Journal of Artificial Intelligence Research 15 (2001) 351-381 Submitted 9/00; published 11/01

Experiments with Infinite-Horizon, Policy-Gradient Estimation

Jonathan Baxter JBAXTER@WHIZBANG.COM
WhizBang! Labs.
4616 Henry Street Pittsburgh, PA 15213
Peter L. Bartlett BARTLETT@BARNHILLTECHNOLOGIES.COM
BIOwulf Technologies.
2030 Addison Street, Suite 102, Berkeley,CA 94704
Lex Weaver LEX.WEAVER@ANU.EDU.AU
Department of Computer Science
Australian National University , Canberra 0200, Australia

Abstract

In this paper, we present algorithms that perform gradient ascent of the average reward in a partially observable Markov decision process (POMDP). These algorithms are based on GPOMDP, an algorithm introduced in a companion paper (Baxter & Bartlett, 2001), which computes biased estimates of the performance gradient in POMDPs. The algorithm's chief advantages are that it uses only one free parameter $\beta \in [0, 1)$, which has a natural interpretation in terms of bias-variance trade-off, it requires no knowledge of the underlying state, and it can be applied to infinite state, control and observation spaces. We show how the gradient estimates produced by GPOMDP can be used to perform gradient ascent, both with a traditional stochastic-gradient algorithm, and with an algorithm based on conjugate-gradients that utilizes gradient information to bracket maxima in line searches. Experimental results are presented illustrating both the theoretical results of Baxter and Bartlett (2001) on a toy problem, and practical aspects of the algorithms on a number of more realistic problems.

1. Introduction

Function approximation is necessary to avoid the curse of dimensionality associated with large-scale dynamic programming and reinforcement learning problems. The dominant paradigm is to use the function to approximate the state (or state and action) values. Most algorithms then seek to minimize some form of error between the approximate value function and the true value function, usually by simulation (Sutton & Barto, 1998; Bertsekas & Tsitsiklis, 1996). While there have been a multitude of empirical successes for this approach (for example, Samuel, 1959; Tesauro, 1992, 1994; Baxter, Tridgell, & Weaver, 2000; Zhang & Dietterich, 1995; Singh & Bertsekas, 1997), there are only weak theoretical guarantees on the performance of the policy generated by the approximate value function. In particular, there is no guarantee that the policy will improve as the approximate value function is trained; in fact performance can degrade even when the function class contains an approximate value function whose corresponding greedy policy is optimal (see Baxter & Bartlett, 2001, Appendix A, for a simple two-state example).

An alternative technique that has received increased attention recently is the "policy-gradient" approach in which the parameters of a stochastic policy are adjusted in the direction of the gradient of some performance criterion (typically either expected discounted reward or average reward). The key problem is how to compute the performance gradient under conditions of partial observability when an explicit model of the system is not available.

This question has been addressed in a large body of previous work (Barto, Sutton, & Anderson, 1983; Williams, 1992; Glynn, 1986; Cao & Chen, 1997; Cao & Wan, 1998; Fu & Hu, 1994; Singh, Jaakkola, & Jordan, 1994, 1995; Marbach & Tsitsiklis, 1998; Marbach, 1998; Baird & Moore, 1999; Rubinstein & Melamed, 1998; Kimura, Yamamura, & Kobayashi, 1995; Kimura, Miyazaki, & Kobayashi, 1997). See the introduction of (Baxter & Bartlett, 2001) for a discussion of the history of policy-gradient approaches. Most existing algorithms rely on the existence of an identifiable recurrent state in order to make their updates to the gradient estimate, and the variance of the algorithms is governed by the recurrence time to that state. In cases where the recurrence time is too large (for instance because the state space is large), or in situations of partial observability where such a state cannot be reliably identified, we need to seek alternatives that do not require access to such a state.

Motivated by these considerations, Baxter and Bartlett (2001, 2000) introduced and analysed GPOMDP—an algorithm for generating a *biased* estimate of the gradient of the average reward in general Partially Observable Markov Decision Processes (POMDPs) controlled by parameterized stochastic policies. The chief advantages of GPOMDP are that it requires only a single sample path of the underlying Markov chain, it uses only one free parameter $\beta \in [0, 1)$, which has a natural interpretation in terms of bias-variance trade-off, and it requires no knowledge of the underlying state.

More specifically, suppose $\theta \in \mathbb{R}^K$ are the parameters controlling the POMDP. For example, θ could be the parameters of an approximate neural-network value-function that generates a stochastic policy by some form of randomized look-ahead, or θ could be the parameters of an approximate Q function used to stochastically select controls[1]. Let $\eta(\theta)$ denote the average reward of the POMDP with parameter setting θ. GPOMDP computes an approximation $\nabla_\beta \eta(\theta)$ to $\nabla \eta(\theta)$ based on a single continuous sample path of the underlying Markov chain. The accuracy of the approximation is controlled by the parameter $\beta \in [0, 1)$, and one can show that

$$\nabla \eta(\theta) = \lim_{\beta \to 1} \nabla_\beta \eta(\theta).$$

The trade-off preventing choosing β arbitrarily close to 1 is that the *variance* of GPOMDP's estimates of $\nabla_\beta \eta(\theta)$ scale as $1/(1-\beta)^2$. However, on the bright side, it can also be shown that the *bias* of $\nabla_\beta(\theta)$ (measured by $\|\nabla_\beta \eta(\theta) - \nabla \eta(\theta)\|$) is proportional to $\tau(1-\beta)$ where τ is a suitable *mixing time* of the Markov chain underlying the POMDP (Bartlett & Baxter, 2000a). Thus for "rapidly mixing" POMDP's (for which τ is small), estimates of the performance gradient with acceptable *bias* and *variance* can be obtained.

Provided $\nabla_\beta \eta(\theta)$ is a sufficiently accurate approximation to $\nabla \eta(\theta)$—in fact, $\nabla_\beta \eta(\theta)$ need only be within 90° of $\nabla \eta(\theta)$—small adjustments to the parameters θ in the direction $\nabla_\beta \eta(\theta)$ will guarantee improvement in the average reward $\eta(\theta)$. In this case, gradient-based optimization algorithms using $\nabla_\beta \eta(\theta)$ as their gradient estimate will be guaranteed to improve the average reward $\eta(\theta)$ on each step. Except in the case of table-lookup, most value-function based approaches to reinforcement learning cannot make this guarantee.

In this paper we present a conjugate-gradient ascent algorithm that uses the estimates of $\nabla_\beta \eta(\theta)$ provided by GPOMDP. Critical to the successful operation of the algorithm is a novel line search

1. Stochastic policies are not strictly necessary in our framework, but the policy must be "differentiable" in the sense that $\nabla \eta(\theta)$ exists.

subroutine that brackets maxima by relying solely upon gradient estimates. This largely avoids problems associated with finding the maximum using noisy value estimates. Since the parameters are only updated after accumulating sufficiently accurate estimates of the gradient direction, we refer to this approach as the "off-line" algorithm. This approach essentially allows us to take a stochastic gradient optimization problem and treat it as a non-stochastic optimization problem, thus enabling the use of a large body of accumulated heuristics and algorithmic improvements associated with such methods. We also present a more traditional, "on-line" stochastic gradient ascent algorithm based on GPOMDP that updates the parameters at every time step. This algorithm is essentially the algorithm proposed in (Kimura et al., 1997).

The off-line and on-line algorithms are applied to a variety of problems, beginning with a simple 3-state Markov decision process (MDP) controlled by a linear function for which the true gradient can be exactly computed. We show rapid convergence of the gradient estimates $\nabla_\beta \eta(\theta)$ to the true gradient, in this case over a large range of values of β. With this simple system we are able to illustrate vividly the bias/variance tradeoff associated with the selection of β. We then compare the performance of the off-line and on-line approaches applied to finding a good policy for the MDP. The off-line algorithm reliably finds a near-optimal policy in less than 100 iterations of the Markov chain, an order of magnitude faster than the on-line approach. This can be attributed to the more aggressive exploitation of the gradient information by the off-line method.

Next we demonstrate the effectiveness of the off-line algorithm in training a neural network controller to control a "puck" in a two-dimensional world. The task in this case is to reliably navigate the puck from any starting configuration to an arbitrary target location in the minimum time, while only applying discrete forces in the x and y directions. Although the on-line algorithm was tried for this problem, convergence was considerably slower and we were not able to reliably find a good local optimum.

In the third experiment, we use the off-line algorithm to train a controller for the call admission queueing problem treated in (Marbach, 1998). In this case near-optimal solutions are found within about 2000 iterations of the underlying queue, 1-2 orders of magnitude faster than the experiments reported in (Marbach, 1998) with on-line (stochastic-gradient) algorithms.

In the fourth and final experiment, the off-line algorithm was used to reliably train a switched neural-network controller for a two-dimensional variation on the classical "mountain-car" task (Sutton & Barto, 1998, Example 8.2).

The rest of this paper is organized as follows. In Section 2 we introduce POMDPs controlled by stochastic policies, and the assumptions needed for our algorithms to apply. GPOMDP is described in Section 3. In Section 4 we describe the off-line and on-line gradient-ascent algorithms, including the gradient-based line-search subroutine. Experimental results are presented in Section 5.

2. POMDPs Controlled by Stochastic Policies

A partially observable, Markov decision process (POMDP) consists of a state space $\mathcal{S}$, observation space $\mathcal{Y}$ and a control space $\mathcal{U}$. For each state $i \in \mathcal{S}$ there is a deterministic reward $r(i)$. Although the results in Baxter and Bartlett (2001) only guarantee convergence of GPOMDP in the case of finite $\mathcal{S}$ (but continuous $\mathcal{U}$ and $\mathcal{Y}$), the algorithm can be applied regardless of the nature of $\mathcal{S}$ so we do not restrict the cardinality of $\mathcal{S}, \mathcal{U}$ or $\mathcal{Y}$.

Consider first the case of discrete $\mathcal{S}$, $\mathcal{U}$ and $\mathcal{Y}$. Each control $u \in \mathcal{U}$ determines a stochastic matrix $P(u) = [p_{ij}(u)]$ giving the transition probability from state i to state j ($i, j \in \mathcal{S}$). For each state $i \in \mathcal{S}$, an observation $Y \in \mathcal{Y}$ is generated independently according to a probability distribution $\nu(i)$ over observations in $\mathcal{Y}$. We denote the probability that $Y = y$ by $\nu_y(i)$. A *randomized policy* is simply a function μ mapping observations into probability distributions over the controls $\mathcal{U}$. That is, for each observation $y \in \mathcal{Y}$, $\mu(y)$ is a distribution over the controls in $\mathcal{U}$. Denote the probability under μ of control u given observation y by $\mu_u(y)$.

For continuous $\mathcal{S}, \mathcal{Y}$ and $\mathcal{U}$, $p_{ij}(u)$ becomes a *kernel* $k_{ij}(u)$ giving the probability density of transitions from i to j, $\nu(i)$ becomes a probability density function on $\mathcal{Y}$ with $\nu_y(i)$ the density at y, and $\mu(y)$ becomes a probability density function on $\mathcal{U}$ with $\mu_u(y)$ the density at u.

To each randomized policy μ there corresponds a Markov chain in which state transitions are generated by first selecting an observation Y in state i according to the distribution $\nu(i)$, then selecting a control U according to the distribution $\mu(Y)$, and finally generating a transition to state j according to the probability $p_{ij}(U)$.

At present we are only dealing with a fixed POMDP. To parameterize the POMDP we parameterize the policies, so that μ now becomes a function $\mu(\theta, y)$ of a set of parameters $\theta \in \mathbb{R}^K$, as well as of the observation y. The Markov chain corresponding to θ has state transition matrix $P(\theta) = [p_{ij}(\theta)]$ given by

$$p_{ij}(\theta) = \mathbf{E}_{Y \sim \nu(i)} \mathbf{E}_{U \sim \mu(\theta, Y)} p_{ij}(U). \tag{1}$$

Note that the policies μ are *purely reactive* or *memoryless* in that their choice of action is based only upon the current observation. All the experiments described in the present paper use purely reactive policies. Aberdeen and Baxter (2001) have extended GPOMDP and the techniques of the present paper to controllers with internal state.

The following technical assumptions are required for the operation of GPOMDP.

Assumption 1. *The derivatives,*

$$\frac{\partial \mu_u(\theta, y)}{\partial \theta_k},$$

exist, and the ratios

$$\frac{\left| \dfrac{\partial \mu_u(\theta, y)}{\partial \theta_k} \right|}{\mu_u(\theta, y)}$$

are uniformly bounded by $B < \infty$, for all $u \in \mathcal{U}$, $y \in \mathcal{Y}$, $\theta \in \mathbb{R}^K$ and $k = 1, \ldots, K$.

The second part of this assumption is needed because the ratio appears in the GPOMDP algorithm. It allows zero-probability actions $\mu_u(\theta, y) = 0$ only if $\nabla \mu_u(\theta, y)$ is also zero, in which case we set $0/0 = 0$. See Section 5 for examples of policies satisfying this requirement.

Assumption 2. *The magnitudes of the rewards, $|r(i)|$, are uniformly bounded by $R < \infty$ for all states i.*

For deterministic rewards, his condition only represents a restriction in infinite state spaces. However, all the results in the present paper apply to bounded stochastic rewards, in which case $r(i)$ is the expectation of the reward in state i.

Assumption 3. *Each $P(\theta), \theta \in \mathbb{R}^K$, has a unique stationary distribution $\pi(\theta) = [\pi_1(\theta, \ldots, \pi_n(\theta)]$, satisfying the* balance equations:

$$\pi(\theta) P(\theta) = \pi(\theta).$$

Assumption 3 ensures that, for all parameters θ, the Markov chain forms a single recurrent class. Since any finite-state Markov chain always ends up in a recurrent class, and it is the properties of this class that determine the long-term average reward, this assumption is mainly for convenience so that we do not have to include the recurrence class as a quantifier in our theorems. Observe that *episodic* problems, such as the minimization of time to a goal state, may be modeled in a way that satisfies Assumption 3 by simply resetting the agent upon reaching the goal state back to some initial starting distribution over states. Examples are described in Section 5.

The *average reward* $\eta(\theta)$ is simply the expected reward under the stationary distribution $\pi(\theta)$:

$$\eta(\theta) = \sum_{i=1}^{n} \pi_i(\theta) r(i). \tag{2}$$

Because of Assumption 3, $\eta(\theta)$ is also equal to the expected long-term average of the reward received when starting from any state i:

$$\eta(\theta) = \lim_{T\to\infty} \mathbf{E}\left(\frac{1}{T} \sum_{t=0}^{T-1} r(X_t) \,\middle|\, X_0 = i \right).$$

Here the expectation is over sequences of states $X_0, \dots, X_{T-1}$ with state transitions generated by $P(\theta)$ (note that the expectation is independent of the starting state i).

3. The GPOMDP Algorithm

GPOMDP (Algorithm 1) is an algorithm for computing a *biased* estimate Δ_T of the *gradient* of the average reward $\nabla\eta(\theta)$. Δ_T satisfies

$$\lim_{T\to\infty} \Delta_T = \nabla_\beta \eta(\theta),$$

where $\nabla_\beta\eta(\theta)$ ($\beta \in [0,1)$) is an approximation to $\nabla\eta(\theta)$ satisfying

$$\nabla\eta(\theta) = \lim_{\beta\to 1} \nabla_\beta \eta(\theta),$$

(Baxter & Bartlett, 2001, Theorems 2, 5). Note that GPOMDP relies only upon a single sample path from the POMDP. Also, it does not require knowledge of the transition probability matrix P, nor of the observation process ν; it only requires knowledge of the randomized policy μ, in particular the ability to compute the gradient of the probability of the chosen control divided by the probability of the chosen control.

We cannot set β arbitrarily close to 1 in GPOMDP, since the *variance* of the estimate is proportional to $1/(1-\beta)^2$. However, on the bright side, it can also be shown that the *bias* of $\nabla_\beta(\theta)$ (measured by $\|\nabla_\beta\eta(\theta) - \nabla\eta(\theta)\|$) is proportional to $\tau(1-\beta)$ where τ is a suitable *mixing time* of the Markov chain underlying the POMDP (Bartlett & Baxter, 2000a). Under Assumption 3, regardless of the initial starting state, the distribution over states converges to the stationary distribution $\pi(\theta)$ when the agent is following policy $\mu(\theta, \cdot)$. Standard Markov chain theory shows that the rate of convergence to $\pi(\theta)$ is exponential, and loosely speaking, the mixing time τ is the time constant in the exponential decay.

Algorithm 1 GPOMDP$(\beta, T, \theta) \to \mathbb{R}^K$

Given:

- $\beta \in [0, 1)$.
- $T > 0$.
- Parameters $\theta \in \mathbb{R}^K$.
- Randomized policy $\mu(\theta, \cdot)$ satisfying Assumption 1.
- POMDP with rewards satisfying Assumption 2, and which when controlled by $\mu(\theta, \cdot)$ generates stochastic matrices $P(\theta)$ satisfying Assumption 3.
- Arbitrary (unknown) starting state X_0.

Set $z_0 = 0$ and $\Delta_0 = 0$ ($z_0, \Delta_0 \in \mathbb{R}^K$).
for $t = 0$ to $T - 1$ **do**
    Observe Y_t (generated according to the observation distribution $\nu(X_t)$)
    Generate control U_t according to $\mu(\theta, Y_t)$
    Observe $r(X_{t+1})$ (where the next state X_{t+1} is generated according to $p_{X_t X_{t+1}}(U_t)$).
    Set $z_{t+1} = \beta z_t + \dfrac{\nabla \mu_{U_t}(\theta, Y_t)}{\mu_{U_t}(\theta, Y_t)}$
    Set $\Delta_{t+1} = \Delta_t + r(X_{t+1}) z_{t+1}$
end for
$\Delta_T \leftarrow \Delta_T / T$
return Δ_T

Thus β has a natural interpretation in terms of a bias/variance trade-off: small values of β give lower variance in the estimates Δ_T, but higher bias in that the expectation of Δ_T may be far from $\nabla\eta(\theta)$, whereas values of β close to 1 yield small bias but correspondingly larger variance. Fortunately, for problems which mix rapidly (small τ), β can be small and still yield reasonable bias. This bias/variance trade-off is vividly illustrated in the experiments of Section 5; see (Bartlett & Baxter, 2000a) for a more detailed theoretical discussion of the bias/variance question.

4. Stochastic Gradient Ascent Algorithms

This section introduces two approaches to exploiting the gradient estimates produced by GPOMDP:

1. an off-line approach based on traditional conjugate-gradient optimization techniques but employing a novel line-search mechanism to cope with the noise in GPOMDP's estimates, and

2. an on-line stochastic optimization approach that uses the core update in GPOMDP ($r(X_t)z_t$) to update the parameters θ on *every* iteration of the POMDP.

4.1 Off-line optimization of the average reward

GPOMDP generates biased and noisy estimates Δ_T of the gradient of the average reward $\nabla\eta(\theta)$ for POMDPs controlled by parameterized stochastic policies. A straightforward algorithm for finding local maxima of $\eta(\theta)$ would be to compute $\Delta_T(\theta)$ at the current parameter settings θ, and then modify θ by $\theta \leftarrow \theta + \gamma\Delta_T(\theta)$. Provided $\Delta_T(\theta)$ is close enough to the true gradient direction $\nabla\eta(\theta)$, and provided the step-sizes γ are suitably decreasing, standard stochastic optimization theory tells us that this technique will converge to a local maximum of $\eta(\theta)$. However, given that each computation of $\Delta_T(\theta)$ requires many iterations of the POMDP to guarantee suitably accurate gradient estimates (that is, in general T needs to be large), we would like to more aggressively exploit the information contained in $\Delta_T(\theta)$ than by simply adjusting the parameters θ by a small amount in the direction $\Delta_T(\theta)$.

There are two techniques for making better use of gradient information that are widely used in *non-stochastic* optimization: better choice of the search direction and better choice of step size. Better search directions can be found by employing *conjugate-gradient* directions rather than the pure gradient direction. Better step sizes are usually obtained by performing some kind of line-search to find a local maximum in the search direction, or through the use of second order methods. Since line-search techniques tend to be more robust to departures from quadraticity in the optimization surface, we will only consider those here (however, see Baxter & Bartlett, 2001, Section 7.3, for a discussion of how second-order derivatives may be computed with a GPOMDP-like algorithm).

CONJPOMDP, described in Algorithm 2, is a version of the Polak-Ribiere conjugate-gradient algorithm (see, *e.g.* Fine, 1999, Section 5.5.2) that is designed to operate using only noisy (and possibly) biased estimates of the gradient of the objective function (for example, the estimates Δ_T provided by GPOMDP). The argument GRAD to CONJPOMDP computes the gradient estimate. The novel feature of CONJPOMDP is GSEARCH, a linesearch subroutine that uses only gradient information to find the local maximum in the search direction. The use of gradient information ensures GSEARCH is robust to noise in the performance estimates. Both CONJPOMDP and GSEARCH can be applied to any stochastic optimization problem for which noisy (and possibly) biased gradient estimates are available.

The argument s_0 to CONJPOMDP provides an initial step-size for GSEARCH. The argument ϵ provides a stopping condition; when $\|\mathrm{GRAD}(\theta)\|^2$ falls below ϵ, CONJPOMDP terminates.

4.2 The GSEARCH algorithm

The key to the successful operation of CONJPOMDP is the linesearch algorithm GSEARCH (Algorithm 3). GSEARCH uses only gradient information to bracket the maximum in the direction θ^*, and then quadratic interpolation to jump to the maximum.

We found the use of gradients to bracket the maximum far more robust than the use of function values. To illustrate why this is so, in Figure 1 we have plotted a stylized view of the average reward $\eta(\theta)$ along some search direction θ^* (labeled "f" in the figure), and its gradient in that direction $\nabla\eta(\theta)\cdot\theta^*$ (labeled "grad(f)"). There are two ways we could search in the direction θ^* to bracket the maximum of $\eta(\theta)$ in that direction (at 0 in this case), one using function values and the other using gradient estimates:

1. Find three points $\theta_1, \theta_2, \theta_3$, all lying in the direction θ^* from θ, such that $\eta(\theta_1) < \eta(\theta_2)$ and $\eta(\theta_3) < \eta(\theta_2)$. Assuming no overshooting, we then know the maximum must lie between θ_1

Algorithm 2 $\mathrm{CONJPOMDP}(\mathrm{GRAD}, \theta, s_0, \epsilon)$

Given:

- $\mathrm{GRAD}\colon \mathbb{R}^K \to \mathbb{R}^K$: a (possibly noisy and biased) estimate of the gradient of the objective function to be maximized.
- Starting parameters $\theta \in \mathbb{R}^K$ (set to maximum on return).
- Initial step size $s_0 > 0$.
- Gradient resolution ϵ.

$g = h = \mathrm{GRAD}(\theta)$
while $\|g\|^2 \geq \epsilon$ **do**
    $\mathrm{GSEARCH}(\mathrm{GRAD}, \theta, h, s_0, \epsilon)$
    $\Delta = \mathrm{GRAD}(\theta)$
    $\gamma = (\Delta - g) \cdot \Delta / \|g\|^2$
    $h = \Delta + \gamma h$
    if $h \cdot \Delta < 0$ **then**
        $h = \Delta$
    end if
    $g = \Delta$
end while

and θ_3 and we can use the three points and quadratic interpolation to estimate the location of the maximum.

2. Find two points θ_1 and θ_2 such that $\nabla\eta(\theta_1) \cdot \theta^* > 0$ and $\nabla\eta(\theta_2) \cdot \theta^* < 0$, and again use quadratic interpolation (which corresponds to linear interpolation of the gradients) to estimate the location of the maximum.

Both of these approaches will be equally satisfactory provided there is no noise in either the function estimates $\eta(\theta)$, or the gradient estimates $\nabla\eta(\theta)$. However, when estimates of $\eta(\theta)$ or $\nabla\eta(\theta)$ are available only through simulation, they will necessarily be noisy and the situation will look more like Figure 2. In this case the use of gradients to bracket the maximum becomes more desirable, because the line-search technique based on value estimates could choose any of the peaks in the plot of f + noise as the location of the maximum, which occur nearly uniformly along the x-axis, whereas the second technique based on gradients would choose any of the *zero-crossings* of the noisy gradient plot, which are far closer to the true maximum[2]. This is illustrated in Figure 3.

Another view of this phenomenon is that regardless of the variance of our estimates of $\eta(\theta)$, the variance of sign $[\eta(\theta_1) - \eta(\theta_2)]$ approaches 1 (the maximum possible) as θ_1 approaches θ_2. Thus, to reliably bracket the maximum using noisy estimates of $\eta(\theta)$ we need to be able to reduce the variance of the estimates when θ_1 and θ_2 are close. In our case this means running the simulation

2. There is an implicit assumption in our argument that the noise processes in the gradient and value estimates are of approximately the same magnitude. If the variance of the value estimates is considerably smaller than the variance of the gradient estimates then we would expect bracketing with values to be superior. In all our experiments we found gradient bracketing to be superior.

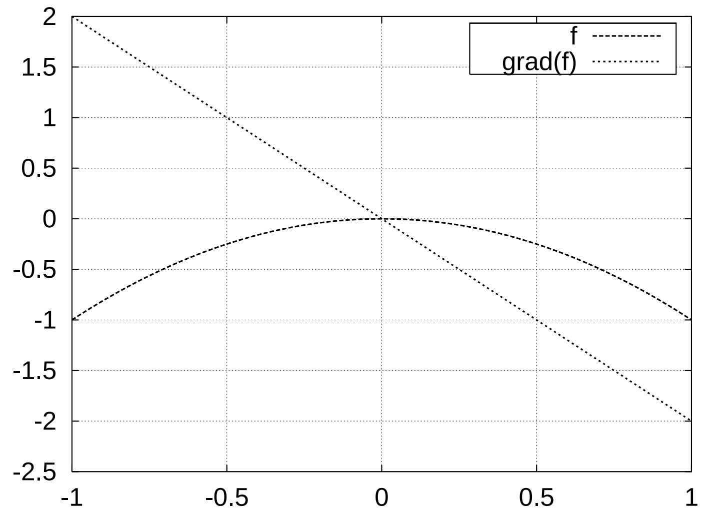

Figure 1: Stylized plot of the average reward $\eta(\theta)$ and the gradient $\nabla\eta(\theta)\cdot\theta^*$ in a search direction θ^*.

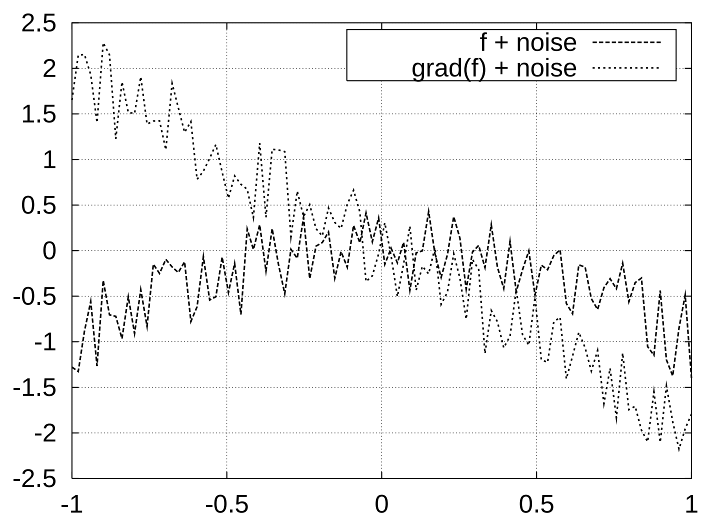

Figure 2: Plot as in Figure 1 but with estimation noise added to both the function and gradient curves.

from which the estimates are derived for longer and longer periods of time. In contrast, the variance of sign $\nabla\eta(\theta_1)\cdot\theta^*$ (and sign $\nabla\eta(\theta_2)\cdot\theta^*$) is independent of the distance between θ_1 and θ_2, and in particular does not grow as the two points approach one another.

One disadvantage to using gradient estimates to bracket is that it is not possible to detect extreme overshooting of the maximum. However, this can be avoided by using value estimates as a "sanity

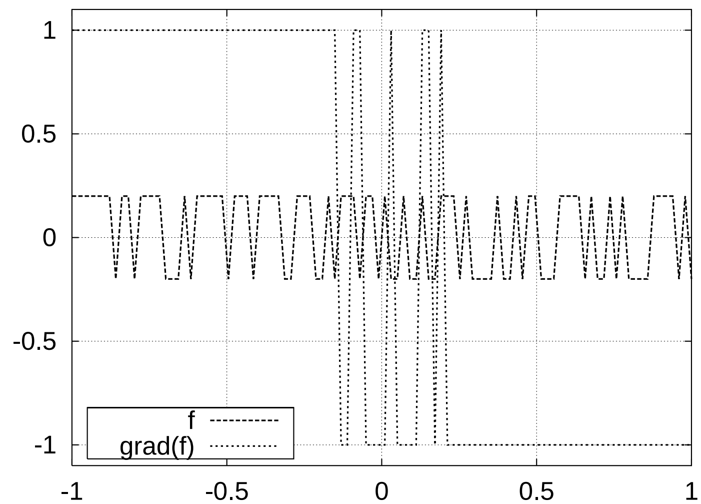

Figure 3: Plot of the possible maximum locations that would be found by a line-search algorithm based on value estimates (f), and one based on gradient estimates (grad(f)), for the curves in Figure 2. The zero-crossings in each case are the possible locations. Note that the gradient-based approach more accurately localizes the maximum.

check" to determine if the value has dropped dramatically, and suitably adjusting the search if this occurs.

In Algorithm 3, lines 5–25 bracket the maximum by finding a parameter setting $\theta_- = \theta_0 + s_-\theta^*$ such that $\mathrm{GRAD}(\theta_-) \cdot \theta^* > -\epsilon$, and a second parameter setting $\theta_+ = \theta_0 + s_+\theta^*$ such that $\mathrm{GRAD}(\theta_+) \cdot \theta^* < \epsilon$. The reason for ϵ rather than 0 in these expressions is to provide some robustness against errors in the estimates $\mathrm{GRAD}(\theta)$. It also prevents the algorithm "stepping to ∞" if there is no local maximum in the direction θ^*. Note that we use the same ϵ as used in CONJPOMDP to determine when to terminate due to small gradient (line 4 in CONJPOMDP).

Provided that the signs of the gradients at the bracketing points θ_- and θ_+ show that the maximum of the quadratic defined by these points lies between them, line 27 will jump to the maximum. Otherwise the algorithm simply jumps to the midpoint between θ_- and θ_+.

4.3 On-line optimization of the average reward: OLPOMDP

CONJPOMDP combined with GSEARCH operates by iteratively choosing "uphill" directions and then searching for a local maximum in the chosen direction. If the GRAD argument to CONJPOMDP is GPOMDP, the optimization will involve many iterations of the underlying POMDP between parameter updates.

In traditional stochastic optimization one typically uses algorithms that update the parameters at *every* iteration, rather than accumulating gradient estimates over many iterations. Algorithm 4, OLPOMDP, presents an adaptation of GPOMDP to this form. See Bartlett and Baxter (2000b) for a proof that OLPOMDP converges to the vicinity of a local maximum of $\eta(\theta)$. Note that OLPOMDP is very similar to the algorithms proposed in Kimura et al. (1995, 1997).

Algorithm 3 $\mathrm{GSEARCH}(\mathrm{GRAD}, \theta_0, \theta^*, s_0, \epsilon)$

Given:

- $\mathrm{GRAD}: \mathbb{R}^K \to \mathbb{R}^K$: a (possibly noisy and biased) estimate of the gradient of the objective function.
- Starting parameters $\theta_0 \in \mathbb{R}^K$ (set to maximum on return).
- Search direction $\theta^* \in \mathbb{R}^K$ with $\mathrm{GRAD}(\theta_0) \cdot \theta^* > 0$.
- Initial step size $s_0 > 0$.
- Inner product resolution $\epsilon >= 0$.

$s = s_0$

$\theta = \theta_0 + s\theta^*$

$\Delta = \mathrm{GRAD}(\theta)$

if $\Delta \cdot \theta^* < 0$ **then**

  Step back to bracket the maximum:

  repeat

    $s_+ = s$

    $p_+ = \Delta \cdot \theta^*$

    $s = s/2$

    $\theta = \theta_0 + s\theta^*$

    $\Delta = \mathrm{GRAD}(\theta)$

  until $\Delta \cdot \theta^* > -\epsilon$

  $s_- = s$

  $p_- = \Delta \cdot \theta^*$

else

  Step forward to bracket the maximum:

  repeat

    $s_- = s$

    $p_- = \Delta \cdot \theta^*$

    $s = 2s$

    $\theta = \theta_0 + s\theta^*$

    $\Delta = \mathrm{GRAD}(\theta)$

  until $\Delta \cdot \theta^* < \epsilon$

  $s_+ = s$

  $p_+ = \Delta \cdot \theta^*$

end if

if $p_- > 0$ and $p_+ < 0$ **then**

  $s = s_- - p_- \frac{s_+ - s_-}{p_+ - p_-}$

else

  $s = \frac{s_- + s_+}{2}$

end if

$\theta_0 = \theta_0 + s\theta^*$

Algorithm 4 OLPOMDP$(\beta, T, \theta_0) \to \mathbb{R}^K$.

Given:

- $\beta \in [0, 1)$.
- $T > 0$.
- Initial parameter values $\theta_0 \in \mathbb{R}^K$.
- Randomized parameterized policies $\left\{\mu(\theta, \cdot) \colon \theta \in \mathbb{R}^K\right\}$ satisfying Assumption 1.
- POMDP with rewards satisfying Assumption 2, and which when controlled by $\mu(\theta, \cdot)$ generates stochastic matrices $P(\theta)$ satisfying Assumption 3.
- Step sizes $\gamma_t, t = 0, 1, \dots$ satisfying $\sum \gamma_t = \infty$ and $\sum \gamma_t^2 < \infty$.
- Arbitrary (unknown) starting state X_0.

Set $z_0 = 0$ ($z_0 \in \mathbb{R}^K$).
for $t = 0$ to $T - 1$ **do**
    Observe Y_t (generated according to $\nu(X_t)$).
    Generate control U_t according to $\mu(\theta, Y_t)$
    Observe $r(X_{t+1})$ (where the next state X_{t+1} is generated according to $p_{X_t X_{t+1}}(U_t)$.
    Set $z_{t+1} = \beta z_t + \dfrac{\nabla \mu_{U_t}(\theta, Y_t)}{\mu_{U_t}(\theta, Y_t)}$
    Set $\theta_{t+1} = \theta_t + \gamma_t r(X_{t+1}) z_{t+1}$
end for
return θ_T

5. Experiments

In this section we present several sets of experimental results. Throughout this section, where we refer to CONJPOMDP we mean CONJPOMDP with GPOMDP as its GRAD argument.

In the first set of experiments, we consider a system in which a controller is used to select actions for a 3-state Markov Decision Process (MDP). For this system we are able to compute the true gradient exactly using the matrix equation

$$\nabla \eta(\theta) = \pi'(\theta) \nabla P(\theta) \left[I - P(\theta) + e\pi'(\theta)\right]^{-1} r, \tag{3}$$

where $P(\theta)$ is the transition matrix of the underlying Markov chain with the controller's parameters set to θ, $\pi'(\theta)$ is the stationary distribution corresponding to $P(\theta)$ (written as a row vector), $e\pi'(\theta)$ is the square matrix in which each row is the stationary distribution, and r is the (column) vector of rewards (see Baxter & Bartlett, 2001, Section 3, for a derivation of (3)). Hence we can compare the estimates Δ_T generated by GPOMDP with the true gradient $\nabla \eta(\theta)$, both as a function of the number of iterations T and as a function of the discount parameter β. We also optimize the performance of the controller using the on-line algorithm, OLPOMDP, and the off-line algorithm CONJPOMDP. CONJPOMDP reliably converges to a near optimal policy with around 100 iterations of the MDP, while the on-line method requires approximately 1000 iterations. This should be contrasted with

Origin State	Action	Destination State Probabilities A	B	C
A	a_1	0.0	0.8	0.2
A	a_2	0.0	0.2	0.8
B	a_1	0.8	0.0	0.2
B	a_2	0.2	0.0	0.8
C	a_1	0.0	0.8	0.2
C	a_2	0.0	0.2	0.8

Table 1: Transition probabilities of the three-state MDP

$$\begin{array}{lll} r(A) = 0 & \phi_1(A) = \frac{12}{18} & \phi_2(A) = \frac{6}{18} \\ r(B) = 0 & \phi_1(B) = \frac{6}{18} & \phi_2(B) = \frac{12}{18} \\ r(C) = 1 & \phi_1(C) = \frac{5}{18} & \phi_2(C) = \frac{5}{18} \end{array}$$

Table 2: Three-state rewards and features.

training a linear *value-function* for this system using $\mathrm{TD}(1)$ (Sutton, 1988), which can be shown to converge to a value function whose one-step lookahead policy is suboptimal (Weaver & Baxter, 1999).

In the second set of experiments, we consider a simple "puck-world" problem in which a small puck must be navigated around a two-dimensional world by applying thrust in the x and y directions. We train a 1-hidden-layer neural-network controller for the puck using $\mathrm{CONJPOMDP}$. Again the controller reliably converges to near optimality.

In the third set of experiments we use $\mathrm{CONJPOMDP}$ to optimize the admission thresholds for the call-admission problem considered in (Marbach, 1998).

In the final set of experiments we use $\mathrm{CONJPOMDP}$ to train a switched neural-network controller for a two-dimensional variant of the "mountain-car" task (Sutton & Barto, 1998, Example 8.2).

In all the experiments we found that convergence of the line-searches was greatly improved if all calls to the GPOMDP algorithm were seeded with the same random number sequence.

5.1 A three-state MDP

In this section we consider a three-state MDP, in each state of which there is a choice of two actions a_1 and a_2. Table 1 shows the transition probabilities as a function of the states and actions. Each state x has an associated two-dimensional feature vector $\phi(x) = (\phi_1(x), \phi_2(x))$ and reward $r(x)$ which are detailed in Table 2. Clearly, the optimal policy is to always select the action that leads to state C with the highest probability, which from Table 1 means always selecting action a_2.

This rather odd choice of feature vectors for the states ensures that a value function linear in those features and trained using $\mathrm{TD}(1)$—while observing the optimal policy—will implement a suboptimal greedy one-step lookahead policy (see (Weaver & Baxter, 1999) for a proof). Thus, in contrast to the gradient based approach, for this system, $\mathrm{TD}(1)$ training a linear value function is guaranteed to produce a worse policy if it starts out observing the optimal policy.

5.1.1 TRAINING A CONTROLLER

Our goal is to learn a stochastic controller for this system that implements an optimal (or near-optimal) policy. Given a parameter vector $\theta = (\theta_1, \theta_2, \theta_3, \theta_4)$, we generate a policy as follows. For any state x, let

$$\begin{aligned} s_1(x) &:= \theta_1\phi_1(x) + \theta_2\phi_2(x) \\ s_2(x) &:= \theta_3\phi_1(x) + \theta_4\phi_2(x). \end{aligned}$$

Then the probability of choosing action a_1 in state x is given by

$$\mu_{a_1}(x) = \frac{e^{s_1(x)}}{e^{s_1(x)} + e^{s_2(x)}},$$

while the probability of choosing action a_2 is given by

$$\mu_{a_2}(x) = \frac{e^{s_2(x)}}{e^{s_1(x)} + e^{s_2(x)}} = 1 - \mu_{a_1}(x).$$

The ratios $\frac{\nabla\mu_{a_i}(x)}{\mu_{a_i}(x)}$ needed by Algorithms 1 and 4 are given by,

$$\frac{\nabla\mu_{a_1}(x)}{\mu_{a_1}(x)} = \frac{e^{s_2(x)}}{e^{s_1(x)} + e^{s_2(x)}} \left[\phi_1(x), \phi_2(x), -\phi_1(x), -\phi_2(x)\right] \tag{4}$$

$$\frac{\nabla\mu_{a_2}(x)}{\mu_{a_2}(x)} = \frac{e^{s_1(x)}}{e^{s_1(x)} + e^{s_2(x)}} \left[-\phi_1(x), -\phi_2(x), \phi_1(x), \phi_2(x)\right] \tag{5}$$

Since the second two components in $\nabla\mu/\mu$ are always the negative of the first two, this shows that two of the parameters are redundant in this case: we could just as well have set $\theta_3 = -\theta_1$ and $\theta_4 = -\theta_2$.

5.1.2 GRADIENT ESTIMATES

With a parameter vector[3] of $\theta = [1, 1, -1, -1]$, GPOMDP was used to generate estimates Δ_T of $\nabla_\beta\eta$, for various values of T and $\beta \in [0, 1)$. To measure the progress of Δ_T towards the true gradient $\nabla\eta$, $\nabla\eta$ was calculated from (3) and then for each value of T the *angle* between Δ_T and $\nabla\eta$ and the relative error $\frac{\|\Delta_T - \nabla\eta\|}{\|\nabla\eta\|}$ were recorded. The angles and relative errors are plotted in Figures 4, 5 and 6.

The graphs illustrate a typical trade-off for the GPOMDP algorithm: small values of β give higher bias in the estimates, while larger values of β give higher variance (the final bias is only shown in Figure 6 for the norm deviation because it was too small to measure for the angular deviation). The bias introduced by having $\beta < 1$ is very small for this system. In the worst case, $\beta = 0.0$, the final gradient *direction* is indistinguishable from the true direction while the relative deviation $\frac{\|\nabla\eta - \Delta_T\|}{\|\nabla\eta\|}$ is only 7.7%.

3. Other initial values of the parameter vector were chosen with similar results. Note that $[1, 1, -1, -1]$ generates a suboptimal policy.

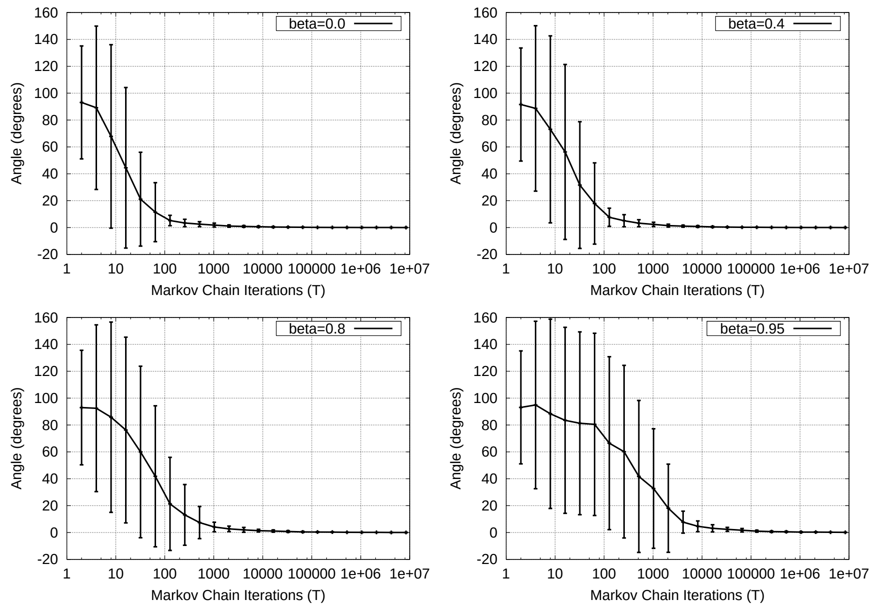

Figure 4: Angle between the true gradient $\nabla\eta$ and the estimate Δ_T for the three-state Markov chain, for various values of the discount parameter β. Δ_T was generated by Algorithm 1. Averaged over 500 independent runs. Note the higher variance at large T for the larger values of β. Error bars are one standard deviation.

5.1.3 TRAINING VIA CONJUGATE-GRADIENT ASCENT

CONJPOMDP with GPOMDP as the "GRAD" argument was used to train the parameters of the controller described in the previous section. Following the low bias observed in the experiments of the previous section, the argument β of GPOMDP was set to 0. After a small amount of experimentation, the arguments s_0 and ϵ of CONJPOMDP were set to 100 and 0.0001 respectively. None of these values were critical, although the extremely large initial step-size (s_0) did considerably reduce the time required for the controller to converge to near-optimality.

We tested the performance of CONJPOMDP for a range of values of the argument T to GPOMDP from 1 to 4096. Since GSEARCH only uses GPOMDP to determine the *sign* of the inner product of the gradient with the search direction, it does not need to run GPOMDP for as many iterations as CONJPOMDP does. Thus, GSEARCH determined its own T parameter to GPOMDP as follows. Initially, (somewhat arbitrarily) the value of T within GSEARCH was set to $1/10$ the value used in CONJPOMDP (or 1 if the value in CONJPOMDP was less than 10). GSEARCH then called GPOMDP to obtain an estimate Δ_T of the gradient direction. If $\Delta_T \cdot \theta^* < 0$ (θ^* being the desired search direction) then T was doubled and GSEARCH was called again to generate a new estimate Δ_T. This procedure was repeated until $\Delta_T \cdot \theta^* > 0$, or T had been doubled four times. If $\Delta_T \cdot \theta^*$ was still negative at the end of this process, GSEARCH searched for a local maximum in the direction $-\theta^*$, and the number of iterations T used by CONJPOMDP was doubled on the next iteration (the conclusion being that the direction θ^* was generated by overly noisy estimates from GPOMDP).

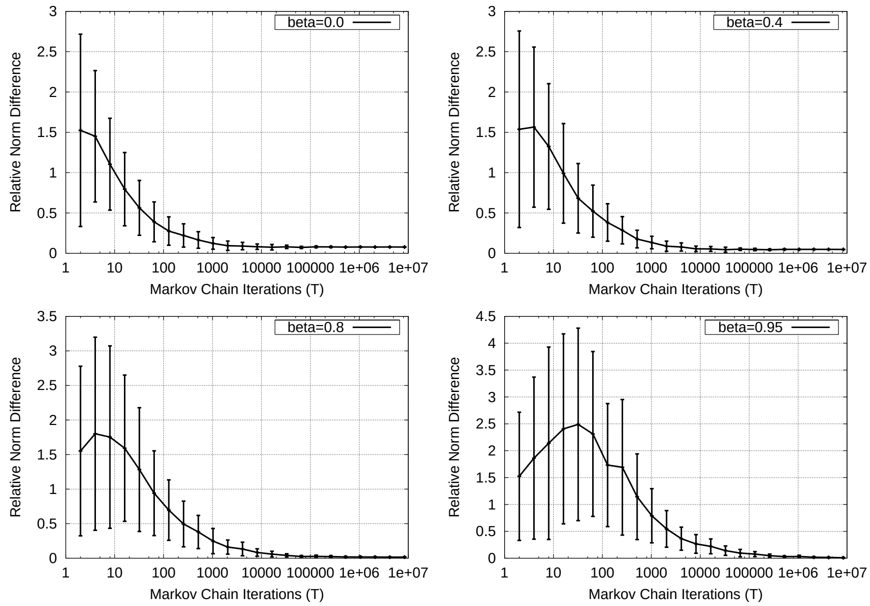

Figure 5: A plot of $\frac{\|\nabla\eta - \Delta_T\|}{\|\nabla\eta\|}$ for the three-state Markov chain, for various values of the discount parameter β. Δ_T was generated by Algorithm 1. Averaged over 500 independent runs. Note the higher variance at large T for the larger values of β. Error bars are one standard deviation.

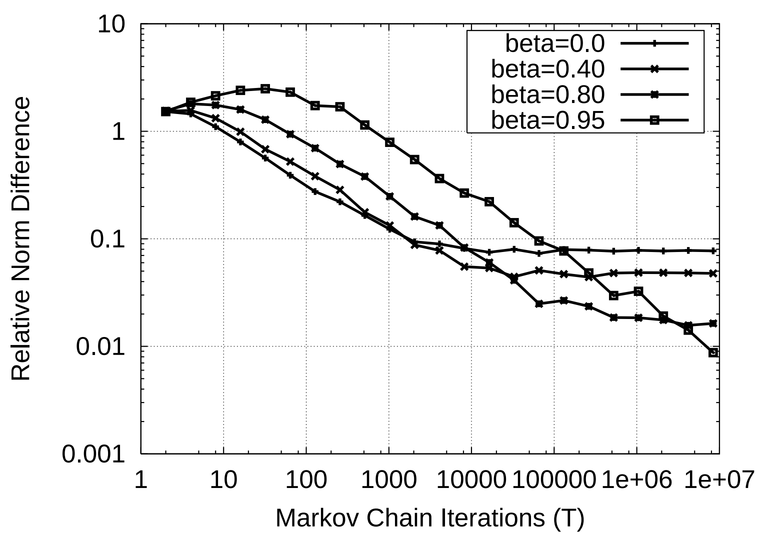

Figure 6: Graph showing the error in the estimate Δ_T (as measured by $\frac{\|\nabla\eta - \Delta_T\|}{\|\nabla\eta\|}$) for various values of β for the three-state Markov chain. Δ_T was generated by Algorithm 1. Note the decrease in the final bias as β increases. Both axes are log scales.

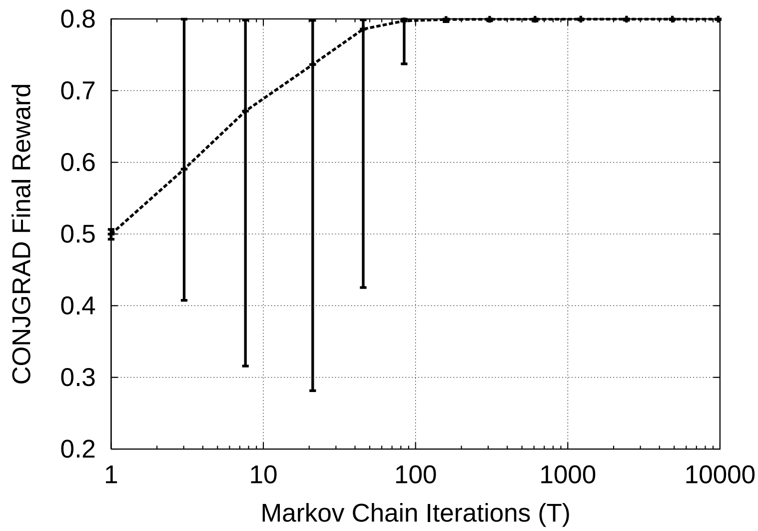

Figure 7: Performance of the 3-state Markov chain controller trained by CONJPOMDP as a function of the total number of iterations of the Markov chain. The performance was computed exactly from the stationary distribution induced by the controller. The average reward of the optimal policy is 0.8. Averaged over 500 independent runs. The error bars were computed by dividing the results into two separate bins depending on whether they were above or below the mean, and then computing the standard deviation within each bin.

Figure 7 shows the average reward $\eta(\theta)$ of the final controller produced by CONJPOMDP, as a function of the total number of simulation steps of the underlying Markov chain. The plots represent an average over 500 independent runs of CONJPOMDP. Note that 0.8 is the average reward of the optimal policy. The parameters of the controller were (uniformly) randomly initialized in the range $[-0.1, 0.1]$ before each call to CONJPOMDP. After each call to CONJPOMDP, the average reward of the resulting controller was computed exactly by calculating the stationary distribution for the controller. From Figure 7, optimality is reliably achieved using approximately 100 iterations of the Markov chain.

5.1.4 TRAINING ON-LINE WITH OLPOMDP

The controller was also trained on-line using Algorithm 4 (OLPOMDP) with fixed step-sizes $\gamma_t = c$ with $c = 0.1, 1, 10, 100$. Reducing step-sizes of the form $\gamma_t = c/t$ were tried, but caused intolerably slow convergence. Figure 8 shows the performance of the controller (measured exactly as in the previous section) as a function of the total number of iterations of the Markov chain, for different values of the step-size c. The graphs are averages over 100 runs, with the controller's weights randomly initialized in the range $[-0.1, 0.1]$ at the start of each run. From the figure, convergence to optimal is about an order of magnitude slower than that achieved by CONJPOMDP, for the best step-size of $c = 1.0$. Step-sizes much greater that $c = 10.0$ failed to reliably converge to an optimal policy.

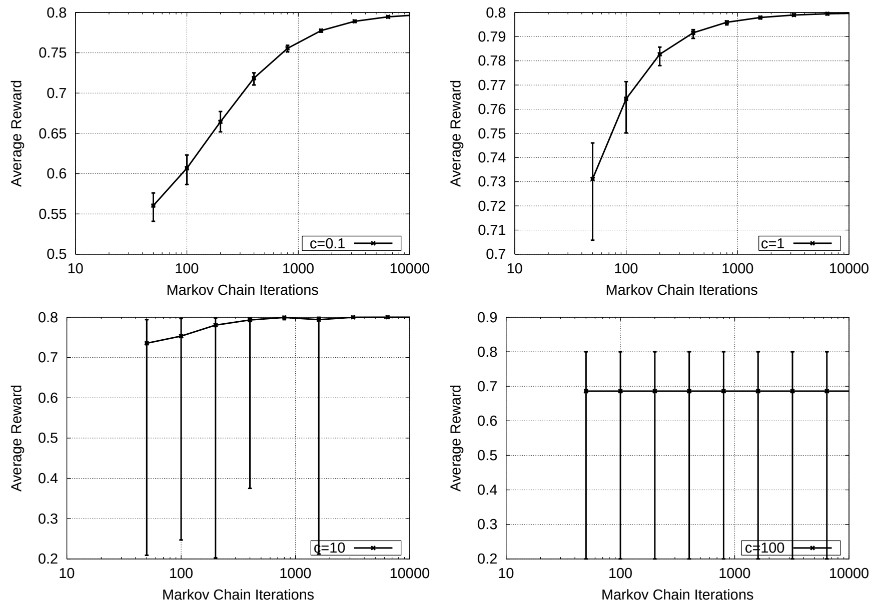

Figure 8: Performance of the 3-state Markov chain controller as a function of the number of iteration steps in the *on-line* algorithm, Algorithm 4, for fixed step sizes of $0.1, 1, 10$, and 100. Error bars were computed as in Figure 7.

5.2 Puck World

In this section, experiments are described in which CONJPOMDP and OLPOMDP were used to train 1-hidden-layer neural-network controllers to navigate a small puck around a two-dimensional world.

5.2.1 THE WORLD

The puck was a unit-radius, unit-mass disk constrained to move in the plane in a region 100 units square. The puck had no internal dynamics (i.e rotation). Collisions with the region's boundaries were inelastic with a (tunable) coefficient of restitution e (set to 0.9 for the experiments reported here). The puck was controlled by applying a 5 unit force in either the positive or negative x direction, and a 5 unit force in either the positive or negative y direction, giving four different controls in total. The control could be changed every $1/10$ of a second, and the simulator operated at a granularity of $1/100$ of a second. The puck also had a retarding force due to air resistance of $0.005 \times \text{speed}^2$. There was no friction between the puck and the ground.

The puck was given a reward at each decision point ($1/10$ of a second) equal to $-d$ where d was the distance between the puck and some designated target point. To encourage the controller to learn to navigate the puck to the target independently of the starting state, the puck state was reset every 30 (simulated) seconds to a random location and random x and y velocities in the range $[-10, 10]$, and at the same time the target position was set to a random location.

Note that the size of the state-space in this example is essentially infinite, being of the order of $2^{\text{PRECISION}}$ where PRECISION is the floating point precision of the machine (64 bits). Thus, the time between visits to a recurrent state is likely to be large. Also, the puck cannot just maximize its immediate reward because this leads to significant overshooting of the target locations.

5.2.2 THE CONTROLLER

A one-hidden-layer neural-network with six input nodes, eight hidden nodes and four output nodes was used to generate a probabilistic policy in a similar manner to the controller in the three-state Markov chain example of the previous section. Four of the inputs were set to the raw x and y locations and velocities of the puck at the current time-step, the other two were the differences between the puck's x and y location and the target's x and y location respectively. The location inputs were scaled to lie between -1 and 1, while the velocity inputs were scaled so that a speed of 10 units per second mapped to a value of 1. The hidden nodes computed a $\tanh$ squashing function, while the output nodes were linear. Each hidden and output node had the usual additional offset parameter. The four output nodes were exponentiated and then normalized as in the Markov-chain example to produce a probability distribution over the four controls (± 5 units thrust in the x direction, ± 5 units thrust in the y direction). Controls were selected at random from this distribution.

5.2.3 CONJUGATE GRADIENT ASCENT

We trained the neural-network controller using $\mathrm{CONJPOMDP}$ with the gradient estimates generated by GPOMDP. After some experimentation we chose $\beta = 0.95$ and $T = 1,000,000$ as the parameters $\mathrm{CONJPOMDP}$ supplied to GPOMDP. $\mathrm{GSEARCH}$ used the same value of β and the scheme discussed in Section 5.1.3 to determine the number of iterations with which to call GPOMDP.

Due to the saturating nature of the neural-network hidden nodes (and the exponentiated output nodes), there was a tendency for the network weights to converge to local minima at "infinity". That is, the weights would grow very rapidly early on in the simulation, but towards a suboptimal solution. Large weights tend to imply very small gradients and thus the network becomes "stuck" at these suboptimal solutions. We have observed a similar behaviour when training neural networks for pattern classification problems. To fix the problem, we subtracted a small quadratic penalty term $\gamma\|\theta\|^2$ from the performance estimates and hence also a small correction $2\gamma\theta_i$ from the gradient calculation[4] for θ_i.

We used a decreasing schedule for the quadratic penalty weight γ (arrived at through some experimentation). γ was initialized to 0.5 and then on every tenth iteration of $\mathrm{CONJPOMDP}$, if the performance had improved by less than 10% from the value ten iterations ago, γ was reduced by a factor of 10. This schedule solved nearly all the local minima problems, but at the expense of slower convergence of the controller.

A plot of the average reward of the neural-network controller is shown in Figure 9, as a function of the number of iterations of the POMDP. The graph is an average over 100 independent runs, with the parameters initialized randomly in the range $[-0.1, 0.1]$ at the start of each run. The four bad runs shown in Figure 10 were omitted from the average because they gave misleadingly large error bars.

Note that the optimal performance (within the neural-network controller class) seems to be around -8 for this problem, due to the fact that the puck and target locations are reset every 30 simulated seconds and hence there is a fixed fraction of the time that the puck must be away from

4. When used as a technique for capacity control in pattern classification, this technique goes by the name "weight decay". Here we used it to condition the optimization problem.

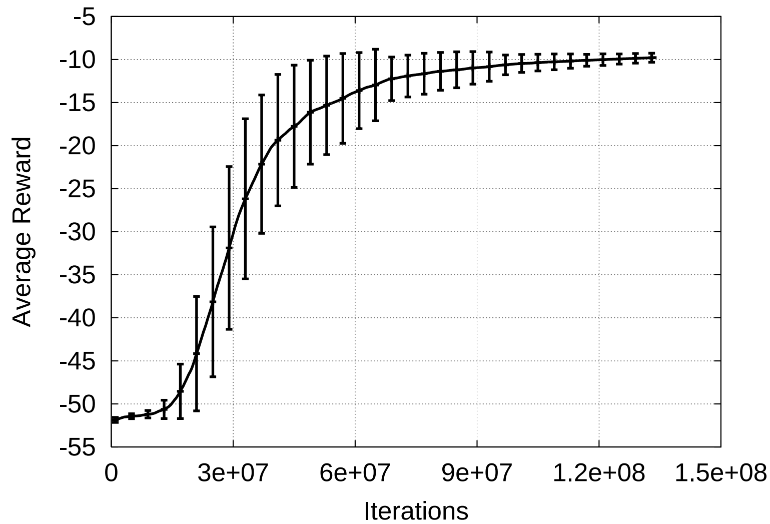

Figure 9: Performance of the neural-network puck controller as a function of the number of iterations of the puck world, when trained using CONJPOMDP. Performance estimates were generated by simulating for $1,000,000$ iterations. Averaged over 100 independent runs (excluding the four bad runs in Figure 10).

the target. From Figure 9 we see that the final performance of the puck controller is close to optimal. In only 4 of the 100 runs did CONJPOMDP get stuck in a suboptimal local minimum. Three of those cases were caused by overshooting in GSEARCH (see Figure 10), which could be prevented by adding extra checks to CONJPOMDP.

Figure 11 illustrates the behaviour of a typical trained controller. For the purpose of the illustration, only the target location and puck velocity were randomized every 30 seconds, not the puck location.

5.3 Call Admission Control

In this section we report the results of experiments in which CONJPOMDP was applied to the task of training a controller for the call admission problem treated by Marbach (1998, Chapter 7).

5.3.1 THE PROBLEM

The call admission control problem treated by Marbach (1998, Chapter 7) models the situation in which a telecommunications provider wishes to sell bandwidth on a communications link to customers in such a way as to maximize long-term average reward.

Specifically, the problem is a queuing problem. There are three different types of call, each with its own call arrival rate $\alpha(1)$, $\alpha(2)$, $\alpha(3)$, bandwidth demand $b(1)$, $b(2)$, $b(3)$ and average holding time $h(1)$, $h(2)$, $h(3)$. The arrivals are Poisson distributed while the holding times are exponentially distributed. The link has a maximum bandwidth of 10 units. When a call arrives and there is sufficient available bandwidth, the service provider can choose to accept or reject the call (if there is not enough available bandwidth the call is always rejected). Upon accepting a call of

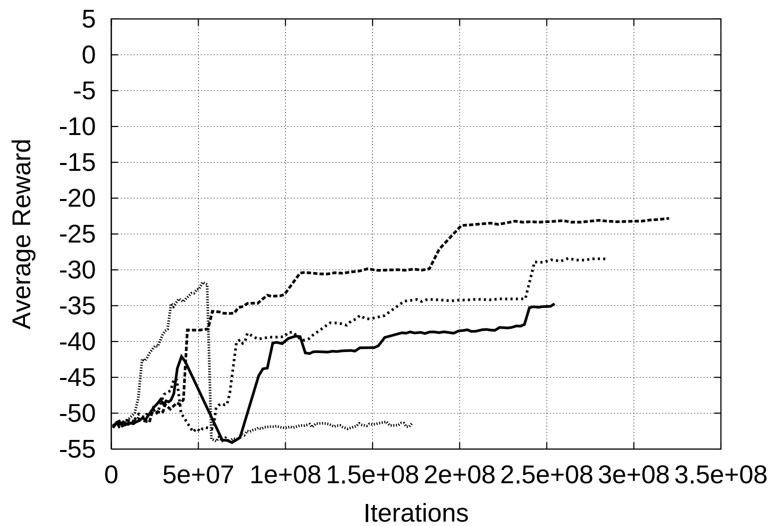

Figure 10: Plots of the performance of the neural-network puck controller for the four runs (out of 100) that converged to substantially suboptimal local minima.

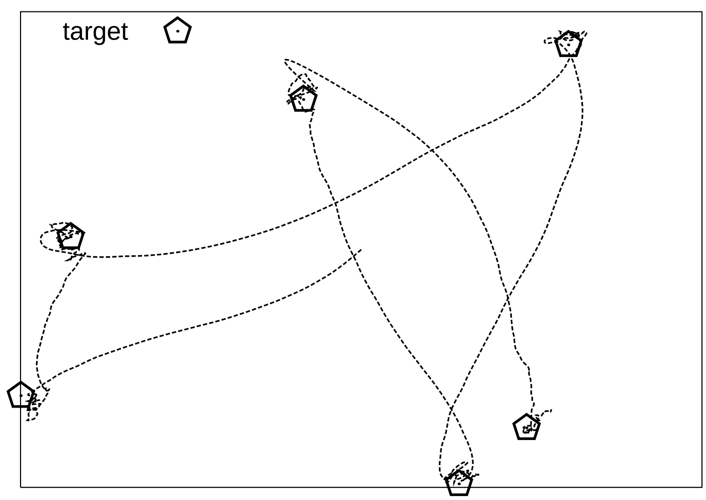

Figure 11: Illustration of the behaviour of a typical trained puck controller.

type m, the service provider receives a reward of $r(m)$ units. The goal of the service provider is to maximize the long-term average reward.

The parameters associated with each call type are listed in Table 3. With these settings, the optimal policy (found by dynamic programming by Marbach (1998)) is to always accept calls of type 2 and 3 (assuming sufficient available bandwidth) and to accept calls of type 1 if the available

Call Type		1	2	3
Bandwidth Demand	b	1	1	1
Arrival Rate	α	1.8	1.6	1.4
Average Holding Time	h	0.6	0.5	0.4
Reward	r	1	2	4

Table 3: Parameters of the call admission control problem.

bandwidth is at least 3. This policy has an average reward of 0.804, while the "always accept" policy has an average reward[5] of 0.784.

5.3.2 THE CONTROLLER

The controller had three parameters $\theta = (\theta_1, \theta_2, \theta_3)$, one for each type of call. Upon arrival of a call of type m, the controller chooses to accept the call with probability

$$\mu(\theta) = \begin{cases} \frac{1}{1+\exp(1.5(b-\theta_m))} & \text{if } b + b(m) \leq 10, \\ 0 & \text{otherwise,} \end{cases}$$

where b is the currently used bandwidth. This is the class of controllers studied by Marbach (1998).

5.3.3 CONJUGATE GRADIENT ASCENT

CONJPOMDP was used to train the above controller, with GPOMDP generating the gradient estimates from a range of values of β and T. The influence of β on the performance of the trained controllers was marginal, so we set $\beta = 0.0$ which gave the lowest-variance estimates. We used the same value of T for calls to GPOMDP within CONJPOMDP and within GSEARCH, and this was varied between 10 and $10,000$. The controller was always started from the same parameter setting $\theta = (8, 8, 8)$ (as was done by Marbach (1998)). The value of this initial policy is 0.691. The graph of the average reward of the final controller produced by CONJPOMDP as a function of the total number of iterations of the queue is shown in Figure 12. A performance of 0.784 was reliably achieved with less than 2000 iterations of the queue.

Note that the optimal policy is not achievable with this controller class since it is incapable of implementing any threshold policy other than the "always accept" and "always reject" policies. Although not provably optimal, a parameter setting of $\theta_1 \approx 7.5$ and any suitably large values of θ_2 and θ_3 generates something close to the optimal policy within the controller class, with an average reward of 0.8. Figure 13 shows the probability of accepting a call of each type under this policy (with $\theta_2 = \theta_3 = 15$), as a function of the available bandwidth.

The controllers produced by CONJPOMDP with $\beta = 0.0$ and sufficiently large T are essentially "always accept" controllers with an average reward of 0.784, within 2% of the optimum achievable in the class. To produce policies even nearer to the optimal policy in performance, CONJPOMDP must keep θ_1 close to its starting value of 8, and hence the gradient estimate $\Delta_T = (\Delta_1, \Delta_2, \Delta_3)$

5. There is some discrepancy between our average rewards and those quoted by Marbach (1998). This is probably due to a discrepancy in the way the state transitions are counted, which was not clear from the discussion in (Marbach, 1998).

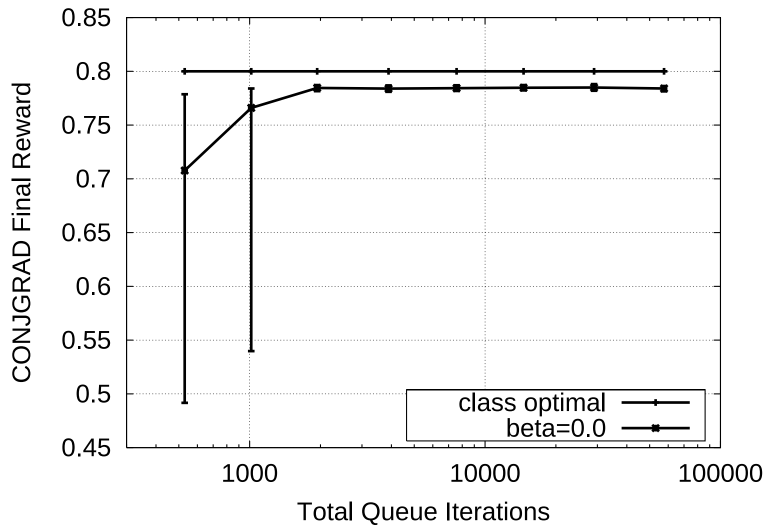

Figure 12: Performance of the call admission controller trained by CONJPOMDP as a function of the total number of iterations of the queue. The performance was computed by simulating the controller for 100,000 iterations. The average reward of the globally optimal policy is 0.804, the average reward of the optimal policy within the class is 0.8, and the plateau performance of CONJPOMDP is 0.784. The graphs are averages from 100 independent runs.

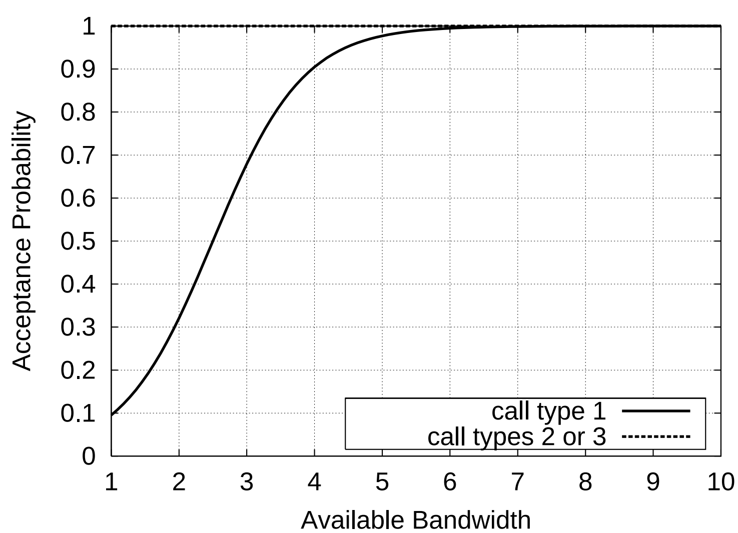

Figure 13: Probability of accepting a call of each type under the call admission policy with near-optimal parameters $\theta_1 = 7.5, \theta_2 = \theta_3 = 15$. Note that calls of type 2 and 3 are essentially always accepted.

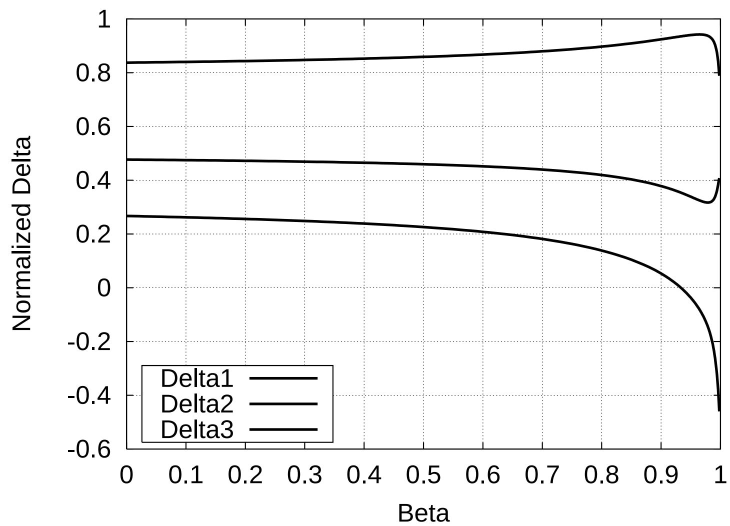

Figure 14: Plot of the three components of Δ_T for the call admission problem, as a function of the discount parameter β. The parameters were set at $\theta = (8, 8, 8)$. T was set to $1,000,000$. Note that Δ_1 does not become negative (the correct sign) until $\beta \approx 0.93$.

produced by GPOMDP must have a relatively small first component. Figure 14 shows a plot of normalized Δ_T as a function of β, for $T = 1,000,000$ (sufficiently large to ensure low variance in Δ_T) and the starting parameter setting $\theta = (8, 8, 8)$. From the figure, Δ_1 starts at a high value which explains why CONJPOMDP produces "always accept" controllers for $\beta = 0.0$, and does not become negative until $\beta \approx 0.93$, a value for which the variance in Δ_T even for moderately large T is relatively high.

A plot of the performance of CONJPOMDP for $\beta = 0.9$ and $\beta = 0.95$ is shown in Figure 15. Approximately half of the remaining 2% in performance can be obtained by setting $\beta = 0.9$, while for $\beta = 0.95$ a sufficiently large choice for T gives most of the remaining performance. For this problem, there is a huge difference between gaining 98% of optimal performance, which is achieved for $\beta = 0.0$ and less than 2000 iterations of the queue, and gaining 99% of the optimal which requires $\beta = 0.9$ and of the order of 500,000 queue iterations. A similar convergence rate and final approximation error to the latter case were reported for the on-line algorithms by Marbach (1998, Chapter 7).

5.4 Mountainous Puck World

The "mountain-car" task is a well-studied problem in the reinforcement learning literature (Sutton & Barto, 1998, Example 8.2). As shown in Figure 16, the task is to drive a car to the top of a one-dimensional hill. The car is not powerful enough to accelerate directly up the hill against gravity, so any successful controller must learn to "oscillate" back and forth until it builds up enough speed to crest the hill.

In this section we describe a variant of the mountain car problem based on the puck-world example of Section 5.2. With reference to Figure 17, in our problem the task is to navigate a puck

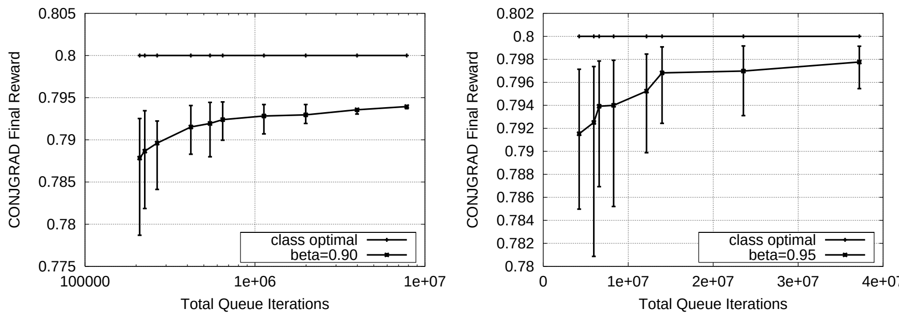

Figure 15: Performance of the call admission controller trained by CONJPOMDP as a function of the total number of iterations of the queue. The performance was calculated by simulating the controller for 1,000,000 iterations. The graphs are averages from 100 independent runs.

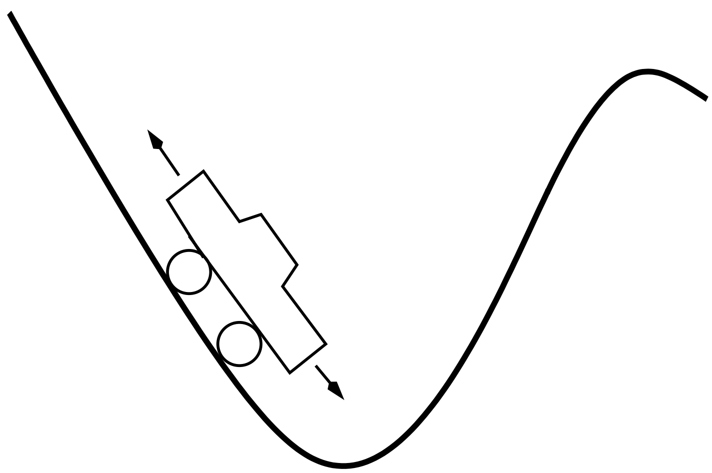

Figure 16: The classical "mountain-car" task is to apply forward or reverse thrust to the car to get it over the crest of the hill. The car starts at the bottom and does not have enough power to drive directly up the hill.

out of a valley and onto a plateau at the northern end of the valley. As in the mountain-car task, the puck does not have sufficient power to accelerate directly up the hill, and so has to learn to oscillate in order to climb out of the valley. Once again we were able to reliably train near-optimal neural-network controllers for this problem, using CONJPOMDP and GSEARCH, and with GPOMDP generating the gradient estimates.

5.4.1 THE WORLD

The world dimensions, physics, puck dynamics and controls were identical to the flat puck world described in Section 5.2, except that the puck was subject to a constant gravitational force of 10 units, the maximum allowed thrust was 3 units (instead of 5), and the height of the world varied as

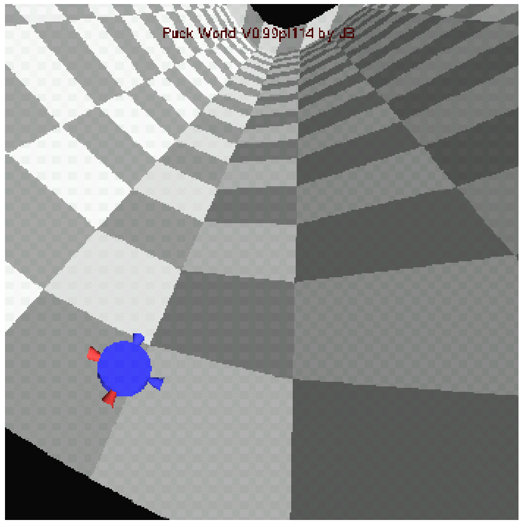

Figure 17: In our variant of the mountain-car problem the task is to navigate a puck out of a valley and onto the northern plateau. The puck starts at the bottom of the valley and does not have enough power to drive directly up the hill.

follows:

$$\text{height}(x, y) = \begin{cases} 15 & \text{if } y < 25 \text{ or } y > 75 \\ 7.5\left[1 - \cos\left(\frac{\pi\left(\frac{y}{2} - 50\right)}{25}\right)\right] & \text{otherwise.} \end{cases}$$

With only 3 units of thrust, a unit mass puck can not accelerate directly out of the valley.

Every 120 (simulated) seconds, the puck was initialized with zero velocity at the bottom of the valley, with a random x location. The puck was given no reward while in the valley or on the southern plateau, and a reward of $100 - s^2$ while on the northern plateau, where s was the speed of the puck. We found the speed penalty helped to improve the rate of convergence of the neural network controller.

5.4.2 THE CONTROLLER

After some experimentation we found that a neural-network controller could be reliably trained to navigate to the northern plateau, or to stay on the northern plateau once there, but it was difficult to combine both in the same controller (this is not so surprising since the two tasks are quite distinct). To overcome this problem, we trained a "switched" neural-network controller: the puck used one controller when in the valley and on the southern plateau, and then switched to a second neural-network controller while on the northern plateau. Both controllers were one-hidden-layer neural-networks with nine input nodes, five hidden nodes and four output nodes. The nine inputs were the normalized ($[-1, 1]$-valued) x, y and z puck locations, the normalized x, y and z locations relative to center of the northern wall, and the x, y and z puck velocities. The four outputs were used to generate a policy in the same fashion as the controller of Section 5.2.2.

An approach requiring less prior knowledge would be to have a third controller that stochastically selects the base neural network controller as a function of the puck's location. This "master"

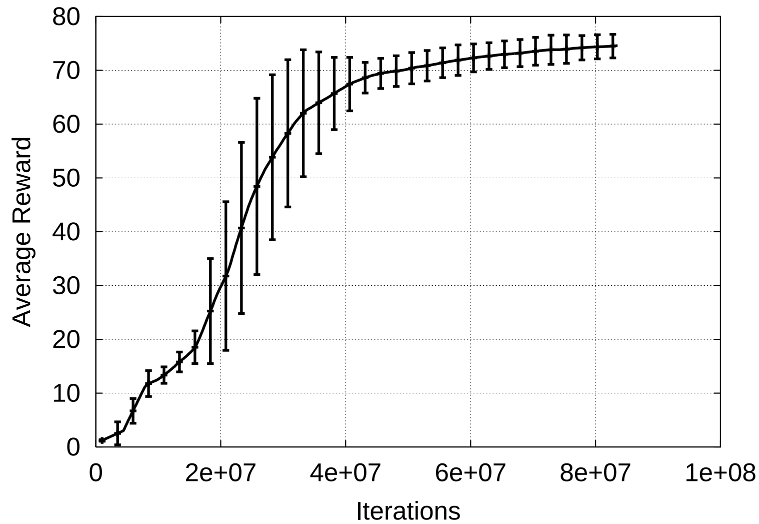

Figure 18: Performance of the neural-network puck controller as a function of the number of iterations of the mountainous puck world, when trained using CONJPOMDP. Performance estimates were generated by simulating for $1,000,000$ iterations. Averaged over 100 independent runs.

controller could itself be parameterized and have its parameters trained along with the base controllers.

5.4.3 CONJUGATE GRADIENT ASCENT

The switched neural-network controller was trained using the same scheme discussed in Section 5.2.3, except this time the discount factor β was set to 0.98.

A plot of the average reward of the neural-network controller is shown in Figure 18, as a function of the number of iterations of the POMDP. The graph is an average over 100 independent runs, with the neural-network controller parameters initialized randomly in the range $[-0.1, 0.1]$ at the start of each run. In this case no run failed to converge to near-optimal performance. From the figure we can see that the puck's performance is nearly optimal after about 40 million total iterations of the puck world. Although this figure may seem rather high, to put it in some perspective note that a random neural-network controller takes about 10,000 iterations to reach the northern plateau from a standing start at the base of the valley. Thus, 40 million iterations is equivalent to only about 4,000 trips to the top for a random controller.

Note that the puck converges to a final average performance around 75, which indicates it is spending at least 75% of its time on the northern plateau. Observation of the puck's final behaviour shows it behaves nearly optimally in terms of oscillating back and forth to get out of the valley.

5.5 Choosing β and the Running Time of GPOMDP

One aspect of these experiments that required some measure of tuning is the choice of the β parameter and running time T used by GPOMDP. Although these were selected by trial and error, we have had some success recently with a scheme for automatically choosing these parameters as follows. Before any training begins, GPOMDP is run for a large number of iterations whilst simultaneously generating gradient estimates for a number of different choices of β. This can be done from a single simulation simply by maintaining a separate eligibility trace z_t for each value of β. Since the bias reduces with increasing β, the largest β that gives a reasonably low-variance gradient estimate at the end of the long run is selected as a "reference" β (the variance is estimated by comparing gradient estimates at reasonably well-separated intervals towards the end of the run). Furthermore, since the variance of the gradient estimate decreases as β decreases, all gradient estimates for values of β smaller than the reference β will typically have smaller variance than that of the reference β. Hence, we can reliably compare the directions for smaller β's with the direction given by the reference β, and choose the *smallest* β whose corresponding direction is sufficiently close to the reference β direction. We take"sufficiently close" to mean within $10°$–$15°$.

Note that this scheme only works if the original run is sufficiently long to get a low-variance direction estimate at the right value of β. If the right value of β is too large then any fixed bound on the run length can be made to fail, but this will be a problem for all algorithms that automatically choose β.

Once a suitable β has been found, we can go back and find the point in the original long run where the direction estimate corresponding to that value of β "settled down" (again, we measure the variance of the estimates by sampling at suitably large intervals, and choose a point where the variance falls below some chosen value). This time is then used as the running time T for GPOMDP when estimating the gradient direction. Finally, the running time used in GPOMDP when bracketing the maximum in GSEARCH can also be automatically tuned by starting with an initial fixed running time that is a fraction of T, and then continuing until the sign of the inner product of the estimates produced by GPOMDP with the search direction "settles down". With this technique, the sign estimation time is usually considerably smaller than the gradient direction estimation time.

Another useful heuristic is to re-estimate β and GPOMDP's running time T whenever the parameters θ change by a large amount, since a large change in θ can lead to significant changes in the mixing time of the POMDP.

6. Conclusion

This paper showed how to use the performance gradient estimates generated by the GPOMDP algorithm (Baxter & Bartlett, 2001) to optimize the average reward of parameterized POMDPs. We described both a traditional "on-line" stochastic gradient algorithm and an "off-line" approach that relied on the use of GSEARCH, a robust line-search algorithm that uses gradient estimates, rather than value estimates, to bracket the maximum. The off-line approach in particular was found to perform well on four quite distinct problems: optimizing a controller for a three-state MDP, optimizing a neural-network controller for navigating a puck around a two-dimensional world, optimizing a controller for a call admission problem, and optimizing a switched neural-network controller in a variation of the classical mountain-car task. One reason for the superiority of the off-line approach is that by searching for a local maximum at each step it makes much more aggressive use of the gradient information than does the on-line algorithm.

For the three-state MDP and the call-admission problems we were able to provide graphic illustrations of how the bias and variance of the gradient estimates $\nabla_\beta \eta$ can be traded against one another by varying β between 0 (low variance, high bias) and 1 (high variance, low bias).

Relatively little tuning was required to generate these results. In addition, the controllers operated on direct and simple representations of the state, in contrast to the more complex representations usually required of value-function based approaches.

It is often the case that value-function methods converge much more rapidly than their policy-gradient counterparts. This is due to the fact that they enforce constraints on the value-function. With this in mind an interesting avenue for further research is Actor-Critic algorithms (Barto et al., 1983; Baird & Moore, 1999; Kimura & Kobayashi, 1998; Konda & Tsitsiklis, 2000; Sutton, McAllester, Singh, & Mansour, 2000) in which one attempts to combine the fast convergence of value-functions with the theoretical guarantees of policy-gradient approaches.

Despite the success of the off-line approach in the experiments described here, the on-line algorithm has advantages in other settings. In particular, when it is applied to multi-agent reinforcement learning, both gradient computations and parameter updates can be performed for distinct agents without any communication beyond the global distribution of the reward signal. This idea has led to a parameter optimization procedure for spiking neural networks, and some successful preliminary results with network routing (Bartlett & Baxter, 1999; Tao, Baxter, & Weaver, 2001).

Acknowledgements

This work was supported by the Australian Research Council, and benefited from the comments of several anonymous referees. Most of this research was performed while the first and second authors were with the Research School of Information Sciences and Engineering, Australian National University.

References

Aberdeen, D., & Baxter, J. (2001). Policy-gradient learning of controllers with internal state. Tech. rep., Australian National University.

Baird, L., & Moore, A. (1999). Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems 11*. MIT Press.

Bartlett, P. L., & Baxter, J. (1999). Hebbian synaptic modifications in spiking neurons that learn. Tech. rep., Research School of Information Sciences and Engineering, Australian National University. http://csl.anu.edu.au/∼bartlett/papers/BartlettBaxter-Nov99.ps.gz.

Bartlett, P. L., & Baxter, J. (2000a). Estimation and approximation bounds for gradient-based reinforcement learning. In *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory*, pp. 133–141.

Bartlett, P. L., & Baxter, J. (2000b). Stochastic optimization of controlled partially observable markov decision processes. In *Proceedings of the 39th IEEE Conference on Decision and Control (CDC00)*.

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

Baxter, J., & Bartlett, P. L. (2000). Reinforcement learning in POMDPs via direct gradient ascent. In *Proceedings of the Seventeenth International Conference on Machine Learning*.

Baxter, J., & Bartlett, P. L. (2001). Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*. To appear.

Baxter, J., Tridgell, A., & Weaver, L. (2000). Learning to play chess using temporal-differences. *Machine Learning*, *40*(3), 243–263.

Bertsekas, D. P., & Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Cao, X.-R., & Chen, H.-F. (1997). Perturbation Realization, Potentials, and Sensitivity Analysis of Markov Processes. *IEEE Transactions on Automatic Control*, *42*, 1382–1393.

Cao, X.-R., & Wan, Y.-W. (1998). Algorithms for Sensitivity Analysis of Markov Chains Through Potentials and Perturbation Realization. *IEEE Transactions on Control Systems Technology*, *6*, 482–492.

Fine, T. L. (1999). *Feedforward Neural Network Methodology*. Springer, New York.

Fu, M. C., & Hu, J. (1994). Smooth Perturbation Derivative Estimation for Markov Chains. *Operations Research Letters*, *15*, 241–251.

Glynn, P. W. (1986). Stochastic approximation for monte-carlo optimization. In *Proceedings of the 1986 Winter Simulation Conference*, pp. 356–365.

Kimura, H., & Kobayashi, S. (1998). An analysis of actor/critic algorithms using eligibility traces: Reinforcement learning with imperfect value functions. In *Fifteenth International Conference on Machine Learning*, pp. 278–286.

Kimura, H., Miyazaki, K., & Kobayashi, S. (1997). Reinforcement learning in POMDPs with function approximation. In Fisher, D. H. (Ed.), *Proceedings of the Fourteenth International Conference on Machine Learning (ICML'97)*, pp. 152–160.

Kimura, H., Yamamura, M., & Kobayashi, S. (1995). Reinforcement learning by stochastic hill climbing on discounted reward. In *Proceedings of the Twelfth International Conference on Machine Learning (ICML'95)*, pp. 295–303.

Konda, V. R., & Tsitsiklis, J. N. (2000). Actor-Critic Algorithms. In *Neural Information Processing Systems 1999*. MIT Press.

Marbach, P. (1998). *Simulation-Based Methods for Markov Decision Processes*. Ph.D. thesis, Laboratory for Information and Decision Systems, MIT.

Marbach, P., & Tsitsiklis, J. N. (1998). Simulation-Based Optimization of Markov Reward Processes. Tech. rep., MIT.

Rubinstein, R. Y., & Melamed, B. (1998). *Modern Simulation and Modeling*. Wiley, New York.

Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, *3*, 210–229.

Singh, S. P., Jaakkola, T., & Jordan, M. I. (1994). Learning Without State-Estimation in Partially Observable Markovian Decision Processes. In *Proceedings of the Eleventh International Conference on Machine Learning*.

Singh, S., & Bertsekas, D. (1997). Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Advances in Neural Information Processing Systems: Proceedings of the 1996 Conference*, pp. 974–980. MIT Press.

Singh, S., Jaakkola, T., & Jordan, M. (1995). Reinforcement learning with soft state aggregation. In Tesauro, G., Touretzky, D., & Leen, T. (Eds.), *Advances in Neural Information Processing Systems*, Vol. 7. MIT Press, Cambridge, MA.

Sutton, R. (1988). Learning to Predict by the Method of Temporal Differences. *Machine Learning*, *3*, 9–44.

Sutton, R. S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge MA. ISBN 0-262-19398-1.

Sutton, R. S., McAllester, D., Singh, S., & Mansour, Y. (2000). Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Neural Information Processing Systems 1999*. MIT Press.

Tao, N., Baxter, J., & Weaver, L. (2001). A multi-agent, policy-gradient approach to network routing. Tech. rep., Australian National University.

Tesauro, G. (1992). Practical Issues in Temporal Difference Learning. *Machine Learning*, *8*, 257–278.

Tesauro, G. (1994). TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, *6*, 215–219.

Weaver, L., & Baxter, J. (1999). Reinforcement learning from state and temporal differences. Tech. rep., Australian National University.

Williams, R. J. (1992). Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, *8*, 229–256.

Zhang, W., & Dietterich, T. (1995). A reinforcement learning approach to job-shop scheduling. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 1114–1120. Morgan Kaufmann.