Policy Gradient Reinforcement Learning for Robotics

Michael C. Koval
mkoval@cs.rutgers.edu

Michael L. Littman
mlittman@cs.rutgers.edu

May 9, 2011

1 Introduction

Learning in an environment with a continuous state-action space is traditionally a very difficult because there are an uncountable number of states-action pairs: explicitly storing any mapping between states, actions, and rewards is not possible. One class of algorithms known as *policy search* algorithms avoids this problem by learning with no explicit model of the environment [3]. Because of the prevalence of in high-dimensional, continuous state-action spaces in robotics, policy search algorithms have been successfully used for a number of real-world learning tasks: from teaching a robotic arm how to throw darts [3] to having a robot imitate human actions [4].

2 Policy Gradient

One policy search algorithm that has seen widespread use for learning on robotic systems is *policy gradient* [3] [4]. Like all policy search algorithms, policy gradient searches the the infinite space of policies to find the optimal policy, π^*, that maximizes *expected long-term reward*

$$J(\theta) = E\left\{\sum_{t=0}^{\infty} \gamma^t r_t\right\} \tag{1}$$

discounted by $\gamma \in (0, 1]$ [5]. Since an arbitrary policy space may contain uncountably many policies, two additional assumptions are required to make the problem tractable: convergence to a local maximum is acceptable and each policy is parametrized by a real-valued parameter vector $\theta \in \mathbb{R}^n$.

By assuming that a policy is fully determined by parameter vector θ, the task of finding the optimal policy becomes that of finding the optimal set of policy parameters θ^*. Once the optimal policy parameters are known, the optimal policy is simply $\pi(s, a|\theta^*)$. Because θ is a vector of real-valued parameters, any general-purpose optimization algorithm can now be applied to solve the reinforcement learning problem. Policy gradient, as its name suggests, uses the standard gradient ascent update rule

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t)$$

to incrementally step the policy towards a local maximum by iteratively moving in the direction of greatest increase in $J(\theta)$ [5]. As long as the estimate of $\nabla J(\theta)$ is unbiased and the time-dependent learning rate α_t satisfies $\sum_{t=0}^{\infty} \alpha_t > 0$ and $\sum_{t=0}^{\infty} \alpha_t^2 =$ constant this algorithm is guaranteed to converge to a local maximum [5].

3 Gradient Estimators

Applying the gradient update rule discussed in Section 2 is trivial if $J(\theta)$ and $\nabla J(\theta)$ have known closed-form solutions. Unfortunately, even simple environments with low-dimensional parameters violate this assumption: even though the parametrization of $\pi(s, a|\theta)$ is known, a closed form solution for $J(\theta)$ may not exist or may be exceedingly difficult to calculate. In such situations, analytically solving for $\nabla J(\theta)$ is not an option

and the gradient must be estimated using an algorithmic *gradient estimator* [4]. Two unbiased gradient estimators that have seen widespread use in reinforcement learning will be discussed next: finite-difference estimators in Section 3.1 and the more popular likelihood-ratio estimator in Section 3.2.

3.1 Finite-Difference Estimator

Originally developed by statisticians, a *finite-difference estimator* estimates the gradient by slightly perturbing θ by some offset $\Delta\theta$ and observing the corresponding, ΔJ [4]. As one of the simplest methods of estimating the gradient of a function, this method is based on directly evaluating the definition of the gradient

$$\nabla J(\theta) = \left(\frac{\delta J}{\delta \theta_1}, \frac{\delta J}{\delta \theta_2}, \cdots, \frac{\delta J}{\delta \theta_n} \right), \tag{2}$$

where each partial derivative is evaluated at point θ.

Expanding each element using the limit definition of the partial derivative, Equation 2 is reduced to

$$\nabla J(\theta) = \lim_{\Delta\theta \to 0} \frac{J(\theta + \Delta\theta) - J(\theta)}{||\Delta\theta||_2}, \tag{3}$$

assuming that the limit exists and $|| \cdot ||_2$ denotes the Euclidean norm of a vector. Since it is not actually possible to evaluate this limit using an infinitesimally small value of $\Delta\theta$, a finite-difference estimator approximates the value of this limit by substituting an extremely small value for $\Delta\theta$ [4].

While Equation 3 appears to require only one policy perturbation to estimate $\nabla J(\theta)$, this is unfortunately not true: a minimum of one policy perturbation is required *per policy parameter*. Assuming each perturbation occurs in only one dimension the elements of the gradient are independent. On a real system, individually perturbing each policy parameter is not always possible: slightly perturbing a stable policy in an unexpected manner may result in an unstable policy that harms the agent. To avoid such situations, the gradient can be estimated using any set of n or more *linearly independent* perturbations [4].

Instead of individually estimating each element of the gradient, instead concatenate all n linearly independent policy perturbations into the matrix $\Delta\Theta = [\theta_1, \cdots, \theta_n]^T$ and the resultant changes in expected long-term reward into the vector $\Delta J = [\Delta J_1, \cdots, \Delta J_n]^T$. Rewriting Equation 3 as a matrix equation

$$\Delta\Theta \nabla J(\theta) = \Delta J$$

expresses the gradient entirely in terms of known quantities [4, 3]. Solving Equation 3.1 using least-squares estimation gives a gradient estimate of

$$\nabla J(\theta) = (\Delta\Theta^T \Delta\Theta)^{-1} \Delta\Theta^T \Delta J. \tag{4}$$

When using Equation 4 to estimate $\nabla J(\theta)$, it is important to notice that ΔJ is never required to be defined between θ and $\theta + \Delta\theta$. This simplest case is known as a *forward difference estimator* (FDE) and is known for over-approximating the gradient on convex functions and underestimating the gradient on concave functions. The complete opposite, known as a *backward difference estimator* (BDE) defines ΔJ between $\theta - \Delta\theta$ and θ and has opposite properties. Combining FDE and BDE, the *central difference estimator* (CDE) defines ΔJ between $\theta + \frac{1}{2}\Delta\theta$ and $\theta - \frac{1}{2}\Delta\theta$. In practice, the CDE requires $2n$ rollouts per gradient estimate as opposed to the $n+1$ needed for the FDE and BDE, but produces estimates with much lower variance. See Algorithm 1 to see how a CDE might be implemented in practice.

3.2 Likelihood Ratio Estimator

Even with the ability to control the policy perturbations required to use a finite-difference estimator, it is even more desirable to not require any policy perturbations at all. Using knowledge of the policy's parametrization and using a mathematical trick known as the *REINFORCE trick* the gradient of the expected long-term reward function can be estimated without ever perturbing the policy. To see how this is possible, begin by expanding the definition of long-term expected average reward (Equation 1) to

$$J(\theta) = \int_S d(s|\theta) \int_A r(s,a)\pi(s,a|\theta)\, da\, ds$$

Algorithm 1 Central Difference Estimator

$E \leftarrow (0, \cdots, 0)$
repeat
  for transition s_t, a_t, r_t **do**
    $e_t \leftarrow r_t \cdot \nabla \log \pi(s_t, a_t|\theta)$
    $E \leftarrow e_t + \gamma E$
  end for
  $\nabla J \leftarrow E + (1 - \gamma)E$
until ∇J has converged

Algorithm 2 Likelihood Ratio Estimator

repeat
  for $k \in \{1, \cdots, n\}$ **do**
    $J_r \leftarrow \text{Rollout}(\theta - \frac{1}{2}\Delta\Theta_k)$
    $J_f \leftarrow \text{Rollout}(\theta + \frac{1}{2}\Delta\Theta_k)$
    $\Delta J_k \leftarrow J_f - J_r$
  end for
  $\nabla J \leftarrow (\Delta\Theta^T \Delta\Theta)^{-1} \Delta\Theta^T \Delta J$
until ∇J has converged

by explicitly averaging over all states and actions. In this form, $d(s|\theta)$ is the *stationary distribution* over the states: the probability that the agent will be in any particular state after enough time has passed that the starting state is inconsequential [5].

While not true of a general MDP, most MDPs considered under reinforcement learning satisfy the *unichain hypothesis*: for every possible starting state, the probability of visiting each other state is one. If the unichain assumption is satisfied, the term $d(s|\theta)$ does not depend upon the current policy and is therefore independent of the parameter vector θ [5]. Because of this assumption, $d(s|\theta)$ can be treated as a constant with respect to θ and

$$\nabla J(\theta) = \int_S d(s|\theta) \int_A r(s,a) \nabla \pi(s,a|\theta), \tag{5}$$

meaning that $\nabla J(\theta)$ depends upon only $r(s,a)$ and $\nabla \pi(s,a|\theta)$ at each time step [5].

While this form of the expression is more convenient than the original, it is still not clear how to directly estimate the value of this double integral from a series of rollouts. Multiplying Equation 5 by $\frac{\pi(s,a|\theta)}{\pi(s,a|\theta)}$ makes the term $\frac{\nabla \pi(s,a|\theta)}{\pi(s,a|\theta)}$ more obvious and allows for the simplification

$$\begin{aligned}
\nabla J(\theta) &= \int_S d(s|\theta) \int_A r(s,a) \pi(s,a|\theta) \left(\frac{\nabla \pi(s,a|\theta)}{\pi(s,a|\theta)} \right) da\, ds \\
&= \int_S d(s|\theta) \int_A r(s,a) \nabla \log \pi(s,a|\theta)\, da\, ds \\
&= E\left\{ r(s,a) \nabla \log \pi(s,a|\theta) \right\}
\end{aligned}$$

where the expectation is taken over all $s \in S$ and $a \in A$. This transformation is known as the *REINFORCE trick* [6] and converts the difficult-to-evaluate double integral in Equation 5 into an expected value that can be estimated using a normal Monte Carlo method [5].

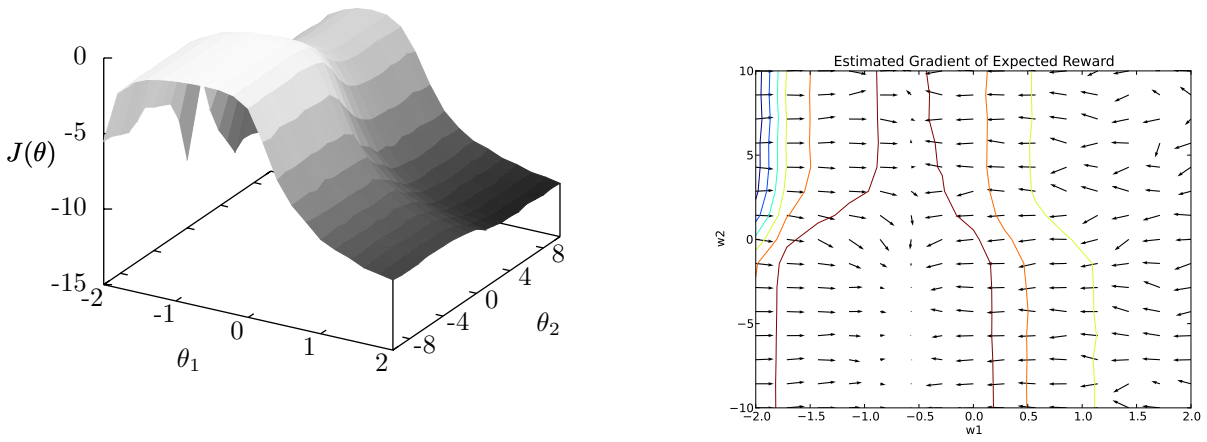
During each state transition in a rollout, the LRE accumulates discounted *eligibility*

$$e_t = r(s_t, a_t) \nabla \log \pi(s_t, a_t|\theta_t)$$

to form an *eligibility trace* that estimates the gradient of the expected long-term reward function. [2] Since the parametrization of the policy is fixed and is known in a closed-form, computing its gradient with respect to θ is simply a matter of using a computer algebra system. By using knowledge of policy parametrization, the LRE technique provides that converges more quickly than the finite-difference estimates without requiring perturbations of the policy parameters.

4 Natural Policy Gradient

Assuming that the gradient estimate is unbiased (such as a finite-difference estimator or a likelihood-ratio estimator) and α meets the appropriate criteria, the *vanilla policy gradient* algorithm described in Section 2 is guaranteed to converge to a local maximum. While it is not immediately obvious from the algorithm, vanilla policy gradient assumes that the parameter space is Euclidean, meaning that $||\theta_1 - \theta_2||_2$ is a meaningful definition of "distance" between policies $\pi(s,a|\theta_1)$ and $\pi(s,a|\theta_2)$ [1]. This is not true in general and



(a) Expected Long-Term Reward

(b) Gradient of Expected Long-Term Reward

Figure 1: Expected long-term reward function, $J(\theta)$, and its gradient $\nabla J(\theta)$. for LQR world. Following the gradient from any starting point will eventually lead to the global maximum at $(-0.58, -\infty)$.

is especially problematic when the parameters have widely different scales: a small change to one policy parameter may have a much greater effect on $J(\theta)$ than the equivalent change to a different parameter.

Unlike the vanilla gradient, the *natural gradient* corrects for the curvature caused by a non-Euclidean distance function of the parameter space. It does so by utilizing the the *Reimann metric tensor*, a matrix whose elements are

$$G_{ij} = \left(\frac{\delta x}{\delta \theta_i}\right)\left(\frac{\delta x}{\delta \theta_j}\right),$$

and relates each point in the parameter space to an infinitesimal distance given by some parametrization-specific distance function. Using the Riemann metric tensor to correct for this curvature, the *natural gradient* at point θ is $\tilde{\nabla} J(\theta)$, is simply

$$\tilde{J}(\theta) = [G(\theta)]^{-1} \nabla J(\theta),$$

the vanilla gradient transformed by the Riemann metric tensor [1].

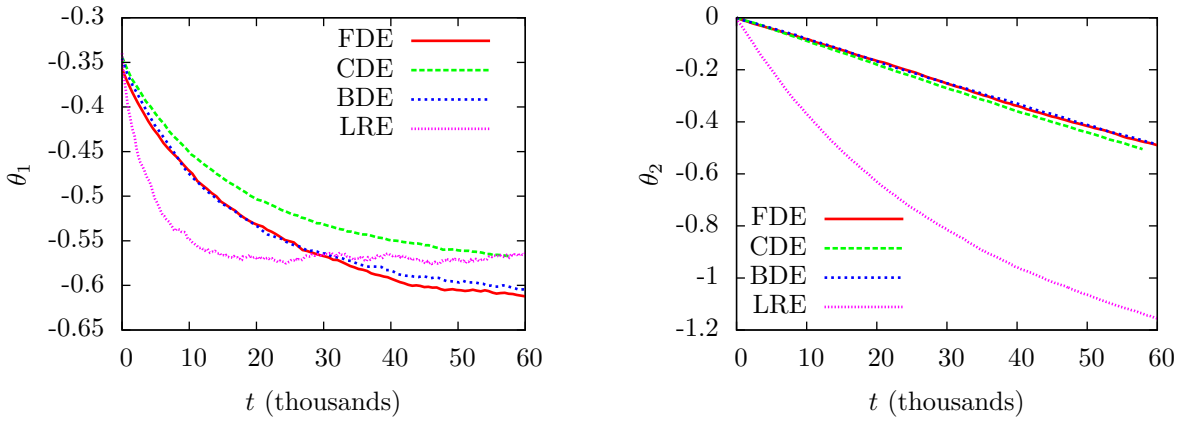
Assuming the distance function defined on the parameter space meets the requirements of distance metric, then G^{-1} is orthonormal and positive definite. With these properties, multiplying a vector by G^{-1} corresponds to a rotation of some angle between zero and 90 degrees in the parameter space. Since the natural gradient is still guaranteed to point in a direction of increase, this rotation does not change the convergence properties discussed in Section 2, but may greatly improve the rate of convergence [1].

5 Simulation

To better understand difference between the finite-difference estimator and the likelihood-ratio estimator, both were simulated in a simple environment. While this environment and parameter space have much lower dimensionality in practice, these simulations are still a good method of comparing the relative merits of the finite-difference and likelihood-ratio gradient estimators.

5.1 Linear Quadratic Regulator Simulation

This algorithm will be simulated in a simple world where the state, $s_t \in \mathbb{R}$ can be thought of as a distance from some set point. At each time step, t, the control algorithm selects action a_t and transitions to state



(a) Convergence of θ_1

(b) Convergence of θ_2

Figure 2: Learning of parameters $\theta = (\theta_1, \theta_2)$ for the LQR problem. Both the CDE and LRE converge to the global maximum at $(-0.58, -\infty)$ and the FDE and BDE converge to slightly different values.

$s_{t+1} = s_t + a_t + \epsilon_{noise}$ where $\epsilon_{noise} \sim N(0, \sigma_{noise})$ [2] After taking action a_t, the controller receives the reward $r_t = -s_t^2 - a_t^2$ that penalizes being distant from the origin and favors small, graduate movements [2]. Known as a *linear quadratic regulator* (LQR), this algorithm can be analytically solved using control theory and has a theoretically optimal set of parameters [2].

Since the policy gradient algorithm has no built-in concept of exploration, a standard linear amplifier will be modified to add stochasticity. Instead of using a constant gain, k, a stochastic linear amplifier draws a time-dependent gain from the normal distribution $k_t \sim N(\mu_t, \sigma_t)$. The mean and standard deviation, μ_t and σ_t, are related to the policy parameters by

$$\mu_t = \theta_1$$
$$\sigma_t = \frac{1}{1 + e^{-\theta_2}}.$$

so μ_t is the mean controller gain and the standard deviation, σ_t, is constrained to fall in the range $(0, 1)$ [2].

5.2 Linear Quadratic Regulator Results

The LQR world described in Section 5.1 was used to evaluate the FDE, CDE, and BDE described in Section 3.1 and the LRE algorithm described in Section 3.2. For all simulations, the constants were fixed at $\gamma = 0.9$, $\epsilon_{noise} = 0.5$, $\alpha = \frac{0.1}{\sigma^2}$, and $||\Delta\theta||_2 = 0.1$. To speed up convergence, the state was constrained to the range $[-4, +4]$ and all actions that exceeded this range were truncated before rewards were assigned. Each gradient estimation algorithm used rollouts that were truncated at 10 timesteps and were iteratively run until two consecutive gradient estimates converged with an L1 residual less than 0.1 [2].

Using control theory, Kimura et al. [2] showed that the theoretically optimal policy parameters are $\theta_* = (0.58, -\infty)$. By plotting the change in parameters θ_1 and θ_2 over time (Figure 2), it is possible to judge the performance of each gradient estimation algorithm by observing the rate at which it converges to the optimal set of parameters. As the plots show, the FDE and BDE have similar convergence rates and do not converge to the true optimal policy parameters due to the relatively large size of $||\Delta\theta||_2$. Unlike the other two finite-difference estimators, CDE does converge to the local maximum, but requires more rollouts and does so more slowly than the FDE and BDE [3].

Largely because of the additional information encoded in the policy parametrization, the LRE quickly converges to the true maximum. This fast convergence is due to the LRE's ability to generate a low-variance gradient estimate from a small number of rollouts. Some of the performance can be also be attributed to not directly perturbing the policy. By not requiring the parameter $||\Delta\theta||_2$, the LRE requires much less tuning to reach its theoretically optimal behavior.

6 Conclusion

While all policy gradient methods are based on the extremely simple update rule described in Section 2, the performance of the algorithm is only as is only as good as that of its underlying gradient estimator. Two of the most common gradient estimation algorithms, the finite-difference estimator and the likelihood-ratio estimator, were described in detail in Section 3 and evaluated in a simulated world in Section 5. As prior work suggested, the LRE outperformed all of the finite-difference estimators and is preferred to finite-difference estimators when controlling robotic systems. For robotic systems that have a highly non-Euclidean parameter space, using a domain-specific distance function and switching to natural policy gradient could greatly speed up the learning rate. Whether it is vanilla policy gradient or natural policy gradient, policy search methods have greatly simplified the application of reinforcement learning to real robotics platforms.

References

[1] S.I. Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

[2] H. Kimura and S. Kobayashi. Reinforcement learning for continuous action using stochastic gradient ascent. *Intelligent Autonomous Systems (IAS-5)*, pages 288–295, 1998.

[3] J. Peters and S. Schaal. Policy gradient methods for robotics. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 2219–2225. IEEE, 2006.

[4] J.R. Peters. *Machine learning of motor skills for robotics*. PhD thesis, University of Southern California, 2007.

[5] R.S. Sutton, S. Singh, and D. McAllester. Comparing Policy-Gradient Algorithms. In *IEEE Transactions on Systems, Man, and Cybernetics*. Citeseer, 1983.

[6] R.J. Williams. On the use of backpropagation in associative reinforcement learning. In *Neural Networks, 1988., IEEE International Conference on*, pages 263–270. IEEE, 1988.