at the goal are 1. In (b) we see an averager, represented as a Markov process: states 1 and 3 are unchanged, while $V(2)$ is replaced by $V(3)$. The derived Markov process is shown in (c); state 3 has been disconnected, so its value estimate will diverge.

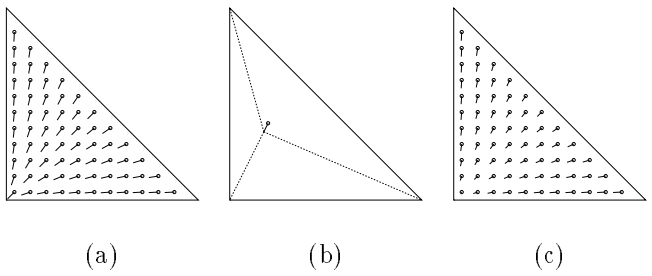

Figure 5: An example of the construction of the derived Markov process. Part (a) shows a deterministic Markov process: its state space is the unit triangle, and on every step the agent moves a constant distance towards the origin. The value of each state is simply its distance from the origin, so the value function is nonlinear. For our function approximator, we will use linear interpolation on the three corners of the triangle. Part (b) shows a representative transition from the derived process: as before, the agent moves towards the goal, but then the averager moves the agent randomly to one of the three corners. On average, this scattering moves the agent back away from the goal, so steps in the derived process don't move the agent as far on average as they did in the original process. Part (c) shows the expected progress the agent makes on each step. The value function for the derived process is $V^*(x, y) = x + y$.

PROOF: Define the derived MDP M' as follows. It will have the same state and action spaces as M, and it will also have the same discount factor and initial distribution. We can assume without loss of generality that state 1 of M is cost-free and absorbing: if not, we can renumber the states of M starting at 2, add a new state 1 which satisfies this property, and make all of its incoming transition probabilities zero. We can also assume, again without loss of generality, that $\beta_1 = 1$ and $k_1 = 0$ (that is, that A always sets $V(1) = 0$) — again, if this property does not already hold for M, we can add a new state 1.

Suppose that, in M, action a in state x takes us to state y with probability p_{axy}. Suppose that A replaces $V(y)$ by $\beta_y k_y + \sum_z \beta_{yz} V(z)$. Then we will define the transition probabilities in M' for state x and action a to be

$$\begin{aligned} p'_{axz} &= \sum_y p_{axy}\beta_{yz} \qquad (z \neq 1) \\ p'_{ax1} &= \sum_y p_{axy}(\beta_{y1} + \beta_y) \end{aligned}$$

These transition probabilities make sense: since A is an averager, we know that $\sum_z \beta_{yz} + \beta_y$ is 1, so

$$\begin{aligned} \sum_z p'_{axz} &= \sum_{z \neq 1} \sum_y p_{axy}\beta_{yz} + \sum_y p_{axy}(\beta_{y1} + \beta_y) \\ &= \sum_y p_{axy} \left(\sum_z \beta_{yz} + \beta_y \right) \\ &= \sum_y p_{axy} = 1 \end{aligned}$$

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

$$c'_{xa} = c_{xa} + \gamma \sum_{y'} p_{axy'}\beta_{y'}k_{y'}$$

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

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

On the other hand, the parallel value backup operator for M is

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

If we replace $V(y)$ by its approximation under A, the operator becomes $T_M \circ M_A$:

$$\begin{aligned} V(x) &\leftarrow \min_a \sum_y p_{axy} \left(c_{xa} + \gamma(\beta_y k_y + \sum_z \beta_{yz} V(z)) \right) \\ &= \min_a \left(c_{xa} + \gamma \sum_y p_{axy} \beta_y k_y + \gamma \sum_y p_{axy} \sum_{z \neq 1} \beta_{yz} V(z) \right) \end{aligned}$$

which is exactly the same as $T_{M'}$ above. □

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 in A. Then it alternately applies 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.) On the other hand, exact value iteration on M' produces the series of functions $V_0, T_M \circ M_A(V_0), T_M \circ M_A(T_M \circ M_A(V_0)), \ldots$. 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 functions $M_A(\ldots)$, while iteration on M' would stop at one of the functions $T_M(\ldots)$.

Now we can see why the combination in figure 4 diverges: the derived MDP has a state with infinite cost. So, in order to prove compatibility, we need to guarantee that the derived MDP is well-behaved.

If the arc costs of a discounted MDP M have finite mean and variance, it is obvious that the arc costs of M' also have finite mean and variance. That means that $T_{M'} = T_M \circ M_A$ converges in max norm at the rate γ — *i.e.*, we have just proven again that M_A is compatible with T_M.

More importantly, if M is a finite nondiscounted process, there are averagers which are compatible with it. For example, if A uses weight decay (*i.e.*, if $\beta_y > 0$ for all y), then M' will have all policies proper, since any action in any state has a nonzero probability of bringing us immediately to state 1.

More generally, if M has only proper policies, we may partition its state space by distance from state 1, as follows (see [BT89]). Let $S_1 = \{1\}$. Now recursively define $U_k = \bigcup_{j<k} S_k$ and S_k as

$$S_k = \left\{ x \,\middle|\, (x \notin U_k) \wedge (\min_{a \in A} \max_{y \in U_k} P(\delta(x,a) = y) > 0) \right\}$$

Bertsekas and Tsitsiklis show that this partitioning is exhaustive; so we may set $k(x)$ to be the unique k so that $x \in S_k$. If M is finite, there will be finitely many nonempty S_k.

We will say that an averager A is *self-weighted* for M if for every state y, either $\beta_y > 0$ or there exists a state x so that $k(x) \leq k(y)$ and $\beta_{yx} > 0$.

Now, if A is self-weighted for a finite nondiscounted MDP M, then M' will have only proper policies: if at some time we are in state x, then no matter what action a we take, there is a nonzero chance that we will immediately follow an arc to a state in S_k for $k < k(x)$. (By definition of the partition, there is a transition in M under a from x to some y so that $k(y) < k(x)$. By the self-weighting property, either β_{yz} must be nonzero for some z so that $k(z) \leq k(y)$, in which case x has a possible transition in M' under a to z; or $\beta_y > 0$, in which case x has a possible transition in M' under a to state 1.) If we follow such an arc, there is then a nonzero chance that we will immediately follow another arc to a state in $S_{k'}$ for $k' < k$, and so forth until we eventually (with positive probability) reach partition S_1 and therefore state 1. Let m be the largest integer so that S_m is nonempty. Then the previous discussion shows that, no matter what state we start from, we have a positive probability of reaching state 1 in $m-1$ steps. Call the smallest such probability ϵ. Then in $k(m-1)$ steps, we have probability at least $1-(1-\epsilon)^k$ of reaching state 1. The limit of this quantity as $k \to \infty$ is 1; so with probability 1 we eventually absorb from any initial state.

We have just proven

Corollary 4.1 *Let M be a finite nondiscounted Markov decision process. Suppose all policies in M are proper. Let A be an averager which is self-weighted for M. Let T_M and M_A be the parallel backup operator for M and the mapping associated with A. Then T_M and M_A are compatible, and the approximate value iteration algorithm based on A will converge if is applied to M.*

If A ignores $V(x)$ for all states x not in some sample X_0, then the states in X_0 will play an important role in the derived MDP M'. While the initial state of a trajectory may be outside of X_0, all transitions in M' lead to states in X_0, so after one step the trajectory will enter X_0 and stay there indefinitely. This means that we can characterize a large part of M' by looking only at its behavior on X_0 — just what we need for a tractable algorithm. (It is worth mentioning that, if M is nondiscounted, X_0 should contain the goal state: if it doesn't, M' will have no transitions into the goal, so all of its values will be infinite.)

5 The online problem and Q-learning

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

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

in which, rather than replacing $V(x)$ by its computed update on each step, we take a weighted average of the old and new values of $V(x)$. (The weights α_x may differ for each x, and may change from iteration to iteration.) That is, we can still construct the derived MDP M' and perform gradual value iteration on it, and gradual value iteration on M' is still the same as gradual approximate value iteration on M.

The results also apply nearly directly to dynamic programming with Watkins' [Wat89] Q^* function

$$Q^*(x,a) \equiv E(c(x,a) + \gamma V^*(\delta(x,a)))$$

and Q-learning operator

$$Q(x,a) \leftarrow_{\alpha_{xa}} E(c(x,a) + \gamma \min_{a'} Q(\delta(x,a),a'))$$

(The learning rates α_{xa} may now be random variables, 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 MDP, up to but not including the current values of the random variables $c(x,a)$ and $\delta(x,a)$.) That is, we can still define a derived MDP so that the behavior of the approximate algorithm on the original MDP is the same as the behavior of the exact algorithm on the derived MDP. The derived MDP is, however, slightly different from the derived MDP for value iteration; see the Appendix for details.

The previous paragraphs imply that, if we could sample transitions at will from the derived MDP (and so compute unbiased estimates of the Q-learning updates), we could apply gradual Q-learning to these transitions and learn a policy. 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 [JJS94, Tsi94].

Unfortunately, there is a catch. Q-learning is designed to work for online problems, where we don't know the cost or transition functions and can only sample transitions from our current state. The power of the approximate value iteration method, on the other hand, comes from the fact that we can pay attention only to transitions from a certain small set of states. So, while the derived MDP for Q-learning will still have only a few relevant states, we won't in general be able to observe many transitions from these states, and so the approximate Q-learning iteration will take a very long time to converge.

There are two ways that the approximate Q-learning algorithm might still be useful. The first is if we encounter a problem which is somewhere between online and offline: we can sample any transition at will, but don't know the cost or transition functions a priori or can't compute the necessary expectations. In such a problem we still can't use value iteration, since it is difficult to compute an unbiased estimate of the value iteration update, so the approximate Q-learning algorithm is helpful.

The second way is if we are willing to accept possible lack of convergence. Suppose our function approximator pays attention to the states in the set X_0. If we pretend that every transition we see from a state $x \notin X_0$ is actually a transition from the nearest state $x' \in X_0$, we will then have enough data to compute the behavior of the derived MDP on X_0. Unfortunately, following this approximation is equivalent to introducing hidden state into the derived MDP; so we now run the risk of divergence.

6 Converging to what?

Until now, we have only considered the convergence or divergence of approximate dynamic programming algorithms. Of course we would like not only convergence, but convergence to a reasonable approximation of the value function. The next section contains some empirical studies of approximate value iteration; this section proves some error bounds, then outlines the types of problems we have encountered while experimenting with approximate value iteration.

6.1 Error bounds

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, we will know that it converges to the right answer.

If M is discounted and has bounded costs, $T_M \circ M_A$ will always converge; the following theorem describes some situations where nondiscounted problems converge.

Theorem 6.1 *Let V^* be the optimal value function for a finite nondiscounted Markov decision process M. Let T_M be the parallel value backup operator for M. Let M_A be the mapping for an averager A, and let V^* also be a fixed point of M_A. Then V^* is a fixed point of $T_M \circ M_A$; and if either*

1. *M has all policies proper and A is self-weighted for M, or*
2. *M has $E(c(x,a)) > 0$ for all $x \neq 1$ and A has $k_x \geq 0$ for all x*

then iteration of $T_M \circ M_A$ converges to V^*.

PROOF: By the value contraction theorem, V^* is a fixed point of T_M. So $T_M \circ M_A(V^*) = T_M(V^*) = V^*$; that is, V^* is a fixed point of $T_M \circ M_A$.

If (1) holds, then by the corollary to the derived MDP theorem, $T_M \circ M_A$ is a contraction in some weighted max norm; so V^* is the unique fixed point of $T_M \circ M_A$, and iteration of $T_M \circ M_A$ converges to V^*.

If (2) holds, then M' has $c'(x,a) > 0$ for all $x \neq 1$; so every improper policy in M' has infinite cost for some initial states. If we can show that M' has at least one proper policy, then $T_M \circ M_A$ must converge (and therefore must converge to V^*).

Note that $V^*(1) = 0$ and $V^*(x) > 0$ for $x \neq 1$, since all arc costs in M are positive. Suppose we start at some non-goal state x in M', and choose an action a so that $V^*(x) = E(c'(x,a) + V^*(\delta'(x,a)))$. (There must be such an action, since V^* is a fixed point of the value backup operator for M'.) Since $c'(x,a) > 0$, we know that $V^*(x) > E(V^*(\delta'(x,a)))$. In particular, there must be a possible transition to some state y so that $V^*(x) > V^*(y)$. If y is not the goal, we can repeat the argument to find a z so that y has a possible transition to z and $V^*(y) > V^*(z)$, and so forth until (with positive probability) we eventually reach the goal. □

The above theorem is useful only when we know that the optimal value function is a fixed point of our averager. For example, it shows that bilinear interpolation will converge to the exact value function for a gridworld, if every arc's cost is equal to its (Manhattan) length, since the value function for this MDP is linear. 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 somewhere near a fixed point of our averager, we can still guarantee an error bound for approximate value iteration.

Theorem 6.2 *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 the mapping for an averager A. 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 $\| V_0 - V^* \| \leq \frac{2\gamma\epsilon}{1-\gamma}$.*

PROOF: By the value contraction theorem, T_M is a contraction in max norm with factor γ. By theorem 3.2, M_A is a nonexpansion in max norm. So, $T_M \circ M_A$ is a contraction in max norm with factor γ, and therefore

converges to some V_0. Repeated application of the triangle inequality and the definition of a contraction give

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

which is what was required. □

If we let $\gamma \to 0$, we can make the above error bound arbitrarily small. This result is somewhat counter-intuitive, since A may not even be able to represent V^* exactly. The reason for this behavior is that the final step in computing V_0 is to apply T_M; when $\gamma = 0$, this step produces V^* immediately.

As mentioned in a previous section, approximate value iteration returns $M_A(V_0)$ rather than V_0 itself. So, an error bound for $M_A(V_0)$ would be useful. The error bound on V_0 leads directly to a bound for $M_A(V_0)$:

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

By way of comparison, Chow and Tsitsiklis [CT89] bound the error introduced by using a grid of side h to compute the value function for a type of continuous-state-space MDP. Writing V_h^* for the approximate value function computed this way, their bound is

$$\| V^* - V_h^* \| \leq \frac{h}{1-\gamma}(K_1 + \gamma K_2 \| V^* \|_Q)$$

(for all sufficiently small h) where $\| \cdot \|_Q$ is the *span quasi-norm*

$$\| F \|_Q = \sup_x F(x) - \inf_x F(x)$$

and K_1 and K_2 are constants. While their formalism allows them to discretize the available actions as well as the state space, an approximation which we don't consider, their bound also applies to processes with a finite number of actions. So, we can compare their bound on V_h^* for the case of non-discretized actions to our bound on $M_A(V_0)$.

Write M_h for the mapping associated with discretization on a grid of side h. (For definiteness, assume that $(M_h(V))(x) = V(x')$ where x' is the center of the grid cell which contains x. Almost any other reasonable convention would also work.) The processes that Chow and Tsitsiklis consider satisfy

$$| V^*(x) - V^*(x') | \leq L \| x - x' \|$$

for some constant L (*i.e.*, V^* is *Lipschitz continuous*). (We can determine L easily from the cost and transition functions.) So, we can find a fixed point V_h of M_h which is close to V^*, as follows. Pick a grid cell

C. Write C_H and C_L for the maximum and minimum values of V^* in C; write x_H and x_L for states $x \in C$ which achieve these values. (Such states exist because V^* is continuous.) We know that $\| x_H - x_L \| \leq h$, since the diameter of C is h. Therefore, by the Lipschitz condition, $| C_H - C_L | \leq hL$. So, if we set $V_h(x)$ to be $\frac{1}{2}(C_H - C_L)$ for all $x \in C$, the largest difference between V^* and V_h on C will be $\frac{hL}{2}$. Applying our error bound now gives

$$\| V^* - V_h^* \| = hL(1 + \frac{\gamma}{1-\gamma})$$

which has slightly different constants but the same behavior as $h \to 0$ or $\gamma \to 1$ as the bound of Chow and Tsitsiklis. Our bound is valid for any h, rather than just sufficiently small h. Their bound depends on $\| V^* \|_Q$; since $\| V^* \|_Q$ is bounded by $\frac{2r}{1-\gamma}$, where r is the largest single-step expected reward, we have folded a similar dependence into the constant L.

The sort of error bound which we have proved is particularly useful for function approximators such as linear interpolation which have many fixed points. (In fact, every function which is representable by a linear interpolator is a fixed point of that interpolator's mapping. The same is true for bilinear interpolation, grids, and 1-nearest-neighbor; it is not true for local weighted averaging or k-nearest-neighbor for $k > 1$.) Because it depends on the maximum difference between V^* and the fixed point of M_A, the error 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^*.

6.2 In practice

The most common problem with approximate value iteration is the presence of barriers in the derived MDP. That is, sometimes the derived MDP can be divided into two pieces so that the first piece contains the goal and the second piece has no transitions into the first. In this case, the estimated values of the states in the second piece will be infinite. (We mentioned a special case of this situation above: if the averager ignores the goal state, then the derived MDP will have no transitions into the goal.) A less drastic but similar problem occurs when the second piece has only low-probability transitions to the first; in this case, the costs for states in the second piece will not be infinite, but will still be artificially inflated.

This sort of problem is likely to happen when the MDP has short transitions and when there are large regions where a single state dominates the averager. For a particularly bad example, suppose our function approximator is 1-nearest-neighbor. If the transitions out of a sampled state x in M are shorter than half the distance to the nearest adjacent sampled state, then the only transitions out of x in M' will lead straight back to x. Similarly, in local weighted averaging with a narrow kernel, a short transition out of x in M will translate to a high probability self loop in M'. In both cases, the effect of the averager is to produce a drag on transitions out of x: actions in M' don't get the agent as far on average as they did in M.

One way to reduce this drag is to make sure that no single state has the dominant weight over a large region. The best way to do so is to sample the state space more densely; but if we could afford to do that, we wouldn't need a function approximator in the first place. Another way is to increase a smoothing parameter such as kernel width or number of neighbors, and so reduce the weight of each sample point in its immediate neighborhood. Unfortunately, increased smoothing brings its own problems: it can remove exactly the features of the value function that we are interested in. For example, if the agent must follow a long, narrow path to the goal, the scattering effect of a wide-kernel averager is almost certain to push it off of the path long before it reaches the end. We will see an example of this problem in the hill-car experiment below.

Both of the above problems — too much smoothing and the introduction of barriers — can be reduced if we can alter our MDP so that the actions move the agent farther. For example, we might look ahead two or more time steps at each value backup. (This strategy corresponds to the dynamic programming operator $T_M^n \circ M_A$ for some $n > 1$. Since T_M^n is the backup operator for an MDP (derived by composing n copies of M), the previous sections' convergence theorems also apply to $T_M^n \circ M_A$.) While in general the cost of looking ahead n steps is exponential in n, there are many circumstances where we can reduce this cost dramatically. For instance, in a physical simulation, we can choose a longer time increment; in a grid world, we can consider only the compound actions which don't contain two steps in opposite directions; and in the

case of a Markov process, where there's only 1 action, the cost of lookahead is linear rather than exponential in n. (In the last case, TD(λ) [Sut88] allows us to combine lookaheads at several depths.) If actions are selected from an interval of $\mathbb{R}$, numerical minimum-finding algorithms such as Newton's method or golden section search can find a local minimum quickly. In any case, if the depth and branching factor are large enough, standard heuristic search techniques can at least chip away at the base of the exponential.

7 Experiments

This section describes our experiments with the Markov decision problems from [BM95].

7.1 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 .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 6.

For a function approximator, we will use bilinear interpolation, defined as follows: to find the predicted value at a point (x, y), first find the corners (x_0, y_0), (x_0, y_1), (x_1, y_0), and (x_1, y_1) of the grid square containing (x, y). Interpolate along the left edge of the square between (x_0, y_0) and (x_0, y_1) to find the predicted value at (x_0, y). Similarly, interpolate along the right edge to find the predicted value at (x_1, y). Now interpolate across the square between (x_0, y) and (x_1, y) to find the predicted value at (x, y).

Figure 6 shows the cost function for one of the actions, the optimal value function computed on a 100×100 grid, an estimate of the optimal value function computed with bilinear interpolation on the corners of a 7×7 grid (*i.e.*, on 64 sample points), and the difference between the two estimates. Since the optimal value function is nearly piecewise linear outside the puddles, but curved inside, the interpolation performs much better outside the puddles: the root mean squared difference between the two approximations is 2.27 within one step of the puddles, and .057 elsewhere. (The lowest-resolution grid which beats bilinear interpolation's performance away from the puddles is 20×20; but even a 5×5 grid can beat its performance near the puddles.)

7.2 Car on a hill

In this world, 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 $[-1, 1] \times [-2, 2]$, which represents the position and velocity of the car; there are two actions, forward and reverse. (This formulation differs slightly from [BM95]: they allowed a third action, coast. We expect that the difference makes the problem no more or less difficult.) The cost function measures time until goal.

There are several interesting features to this world. 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 performance of approximate value iteration in this situation. Second, there is a long, narrow region of state space near the goal through which all optimal trajectories must pass (it is the region where the car is partway up the hill and moving quickly forward). So, excessive smoothing will cause errors over large regions of the state space. Finally, the physical simulation uses a fairly small time step, .03 seconds, so we need fine resolution in our function approximator just to make sure that we don't introduce a barrier.

The results of our experiments appear in figure 7. For a reference model, we fit a 128×128 grid. While this model has 16384 parameters, it is still less than perfect: the right end of the discontinuity is somewhat rough. (Boyan and Moore used a 200 by 200 grid to compute their optimal value function, and it shows no perceptible roughness at this boundary.) We also fit two smaller grids, one 64×64 and one 32×32. Finally, we fit a weighted 4-nearest neighbor model using the 1024 centers of the cells of the 32×32 grid as sample points, and another using a uniform random sample of 1000 points from the state space. Note that

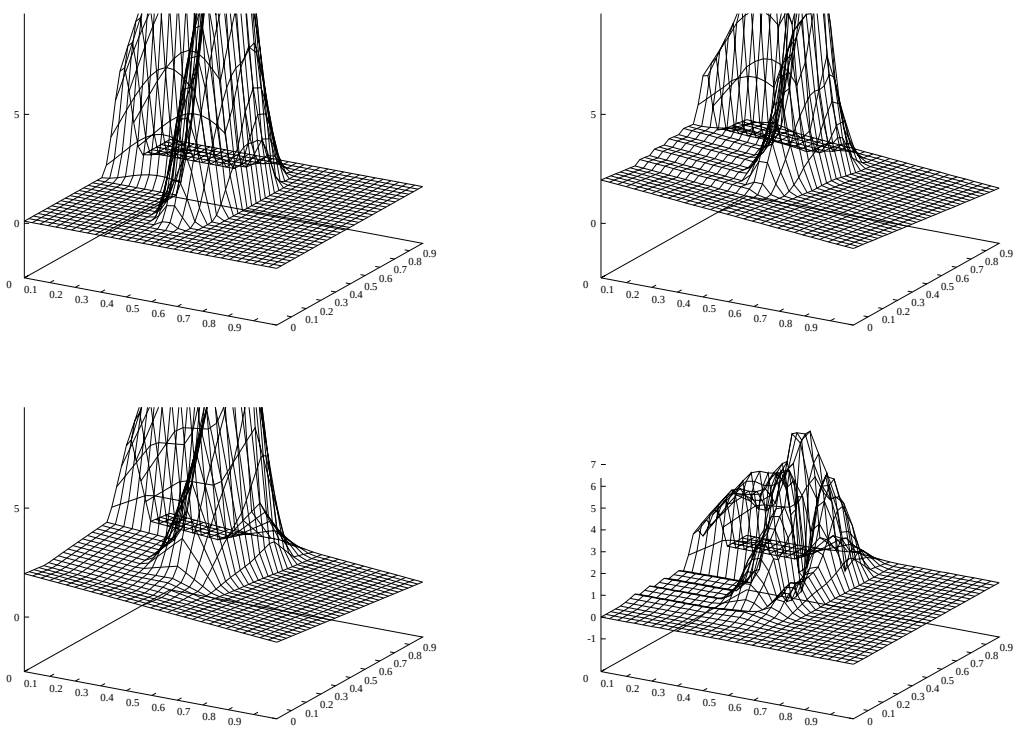

Figure 6: The puddle world. From top left: the cost of moving up, the optimal value function as seen by a 100×100 grid, the optimal value function as seen by bilinear interpolation on the corners of a 7×7 grid, and the difference between the two value functions.

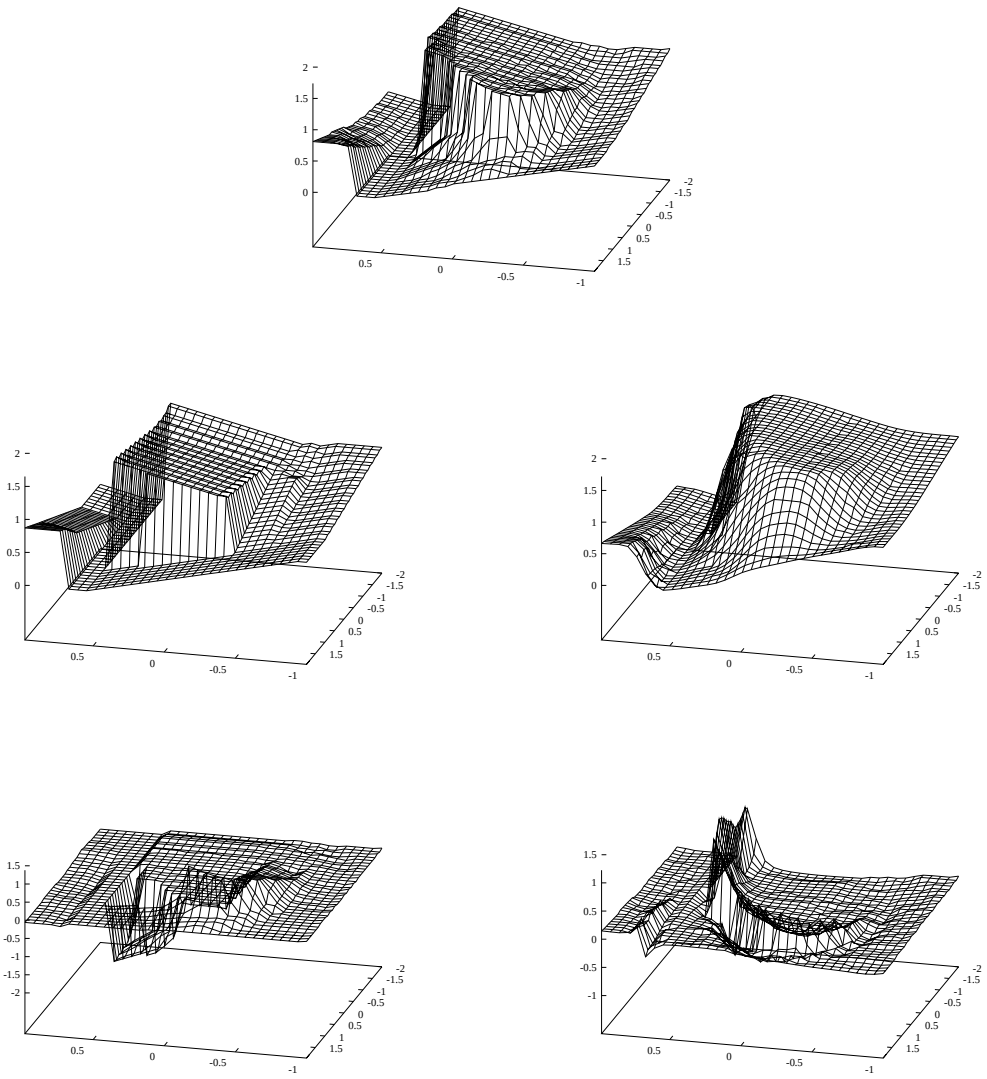

Figure 7: Approximations to the value function for the hill-car problem. From the top: the reference model, a 32×32 grid, a k-nearest-neighbor model, the error of the 32×32 grid, and the error of the k-nearest-neighbor model. In each plot, the horizontal axes represent the agent's position and velocity, and the vertical axis represents the estimated time remaining until reaching the summit at $x = .6$.

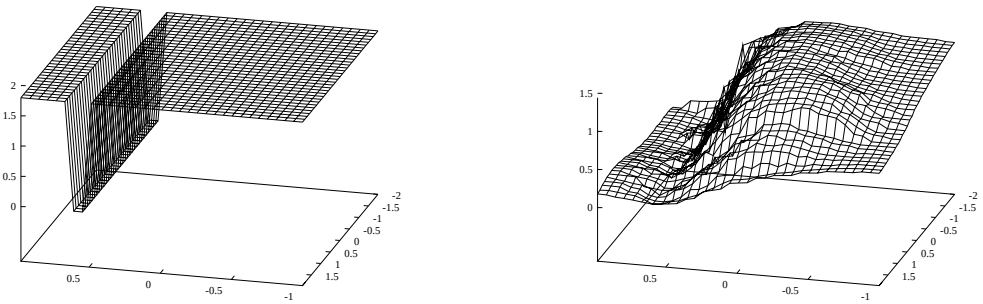

Figure 8: Two smaller models for the hill-car world: a divergent 12×12 grid, and a convergent nearest-neighbor model on the same 144 sample points.

the nearest-neighbor methods are roughly comparable in complexity to the 32×32 grid: each one requires us to evaluate about two thousand transitions in the MDP for every value backup.

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 grids perform better than the nearest-neighbor models (as we would expect, since the nearest-neighbor models tend to smooth out discontinuities). But away from the discontinuity, the nearest-neighbor models win. The 32×32 nearest-neighbor model also beats the 32×32 grid at the right end of the discontinuity: the car is moving slowly enough here that the grid thinks that one of the actions keeps the car in exactly the same place. The nearest-neighbor model, on the other hand, since it smooths more, doesn't introduce as much drag as the grid does and so doesn't have this problem. The root mean square error of the 64×64 grid (not shown) from the reference model is $0.190s$, and of the 32×32 grid is $0.336s$. The RMS error of the 4-nearest-neighbor fitter with samples at the grid points is $0.205s$. The nearest-neighbor fitter with a random sample (not shown) performs slightly worse, but still significantly better than the 32×32 grid (one-tailed t-test gives $p = .971$): its error, averaged over 5 runs, is $0.235s$.

All of the above models are fairly large: the smallest one requires us to evaluate 2000 transitions for every value backup. Figure 8 shows what happens when we try to fit a smaller model. The 12×12 grid is shown after 60 iterations; it is in the process of diverging, since the transitions are too short to reach the goal from adjacent grid cells. The 4-nearest-neighbor fitter on the same 144 grid points has converged; its RMS error from the reference model is $0.278s$ (better than the 32×32 grid, despite needing to simulate fewer than one-seventh as many transitions). A 4-nearest-neighbor fitter on a random sample of size 150 (not shown) also converged, with RMS error $0.423s$.

8 Conclusions and further research

We have proved convergence for a wide class of approximate 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, linear regression, and CMACs, 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. (In some cases, though, it is possible to detect and compensate for this instability. The grow-support algorithm of [BM95], which detects instability by interspersing TD(1) "rollouts," is a good example.)

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 important 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 still does not exaggerate; but it 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 [Moo94, Moo91], so there is hope that a proof is possible.

A The derived process for Q-learning

Here is the analog for Q-learning of the derived MDP theorem. The chief difference is that, where the theorem for value iteration considered the combined operator $T_M \circ M_A$, this version considers $M_A \circ T_M$. The difference is necessary to keep the min operation in the Q-learning backup from getting in the way. Of course, if we show that either $T_M \circ M_A$ or $M_A \circ T_M$ converges from any initial guess, then the other must also converge.

Theorem A.1 (Derived MDP for Q-learning) *For any averager A with mapping M_A, and for any MDP M (either discounted or nondiscounted) with Q-learning backup operator T_M, the function $M_A \circ T_M$ is the Q-learning backup operator for a new Markov decision process M'.*

PROOF: The domain of A will now be pairs of states and actions. Write β_{xayb} for the coefficient of $Q(y,b)$ in the approximation of $Q(x,a)$; write k_{xa} and β_{xa} for the constant and its coefficient.

Take an initial guess $Q(x,a)$. Write Q' for the result of applying T_M to Q; write Q'' for the result of applying M_A to Q'. Then we have

$$\begin{aligned}
Q'(x,a) &= E(c(x,a) + \gamma \min_b Q(\delta(x,a),b)) \\
&= c_{xa} + \gamma \sum_y p_{xay} \min_b Q(y,b) \\
Q''(z,c) &= \sum_x \sum_a \beta_{zcxa} Q'(x,a) + \beta_{zc} k_{zc} \\
&= \sum_x \sum_a \beta_{zcxa} \left(c_{xa} + \gamma \sum_y p_{xay} \min_b Q(y,b) \right) + \beta_{zc} k_{zc} \\
&= \sum_x \sum_a \beta_{zcxa} c_{xa} + \gamma \sum_x \sum_a \beta_{zcxa} \sum_y p_{xay} \min_b Q(y,b) + \beta_{zc} k_{zc} \\
&= \left(\sum_x \sum_a \beta_{zcxa} c_{xa} + \beta_{zc} k_{zc} \right) + \gamma \sum_y \left(\sum_x \sum_a \beta_{zcxa} p_{xay} \right) \min_b Q(y,b)
\end{aligned}$$

We now interpret the first parenthesis above as the cost of taking action c from state z in M'; the second parenthesis is the transition probability p'_{zcy} for M'. Note that the sum $\sum_y p'_{zcy}$ will generally be less than 1; so we will make up the difference by adding a transition in M' from state z with action c to state 1 (which is assumed as before to be cost-free and absorbing and to have $V(1) \equiv 0$). □

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. This material is based on work supported under a National Science Foundation Graduate Research Fellowship, and by NSF grant number BES-9402439. 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.

References

[BD59] R. Bellman and S. Dreyfus. Functional approximations and dynamic programming. *Mathematical Tables and Aids to Computation*, 13:247–251, 1959.

[Bel58] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16(1):87–90, 1958.

[Bel61] R. Bellman. *Adaptive Control Processes*. Princeton University Press, 1961.

[BKK63] 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.

[Bla65] D. Blackwell. Discounted dynamic programming. *Annals of Mathematical Statistics*, 36:226–235, 1965.

[BM95] 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.

[BT89] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice Hall, 1989.

[CT89] 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.

[Day92] P. Dayan. The convergence of TD(λ) for general lambda. *Machine Learning*, 8(3–4):341–362, 1992.

[FF62] L. R. Ford, Jr. and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[JJS94] 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.

[Lin92] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning, and teaching. *Machine Learning*, 8(3–4):293–322, 1992.

[Moo91] 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.

[Moo94] 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.

[Sab93] P. Sabes. Approximating Q-values with basis function representations. In *Proceedings of the Fourth Connectionist Models Summer School*, Hillsdale, NJ, 1993. Lawrence Erlbaum.

[Sam59] A. L. Samuels. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959.

[SS94] S. P. Singh and R. S. Sutton. Reinforcement learning with replacing eligibility traces. Manuscript in preparation, 1994.

[Sut88] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[Tes90] G. Tesauro. Neurogammon: a neural network backgammon program. In *IJCNN Proceedings III*, pages 33–39, 1990.

[TS93] 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.

[Tsi94] J. N. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16(3):185–202, 1994.

[Wat89] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, England, 1989.

[WD92] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3–4):279–292, 1992.

[Wit77] I. H. Witten. An adaptive optimal controller for discrete-time Markov environments. *Information and Control*, 34:286–295, 1977.