Policy Gradients, Reinforcement Learning and Control*

David Meyer

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

Last update: November 30, 2018

1 Introduction

This document outlines how policy gradients for Reinforcement Learning (RL) are derived and how they work[1]. These methods directly optimize the policy parameters, and many of the methods described here attempt to lower the variance of the "naive" likelihood ratio policy gradient (which is known to have hight variance), leading to actor-critic designs [2]. The basic reinforcement learning setup is shown in Figure 1.

This document is is part of an approach to combining RL with more traditional Network Utility Maximization (NUM) approaches and is oriented towards traditional control theorists. As such we note that in traditional control theory parlance the controller (control theory) is the policy (RL) and the plant model (control theory) is what RL calls the "dynamics". RL is, in general, an instance of optimal (risk-aware) control [3].

Policy Gradient algorithms have a long history in Operations Research, Statistics, Control Theory, Discrete Event Systems and Machine Learning. These Policy Gradient methods have been known for some time, at least since Aleksandrov, V. M., Sysoyev, V. I., & Shemeneva, V. V. [4], and today including Barto, Sutton, and Anderson [5], Williams [6], Baxter and Bartlett [7], and many others. Essentially, our problem is that the performance gradient ($\nabla U(\theta)$ in the below) is unlikely to be computable in closed form, especially when learning control in large-scale problems or problems where the system dynamics are unknown. Hence the challenging aspect of the policy-gradient approach is to find an algorithm for estimating the gradient via *simulation*. Thinking forward a bit, for arbitrary networks, where does such a simulation come from?

*This report fulfills a deliverable for Project Number HE2017070001.

[1]Here we follow the notation used in [1].

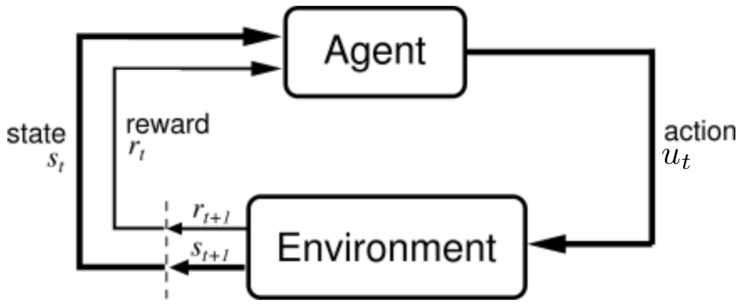

Figure 1: Basic Reinforcement Learning Setup (Figure courtesy [1])

One of the critical challenges for policy gradient methods is the high variance of the gradient estimator. This high variance is caused in part due to difficulty in credit assignment to the actions which affected the future rewards. Such issues are further exacerbated in long horizon problems, where assigning credit properly becomes even more challenging. Some approaches to reducing the variance of the gradient estimator are discussed in Section 4.

In these notes we will consider methods that can directly learn a parameterized policy $\pi(u|s,\theta)$ (sometimes written $\pi_\theta(u|s)$) which can select actions without consulting a value function, again via simulation[2]. A value function may still be used to learn the policy weights (e.g., in Actor-Critic methods, see Section 4.4), but that is not required for action selection. The notation $\theta \in \mathbb{R}^n$ is typically used for the primary learned weight vector, and $\pi_\theta(u|s) = \pi(u|s,\theta) = P(u_t = u|s_t = s, \theta_t = \theta)$ is the probability that action u is taken at time t given that the agent is in state s at time t with weight vector θ. If a method uses a learned value function as well, then the value function's weight vector is denoted w to distinguish it from θ, such as in $\hat{v}(s,w)$.

We wish to consider methods for learning the policy weights θ based on the gradient of some utility/performance measure $U(\theta)$) with respect to the policy weights. These methods seek to maximize performance, so their updates approximate gradient ascent in U. As is typical for gradient descent/ascent,

$$\boldsymbol{\theta}_{t+1} \coloneqq \boldsymbol{\theta}_t + \alpha\widehat{\nabla U(\boldsymbol{\theta}_t)} \tag{1}$$

where $\widehat{\nabla U(\boldsymbol{\theta}_t)}$ is a stochastic estimate whose expectation approximates the gradient of the performance measure U with respect to its argument θ_t. As we will see, $\widehat{\nabla U(\boldsymbol{\theta}_t)}$ will turn

[2]The simulated trajectories are frequently called rollouts.

out to be something like $\nabla_\theta \mathbb{E}_\tau[R(\tau)]$, where τ is a *trajectory* and $R(\tau)$ is the return for path τ (see Section 2).

2 Policy Optimization

In policy optimization we consider control policy π_θ, the parameterized policy (parameterized by parameter vector θ), and a utility/performance function $U(\theta)$, defined as follows:

$$U(\theta) = \max_\theta \mathbb{E}\left[\sum_{t=0}^{H-1} R(s_t)|\pi_\theta\right] \tag{2}$$

where $\pi_\theta(u|s)$ is a stochastic policy, that is, the probability of action u in state s. Note that frequently the action is denoted by a, but we will follow the notation of the control theory community and call the actions u. In addition, the utility (sometimes performance) function $U(\theta)$ is frequently referred to as $\eta(\theta)$ (for example, in [7]); however we shall use $U(\theta)$ here.

2.1 Why Policy Optimization?

There are several reasons we might want to directly optimize our policy rather than say, a value function. These include

- Stochastic policies tend to smooth out the problem[3]
- Policy optimization directly optimizes what we are interested in, namely $\pi_\theta(s, u)$
- Frequently the policy π is simpler that either value function Q or V
- The state value function V does not prescribe actions, so would need a dynamics model (plus one or more Bellman backups)
- Using the action-value function Q is problematic (difficult if not impossible) when the action space (the u's) and or the state spaces (the s's) are continuous or high dimensional (such as in robotic arm manipulation). This is because instead of directly parameterizing a policy, Q-value learning methods estimate the Q-function as $Q_\theta(s, u)$. The greedy policy selects the (discrete) action maximizing value: $u^* = \underset{u}{\text{argmax}}\, Q_\theta(s, u)$. Exploration can be performed using an ϵ-greedy policy, which chooses a uniform random action with probability ϵ and otherwise uses the greedy action. By its off-policy nature, Q-learning permits repeated training use of samples.

[3]Note that there are Deterministic Policy Gradient approaches, e.g. [8].

Note that on-policy TD(0) approaches such as SARSA [1] suffer from the same problem(s) since both SARSA and Q-learning use an ϵ-greedy policy (i.e. $\max_u Q(s,u)$) to strike the balance between exploration and exploitation.

3 Likelihood Ratio Policy Gradients

We want to compute the gradient $\nabla_\theta U(\theta)$ so that we can use gradient ascent/descent to improve the probability of good trajectories τ (Equation 1). In this section we will derive the Likelihood Ratio Policy Gradient which we can use for this purpose. Before doing so, however, first notice that Likelihood Ratio methods only change the probabilities of experienced paths, and further, these methods do not try to change the actions taken in a given path.

Now, let τ denote a state-action sequence with *horizon* H, $s_0, u_0, \cdots, s_{H-1}, u_{H-1}$ and let the total reward from trajectory τ be

$$R(\tau) = \sum_{t=0}^{H-1} R(s_t, u_t)$$

Notes:

- Note that $R(s_t, u_t)$ is a single rollout estimate of the action value function $Q(s_t, a_t)$
- Lack of discounting (or $\gamma = 1$); we will add discounting later as a means to lower the variance of our estimator.

Rewriting $U(\theta)$ we have

$$U(\theta) = \mathbb{E}_{\tau \sim P(\tau;\theta)} \left[\sum_{t=0}^{H-1} R(s_t, u_t); \pi_\theta \right] = \sum_\tau P(\tau;\theta) R(\tau) \tag{3}$$

Here $H = |\tau|$ and τ is a trajectory (rollout) with $\tau = [s_0, u_0, s_1, u_1, \cdots s_{H-1}, u_{H-1}]$.

Notes:

- State action pairs $[s_i, u_i]_{i=0}^{H-1}$ in a trajectory τ are inconsistently numbered in the literature. My guess: In the Monte Carlo policy gradient estimator setting, s_0 is special and is dealt with in a kind of ad-hoc fashion.

Given this notation/machinery, we can rewrite our goal as follows: Find θ such that

$$\max_{\theta} U(\theta) = \max_{\theta} \sum_{\tau} P(\tau;\theta) R(\tau) \tag{4}$$

Because we want to solve our optimization problem (Equation 4) using Stochastic Gradient Ascent (Equation 1), we need to find the gradient of our performance function $U(\theta)$ with respect to its parameters θ:

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta \sum_{\tau} P(\tau;\theta) R(\tau) && \text{\# definition of } U(\theta) && (5)\\
&= \sum_{\tau} \nabla_\theta P(\tau;\theta) R(\tau) && \text{\# Leibniz integral rule: swap } \nabla \text{ and } \textstyle\sum && (6)\\
&= \sum_{\tau} \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta) R(\tau) && \text{\# multiply by } 1 = \frac{P(\tau;\theta)}{P(\tau;\theta)} && (7)\\
&= \sum_{\tau} P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} R(\tau) && \text{\# rearrange} && (8)\\
&= \sum_{\tau} P(\tau;\theta) \nabla_\theta \log P(\tau;\theta) R(\tau) && \text{\# } \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} = \nabla_\theta \log P(\tau;\theta) && (9)\\
&= \mathbb{E}_{\tau \sim P(\tau;\theta)} \Big[\nabla_\theta \log P(\tau;\theta) R(\tau) \Big] && \text{\# definition of expectation} && (10)
\end{aligned}$$

Notes:

- Equation 7: Multiplying by $\frac{P(\tau;\theta)}{P(\tau;\theta)}$ is sometimes called the "Probabilistic Identity Trick" [9]. This allows us to form the score function (see "Log Derivative Trick" in the next bullet). Ed: Not sure why these are called tricks.
- Equation 9: $\frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} = \nabla_\theta \log P(\tau;\theta)$ is known as the "Log Derivative Trick" [9] or sometimes the "likelihood ratio trick" or even the "REINFORCE trick" [6] . $\frac{\nabla_\theta p(x|\theta)}{p(x|\theta)}$ is called the "likelihood ratio" or "score function" in classical statistics. The log derivative trick is sometimes framed up like this: The gradient of something divided by something is the gradient of the log of something. Also frequently pronounced "grad something divided by something is grad log something".
- Equation 10: This derivation reduces the computation of the gradient $\nabla_\theta U(\theta)$ to a simple expectation. This means, among other things, that we know from the Strong Law of Larger Numbers (see Section 6.1) that an *unbiased* estimate of the gradient $\hat{g}$ can be computed directly from our samples (Equation 12).

- Appendix A gives a slightly more general version of this derivation.

So our result is that

$$\nabla_\theta U(\theta) = \mathbb{E}_{\tau \sim P(\tau;\theta)} \Big[\nabla_\theta \log P(\tau;\theta) R(\tau) \Big] \tag{11}$$

An immediate implication of the fact that the gradient is reduced to an expectation is that we can approximate the gradient ($\hat{g}$) with the empirical estimate for m sample paths under policy π_θ, as follows:

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta) R(\tau^{(i)}) \tag{12}$$

A couple of notes here. First, we know by the law of large numbers (Appendix B) that $\frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta) R(\tau^{(i)}) \to \nabla_\theta U(\theta)$ with probability one; this is another way of saying our estimator $\hat{g}$ is unbiased. Next, the estimate $\hat{g}$ holds even if

- $R(\tau)$ is not continuous and/or unknown
- The sample space of paths is a discrete set

So for each sample path we need to be able to compute $\nabla_\theta \log P(\tau^{(i)};\theta)$. How might we do this? We can decompose $P(\tau^{(i)};\theta)$ into a states and actions, as follows:

$$\nabla_\theta \log P(\tau^{(i)};\theta) = \nabla_\theta \log \left[\prod_{t=0}^{H-1} \underbrace{P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)})}_{\text{dynamics/plant model}} \cdot \underbrace{\pi_\theta(u_t^{(i)} | s_t^{(i)})}_{\text{policy}} \right] \tag{13}$$

$$= \nabla_\theta \left[\sum_{i=0}^{H-1} \log P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}) + \sum_{i=0}^{H-1} \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \right] \tag{14}$$

$$= \nabla_\theta \sum_{i=0}^{H-1} \log P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}) + \nabla_\theta \sum_{i=0}^{H-1} \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \tag{15}$$

$$= \sum_{i=0}^{H-1} \nabla_\theta \log P(s_{t+1}^{(i)} | s_t^{(i)}, u_t^{(i)}) + \sum_{i=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \tag{16}$$

$$= \sum_{i=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \tag{17}$$

So $\nabla_\theta \log P(\tau^{(i)};\theta) = \sum_{i=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})$. This result is stated in a slightly different form in Baxter & Bartlett [7]:

$$\frac{\nabla_\theta P(\tau^{(i)};\theta)}{P(\tau^{(i)};\theta)} = \sum_{i=0}^{H-1} \frac{\nabla_\theta \pi_\theta(u_t^{(i)}|s_t^{(i)})}{\pi_\theta(u_t^{(i)}|s_t^{(i)})} \tag{18}$$

You can see that this is equivalent to Equation 17 by applying the "log derivative trick" [9], $\nabla \log f(X) = \frac{\nabla f(X)}{f(X)}$, to both sides of Equation 17.

Notice that the dynamics/plant model $P(s_{t+1}^{(i)}|s_t^{(i)}, u_t^{(i)})$ drops out of Equation 16. This is because the dynamics of the system do not depend on the parameters θ of our model, so the gradient is zero. This is a handy result as we are trying to do model-free learning (model-based learning would require the dynamics as part of the model). Now we can compute an unbiased estimate[4] of the gradient without the need for a dynamics model:

$$\hat{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta) R(\tau^{(i)}) \tag{19}$$

where

$$\nabla_\theta \log P(\tau^{(i)};\theta) = \sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)}) \tag{20}$$

Note that the gradients $\nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})$ required by Equation 20 are exactly what are computed by the backpropagation algorithm on the (log) policy neural network.

Combining Equations 19 and 20 gives

$$\hat{g} = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)}) R(\tau^{(i)}) \tag{21}$$

Equation 21 is our estimate of the gradient $g = \nabla_\theta U(\theta)$, where

[4] Unbiased means that $\mathbb{E}[\hat{g}] = \nabla_\theta U(\theta)$.

$$g = \mathbb{E}_{\tau \sim P(\tau;\theta)} \left[\nabla_\theta \log P(\tau;\theta) R(\tau) \right] \tag{22}$$

$$= \mathbb{E}_{s_t, u_t \sim P(\tau;\theta)} \left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t|s_t) \phi(s_t) \right] \tag{23}$$

Here $s_0 \sim \rho(s_0)$ is a distribution over initial states and $\phi(s_t) = R(u_t, s_t)$; we will explore the tradeoff between variance and bias for different options for ϕ_t below.

We can also relax the "episodic" assumption (changing the inner sum from $t = 0$ to $H - 1$ into a sum from $t = 0$ to ∞ in Equation 21) by using a *discount factor* $\gamma \in [0, 1]$; as we will see below that we can use γ to reason about the termination of $\hat{g}$.

4 Reducing the Variance of the (Gradient) Estimator

The likelihood ratio estimate of the policy gradient as described so far can be very sample inefficient due to its high variance. The next few sections describe straight forward ways to reduce the variance of the estimator $\hat{g}$. The methods described below are part of a more general approach to variance reduction which uses one ore more *control variates* [10]:

- Centering our estimate around a constant baseline [6]. As we shall see (Section 4.2, baselines effectively account for and remove the effects of past actions.
- Taking advantage of temporal structure based on the observation that future actions do not depend on past rewards (unless the policy has been changed) . This can result in a significant reduction of the variance of the policy gradient estimate. See Section II. B (1) of [11].
- Use an estimate of the state value function, $V^\pi(s_t^{(i)})$ as a control variate. The value function is a typical control variate used with Monte Carlo estimators [12]. See for example Equation 68.
- More advanced Policy Gradient methods such as Trust Region Policy Optimization [13] and Proximal Policy Optimization Algorithms [14] provide still better estimates. These ideas relate to the one of the main reasons for their use in robotics: how to make "small" changes to the policy π_θ while improving it. Section II C [11] discusses this point in some detail.

4.1 Aside on Control Variates

Suppose we wish to estimate $\mu = \mathbb{E}[f]$ where we know that f has high variance (such as is the case for Monte Carlo gradient estimators). The control variate method is to find a function g, the control variate, which without loss of generality has $\mathbb{E}[g] = 0$. Then

$$\mu = \mathbb{E}[f] = \mathbb{E}[f - g] \approx \frac{1}{n} \sum_{i=1}^{n} \Big[f(x_i) - g(x_i) \Big]$$

If we can choose g such that $\sigma^2_{(f-g)}$ is small, then we will have found a way to reduce the variance of our original estimator. This means that we want to find an identity $\mathbb{E}[g] = 0$ for a large class of functionals g so that we will have flexibility in choosing g. The identity typically used in the Policy Gradient setting is

$$\mathbb{E}_{s \sim P(s), u \sim \pi_\theta(u|s)} \Big[\nabla_\theta \log \pi_\theta(u|s) \phi(s) \Big] = 0 \tag{24}$$

where $\phi(s)$ is the control variate. You can see this pretty easily:

$$\begin{aligned}
\mathbb{E}_{s,u} \Big[\nabla_\theta \log \pi_\theta(u|s) \phi(s) \Big] &= \mathbb{E}_s \left[\mathbb{E}_u \Big[\nabla_\theta \log \pi_\theta(u|s) \phi(s) \Big] \right] && \text{\# def expectation} \\
&= \mathbb{E}_s \left[\phi(s) \cdot \mathbb{E}_u \Big[\nabla_\theta \log \pi_\theta(u|s) \Big] \right] && \text{\# } \phi(s) \text{ doesn't depend on } u \\
&= \mathbb{E}_s \Big[\phi(s) \cdot 0 \Big] && \text{\# } \mathbb{E} \Big[\nabla_\theta \log \pi(u|s) \Big] = 0 \\
&= \mathbb{E}_s \Big[0 \Big] \\
&= 0
\end{aligned}$$

We can also see pretty easily that $\mathbb{E} \Big[\nabla_\theta \log \pi(u|s) \Big] = 0$. More generally

$$\mathbb{E}_{x\sim p_\theta(x)}\left[\nabla_\theta \log p_\theta(x)\right] = \int p_\theta(x)\nabla_\theta \log p_\theta(x)dx \quad \# \text{ def expectation} \tag{25}$$

$$= \int \frac{\nabla_\theta p_\theta(x)}{p_\theta(x)} p_\theta(x)dx \quad \# \nabla_\theta \log p_\theta(x) = \frac{\nabla_\theta p_\theta(x)}{p_\theta(x)} \tag{26}$$

$$= \int \frac{p_\theta(x)}{p_\theta(x)} \nabla_\theta p_\theta(x)dx \quad \# \text{ rearrange} \tag{27}$$

$$= \int \nabla_\theta p_\theta(x)dx \quad \# \frac{p_\theta(x)}{p_\theta(x)} = 1 \tag{28}$$

$$= \nabla_\theta \int p_\theta(x)dx \quad \# \text{ swap } \int \text{ and } \nabla \tag{29}$$

$$= \nabla_\theta 1 \quad \# \int_x p_\theta(x)dx = 1 \ (p \text{ a density}) \tag{30}$$

$$= 0 \quad \# \text{ for c any constant } \nabla c = 0 \tag{31}$$

There are several options for $\phi(s_t)$ [15]. As we will see, these include (abusing notation slightly):

$$\phi(s_t) = \sum_{t=0}^{H-1} r_t \quad \# \text{ total reward of the trajectory } \tau$$

$$\phi(s_t) = \sum_{t'=t}^{H-1} r_t \quad \# \text{ total reward following action } u_t$$

$$\phi(s_t) = \sum_{t'=t}^{H-1} r_t - b(s_t) \quad \# \text{ baselined version of previous expression}$$

$$\phi(s_t) = r_t + V^\pi(s_{t+1}) - V^{\pi,\gamma}(s_t) \quad \# \text{ TD residual, denoted } \delta_t^V(s_{t+1})$$

$$\phi(s_t, u_t) = Q^\pi(s_t, u_t) \quad \# \text{ state-action value function}$$

$$\phi(s_t, u_t) = A^\pi(s_t, u_t) \quad \# \text{ advantage function: } A^\pi(s_t, u_t) = Q^\pi(s_t, u_t) - V^\pi(s_t)$$

where $r_t \sim P(r_t \mid s_t, u_t)$ is the probability of reward r_t when taking action u_t in state s_t,

and

$$V^\pi(s_t) = \mathbb{E}_{s_{t+1}\sim p(s_{t+1}|s_t,u_t),u_t\sim\pi(u_t|s_t)}\left[\sum_{l=0}^{H-1} r_{t+l}\right] \qquad \text{\# finite horizon state value function}$$

$$Q^\pi(s_t,u_t) = \mathbb{E}_{s_{t+1}\sim p(s_{t+1}|s_t,u_t),u_{t+1}\sim\pi(u_{t+1}|s_{t+1})}\left[\sum_{l=0}^{H-1} r_{t+l}\right] \qquad \text{\# finite horizon state-action value function}$$

$$A^\pi(s_t,u_t) = Q^\pi(s_t,u_t) - V^\pi(s_t) \qquad \text{\# finite horizon advantage function}$$

Notes:

- if $\phi(\cdot)$ depends on u (e.g., $\phi(s_t,u_t)) = A^\pi(s_t,u_t)$) the estimator will typically be biased
- Recall that an estimator $\hat{g}$ is *unbiased* if $\mathbb{E}[\hat{g}] = g$

Interestingly, $\mathbb{E}_{s,u}\left[A^\pi(s,u)\nabla\log\pi(u|s)\right] = E_{s,u}\left[Q^\pi(s,u)\nabla\log\pi(u|s)\right]$. Here's one way to see this:

$$\begin{aligned}
\mathbb{E}_{s,u}\left[A^\pi(s,u)\nabla\log\pi(u|s)\right] &= \mathbb{E}_{s,u}\left[(Q^\pi(s,u) - V^\pi(s))\nabla\log\pi(u|s)\right] \\
&= \mathbb{E}_s\left[\mathbb{E}_u\left[\big(Q^\pi(s,u) - V^\pi(s)\big)\nabla\log\pi(u|s)\right]\right] \\
&= \mathbb{E}_s\left[\mathbb{E}_u\left[Q^\pi(s,u)\nabla\log\pi(u|s) - V^\pi(s)\nabla\log\pi(u|s)\right]\right] \\
&= \mathbb{E}_s\left[\mathbb{E}_u\left[Q^\pi(s,u)\nabla\log\pi(u|s)\right] - \mathbb{E}_u\left[V^\pi(s)\nabla\log\pi(u|s)\right]\right] \\
&= \mathbb{E}_s\left[\mathbb{E}_u\left[Q^\pi(s,u)\nabla\log\pi(u|s)\right] - V^\pi(s)\cdot\mathbb{E}_u\left[\nabla\log\pi(u|s)\right]\right] \\
&= \mathbb{E}_s\left[\mathbb{E}_u\left[Q^\pi(s,u)\nabla\log\pi(u|s)\right] - V^\pi(s)\cdot 0\right] \\
&= \mathbb{E}_s\left[\mathbb{E}_u\left[Q^\pi(s,u)\nabla\log\pi(u|s)\right]\right] \\
&= \mathbb{E}_{s,u}\left[Q^\pi(s,u)\nabla\log\pi(u|s)\right]
\end{aligned}$$

The key step here is that in the term $\mathbb{E}_u\left[V^\pi(s)\nabla\log\pi(u|s)\right]$, $V^\pi(s)$ doesn't depend on u, so we can factor it out of the expectation. This leaves us with the term $V^\pi(s)\cdot\mathbb{E}_u\left[\nabla\log\pi(u|s)\right]$ and we know that by Equation 25 $\mathbb{E}_u\left[\nabla\log\pi(u|s)\right] = 0$. So the whole term is zero.[5]

[5]Sorry there wasn't enough space to write comments in-line.

As mentioned above (and I'll just note here), one weakness with the identity given in Equation 24 (and hence Equation 32) is that $\phi(s)$ can not depend on the action u. Q-Prop [16], approaches based on Stein's Identity [17], and Action-Dependent Factorized Baselines [18] for example, overcome this limitation by introducing action-dependent baseline functions. However, recent work questions the effectiveness of action-dependent baselines [19].

If we combine Equation 24 with the Policy Gradient Theorem [20] we get

$$\nabla J(\theta) = \mathbb{E}_{u\sim\pi_\theta(u|s)}\Big[\nabla_\theta \log \pi_\theta(u|s)\big(Q^\pi(s,t) - \phi(s)\big)\Big] \tag{32}$$

which generalizes quite a few policy gradient algorithms including REINFORCE [6], where $\phi(s) = b$ (b is an empirically determined constant baseline function), and Advantage Actor-Critic (A2C) [1], where $\phi(s) = V^\pi(s)$ ($V^\pi(s)$ is a state-dependent baseline function). We'll see more detail on all of this below.

To fill in some detail and see how this all works, consider a Monte Carlo approximation of $\mathbb{E}[X]$

$$\mathbb{E}[X] \approx \overline{X}(n) = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{33}$$

where the n samples from $p(X)$ are independent and identically distributed (iid)[6]. Now, let C be any other random variable over the same space as X with known mean $\mathbb{E}[C]$ and let b be some constant. Then consider the random variable Y given by

$$Y = X - b(C - \mathbb{E}[C]) \tag{34}$$

Here we can think of Y a "corrected" or controlled estimator of $\mathbb{E}[Y]$ where the deviations of C_i's from their known means $\mathbb{E}[C_i]$ are used to correct samples of X, drawing the X_i's closer to $\mathbb{E}[X]$ and hence reducing the variance of the estimator. Here information about the C_i's and their means is being *extracted* and *transferred* to our estimate of $\mathbb{E}[X]$, hopefully reducing the variance.

This method of introducing a random variable C for purposes of reducing variance of another random variable (in this case X) is called the control variates method, and $C - \mathbb{E}[C]$ is frequently called the control variate for estimating $\mathbb{E}[X]$.

[6] $p(X)$ is the (unknown) Data Generating Distribution.

Note that among other properties, Y has the same mean as X, than is, $\mathbb{E}[Y] = \mathbb{E}[X]$. To see this, notice that

$$\begin{aligned}
\mathbb{E}[Y] &= \mathbb{E}[X - b(C - \mathbb{E}[C])] && \text{\# Expected value of } Y \text{ (Equation 34)} && (35)\\
&= \mathbb{E}[X] - \mathbb{E}[b(C - \mathbb{E}[C]] && \text{\# } \mathbb{E}[X - Y] = \mathbb{E}[X] - \mathbb{E}[Y] && (36)\\
&= \mathbb{E}[X] - b\,\mathbb{E}[(C - \mathbb{E}[C]] && \text{\# } \mathbb{E}[bX] = b\,\mathbb{E}[X] && (37)\\
&= \mathbb{E}[X] - b\big(\mathbb{E}[C] - \mathbb{E}[\mathbb{E}[C]]\big) && \text{\# } \mathbb{E}[X - Y] = \mathbb{E}[X] - \mathbb{E}[Y] && (38)\\
&= \mathbb{E}[X] - b\big(\mathbb{E}[C] - \mathbb{E}[C]\big) && \text{\# } \mathbb{E}[\mathbb{E}[X]] = \mathbb{E}[X] \text{ } (\mathbb{E}[X] \text{ is a constant}) && (39)\\
&= \mathbb{E}[X] - b \cdot 0 && \text{\# } f(X) - f(X) = 0 && (40)\\
&= \mathbb{E}[X] && \text{\# } \mathbb{E}[Y] = \mathbb{E}[X] && (41)
\end{aligned}$$

This is handy since if we could simulate iid instances of Y, $\{Y_i\}_{i=1}^n$, then we could use $\overline{Y}(n)$ as our estimator instead of $\overline{X}(n)$. The reason we would like to use $\overline{Y}(n)$ rather than $\overline{X}(n)$ is that, in addition to X and Y having the same mean, we can use the basic properties of variance[7] to see how to directly reduce the variance of $\overline{Y}(n)$. To see this, first note that

$$\sigma_Y^2 = \sigma_X^2 + b^2\sigma_C^2 - 2b\sigma_{X,C}^2 \tag{42}$$

where $\sigma_{X,C}^2$ is the *covariance*[8] between X and C.

Rearranging Equation 42 a bit we see that

$$\sigma_Y^2 = \sigma_X^2 - b\big(2\sigma_{X,C}^2 - b\sigma_C^2\big) \tag{43}$$

So $\overline{Y}(n)$ can have lower variance than $\overline{X}(n)$ if we can manage to choose b and C such that $2\sigma_{X,C}^2 - b\sigma_C^2 > 0$, that is, $\sigma_{X,C}^2 > \frac{b}{2}\sigma_C^2$. Said another way, our new estimator $\overline{Y}(n)$ can have lower variance than $\overline{X}(n)$ whenever $\sigma_{X,C}^2 > \frac{b}{2}\sigma_C^2$. The covariance between X and C, $\sigma_{X,C}^2$, can be thought of as the information that C contains about X.

For a given C, we can view Equation 42 as a function of b, namely $g(b) = \sigma_Y^2(b)$, and then using a bit of calculus we see that

[7] In particular, $\sigma_{(X-Y)}^2 = \sigma_X^2 + \sigma_Y^2 - 2\sigma_{X,C}^2$.

[8] $\sigma_{X,C}^2$ and $\mathrm{Cov}(X, C)$ seem to be used interchangeably in the literature.

$$\frac{\partial g(b)}{\partial b} = \frac{\partial}{\partial b}\left(\sigma_X^2 + b^2\sigma_C^2 - 2b\sigma_{X,C}^2\right) \qquad \text{\# definition of } g(b), \text{ Equation 42} \tag{44}$$

$$= \frac{\partial}{\partial b}\sigma_X^2 + \frac{\partial}{\partial b}b^2\sigma_C^2 - \frac{\partial}{\partial b}2b\sigma_{X,C}^2 \qquad \text{\# } \frac{\partial(f+g)}{\partial\theta} = \frac{\partial f}{\partial\theta} + \frac{\partial g}{\partial\theta} \tag{45}$$

$$= \frac{\partial}{\partial b}b^2\sigma_C^2 - \frac{\partial}{\partial b}2b\sigma_{X,C}^2 \qquad \text{\# } \sigma_X^2 \text{ doesn't depend on } b \text{ so } \frac{\partial}{\partial b}\sigma_X^2 = 0 \tag{46}$$

$$= 2b\sigma_C^2 - \frac{\partial}{\partial b}2b\sigma_{X,C}^2 \qquad \text{\# power rule: } \frac{\partial}{\partial x}x^n = nx^{n-1} \tag{47}$$

$$= 2b\sigma_C^2 - 2\sigma_{X,C}^2 \qquad \text{\# power rule again} \tag{48}$$

To find b^*, the minimum value of b, set $\frac{\partial g(b)}{\partial b} = 2b\sigma_C^2 - 2\sigma_{X,C}^2 = 0$ and solve for b. This yields:

$$b^* = \frac{\sigma_{X,C}^2}{\sigma_C^2} \tag{49}$$

The variance of b^*, $\sigma_Y^2(b^*)$, is then defined as follows:

$$\sigma_Y^2(b^*) = \sigma_X^2(1 - \rho_{X,C}^2) \tag{50}$$

where the *correlation coefficient* $\rho_{X,C} = \sigma_{X,C}^2/\sigma_X\sigma_C$. Thus by choosing any C for which $\sigma_{X,C}^2 \neq 0$ we can always reduce the variance of the estimator.

All good but in practice we likely won't be able to compute the value of b^* since it is unlikely that we would know $\sigma_{X,C}^2$ and we might not even know σ_C^2. However, we could estimate b^* in advance by simulation: Choose a large n and estimate b^* as follows:

$$b^* \approx b^*(n) = \frac{\sum_{i=1}^{n}(X_i - \overline{X}(n))(C_i - \mathbb{E}[C])}{\sum_{i=1}^{n}(C_i - \mathbb{E}[C])^2}$$

Said another way, first run one simulation with large n to obtain the estimate $b^*(n)$, and then use that fixed value throughout our desired Monte Carlo simulation.

An effective control variate for X, say Y, needs to satisfy two requirements (to simplify the discussion we consider a scalar control):

1. X and Y need to be correlated[9]. That is, Y must carry some information about X.
2. $\mathbb{E}[Y]$ needs to be known

As mentioned ablve, the use of different control variates yield different variance reduction methods. Examples from Reinforcement Learning include REINFORCE [6], where a constant baseline function is chosen as a control variate. Advantage Actor-Critic (A2C) [1] uses a state-dependent baseline function as the control variate, which is often set to be an estimation of the state-value function $V^\pi(s) = \mathbb{E}_{a\sim\pi(a|s)}[Q^\pi(s,a)]$ so that $\hat{Q}^\pi(s,a) - \hat{V}^\pi(s)$ is an estimator of the advantage function A. Q-prop [16] proposes a more general baseline function that linearly depends on the actions. [17] extends the control variate methods used in REINFORCE and A2C by introducing more general action-dependent baseline functions based on Stein's Identity.

In summary, the basic idea behind the control variate method is to subtract some baseline function which has (analytically) zero expectation from a Monte Carlo gradient estimator. It is pretty easy to show that the resultant estimator does not in theory introduce additional bias (see Equation 53) but can have significantly lower variance. The reduction in variance becomes possible when the control variate is chosen so that its variance effectively cancels out the variance of the original gradient estimator.

4.2 Using a (constant) Baseline

This idea seems to have originated in Williams [6]. To see how adding a baseline can reduce the variance of our estimator, recall that the gradient estimator $\hat{g}$ is defined as follows:

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta) R(\tau^{(i)}) \tag{51}$$

We can add a baseline b to capture the idea that what we really care about is how good the reward $R(t)$ is compared to some baseline, such as the mean reward. The way we might formalize this is as follows:

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta)(R(\tau^{(i)})b) \tag{52}$$

b here is assumed to be constant, but in reality as long as b doesn't depend on the action u, adding a baseline b does not introduce bias to the gradient estimate $\hat{g}$ [6]. Why? We can see this pretty easily. The following result is used in Equation 7 of [11]; there they

[9] That is, $\sigma^2_{X,Y} \neq 0$.

state that $\int_{\mathbb{T}} \nabla_\theta p_\theta(\tau)\, d\tau = 0$, which is important as can be seen (with a bit more detail) in Equations 58 through 61 of the following:

$$\begin{aligned}
\mathbb{E}[\hat{g}] &= \mathbb{E}_{\tau \sim P(\tau;\theta)} \Big[\nabla_\theta \log P(\tau;\theta)[R(\tau) - b] \Big] && \text{\# definition of } \hat{g} && (53)\\
&= \sum_\tau P(\tau;\theta) \nabla_\theta \log P(\tau;\theta)[R(\tau) - b] && \text{\# definition of expectation} && (54)\\
&= \sum_\tau \nabla_\theta P(\tau;\theta)[R(\tau) - b] && \text{\# } \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} = \nabla_\theta \log P(\tau;\theta) && (55)\\
&= \sum_\tau \Big[\nabla_\theta P(\tau;\theta) R(\tau) - b \cdot \nabla_\theta P(\tau;\theta) \Big] && \text{\# multiply through} && (56)\\
&= \sum_\tau \nabla_\theta P(\tau;\theta) R(\tau) - \sum_\tau b \cdot \nabla_\theta P(\tau;\theta) && \text{\# distribute } \textstyle\sum && (57)\\
&= \sum_\tau \nabla_\theta P(\tau;\theta) R(\tau) - b \cdot \sum_\tau \nabla_\theta P(\tau;\theta) && \text{\# factor out } b && (58)\\
&= \sum_\tau \nabla_\theta P(\tau;\theta) R(\tau - b \cdot \nabla_\theta \sum_\tau P(\tau;\theta) && \text{\# swap } \textstyle\sum_\tau \text{ and } \nabla_\theta && (59)\\
&= \sum_\tau \nabla_\theta P(\tau;\theta) R(\tau) - b \cdot \nabla_\theta 1 && \text{\# } \textstyle\sum_\tau P(\tau;\theta) = 1 && (60)\\
&= \sum_\tau \nabla_\theta P(\tau;\theta) R(\tau) - b \cdot 0 && \text{\# } \nabla c = 0 \text{ for constant } c && (61)\\
&= \sum_\tau \nabla_\theta P(\tau;\theta) R(\tau) && \text{\# } b \cdot 0 = 0 && (62)\\
&= \nabla_\theta U(\theta) && \text{\# } \textstyle\sum_\tau \nabla_\theta P(\tau;\theta) R(\tau) = \nabla_\theta U(\theta) && (63)
\end{aligned}$$

So $\mathbb{E}[\hat{g}] = \nabla_\theta U(\theta)$, that is, $\hat{g}$ is unbiased, even if we subtract an arbitrary baseline b (so long as b doesn't depend on the action u).

4.3 Exploiting Temporal Structure

We know that in general, removing terms that don't depend on the current action can reduce variance. We can take advantage of this observation and of temporal structure of our problem. In particular, we know that future actions do not depend on past rewards (unless the policy has been changed). So if we can remove past rewards from our current estimate $\hat{g}$, we would expect a significant reduction of the variance of our gradient estimate (which in general we observe). This observation seems to have originated in Williams [6]. See also Section II. B (1) of [11]. We can implement this strategy in our current estimate

$\hat{g}$ by simply removing rewards received at times $j < t-1$, where t is the current timestep in the current rollout i, as follows:

$$\hat{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta)(R(\tau^{(i)}) - b) \tag{64}$$

$$= \frac{1}{m}\sum_{i=1}^{m} \left(\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})\right)\left(\sum_{t=0}^{H-1} R(s_t^{(i)}, u_t^{(i)}) - b\right) \tag{65}$$

$$= \frac{1}{m}\sum_{i=1}^{m} \left(\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})\left[\left(\sum_{j=0}^{t-1} R(s_j^{(i)}, u_j^{(i)})\right) + \left(\sum_{k=t}^{H-1} R(s_k^{(i)}, u_k^{(i)})\right) - b\right]\right. \tag{66}$$

$$= \frac{1}{m}\sum_{i=1}^{m} \left(\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)})\left[\left(\sum_{k=t}^{H-1} R(s_k^{(i)}, u_k^{(i)})\right) - b\right]\right) \tag{67}$$

Note that the term $\sum_{j=0}^{t-1} R(s_t^{(i)}, u_t^{(i)})$ drops out going from Equation 66 to Equation 67. This is again because future actions do not depend on past rewards.

4.4 How to choose b?

That's all good, but now how do we choose b? The next few sections will look at this question, which will lead to actor-critic models, in which b is chosen to be the value function V^π. The basic choices for b are part of a more general variance reduction approach known as "control variates" [12]. These include

- Constant baseline: $b = \mathbb{E}[R(t)] \approx \frac{1}{m}\sum_{i=1}^{m} R(\tau^{(i)})$
- Optimal baseline: $b = \frac{\sum_i (\nabla_\theta \log P(\tau^{(i)};\theta))^2 R(\tau^{(i)})}{\sum_i (\nabla_\theta \log P(\tau^{(i)};\theta))^2}$
- Time dependent baseline: $b_t = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=t}^{H-1} R(s_t^{(i)}, u_t^{(i)})$
- State-dependent expected return: $b(s_t^{(i)}) = \mathbb{E}[r_t + r_{t+1} + \ldots + r_{H-1}] = V^\pi(s_t^{(i)})$

Using the state-dependent expected return, i.e. the value function $V^\pi(s_t^{(i)})$, as a control variate[10] gives us what are called Actor-Critic (AC) models:

[10] Alternatively we can think of this as using a baseline $b = b(s_t) = V^\pi(s_t^{(i)})$ in Equation 53.

$$\hat{g} = \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(u_t^{(i)} | s_t^{(i)}) \left[\sum_{k=t}^{H-1} R(s_t^{(i)}, u_t^{(i)}) - V^\pi(s_t^{(i)}) \right] \right) \tag{68}$$

Equation 68 is generally referred to as the Advantage Actor-Critic (A2C) gradient [21]. Here the agent estimates $V^\pi(s_t^{(i)})$ from the data, for instance using a separate output from the same network used to estimate $\log \pi_\theta(u_t^{(i)} | s_t^{(i)})$.

Note that $R(s_t^{(i)}, u_t^{(i)}) - V^\pi(s_t^{(i)})$ estimates the advantage $A(s, a) = Q(s, a) - V(s)$. $R(s_t^{(i)}, u_t^{(i)})$ is typically computed using the discounted sum of as many future returns as are observed in a given batch, up to H , and is bootstrapped with $V^\pi(s_H^{(i)})$, appropriately discounted. $R(s_t^{(i)}, u_t^{(i)})$ can also be seen as part of a single rollout estimate of the action value function $Q(s_t^{(i)}, u_t^{(i)})$; more on this later.

The estimated advantage $R(s_t^{(i)}, u_t^{(i)}) - V^\pi(s_t^{(i)})$ gives us a way to train the estimator $V^\pi(s_t^{(i)})$: it is simply a supervised learning problem where $R(s_t^{(i)}, u_t^{(i)})$, the reward observed at step t on the i^{th} rollout, is the label and $V^\pi(s_t^{(i)})$ is our estimate. Thus we can train $V^\pi(s_t^{(i)})$ using something like the squared-error or cross-entropy loss at the same time we are training π_θ. Lastly, A2C adds an entropy term, $\nabla_\theta H(\pi_\theta(u_t^{(i)} | s_t^{(i)}))$, to the gradient to promote exploration and discourage premature convergence.

In a slight variation on A2C, Asynchronous Advantage Actor-Critic (A3C) [22], separate actor-learner threads sample environment steps and update a centralized copy of the parameters asynchronously to each other. In (batched) A2C, which performs similarly to A3C [14], separate environment instances are sampled, but the learner is single-threaded and gathers all data into one mini-batch to compute the gradient.

Interestingly, Actor-Critic methods share important architectural features with Generative Adversarial Networks (GANs) [22]. In particular, in AC models the actor has access only to the gradient estimate (Equation 68) to update the policy, whereas the critic ($V^\pi(s_t^{(i)})$) is trained on the actual data instances (the rollouts). In the GAN setting, the generator has access only to the gradient to update the generator, and the discriminator is trained on the actual data. Thus the actor in AC methods is similar to the generator in a GAN; similarily, the critic in AC models and the discriminator in GANs are trained on the actual data and are used to form the gradient used to train the actor/generator.

5 Appendix A

We can think about he derivation of the likelihood ratio policy gradient in a more general way. The basic result we would like to prove is

$$\nabla_\theta \mathbb{E}_{x\sim p(x|\theta)}[f(x)] = \mathbb{E}_x[f(x)\nabla_\theta \log p(x \mid \theta)] \tag{69}$$

To see this, imagine that $f(x) = R(\tau)$.

So the proof of this is amazingly simple given the log derivative trick [9]. First, consider a function $f(x)$ for which we wish to find $\nabla_\theta \mathbb{E}_{x\sim p(x|\theta)}[f(x)]$. Then

$$\begin{aligned}
\nabla_\theta \mathbb{E}_{x\sim p(x|\theta)}[f(x)] &= \nabla_\theta \int_x p(x \mid \theta) f(x) dx && \text{\# definition} && (70)\\
&= \int_x \nabla_\theta p(x \mid \theta) f(x) dx && \text{\# exchange grad and sum} && (71)\\
&= \int_x p(x \mid \theta) \frac{\nabla_\theta p(x \mid \theta)}{p(x \mid \theta)} f(x) dx && \text{\# multiply by } \frac{p(x \mid \theta)}{p(x \mid \theta)} && (72)\\
&= \int_x p(x \mid \theta) \nabla_\theta \log p(x \mid \theta) f(x) && \text{\# log-derivative trick} && (73)\\
&= \mathbb{E}_{x\sim p(x|\theta)}[f(x)\nabla_\theta \log p(x \mid \theta)] && \text{\# defn expectation} && (74)
\end{aligned}$$

Define $\hat{g}_i = f(x_i)\nabla_\theta \log p(x_i \mid \theta)$. An empirical estimate the gradient for m samples is

$$\nabla_\theta \mathbb{E}_{x\sim p(x|\theta)}[f(x)] = g \approx \frac{1}{m}\sum_{i=1}^{m} \hat{g}_i = \frac{1}{m}\sum_{i=1}^{m} f(x_i)\nabla_\theta \log p(x_i \mid \theta) \tag{75}$$

So $\nabla_\theta E_{x\sim p(x|\theta)}[f(x)] = E_{x\sim p(x|\theta)}[f(x)\nabla_\theta \log p(x \mid \theta)]$. This gives us an unbiased gradient estimator; to compute the gradient estimate, just sample $x_i \sim p(x \mid \theta)$ and then compute the gradient estimate $\hat{g}_i = f(x_i)\nabla_\theta \log p(x_i \mid \theta)$. Now, let the trajectory (sometimes path) τ be the state-action sequence $(s_0, a_0, r_0, s_1, a_1, r_1, \cdots, a_{T-1}, a_{T-1}, r_{T-1})$ and suppose $f(x) = R(\tau)$, the total return for *path* τ. Then

$$\nabla_\theta \mathbb{E}_\tau[R(\tau)] = E_\tau[\nabla_\theta \log p(\tau \mid \theta) R(\tau)] \tag{76}$$

so that all we really need to do is see that we can compute $p(\tau \mid \theta)$:

$$p(\tau \mid \theta) = \mu_0(s_0) \prod_{t=0}^{T-1} \underbrace{\pi(a_t \mid s_t, \theta)}_{\text{policy}} \underbrace{P(s_{t+1}, r_t) \mid s_t, a_t)}_{\text{dynamics}} \qquad \#\text{definition} \tag{77}$$

$$\log p(\tau \mid \theta) = \log \mu_0(s_0) + \sum_{t=0}^{T-1} \log \pi(a_t \mid s_t, \theta) + \log P(s_{t+1}, r_t) \mid s_t, a_t) \tag{78}$$

$$\nabla_\theta \log p(\tau \mid \theta) = \nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t \mid s_t, \theta) \tag{79}$$

$$= \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta) \tag{80}$$

So $\nabla_\theta \mathbb{E}_\tau[R] = \mathbb{E}_\tau[R(\tau) \nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t \mid s_t, \theta)]$, which gives a direct way of computing a gradient we can use with standard Stochastic Gradient Ascent (or Descent), where the parameter update rule would be something like $\theta \coloneqq \theta + \alpha \nabla_\theta \mathbb{E}_\tau[R(\tau)]$. Essentially move θ in the direction of better expected return. Note also that the model dynamics, $P(s_{t+1}, r_t) \mid s_t, a_t)$, conveniently drops out of the gradient (good thing given that we're trying to learn model-free control).

6 Appendix B

6.1 Strong Law of Large Numbers

Let $X_1, X_2, \ldots, X_M$ be a sequence of **independent** and **identically distributed** random variables, each having a finite mean $\mu_i = E[X_i]$.

Then with probability 1

$$\frac{1}{M}\sum_{i=1}^{M} X_i \to E[X] \tag{81}$$

as $M \to \infty$.

6.2 Ergodic Theorem

Let $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(M)}$ be M samples from a Markov chain that is *aperiodic*, *irreducible*, and *positive recurrent*[11], and $E[g(\theta)] < \infty$.

Then with probability 1

$$\frac{1}{M}\sum_{i=1}^{M} g(\theta_i) \to E[g(\theta)] = \int_{\Theta} g(\theta)\, \pi(\theta)\, d\theta \tag{82}$$

as $M \to \infty$ and where π is the stationary distribution of the Markov chain.

7 Acknowledgements

[11] In this case, the chain is said to be *ergodic*.

References

[1] R. S. Sutton and A. G. Barto., *Reinforcement learning: An Introduction*. MIT Press, 1998.

[2] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems 12* (S. A. Solla, T. K. Leen, and K. Müller, eds.), pp. 1008–1014, MIT Press, 2000.

[3] E. Todorov, "Optimal control theory," 2006.

[4] V. M. Aleksandrov, V. I. Sysoyev, and V. V. Shemeneva, "Stochastic optimization," *Engineering Cybernetics*, vol. 5, pp. 11–16, 1968.

[5] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Artificial neural networks," ch. Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems, pp. 81–93, Piscataway, NJ, USA: IEEE Press, 1990.

[6] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[7] J. Baxter and P. L. Bartlett, "Infinite-horizon policy-gradient estimation," *J. Artif. Int. Res.*, vol. 15, pp. 319–350, Nov. 2001.

[8] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pp. I–387–I–395, JMLR.org, 2014.

[9] S. Mohamed, "The log derivative trick." `http://blog.shakirm.com/2015/11/machine-learning-trick-of-the-day-5-log-derivative-trick/`, Nov. 2015. Accessed: Sun Jun 3 11:33:18 PDT 2018.

[10] R. Szechtman, "Control variates techniques for monte carlo simulation," in *Proceedings of the 2003 Winter Simulation Conference, 2003.*, vol. 1, pp. 144–149 Vol.1, Dec 2003.

[11] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Beijing, China), 2006.

[12] E. Greensmith, P. L. Bartlett, and J. Baxter, "Variance reduction techniques for gradient estimates in reinforcement learning," *J. Mach. Learn. Res.*, vol. 5, pp. 1471–1530, Dec. 2004.

[13] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015.

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *ArXiv e-prints*, July 2017.

[15] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," *ArXiv e-prints*, June 2015.

[16] S. Gu, T. Lillicrap, Z. Ghahramani, R. E. Turner, and S. Levine, "Q-Prop: Sample-Efficient Policy Gradient with An Off-Policy Critic," *ArXiv e-prints*, Nov. 2016.

[17] H. Liu, Y. Feng, Y. Mao, D. Zhou, J. Peng, and Q. Liu, "Action-depedent Control Variates for Policy Optimization via Stein's Identity," *ArXiv e-prints*, Oct. 2017.

[18] C. Wu, A. Rajeswaran, Y. Duan, V. Kumar, A. M. Bayen, S. Kakade, I. Mordatch, and P. Abbeel, "Variance Reduction for Policy Gradient with Action-Dependent Factorized Baselines," *ArXiv e-prints*, Mar. 2018.

[19] G. Tucker, S. Bhupatiraju, S. Gu, R. E. Turner, Z. Ghahramani, and S. Levine, "The Mirage of Action-Dependent Baselines in Reinforcement Learning," *ArXiv e-prints*, Feb. 2018.

[20] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, (Cambridge, MA, USA), pp. 1057–1063, MIT Press, 1999.

[21] A. Stooke and P. Abbeel, "Accelerated Methods for Deep Reinforcement Learning," *ArXiv e-prints*, Mar. 2018.

[22] V. Mnih, A. Puigdomènech Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," *ArXiv e-prints*, Feb. 2016.