The Cross Entropy method for Fast Policy Search

Shie Mannor SHIE@MIT.EDU
Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139

Reuven Rubinstein IERRR01@IE.TECHNION.AC.IL
Yohai Gat YOHAIG@TX.TECHNION.AC.IL
Faculty of Industrial Engineering and Management, Technion, 32000 Israel

Abstract

We present a learning framework for Markovian decision processes that is based on optimization in the policy space. Instead of using relatively slow gradient-based optimization algorithms, we use the fast Cross Entropy method. The suggested framework is described for several reward criteria and its effectiveness is demonstrated for a grid world navigation task and for an inventory control problem.

1. Introduction

The *Markov decision processes* (MDP) model is standard in artificial intelligence, machine learning, operation research and related fields. When the transition probability or the reward in an MDP are unknown, the problem is referred to as a *learning* one. In *Reinforcement Learning* (RL) an agent learns the behavior of the system through trial-and-error with an unknown dynamic environment (see Kaelbling et al., 1996). For reviews of RL see Sutton and Barto (1998); Bertsekas and Tsitsiklis (1996); Kaelbling et al. (1996). There are several approaches for RL, which can be roughly divided to the following three classes: model-based, model-free, and policy search. In the model-based approach, first a model of the environment is constructed. The estimated MDP is then solved using standard tools like dynamic programming (see Kearns & Singh, 1998). In the model-free approach one learns a utility function, instead of learning the model. The optimal policy is to choose at each state an action, which maximizes the expected utility. The popular Q-learning (e.g. Watkins, 1989) algorithm is an example of this approach. In the policy search approach a subspace of the policy space is searched, and the performance of policies is evaluated based on their empirical performance (e.g. Barto et al., 1983; Sutton & Barto, 1998). Examples to gradient-based policy search methods include the REINFORCE algorithm of Williams (1992), and certain variants of the actor-critic framework (e.g. Konda & Tsitsiklis, 2003). A detailed account of policy gradient can be found in Baxter et al. (2001). For a direct search in the policy space approach see Rosenstein and Barto (2001). The learning algorithms suggested in this paper belong to the policy search approach.

Many RL algorithms are essentially based on the classic *Stochastic Approximation* (SA) algorithm. To explain SA, assume that we need to find the unique solution v^* of the nonlinear equation $\mathbb{E}S(v) = 0$, where $S(v)$ is a random variable (noisy function) with *unknown* expectation $\mathbb{E}S(v)$. The SA algorithm for estimating v^* is $v_{t+1} = v_t + \beta_t S(v_t)$. The connection between SA and Q-learning is given by Tsitsiklis (1994). This work has made an important impact on the entire field of RL. Standard SA is known to converge slowly, because of the requirement that $\sum_{t=1}^{\infty} \beta_t^2 < \infty$ ($\beta_t \to 0$). Even if β_t remains bounded away from 0, (and thus convergence is not guaranteed) it is still required that β_t is small in order to ensure convergence to a reasonable solution. For details see Borkar and Meyn (2000).

The main goal of this paper is to introduce a fast learning algorithm based on the Cross Entropy (CE) method instead of the slow SA algorithms. CE has become a standard tool in Monte Carlo estimation and both, combinatorial and continuous multi-extremal optimization, see Rubinstein (1999); de-Boer et al. (2003) for details. As opposed to most RL methods our framework leads to fast convergence and can be easily extended to parameterized policy architecture, as outlined in Section 4. Some experiments in Section 5 with a maze world and inventory control problems indicate the suggested framework converges in a small number of iterations, with high probability, to a small neighborhood of the optimal solution.

2. Preliminaries

In this section we present some background on MDPs and the CE method.

2.1. Markov Decision Process (MDP)

We review briefly some of the basic definitions and concepts in MDP. For details see, e.g., Puterman (1994); Bertsekas (1995). An MDP is defined by a tuple $(\mathcal{M}, \mathcal{A}, \mathcal{P}, r)$ where: $\mathcal{M} = \{1, \ldots, M\}$ is the set of states, which we assume to be finite; $\mathcal{A} = \{1, \ldots, A\}$ is the set of possible actions of the decision maker, which we assume to be the same for every state to ease notations; $\mathcal{P}$ is the transition probability matrix with the elements $\mathcal{P}(m'|m, a)$ presenting the transition probability from state m to state m', when action a is taken; and $r(m, a)$ is the reward for performing action a in state m, which we assumed to be bounded by r_{max}.

At each time instance t, the decision maker observes the current state, m_t, and determines the action to be taken, a_t. As a result, reward $r(m_t, a_t)$, denoted by r_t, is received and a new state is chosen according to the transition probability $\mathcal{P}(m'|m_t, a_t)$. A mapping from the state-action-reward histories to the probability distribution of the decision maker's actions is called a *strategy*. A strategy is called *stationary* if it depends only on the current state. The goal of the decision maker is to maximize a certain reward function. The following are standard reward criteria:

1. Finite horizon reward. We will consider the special case of shortest path MDPs where it is assumed that the process starts from a specific initial state m_0, and that there is a absorbing state m_{ter} with zero reward. The objective is to maximize $V^T(m_0) = \sup_\pi \mathbb{E}_\pi \sum_{t=0}^{T-1} r_t$, where $\mathbb{E}_\pi$ denotes the expectation with respect to some probability measure induced by the strategy π and the supremum is taken over all strategies.
2. Infinite horizon discounted reward. The objective is to find a strategy π, which maximizes $V^\alpha(m) = \sup_\pi \mathbb{E}_\pi \sum_{t=0}^{\infty} \alpha^t r_t$, for all m, where $0 < \alpha < 1$ is the discount factor.
3. Average reward. The objective is to maximize $V(m) = \sup_\pi \liminf_{T\to\infty} \frac{1}{T}\mathbb{E}_\pi \sum_{t=0}^{T-1} r_t$.

It is well known (Puterman, 1994) that there exists a deterministic stationary optimal strategy for the above three cases. Note that if the model (r and $\mathcal{P}$) is known, then there are several efficient methods for finding the optimal strategy (Puterman, 1994). However, since the model is assumed to be unknown, a learning scheme is required. As mentioned, we shall employ in this situation the CE method instead of SA and shall demonstrate its high performance.

2.2. The Cross Entropy method

In this section we review the CE method, which is a state-of-the-art method for solving combinatorial and multi-extremal continuous optimization problems. We refer the reader to de-Boer et al. (2003) and references therein for context, extensions, and applications. The main idea behind the CE method is to transform the original optimization problem to an *associated stochastic problem* (ASP) and then to tackle the ASP efficiently by an adaptive algorithm. By doing so one constructs a random sequence of solutions which converges probabilistically to the optimal or near-optimal one. As soon as the ASP is defined, the CE method employs the following two phases:

1. Generation of a sample of random data (trajectories, vectors, etc.) according to a specified random mechanism.
2. Updating the parameters of the random mechanism, typically parameters of pdfs, on the basis of the data, in order to produce a "better" sample in the next iteration.

To proceed, suppose we wish to maximize some performance function $S(\mathbf{x})$ over all $\mathbf{x}$ in some set $\mathcal{X}$. Let us denote the unique maximum by γ^*, thus

$$\gamma^* = \max_{\mathbf{x}\in\mathcal{X}} S(\mathbf{x}) \; . \tag{1}$$

As mentioned, we randomize our deterministic problem by defining a family of auxiliary pdfs $\{f(\cdot; \mathbf{v}), \mathbf{v} \in \mathcal{V}\}$ on $\mathcal{X}$ and we associate with Eq. (1) the following estimation problem

$$\ell(\gamma) = \mathbb{P}_{\boldsymbol{u}}(S(\mathbf{X}) \geq \gamma) = \mathbb{E}_{\boldsymbol{u}} \mathbf{I}_{\{\mathbf{S}(\mathbf{X})\geq\gamma\}},$$

where u is some known parameter ($\mathbf{X} \in \mathcal{X}$ has a pdf $f(\cdot; \boldsymbol{u})$) and γ an unknown scalar. We consider the event "score is high" to be the rare event of interest. To estimate this event, the CE method generates a sequence of tuples $\{(\hat{\gamma}_t, \hat{\mathbf{v}}_t)\}$, which converges quickly (with high probability) to a small neighborhood of the optimal tuple $(\gamma^*, \mathbf{v}^*)$.

We denote by ρ the fraction, that the best (maximal values, sometimes termed elite) samples, that are used to find the threshold γ, constitute in the entire sample. Following standard terminology in simulation theory, the process that is based on sampled data is called the *stochastic counterpart* as it is based on stochastic samples of data. The number of samples in each stage of the stochastic counterpart is denoted by N, which is assumed to be a fixed predefined number. The following is a standard CE procedure for a maximization problem borrowed from de-Boer et al. (2003). We initialize by setting $\hat{\mathbf{v}}_0$ uniform, and choose a not very small ρ, say $10^{-2} \leq \rho$, and then we proceed iteratively as follows:

1. **Adaptive updating of** γ_t. For a fixed $\mathbf{v}_{t-1}$, let γ_t be a $(1-\rho)100\%$-*percentile* of $S(\mathbf{X})$ under $\mathbf{v}_{t-1}$. That is, γ_t satisfies $\mathbb{P}_{\mathbf{v_{t-1}}}(S(\mathbf{X}) \geq \gamma_t) \geq \rho$ and $\mathbb{P}_{\mathbf{v_{t-1}}}(S(\mathbf{X}) \leq \gamma_\mathbf{t}) \geq \mathbf{1} - \rho$ where $\mathbf{X} \sim \mathbf{f}(\cdot; \mathbf{v_{t-1}})$. A simple estimator $\widehat{\gamma}_t$ of γ_t can be obtained by taking a random sample $\mathbf{X^{(1)}}, \ldots, \mathbf{X^{(N)}}$ from the pdf $f(\cdot; \mathbf{v}_{t-1})$, calculating the performances $S(\mathbf{X^{(i)}})$ for all i, ordering them from smallest to biggest as $S_{(1)} \leq \ldots \leq S_{(N)}$ and finally evaluating the $(1-\rho)100\%$ sample percentile as $\widehat{\gamma}_t = S_{(\lceil (1-\rho)N \rceil)}$.

2. **Adaptive updating of $\mathbf{v_t}$**. For a fixed γ_t and $\mathbf{v_{t-1}}$, derive $\mathbf{v_t}$ from the solution of the program

$$\max_{\mathbf{v}} D(\mathbf{v}) = \max_{\mathbf{v}} \mathbb{E}_{\mathbf{v_{t-1}}} \mathbf{I}_{\{\mathbf{S(X)} \geq \gamma_t\}} \log \mathbf{f}(\mathbf{X}; \mathbf{v}) \ . \tag{2}$$

The stochastic counterpart of (2) is as follows: for fixed $\widehat{\gamma}_t$ and $\widehat{\mathbf{v}}_{t-1}$, derive $\widehat{\mathbf{v}}_t$ from the following program:

$$\max_{\mathbf{v}} \hat{D}(\mathbf{v}) = \max_{\mathbf{v}} \frac{\mathbf{1}}{\mathbf{N}} \sum_{\mathbf{i=1}}^{\mathbf{N}} \mathbf{I}_{\{\mathbf{S(X^{(i)})} \geq \widehat{\gamma}_\mathbf{t}\}} \log \mathbf{f}(\mathbf{X^{(i)}}; \mathbf{v}) \ . \tag{3}$$

We note that if f belongs to the Natural Exponential Family (NEF; e.g., Gaussian, discrete Bernoulli), then Eq. (3) has a closed form solution (see de-Boer et al., 2003). The CE optimization algorithm is summarized in Algorithm 2.1.

Remark: Instead of the updating the parameter vector $\mathbf{v}$ directly via the solution of Eq. (3) we use the following *smoothed* version

$$\widehat{\mathbf{v}}_t = \alpha \tilde{\mathbf{v}}_t + (1-\alpha)\widehat{\mathbf{v}}_{t-1}, \tag{4}$$

where $\tilde{\mathbf{v}}_t$ is the parameter vector obtained from the solution of (3), and α is called the *smoothing parameter*, with $0.7 < \alpha < 1$. Clearly, for $\alpha = 1$ we have our original updating rule. The reason for using the smoothed (4) instead of the original updating rule is twofold: (a) to smooth out the values of $\hat{\mathbf{v}}_t$, (b) to reduce the probability that some component $\widehat{v}_{t,i}$ of $\hat{\mathbf{v}}_t$ will be zeros or unities at an early stage, and a the algorithm will get stuck in a local maxima. Note that for $0 < \alpha < 1$ we always have that $\widehat{v}_{t,i} > 0$, while for $\alpha = 1$ one might have (even at the first iterations) that either $\widehat{v}_{t,i} = 0$ or $\widehat{v}_{t,i} = 1$ for some indices i. As result, the algorithm may converge to a wrong solution.

Remark: The performance CE method is insensitive to the exact choice of parameters. As long as ρ is not too small, $\alpha < 1$, and N is large enough, the results

Algorithm 2.1 The CE Method for Stochastic Optimization

```
1. Choose some $\widehat{\mathbf{v}}_0$. Set $t = 1$ (level counter).
2. Generate a sample $\mathbf{X^{(1)}}, \ldots, \mathbf{X^{(N)}}$ from the density $f(\cdot; \mathbf{v_{t-1}})$ and compute the sample $(1-\rho)100\%$-percentile $\widehat{\gamma}_t$ of the sample scores.
3. Use the **same** sample $\mathbf{X^{(1)}}, \ldots, \mathbf{X^{(N)}}$ and solve the stochastic program (3). Denote the solution by $\widetilde{\mathbf{v}}_t$.
4. Apply (4) to smooth out the vector $\widetilde{\mathbf{v}}_t$.
5. If for some $t \geq d$, say $d = 5$, $\widehat{\gamma}_t = \widehat{\gamma}_{t-1} = \cdots = \widehat{\gamma}_{t-d}$ then **stop** (let $T$ denote the final iteration); otherwise set $t = t + 1$ and reiterate from step 2.
```

of the algorithm are robust. In our numerical studies below we did not tweak the parameters. We chose a typical $0.01 \leq \rho \leq 0.03$, and the smoothing parameter $\alpha = 0.7$. The sample size N was chosen so that $\eta = \rho N$ which corresponds to the number of sample performances $S_{(j)}, \ j = 1, \ldots, N$, that lie in the upper $100\rho\%$ was a few tens. It is the best samples that allow Algorithm 2.1 to avoid local extrema and to settle down in the global maximum with high probability.

Improvements of Algorithm 2.1 include the Fully Adaptive CE (FACE) variant, where the parameters N and ρ are updated online, and some alternatives to the threshold (indicator) sample functions, H, (like Boltzmann functions). See de-Boer et al. (2003) for more details. For a convergence proof of the CE method see Homem de Mello and Rubinstein (2002).

3. Policy Learning using the CE method

This section deals with the application of the CE method to learning in MDPs. To proceed, we define an auxiliary $M \times A$ probability matrix $\boldsymbol{P} = (P_{ma})$ with elements $P_{ma}, \ m = 1, \ldots, M; \ a = 1, \ldots, A$ denoting the probability of taking action a at state m, $(\sum_{a=1}^{A} P_{ma} = 1, \forall m)$. Once the matrix $\boldsymbol{P}$ is defined, the above two phases (with $f(\cdot, v)$ replaced by P_{ma}) can be written as:

1. Generation of random trajectories (samples) using the auxiliary probability matrix $\boldsymbol{P} = (P_{ma})$ and simultaneous calculation of the sample function S.

2. Updating of the parameters of the probability matrix (P_{ma}) on the basis of the data collected at the first phase. This is done via the CE Algorithm 2.1.

The matrix $\boldsymbol{P}$ is typically initialized to a uniform matrix ($\boldsymbol{P}_{ma} = 1/A$.) The generation of random trajectories for an MDP according to the probability matrix $\boldsymbol{P}$ is quite straightforward. We shall show that in calculating the associated sample reward function S, one can take into the Markovian nature of the problem and to speed up the Monte-Carlo process. The following three subsections discuss in more details trajectory generation, the sample performance calculation and the stopping criteria for the finite horizon shortest path problem, infinite horizon discounted, and average rewards.

3.1. Trajectory generation for the shortest path problem

As mentioned, in shortest path problems there is a terminal state which corresponds to zero reward. With this in mind we may either stop the trajectory when it reaches the terminal state or alternatively we can stop the trajectory if it becomes too long (and discard that trajectory, or penalize states in the trajectory, depending on the application). We will assume that every policy is proper (Bertsekas & Tsitsiklis, 1996) and that the terminal state is always reached.

Algorithm 3.1 Trajectory generation for the Shortest Path problem

```
Input: $\boldsymbol{P}$ — action probability.
For ($i = 1$ to $N$):
  1. Start from some given initial state $m_0$, set $t = 0$.
  2. Repeat until $m_t = m_{ter}$
     (a) Generate an action $a_t$ according to $\boldsymbol{P}_{m_t a}$ and apply it.
     (b) Observe a reward $r_t$ and a new state $m_{t+1}$.
     (c) Set $t = t + 1$.
  3. Given a trajectory
       $\mathbf{X}^{(i)} = \{m_0^{(i)}, a_0^{(i)}, r_0^{(i)}, m_1^{(i)}, a_1^{(i)}, r_1^{(i)}, \ldots, a_{t-1}^{(i)}, r_{t-1}^{(i)}, m_t^{(i)}\}$,
     calculate the cumulative reward on the trajectory as
       $S(\mathbf{X}^{(i)}) = \sum_{k=0}^{t-1} r_k^{(i)}$ .
Output: Score $S$.
```

Given the N trajectories $\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(N)}$ and their scores, $S(\mathbf{X}^{(1)}), \ldots, S(\mathbf{X}^{(N)})$, one can update the parameters matrix (P_{ma}) using the CE method, namely as per (3). Since for each m the entry P_{ma} presents a discrete pdf, and thus NEF, the update formula (see de-Boer et al., 2003) is:

$$P_{ma} = \frac{\sum_{k=1}^{N} I_{\{S(\mathbf{X}^{(k)}) \leq \gamma\}} I_{\{\mathbf{X}^{(k)} \in \mathcal{X}_{ma}\}}}{\sum_{k=1}^{N} I_{\{S(\mathbf{X}^{(k)}) \leq \gamma\}} I_{\{\mathbf{X}^{(k)} \in \mathcal{X}_{m}\}}} , \qquad (5)$$

where the event $\{\mathbf{X}^{(k)} \in \mathcal{X}_m\}$ means that the trajectory $\mathbf{X}^{(k)}$ contains a visit to state m and the event $\{\mathbf{X}^{(k)} \in \mathcal{X}_{ma}\}$ means the trajectory $\mathbf{X}^{(k)}$ contains a visit to state m in which action a was taken.

We note that the score that is assigned to two states in the same trajectory is correlated, which creates a bias. This bias is inherent to our method, but by using several uncorrelated trajectories, the effect of this biased is reduced.

We now explain how to take advantage of the Markovian nature of the problem. Let us think of a maze where a certain trajectory starts badly, that is the path is not efficient in the beginning, but after some time it starts moving quickly towards the goal. According to (5), all the updates are performed in a similar manner in every state in the trajectory. However, the actions taken in the states that were sampled near the target were successful, so one would like to encourage these actions. The Markov property suggests an efficient way to improve the above algorithm by considering for each state the part of the reward from the visit to that state onwards. We therefore use the same trajectory and simultaneously calculate the score for every state in the trajectory separately. The idea here is that each choice of action in a given state affects the reward from that point on, disregarding the past.

The sampling algorithm of Algorithm 3.1 does not change in steps 1 and 2. The difference is in step 3. Given a trajectory

$$\mathbf{X}^{(i)} = \{m_0^{(i)}, a_0^{(i)}, r_0^{(i)}, m_1^{(i)}, a_1^{(i)}, r_1^{(i)}, \ldots, a_{t-1}^{(i)}, r_{t-1}^{(i)}, m_t^{(i)}\}$$

we calculate the reward from every state until termination. For every state in the trajectory the score is $S_{m_j}(\mathbf{X}^{(i)}) = \sum_{k=j}^{t-1} r_k^{(i)}$. The update formula for P_{ma} is similar to (5), however *each state is updated separately* according to the reward $S_{m_j}(\mathbf{X}^{(i)})$ obtained from state m_j onwards.

$$P_{ma} = \frac{\sum_{k=1}^{N} I_{\{S_m(\mathbf{X}^{(k)}) \leq \gamma_m\}} I_{\{\mathbf{X_m}^{(k)} \in \mathcal{X}_{ma}\}}}{\sum_{k=1}^{N} I_{\{S_m(\mathbf{X}^{(k)}) \leq \gamma_m\}} I_{\{\mathbf{X}^{(k)} \in \mathcal{X}_{m}\}}} . \qquad (6)$$

A crucial point here is to understand that in contrast to (5) the CE optimization is carried for every state separately and a different threshold parameter γ_m is used for every state m. This facilitates faster convergence for "easy" states where the optimal strategy is easy to find. The above trajectory sampling method can be viewed as a variance reduction method. Numerical results indicate that the CE algorithm with updating (6) is much faster then that with updating (5).

3.2. Trajectory generation for the discounted reward MDP

Sampling for the discounted reward criterion is a bit more difficult since there is no obvious reason to assume the existence of a terminal state. However, because of the discount factor we can stop sampling when some precision level ε is guaranteed. Indeed, recall that r_{max} denotes a known upper bound on the immediate reward, then for a given discount factor α the time horizon which guarantees the accuracy up to ε equals to $T_\varepsilon = \log(\frac{\varepsilon(1-\alpha)}{r_{max}})/\log(\alpha)$. Trajectory generation for discounted reward is similar to Algorithm 3.1, with the exception that now each trajectory is T_{max} long (T_{max} is an application dependent parameter), as there may not be a natural termination time. Given a trajectory

$$\mathbf{X}^{(i)} = \{m_0^{(i)}, a_0^{(i)}, r_0^{(i)}, m_1^{(i)}, a_1^{(i)}, r_1^{(i)}, \ldots, a_{T_{max}-1}^{(i)}, \\ r_{T_{max}-1}^{(i)}, m_{T_{max}}^{(i)}\}$$

the reward for states $m_0, \ldots, m_{T_{max}-T_\varepsilon}$ in the trajectory as

$$S_{m_j}(\mathbf{X}^{(i)}) = \sum_{k=j}^{T_{max}-1} \alpha^{k-j} r_j^{(i)} . \qquad (7)$$

Instead of (7) one may calculate the reward based only on the effective horizon T_ε, that is $S_{m_j}(\mathbf{X}^{(i)}) = \sum_{k=j}^{j+T_\varepsilon} \alpha^{k-j} r_j$, and obtain an additional speed up. The update equation remains exactly as in (6).

3.3. Average reward MDP

The average reward criterion requires a more elaborated scheme for trajectory generations. The main problem is that there is no finite horizon to consider as in the previous cases. One may employ the following two alternatives. The first is to adopt the finite horizon reward Algorithm 3.1 and then calculate the average reward as the cumulative reward divided by the total time. The drawback of this approach is that the reward of actions that were performed in the distant past or future affect the score of the whole trajectory. Under the assumption that for every strategy there exists at least one recurrent state a second alternative may be suggested based on the concept of *regeneration*. According to this concept, when reaching the recurrent state the process "starts over" again. Let $m_{rec} \in \mathcal{M}$ be a recurrent state. According to the regenerative method the average reward can be calculated as the ratio between the expected reward per cycle and the expected cycle time. Note that a cycle is defined as the time between two consecutive visits of the process to the same recurrent state. We describe this idea in Algorithm 3.2.

Algorithm 3.2 Trajectory generation for the average reward MDP

```
Input: $\boldsymbol{P}$ — Action probability; $m_{rec}$ — A recurrent state; $T_{max}$ — Trajectory length.
For ($i = 1$ to $N$):
  1. Start from some given initial state $m_0$, set $t = 0$.
  2. Repeat until $t = T_{max}$
     Same sampling as steps 2(a)-(c) in Algorithm 3.1.
  3. Let $\tau_0 = 0$, and let $\tau_\ell = \min\{t > \tau_{\ell-1}, \text{s.t. } m_t = m_{rec}\}$, (if the minimum is not attained we set $\tau_\ell = \infty$). Let $last = \max\{\ell \text{ s.t. } \tau_\ell < \infty\}$ denote the total number of generated regenerative cycles (last return to the $m_{rec}$).
     Calculate the reward per cycle for each state in the trajectory, starting at $\tau_1$ until $\tau_{last}$. That is, for $\tau_1 < j < \tau_{last}$:
     $S_{m_j}(\mathbf{X}^{(i)}) = \frac{\sum_{k=\max\{\tau_\ell:\tau_\ell \le j\}}^{\min\{\tau_\ell:\tau_\ell > j\}-1} r_k^{(i)}}{\min\{\tau_\ell : \tau_\ell > j\} - \max\{\tau_\ell : \tau_\ell \le j\}}$.
Output: Score vector per state $\{S_{m_j}\}_{i=1}^N$.
```

The score of each action is taken as the average reward of the cycle. Note that for every action in a given cycle has the same score. Observe that we can calculate the score only for states from the first regeneration time (τ_1) until the last regeneration time (τ_{last}) since the cycle reward cannot be estimated without reference to the regeneration time. The update of $\boldsymbol{P}$ remains identical to (6).

4. Parameterized Policies

Until now we assumed that the state space is finite and that sampling was performed according to the matrix $\boldsymbol{P}$ which was assumed to be $M \times A$ matrix. We now extend our framework to a large state space. We assume that the policy is parameterized by a small number of parameters (as in Konda & Tsitsiklis, 2003). The CE method replaces the traditional gradient-based method and is used for optimizing over the parameterized policy. Let us denote the parameter space by Θ.

Assume that every $\theta \in \Theta$ induces a strategy $\mu(a|m,\theta)$. The strategy $\mu(a|m,\theta)$ is defined by the probability of choosing action a when in state m according to the parameter θ. Instead of looking for the best strategy, we look for the best parameter $\theta \in \Theta$. All three reward criteria that were discussed above are still relevant, with appropriate modifications, as the sampling algorithm is virtually the same.

The CE optimization is performed by assuming that the parameters $\theta \in \Theta$ are drawn from a distribution $f(\theta; v)$ and optimization is performed by solving (3). As before, if $f(\theta; v)$ belongs to a NEF, then an analytical solution of Eq. (3) is available. We emphasize that by using the CE method there is no need to assume anything on the parameterization of $\mu(\cdot|m,\theta)$. Specifically, $\mu(\cdot|m,\theta)$ may be non-differentiable with respect to θ (as opposed to most policy search algorithms).

5. Experiments

In this subsection we describe experiments with provide two "traditional" domains. The first domain is a maze world with stochastic transitions and the second is an inventory control problem.

5.1. The Maze Problem

Algorithm 3.1 for stochastic shortest path MDPs was tested for a maze problem, which presents a two-dimensional grid world. The agent moves in the grid in four possible directions. The goal of the agent is to move from the upper-left corner (the starting state) to the lower-right corner (the goal). The maze contains obstacles ("walls") into which movement is not allowed. The reward for every allowed movement until reaching the goal is -1. In addition we introduce:

1. A small (failure) probability not to succeed moving in an allowed direction.
2. A small probability of succeeding moving in the forbidden direction ("moving into a wall").[1]
3. A high cost for trying to move in a forbidden direction.

We run the algorithm for several mazes and the optimal policy was always found.

In Figure 1 we present the results for 20×20 maze. We set the following parameters: $N = 1000$, $\rho = 3\%$, $\alpha = 0.7$ and $T = 1500$ (maximal path length). The initial policy was a uniformly random one. The cost of moves were random variables uniformly distributed between 0.5 and 1.5 and between 25 and 75 (expected cost are equal to 1 and 50) for the allowed and forbidden moves, respectively. The success probabilities in the allowed and forbidden states were taken 0.95 and 0.05, respectively. In Figure 1 we plot the possible ($\boldsymbol{P}_{ma} > 0.01$) trajectories of the algorithm in the end of each iteration. The convergence is quite fast and very accurate. In all our experiments CE found the target exactly, within 5-10 iterations and CPU time was less than one minute (on a 500MHz Pentium processor). Note the successive refinement of the policy in Figure 1, as the $\boldsymbol{P}_{ma}$ of suboptimal entries was reduced quickly to 0.

[1]This is required, for some extent, for making sure that the goal is reached in the first iteration.

5.2. The Inventory Control Problem

The inventory control (IC) is a well studied problem in the operation research community. The decision maker in this problem can be viewed as a shop owner, who buys and sells certain commodities (say, k commodities). At each stage (day) t he faces the dilemma of how much stock to order from each type of commodity, provided that the customers' demand of the ith commodity $d_t(i)$ is a random variable with an *unknown* distribution. The decision maker's expenses consist of the purchase price, a price paid for holding surplus stock, and penalty for back-logged demand. In MDP terms, the state space is the amount of available stock of each commodity ($\mathcal{M} = \mathbb{R}^k$ for continuous stock or $\mathcal{M} = \{\dots,-2,-1,0,1,\dots\}^k$ for discrete stock), with negative stock representing unsatisfied demand. Denote by $m_t(i)$ the inventory of commodity i at the beginning of period t, by $a_t(i)$ the ordered stock at that period and by $d_t(i)$ the demand of it, then $m_{t+1}(i)$, which is the inventory of commodity i at the beginning of period $t+1$, is given by:

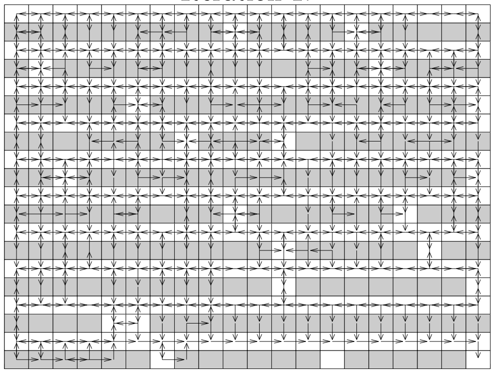
$$m_{t+1}(i) = m_t(i) + a_t(i) - d_t(i) \quad i = 1,\dots,k.$$

There are many possible cost functions of interest. We shall assume the following linear cost function:

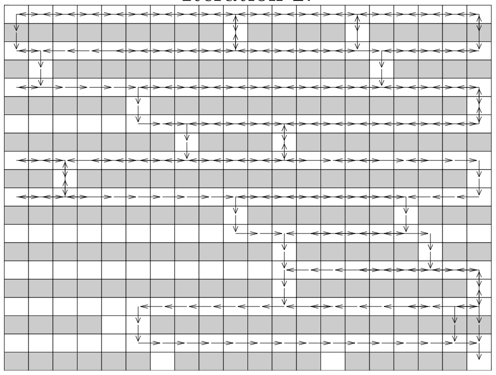
$$\begin{aligned} r(m_t, a_t) \quad = \quad & \sum_{i=1}^{k} \big(h(i)\max\{0, m_t(i)\} + \\ & b(i)\max\{0, -m_t(i)\} + c(i)a_t(i)\big), \end{aligned}$$

where for the ith commodity $h(i)$ is the holding cost, $b(i)$ is the back-logged demand cost, and $c(i)$ is the price of one unit stock. We let D_{max} denote the maximal number of items that can be held in stock. Obviously, the state and action spaces are huge even for relatively small k. For a single commodity problem and the average reward criterion, if the demand d_t is IID, then there exists (Bertsekas, 1995) an optimal stationary threshold policy $a^*(m)$ such that $a^*(m) = \max\{M_{max} - m, 0\}$ for some M_{max}, which depends on the problem parameters. Note that in contrast to SA (see, e.g. Konda & Tsitsiklis, 2003) one does not have

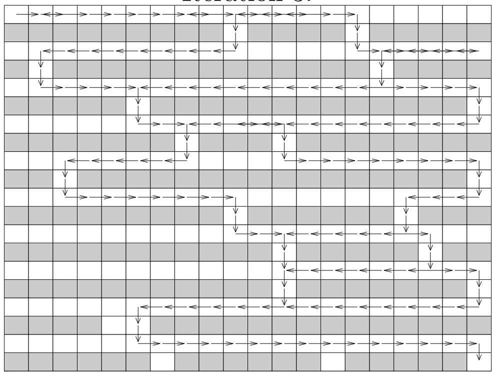
Iteration 1:



Iteration 2:



Iteration 3:



Iteration 4:

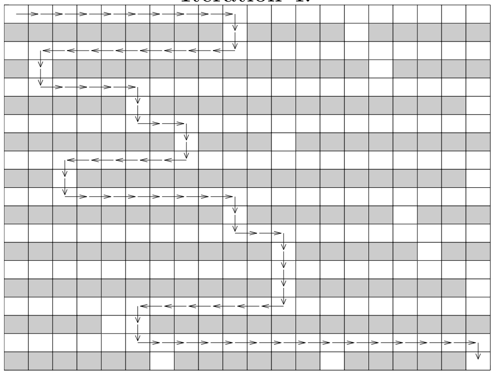
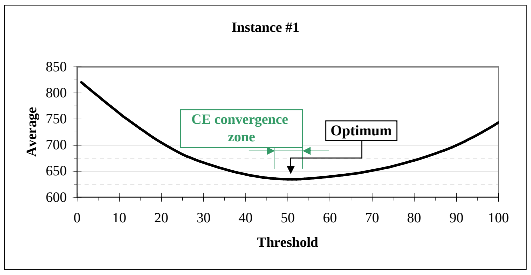


Figure 1. Results for the 20 × 20 maze. The arrows represent a probability of going in a direction which is higher than 0.01.

Figure 2. Results for instance 1 of the IC problem



to make any smoothness assumptions for applying CE policy search. In the case of multiple commodities, an educated guess would be to use a threshold policy for every commodity separately (but this may lead to a suboptimal solution.)

We have tested the CE policy search algorithm for parameterized policies on several IC problems while minimizing the average cost. As regeneration points we took the zero state (zero stock value). We look for a threshold policy, and all we need to do is to find the optimal threshold (or thresholds if there are multiple commodities.) Each observation cost was averaged over ℓ=10 regenerative cycles to reduce variance. Obviously, since the threshold is non-negative and since it can not exceed D_{max}, we can sample it from a Beta distribution, multiply it by the maximal demand and round it to the closest integer number. At the first iteration the threshold was generated from a Beta(1,1) distribution, i.e. from a uniform distribution. At each iteration we updated the parameters in the Beta distribution according to Algorithm 2.1. The update of the parameters is somewhat more complex in this case as the distribution is not NEF. However, the update equation can be efficiently solved numerically.

We started with running an IC problem with a single commodity. We let $c = 10$, $h = 5$, $b = 7$, and $D_{max} = 100$. The demand distribution for was a fixed demand function chosen randomly (in the initial phase we sampled the demand for every d from a uniform distribution, and normalized.) We run the CE method with $N = 100$ and $\rho = 1\%$. The number of iterations was 5-11, and the execution time 1-4 seconds. Figure 2 shows the calculated average cost curves for and the range of thresholds that the algorithm converged to. The CE policy search algorithm was also tested for a multi-commodity IC problem. In this problem there are seven commodities with different costs parameters and demand distributions[2], as specified in

[2]By $U[0, a]$ we mean uniform over $\{0, 1, \ldots, a\}$; Poisson and exponential distributions were truncated at 11 and normalized so that the sum is 1; the reverse exponential

Table 1. The total storage space was $D_{max} = 30$. The multi-commodity IC problem is in general a difficult problem, but a reasonable (sub-optimal) heuristic is to use a threshold policy for each commodity. We run the policy search 10 times using threshold policies. The relative error (comparing to the optimal threshold policy, obtained using a time consuming heuristic branch and bound search) was 0.1% (±0.05%) with 20 iterations (±5) and average execution time of 650 seconds. The advantage of the CE method is that by understanding the problem (and the structure of the solution, in this case), a robust and efficient method for learning a nearly optimal strategy can be easily derived.

Table 1. IC multi-commodity problem parameters

i	**c(i)**	**h(i)**	**b(i)**	Distribution
1	10	3	7	$U[0,11]$
2	14	7	4	$U[0,24]$
3	11	1	7	$U[0,9]$
4	17	8	6	$U[0,19]$
5	12	4	2	Poisson ($\mu = 4$)
6	8	6	8	Exponential ($\lambda = 0.3$)
7	10	9	5	Reverse exp. ($\lambda = 0.3$)

6. Conclusion

We presented a framework for policy search when the environment is unknown. A significant advantage of the CE method which was exploited in the IC problem is that knowledge of the structure of "good" policies can be easily exploited to facilitate an efficient search. Another advantage of the CE method is the speed of convergence and the small number of parameters that need to be tweaked for guaranteeing convergence.

There are plenty of off-the-shelf optimization algorithms that may be considered for policy search. The advantage of using the CE method is that there is no need to estimate gradients as required by many algorithms (e.g. steepest or conjugate gradient). Algorithms that are based on proximity relation (such as simulated annealing or guided local search) are also sensitive to the sampling error. Since gradients are not used when using the CE method, the CE method is expected to be more robust than other methods.

Future research includes: theoretical study of convergence; extension of the CE framework to a hierarchical framework; incorporating exploration mechanisms, as currently exploration is based on the initial random policy; and experimentation the CE method for multi-agent problems.

Acknowledgements. We are grateful for three anonymous reviewers for significantly improving the presentation of this work. S.M. was partially supported by the ARO under grant DAAD10-00-1-0466.

distribution was created from the exponential distribution by reversing it.

References

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

Baxter, J., Bartlett, P. L., & Weaver, L. (2001). Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research*, *15*, 351–381.

Bertsekas, D. (1995). *Dynamic programming and optimal control*, vol. I. Athena Scientific.

Bertsekas, D., & Tsitsiklis, J. (1996). *Neuro-dynamic programming.* Athena Scientific.

Borkar, V., & Meyn, S. (2000). The O.D.E. method for convergence of stochastic approximation and reinforcement learning. *SIAM J. Control Optim.*, *38*, 447–469.

de-Boer, P., Kroese, D., Mannor, S., & Rubinstein, R. (2003). A tutorial on the cross-entropy method. Submitted to the Annals of Operation Research.

Homem de Mello, T., & Rubinstein, R. (2002). Rare event estimation for static models via cross-entropy and importance sampling. Manuscript, Technion, Haifa, Israel.

Kaelbling, L., Littman, M., & Moore., A. (1996). Reinforcement learning - a survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Kearns, M., & Singh, S. (1998). Near-optimal reinforcement learning in polynomial time. *Proc. of the 15th Int. Conf. on Machine Learning* (pp. 260–268). Morgan Kaufmann.

Konda, V. R., & Tsitsiklis, J. N. (2003). Actor-critic algorithms. To appear in SIAM Journal on Control and Optimization.

Puterman, M. (1994). *Markov decision processes.* Wiley-Interscience.

Rosenstein, M. T., & Barto, A. G. (2001). Robot weightlifting by direct policy search. *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence* (pp. 839–846). Morgan Kaufmann.

Rubinstein, R. Y. (1999). The simulated entropy method for combinatorial and continuous optimization. *Methodology and Computing in Applied Probability*, *2*, 127–190.

Sutton, R., & Barto, A. (1998). *Reinforcement learning.* MIT Press.

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

Watkins, C. (1989). *Learning from delayed rewards.* Doctoral dissertation, Cambridge University.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*, 229–256.