CS294-40 Learning for Robotics and Control Lecture 15 - 10/16/2008

Policy Gradient Methods

Lecturer: Pieter Abbeel *Scribe: Fernando Garcia Bermudez*

1 Lecture outline

- Introduction
- Finite difference methods
- Likelihood ratio methods

2 Introduction

Let's choose a parametrized policy class $\pi_\theta : \theta \in \mathbb{R}^d$. As an example, we will consider helicopter flight. We will consider a policy parameterization as the one used by Ng et al. (2004).

x, y, z: x points forward along the helicopter, y sideways to the right, z downward.
n_x, n_y, n_z: rotation vector that brings helicopter back to level position (expressed in the helicopter frame).

$$u_{collective} = \theta_1 \cdot f_1(z^* - z) + \theta_2 \cdot \dot{z}$$
$$u_{elevator} = \theta_3 \cdot f_2(x^* - x) + \theta_4 f_4(\dot{x}) + \theta_5 \cdot q + \theta_6 \cdot n_y$$
$$u_{aileron} = \theta_7 \cdot f_3(y^* - y) + \theta_8 f_5(\dot{y}) + \theta_9 \cdot p + \theta_{10} \cdot n_x$$
$$u_{rudder} = \theta_{11} \cdot r + \theta_{12} \cdot n_z$$

Note that policy search depends on the number of parameters versus on the linearity of them. One can solve linearly per axis because integration of them over time yields the nonlinearity.

3 Finite difference methods

We estimate $U(\theta)$ from sample means (in a simulated environment or in the real world).

We can compute the gradient g using standard finite difference methods, as follows:

$$g_j = \frac{U(\theta^{(i)} + \epsilon e_j) - U(\theta^{(i)} - \epsilon e_j)}{2\epsilon} \qquad e_j = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \quad \leftarrow \text{j}^{\text{th}} \text{ entry} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

Then we can make a (small) step in the direction of the gradient to try to improve the utility.

However, for some robotics settings, we might not be willing to evaluate the performance of the policy according to these finite differences, especially so when testing on real systems (rather than simulation). We can, alternatively, find the gradient through the following computation:

Recall that, locally around our current estimate θ, we can approximate the utility $U(\theta^{(i)})$ at an alternative point $\theta^{(i)}$ with a linear approximation:

$$U(\theta) \approx U(\theta) + g^T(\theta^{(i)} - \theta)$$

Let's say we have a set of small perturbations $\theta^{(0)}, \theta^{(1)}, \ldots, \theta^{(m)}$ for which we are willing to evaluate the utility $U(\theta^{(i)})$. We can get an estimate of the gradient through solving the following set of equations for the gradient g through least squares:

$$\begin{aligned} U(\theta^{(0)}) &= U(\theta) + (\theta^{(0)} - \theta)^\top g \\ U(\theta^{(1)}) &= U(\theta) + (\theta^{(1)} - \theta)^\top g \\ &\ldots \\ U(\theta^{(m)}) &= U(\theta) + (\theta^{(m)} - \theta)^\top g \end{aligned}$$

Note that estimating the derivative of a stochastic function may be problematic as Figure 1 illustrates. To get a better, but still noisy measurement one could get a lot of samples and average.

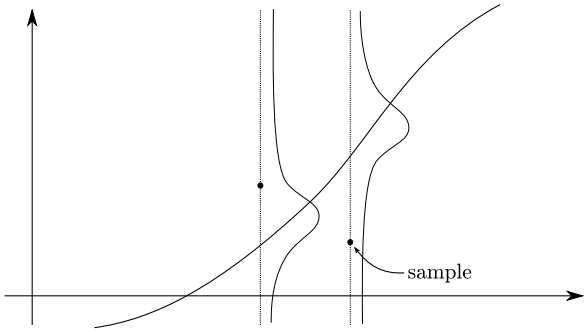

Figure 1: Problem estimating the derivative of a stochastic function. In the figure one can see a stochastic function for which we want to estimate the slote between the two vertical lines, and because we need to sample a point from a random distribution for the two specific values of θ, we end up getting two points that estimate the wrong slope for the function between those points.

For the simulated setting, rather than requiring a large number of samples, the PEGASUS algorithm (Ng and Jordan, 2000) turns the optimization into an optimization over a deterministic function. In particular, they observe that when you fix the random seed of the simulation, the simulation becomes deterministic, and computing gradients will be less susceptible to noise introduced by stochasticity.

Essentially, every random seed corresponds to a function, and one averages them to get the (deterministic) function one is optimizing (see Figure 2). This gives less noisy gradient estimates.

Using the PEGASUS algorithm, it took about 2 hours to search the parameters of a new helicopter.

4 Likelihood ratio methods

Let τ denote a trajectory and time H the horizon:

$$\tau = (s_0, a_0, s_1, a_1, \ldots, s_H, a_H)$$

Let,

$$R(\tau) = \sum_{t=0}^{H} R(s_t, a_t)$$

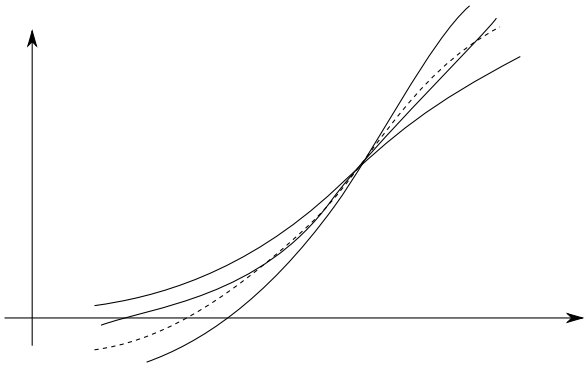

Figure 2: Target and the estimated functions using a fixed random seed.

$$U(\theta) = \sum_{\tau} p_\theta(\tau) R(\tau)$$

The last equation represents the utility and is smooth for a discrete system, which allows the following:

$$\begin{aligned}
\nabla_\theta U(\theta) &= \sum_{\tau} \nabla_\theta p_\theta(\tau) R(\tau) \\
&= \sum_{\tau} \frac{p_\theta(\tau)}{p_\theta(\tau)} \nabla_\theta p_\theta(\tau) R(\tau) \\
&= \sum_{\tau} p_\theta(\tau) \frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} R(\tau) \\
&\approx \frac{1}{m} \sum_{i=1}^{m} \frac{\nabla_\theta p_\theta(\tau^{(i)})}{p_\theta(\tau^{(i)})} R(\tau^{(i)})
\end{aligned}$$

Here i indexes over the m sample trajectories $\tau^{(i)}$ which are obtained while executing the current policy π_θ and which are used to obtain a finite sample estimate of the above expectation.

The above derivation allows the empirical average to be used to estimate the gradient. Note in the last step that the ratio is often easier to evaluate than just the gradient.

$$p_\theta(\tau) = \prod_{t=0}^{H} p(s_{t+1}|s_t, s_t) \pi_\theta(a_t|s_t)$$

$$\begin{aligned}
\nabla_\theta p_\theta(\tau) &= \prod_{t=0}^{H} p(s_{t+1}|s_t, a_t) \nabla_\theta \left(\prod_{t=0}^{H} \pi_\theta(a_t|s_t) \right) \\
&= \prod_{t=0}^{H} p(s_{t+1}|s_t, a_t) \prod_{t=0}^{H} \pi_\theta(a_t|s_t) \sum_{t'=0}^{H} \frac{\nabla_\theta \pi_\theta(a_{t'}|s_{t'})}{\pi_\theta(a_{t'}|s_{t'})}
\end{aligned}$$

$$\begin{aligned}\frac{\nabla_\theta p_\theta(\tau)}{p_\theta(\tau)} &= \frac{\prod_{t=0}^{H} p(s_{t+1}|s_t, s_t) \prod_{t=0}^{H} \pi_\theta(a_t|s_t)}{\prod_{t=0}^{H} p(s_{t+1}|s_t, s_t) \prod_{t=0}^{H} \pi_\theta(a_t|s_t)} \sum_{t'=1}^{H} \frac{\nabla_\theta \pi_\theta(a_{t'}|s_{t'})}{\pi_\theta(a_{t'}|s_{t'})} \\ &= \sum_{t'=0}^{H} \frac{\nabla_\theta \pi_\theta(a_{t'}|s_{t'})}{\pi_\theta(a_{t'}|s_{t'})}\end{aligned}$$

Pay close attention to the last equality in which by getting rid of the first term due to computing the ratio, the dynamics now dissapeared and we don't need to know them to perform this calculation. Plugging in this last result in the gradient of the utility function one gets:

$$\begin{aligned}\nabla_\theta U(\theta) &= \sum_\tau p_\theta(\tau) \left(\sum_{t'=0}^{H} \frac{\nabla_\theta \pi_\theta(a_{t'}|s_{t'})}{\pi_\theta(a_{t'}|s_{t'})} \right) R(\tau) \\ &\approx \frac{1}{m} \sum_{i=1}^{m} \sum_{t'=0}^{H} \frac{\nabla_\theta \pi_\theta(a_{t'}^{(i)}|s_{t'}^{(i)})}{\pi_\theta(a_{t'}^{(i)}|s_{t'}^{(i)})} R(\tau^{(i)})\end{aligned}$$

The last equality shows that the gradient of reinforce, g_{RF}, can be calculated as an empirical estimate without having the dynamic model. Note that this gradient doesn't change when offsetting the reward by a constant, so we could add this to the equation in order to reduce the variance of g_{RF}. Note also that this is an unbiased estimator since $\mathbb{E}[g]_{RF} = \nabla_\theta U(\theta)$.

References

[1] Greensmith, Evan, Peter L. Bartlett and Jonathan Baxter (2004). Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning. *Machine Learning Research*, vol. 5, p. 1471-1530.

[2] Ng, Andrew Y., Daishi Harada and Stuart Russell (1999). Policy invariance under reward transformations: Theory and application to reward shaping. *Proceedings of the 16th International Conference on Machine Learning.*

[3] Ng, Andrew Y. and Michael I. Jordan (2000). PEGASUS: A policy search method for large MDPs and POMDPs. *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence.*

[4] Ng, Andrew Y., H. Jin Kim, Michael I. Jordan and Shankar Sastry (2003). Autonomous helicopter flight via Reinforcement Learning. *Advances in Neural Information Processing Systems*, vol. 16.

[5] Ng, Andrew Y., Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger and Eric Liang (2004). Autonomous inverted helicopter flight via reinforcement learning. *International Symposium on Experimental Robotics.*

[6] Williams, Ronald J. (1992). Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*, vol. 8, p. 229-256.