Notes on policy gradients and the log derivative trick for reinforcement learning

David Meyer
dmm@{1-4-5.net,uoregon.edu}

February 18, 2014

1 Introduction

The *log derivative trick*[1] is a widely used identity that allows us to find various gradients required for policy learning. For policy-based reinforcement learning, we directly parameterize the policy. In value-based learning, we imagine we have value function approximator (either state-value or action-value) parameterized by θ:

$$V_\theta(s) \approx V^\pi(s) \qquad \text{\# state-value approximation} \tag{1}$$
$$Q_\theta(s,a) \approx Q^\pi(s,a) \qquad \text{\# action-value approximation} \tag{2}$$

Here our goal is to directly parameterize the policy (i.e., *model-free reinforcement learning*):

$$\pi_\theta(s,a) = \mathbb{P}[a|s,\theta] \qquad \text{\# parameterized policy} \tag{3}$$

2 Policy Objective Functions

There are three basic policy objective functions, each of which has the goal of given a policy $\pi_\theta(s,a)$ with paramters θ, find the best θ.

- In episodic environments we can use the *start value*: $J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$
- In continuing environments we can use the *average value*: $J_{aaV}(\theta) = \sum\limits_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$

[1] http://blog.shakirm.com/2015/11/machine-learning-trick-of-the-day-5-log-derivative-trick

- Or we can use the *average reward per time step*: $J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(s,a)\mathcal{R}_s^a$

where $d^{\pi_\theta}(s)$ is the *stationary distribution* of a Markov chain for π_θ.

So now we're casting policy based reinforcement learning as an optimization problem (e.g., there is a neural network that we want to learn the policy, e.g., via gradient ascent). Now, let $J(\theta)$ be a *policy objective function*. A policy gradient algorithm searches for a local maximum[2] in $J(\theta)$ by ascending the gradient of the policy with respect to the parameters θ. The update rule for θ is

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{4}$$

where $\nabla_\theta J(\theta)$ is the *policy gradient* and α is the *learning rate* (sometimes step-size parameter). In particular

$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \frac{\partial J(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix} \tag{5}$$

3 Computing the gradient analytically

First, we assume that the policy π_θ is differentiable wherever it is non-zero (this is a softer requirement than requiring π_θ be differentiable *everywhere*). In addition, we know the gradient: $\nabla_\theta J(\theta)$. In this case, let $p(\mathbf{x};\theta)$ be the likelihood parametrized by θ and let $\log p(\mathbf{x};\theta)$ be the *log likelihood*. Define

$$\begin{aligned} y &= p(\mathbf{x};\theta) && \text{\# define } y \text{ to be the likelihood parametrized by } \theta \\ z &= \log y = \log p(\mathbf{x};\theta) && \text{\# } z \text{ is the log likelihood} \end{aligned}$$

Then

[2] in the case that $J(\theta)$ is *non-convex*.

$$
\begin{aligned}
\frac{dz}{d\theta} &= \frac{dz}{dy}\cdot\frac{dy}{d\theta} && \text{\# chain rule} \\
\frac{dz}{dy} &= \frac{1}{p(\mathbf{x};\theta)} && \text{\#} \frac{d}{dx}\log x \approx \tfrac{1}{x} \\
\frac{dy}{d\theta} &= \frac{d}{d\theta}\,p(\mathbf{x};\theta) = \nabla_\theta p(\mathbf{x};\theta) && \text{\# definition (chain rule, again)} \\
\frac{dz}{d\theta} &= \frac{dz}{dy}\cdot\frac{dy}{d\theta} = \frac{\nabla_\theta\; p(\mathbf{x};\theta)}{p(\mathbf{x};\theta)} && \text{\# apply chain rule} \\
\frac{dz}{d\theta} &= \nabla_\theta \log p(\mathbf{x};\theta) && \text{\# set } w = p(\mathbf{x};\theta) \text{ and noting that } \nabla_\theta \log w = \tfrac{1}{w}\nabla_\theta w
\end{aligned}
$$

Here $\nabla_\theta \log p(\mathbf{x};\theta)$ is known as the score or sometimes the *Fischer* information. So the *log derivative trick* (sometimes *likelihood ratio*) is

$$\nabla_\theta \log p(\mathbf{x};\theta) = \frac{\nabla_\theta\; p(\mathbf{x};\theta)}{p(\mathbf{x};\theta)}$$

Setting $\pi_\theta(s,a) = p(\mathbf{x};\theta)$ we see that

$$
\begin{aligned}
\nabla_\theta \pi_\theta(s,a) &= \nabla_\theta \pi_\theta(s,a) \cdot \left[\tfrac{\pi_\theta(s,a)}{\pi_\theta(s,a)}\right] && \text{\# multiply by } 1 = \tfrac{\pi_\theta(s,a)}{\pi_\theta(s,a)} \\
&= \left[\tfrac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)}\right] \cdot \pi_\theta(s,a) && \text{\# rearrange} \\
&= \nabla_\theta \log \pi_\theta(s,a) \cdot \pi_\theta(s,a) && \text{\# log derivative trick: } \tfrac{\nabla_\theta \pi_\theta(s,a)}{\pi_\theta(s,a)} = \nabla_\theta \log \pi_\theta(s,a)
\end{aligned}
$$

and the score function is $\nabla_\theta \log \pi_\theta(s,a)$.

Another yet similar way to look policy gradients is as follows:[3] First, as Andrej points out policy gradients are a special case of a more general *score function gradient estimator*. Here we have an expectation of the form $\mathrm{E}_{x\sim p(x|\theta)}\big[f(x)\big]$. This is the expectation of a scalar valued function $f(x)$ under some probability distribution $p(x|\theta)$. Here $f(x)$ can be thought of as a *reward function* and $p(x|\theta)$ is the *policy* network.

The problem we want to solve is how we should shift the distribution (though its parameters θ) to increase its score as judged by f. Since the general gradient decent update rule is

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta) \tag{6}$$

[3] h/t Andrej Karpathy http://karpathy.github.io/2016/05/31/rl/

our goal is to find $\nabla_\theta E_{x\sim p(x|\theta)}\big[f(x)\big]$ and update θ in the direction indicated by the gradient. However, what we know is f and p.

$$\begin{aligned}
\nabla_\theta E_{x\sim p(x|\theta)}\big[f(x)\big] &= \nabla_\theta \sum_x p(x)f(x) && \text{\# defn expectation} \\
&= \sum_x \nabla_\theta p(x)f(x) && \text{\# swap sum and gradient} \\
&= \sum_x p(x)\frac{\nabla_\theta p(x)}{p(x)}f(x) && \text{\# multiply/divide by } p(x) \\
&= \sum_x p(x)\nabla_\theta \log p(x)f(x) && \# \ \frac{1}{z}\nabla_\theta z = \nabla_\theta \log(z) \\
&= E_{x\sim p(x|\theta)}\big[f(x)\nabla_\theta \log p(x)\big] && \text{\# defn expectation again}
\end{aligned}$$

The basic idea here is that we have some distribution $p(x|\theta)$ which we can sample from,[4] and for each sample we evaluate its score $f(x)$; then the gradient

$$\nabla_\theta E_{x\sim p(x|\theta)}\big[f(x)\big] = E_{x\sim p(x|\theta)}\big[f(x)\nabla_\theta \log p(x)\big] \tag{7}$$

is telling us how we should shift the distribution (through its parameters θ) if we wanted its samples to achieve higher scores (as judged by f). The second term $\nabla_\theta \log p(x)$ is telling us which direction in parameter space would lead to increase of the probability assigned to a given x. That is, if we were to move θ in the direction $\nabla_\theta \log p(x)$ we would see the new probability assigned to some x slightly increase (see Equation 6).

4 The Policy Gradient Theorem

Recall that the the start value policy for episodic environments was

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1] = \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \pi_\theta(s,a)\mathcal{R}_s^a \tag{8}$$

and thus

$$\begin{aligned}
\nabla_\theta J(\theta) &= \sum_{s\in\mathcal{S}} d(s) \sum_{a\in\mathcal{A}} \pi_\theta(s,a)\nabla_\theta \log \pi_\theta(s,a)\mathcal{R}_s^a && (9) \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) r] && (10)
\end{aligned}$$

[4] for example, $p(x|\theta)$ could be a Gaussian

A few things to notice:

- The policy gradient theorem generalizes the likelihood ratio
- In the policy gradient theorem we replace r with the long-term value $Q^{\pi}(s,a)$ (our estimate of r).
- The policy gradient theorem applies to the start-state, average reward and average value objectives

For any differentiable policy $\pi_\theta(s,a)$ and for any policy objective J_1, J_{avR} or $\frac{1}{1-\gamma}J_{avV}$, the **policy gradient** is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\Big[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)\Big] \tag{11}$$

Now that we have $\nabla_\theta J(\theta)$, we can use this gradient to train a neural network (e.g., the policy networks of AlphaGo).