CS234 Notes - Lecture 8 & 9
Policy Gradient

Luke Johnston, Emma Brunskill

March 20, 2018

1 Introduction to Policy Search

So far, in order to learn a policy, we have focused on **value-based** approaches where we find the optimal state value function or state-action value function with parameters θ,

$$V_\theta(s) \approx V^\pi(s)$$

$$Q_\theta(s,a) \approx Q^\pi(s,a)$$

and then use V_θ or Q_θ to extract a policy, e.g. with ϵ-greedy. However, we can also use a **policy-based** approach to directly parameterize the policy:

$$\pi_\theta(a|s) = \mathbb{P}[a|s;\theta]$$

policy-based v.s. value-based

In this setting, our goal is to directly find the policy with the highest value function V^π, rather than first finding the value-function of the optimal policy and then extracting the policy from it. Instead of the policy being a look-up table from states to actions, we will consider stochastic policies that are parameterized. Finding a good policy requires two parts:

- Good policy parameterization: our function approximation and state/action representations must be expressive enough
- Effective search: we must be able to find good parameters for our policy function approximation

Policy-based RL has a few advantages over value-based RL:

- Better convergence properties (see Ch 13.3 of Sutton and Barto)
- Effectiveness in high-dimensional or continuous action spaces, e.g. robotics. One method for continuous action spaces is covered in section 6.2.
- Ability to learn stochastic policies. See the following section.

data efficient to train

The disadvantages of policy-based RL methods are:

- They typically converge to locally rather than globally optimal policies, since they rely on gradient descent.
- Evaluating a policy is typically data inefficient and high variance.

At least we have local optima convergence, instead of no guarantee in value based approaches

Every time we update the policy, we can no longer use the previous episode generated by different policy. → can be solved by Importance Sampling.

2 Stochastic policies

In this section, we will briefly go over two environments in which a stochastic policy is better than any deterministic policy.

2.1 Rock-paper-scissors

For a relatable example, in the popular zero-sum game of rock-paper-scissors, any policy that is not uniformly random:

$$P(\texttt{rock}|s) = 1/3$$
$$P(\texttt{scissors}|s) = 1/3$$
$$P(\texttt{paper}|s) = 1/3$$

can be exploited.

motivation

2.2 Aliased gridworld

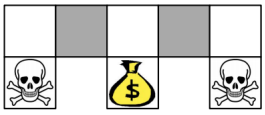

Figure 1: In this partially observable gridworld environment, the agent cannot distinguish between the gray states.

In the above gridworld environment, suppose that the agent can move in the four cardinal directions, so its actions space is $A = \{N, S, E, W\}$. However, suppose that it can only sense the walls around its current location. Specifically, it observes features of the following form for each direction:

$$\phi(s) = \begin{bmatrix} \mathbf{1}(\text{wall to N}) \\ \dots \\ \mathbf{1}(\text{wall to W}) \end{bmatrix}$$

Note that its observations are not fully representative of the environment, as it cannot distinguish between the two gray squares. This also means that its domain is not Markov. Hence, a deterministic policy must either learn to always go left in the gray squares, or always go right. Neither of these policies is optimal, since the agent can get stuck in one corner of the environment:

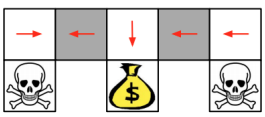

Figure 2: For this deterministic policy, the agent cannot "escape" from the upper-left two states.

However, a stochastic policy can learn to randomly select a direction in the gray states, guaranteeing that it will eventually reach the reward from any starting location. In general, stochastic policies can help overcome an adversarial or non-stationary domain and cases where the state-representation is not Markov.

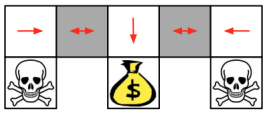

Figure 3: A stochastic policy which moves E or W with equal probability in the gray states will reach the goal in a few time steps with high probability.

3 Policy optimization

3.1 Policy objective functions

Once we have defined a policy $\pi_\theta(a|s)$, we need to able to measure how it is performing in order to optimize it. In an episodic environment, a natural measurement is the **start value** of the policy, which is the expected value of the start state:

$$J_1(\theta) = V^{\pi_\theta}(s_1) = \mathbb{E}_{\pi_\theta}[v_1]$$

In continuing environments we can use the **average value** of the policy, where $d^{\pi_\theta}(s)$ is the stationary distribution of π_θ:

$$J_{avV}(\theta) = \sum_s d^{\pi_\theta}(s) V^{\pi_\theta}(s)$$

or alternatively we can use the **average reward** per time-step:

$$J_{avR}(\theta) = \sum_s d^{\pi_\theta}(s) \sum_a \pi_\theta(a|s) R(s,a)$$

In these notes we discuss the episodic case, but all the results we derive can be easily extended to the non-episodic case. We will also focus on the case where the discount $\gamma = 1$, but again, the results are easily extended to general γ.

3.2 Optimization methods

With an objective function, we can treat our policy-based reinforcement learning problem as an optimization problem. In these notes, we focus on gradient descent, because recently that has been the most common optimization method for policy-based RL methods. However, it is worth considering some **gradient-free optimization methods**, including the following:

- Hill climbing
- Simplex / amoeba / Nelder Mead
- Genetic algorithms
- Cross-Entropy method (CEM)
- Covariance Matrix Adaptation (CMA)
- Evolution strategies

These methods have the advantage over gradient-based methods in that they do not have to compute a gradient of the objective function. This allows the policy parameterization to be non-differentiable, and these methods are also often easy to parallelize. Gradient-free methods are often a useful baseline to try, and sometimes they can work embarrassingly well [1]. However, this methods are usually not very sample efficient because they ignore the temporal structure of the rewards - updates only take into account the total reward over the entire episode, and they do not break up the reward into different rewards for each state in the trajectory. (See section 4.3).

4 Policy gradient

Let us define $V(\theta)$ to be the objective function we wish to maximize over θ. Policy gradient methods search for a *local* maximum in $V(\theta)$ by ascending the gradient of the policy, w.r.t parameters θ

Maximize V(θ) through $\Delta\theta = \alpha\nabla_\theta V(\theta)$

where α is a step-size parameter and $\nabla_\theta V(\theta)$ is the policy gradient

$$\nabla_\theta V(\theta) = \begin{pmatrix} \frac{\partial V(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial V(\theta)}{\partial \theta_n} \end{pmatrix}$$

4.1 Computing the gradient

With this setup, all we have to do is compute the gradient of the objective function $V(\theta)$, and we can optimize it! The method of **finite difference** from calculus provides an approximation of the gradient:

$$\frac{\partial V(\theta)}{\partial \theta_k} \approx \frac{V(\theta + \epsilon u_k) - V(\theta)}{\epsilon}$$

where u_k is a unit vector with 1 in kth component, 0 elsewhere. This method uses n evaluations to compute the policy gradient in n dimensions, so it is quite inefficient, and it usually only provides a noisy approximation of the true policy gradient. However, it has the advantage that it works for non-differentiable policies. An example of a successful use of this method to train the AIBO robot gait can be found in [2].

4.1.1 Analytic gradients

Let us set the objective function $V(\theta)$ to be the expected rewards for an episode,

$$V(\theta) = \mathbb{E}_{(s_t,a_t)\sim\pi_\theta}\left[\sum_{t=0}^{T} R(s_t, a_t)\right] = \mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\right] = \sum_\tau P(\tau;\theta)R(\tau)$$

where τ is a trajectory,

$$\tau = (s_0, a_0, r_0, \cdots, s_{T-1}, a_{T-1}, r_{T-1}, s_T)$$

$P(\tau;\theta)$ denotes the probability over trajectories when following policy π_θ, and $R(\tau)$ is the sum of rewards for a trajectory. Note that this objective function is the same as the **start value** $J_1(\theta)$ as mentioned in section 3.1 when the discount $\gamma = 1$.

If we can mathematically compute the policy gradient $\nabla_\theta \pi_\theta(a|s)$, then we can go right ahead and

compute the gradient of this objective function with respect to θ:

$$\nabla_\theta V(\theta) = \nabla_\theta \sum_\tau P(\tau;\theta) R(\tau) \tag{1}$$

$$= \sum_\tau \nabla_\theta P(\tau;\theta) R(\tau) \tag{2}$$

$$= \sum_\tau \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta) R(\tau) \tag{3}$$

$$= \sum_\tau P(\tau;\theta) R(\tau) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} \tag{4}$$

$$= \sum_\tau P(\tau;\theta) R(\tau) \nabla_\theta \log P(\tau;\theta) \tag{5}$$

$$= \mathbb{E}_{\tau\sim\pi_\theta} [R(\tau) \nabla_\theta \log P(\tau;\theta)] \tag{6}$$

The expression $\frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)}$ in equation (4) is known as the **likelihood ratio.**

The trick in steps (3)-(6) helps for two reasons. First, it helps us get the gradient into the form $\mathbb{E}_{\tau\sim\pi_\theta}[\dots]$, which allows us to approximate the gradient by sampling trajectories $\tau^{(i)}$:

$$\nabla_\theta V(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} R(\tau^{(i)}) \nabla_\theta \log P(\tau^{(i)};\theta) \tag{7}$$

use $\frac{1}{m}\sum_{i=1}^{m}$ to approximate $\sum_\tau P(\tau;\theta)$

Second, computing $\nabla_\theta \log P(\tau^{(i)};\theta)$ is easier than working with $P(\tau^{(i)};\theta)$ directly:

$$\nabla_\theta \log P(\tau^{(i)};\theta) = \nabla_\theta \log \left[\underbrace{\mu(s_0)}_{\text{Initial state distribution}} \prod_{t=0}^{T-1} \underbrace{\pi_\theta(a_t|s_t)}_{\text{policy}} \underbrace{P(s_{t+1}|s_t,a_t)}_{\text{dynamics model}} \right] \tag{8}$$

$$= \nabla_\theta \left[\log \mu(s_0) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) + \log P(s_{t+1}|s_t,a_t) \right] \tag{9}$$

$$= \sum_{t=0}^{T-1} \underbrace{\nabla_\theta \log \pi_\theta(a_t|s_t)}_{\text{no dynamics model required!}} \tag{10}$$

Working with $\log P(\tau^{(i)};\theta)$ instead of $P(\tau^{(i)};\theta)$ allows us to represent the gradient without reference to the initial state distribution, or even the environment dynamics model!

model-based or model-free is not relevant.

The expression $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is known as the **score function.**

Putting equations (7) and (10) together, we get

$$\nabla_\theta V(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} R(\tau^{(i)}) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta \left(a_t^{(i)} \mid s_t^{(i)} \right)$$

which we can convert into a concrete algorithm for optimizing π_θ (section 5). But before that, we will mention the generalized version of this result and cover an optimization of the above derivation that takes advantage of decomposing $R(\tau^{(i)})$ into a sum of reward terms $r_t^{(i)}$ (section 4.3).

4.2 The policy gradient theorem

Theorem 4.1. *For any differentiable policy $\pi_\theta(a|s)$ and for any of the policy objective functions $V(\theta) = J_1, J_{avR}$, or $\frac{1}{1-\gamma} J_{avV}$, the policy gradient is*

$$\nabla_\theta V(\theta) = \mathbb{E}_{\pi_\theta}[Q^{\pi_\theta}(s,a) \cdot \nabla_\theta \log \pi_\theta(a|s)]$$

We will not go over the derivation of this more general theorem, but the same concepts discussed in this lecture apply to non-episodic (continuing) environments. In our discussion thus far, the total episode rewards $R(\tau)$ have been substituted in place of the Q values of this theorem, but in the following section we will use the temporal structure to get our result into a form that looks more like this theorem, where the future returns G_t (which are unbiased estimates of $Q(s_t, a_t)$) appear in place of $Q^{\pi_\theta}(s,a)$.

4.3 Using temporal structure of rewards for the policy gradient

Equation (6) above can be written

$$\nabla_\theta V(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta}\left[R(\tau) \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \tag{11}$$

Notice that the rewards $R(\tau^{(i)})$ are treated as a single number which is a function of an entire trajectory $\tau^{(i)}$. We can break this down into the sum of all the rewards encountered in the trajectory,

$$R(\tau) = \sum_{t=1}^{T-1} R(s_t, a_t)$$

Using this knowledge, we can derive the gradient estimate for a single reward term $r_{t'}$ in exactly the same way we derived equation (11):

$$\nabla_\theta \mathbb{E}_{\pi_\theta}[r_{t'}] = \mathbb{E}_{\pi_\theta}\left[r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

Since $\sum_{t'=t}^{T-1} r_{t'}^{(i)}$ is the return $G_t^{(i)}$, we can sum this up over all time steps for a trajectory to get

$$\nabla_\theta V(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] = \mathbb{E}_{\pi_\theta}\left[\sum_{t'=0}^{T-1} r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \tag{12}$$

$$= \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T-1} r_{t'}\right] \tag{13}$$

$$= \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta \log \pi_\theta(a_t|s_t)\right] \tag{14}$$

Going from (12) to (13) may not be obvious, so let's go over a quick example. Say we have a trajectory that is three time steps long. Then equation (12) becomes

$$\begin{aligned}\nabla_\theta V(\theta) = \mathbb{E}_{\pi_\theta}\big[&r_0 \nabla_\theta \log \pi_\theta(a_0|s_0) + \\ &r_1(\nabla_\theta \log \pi_\theta(a_0|s_0) + \nabla_\theta \log \pi_\theta(a_1|s_1)) + \\ &r_2(\nabla_\theta \log \pi_\theta(a_0|s_0) + \nabla_\theta \log \pi_\theta(a_1|s_1) + \nabla_\theta \log \pi_\theta(a_2|s_2))\big]\end{aligned}$$

Regrouping the terms, we get

$$\begin{aligned}\nabla_\theta V(\theta) = \mathbb{E}_{\pi_\theta}\big[&\nabla_\theta \log \pi_\theta(a_0|s_0)(r_0 + r_1 + r_2) + \\ &\nabla_\theta \log \pi_\theta(a_1|s_1)(r_1 + r_2) + \\ &\nabla_\theta \log \pi_\theta(a_2|s_2)(r_2)\big]\end{aligned}$$

which equals equation (13) as expected. The main idea is that the policy's choice at a particular time step t only affects rewards received in later steps of the episode, and has no effect on rewards received in previous time steps. Our original expression in equation (11) did not take this into account.

Our final expression that we will use in the policy gradient algorithm in the next section is

$$\nabla_\theta V(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] \approx \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{T-1} G_t^{(i)} \cdot \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)}) \tag{15}$$

5 REINFORCE: A Monte-Carlo policy gradient algorithm

We've done most of the work towards our first policy gradient algorithm in the sections above. The algorithm simply samples multiple trajectories following the policy π_θ while updating θ using the estimated gradient (15).

Algorithm 1 REINFORCE: Monte-Carlo policy gradient algorithm

procedure REINFORCE(α)
    Initialize policy parameters θ arbitrarily
    for each episode $\{s_1, a_1, r_2, \cdots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
        for $t = 1$ to $T - 1$ **do**
            $\theta \leftarrow \theta + \alpha \cdot G_t \nabla_\theta \log \pi_\theta(a_t|s_t)$
    return θ

6 Differentiable policy classes

6.1 Discrete action space: softmax policy

In discrete action spaces, the softmax function is commonly used to parameterize the policy:

$$\pi_\theta(a|s) = \frac{e^{\phi(s,a)^T \theta}}{\sum_{a'} e^{\phi(s,a')^T \theta}}$$

The score function is then

$$\begin{aligned}
\nabla_\theta \log \pi_\theta(a|s) &= \nabla_\theta \left[\phi(s,a)^T\theta - \log \sum_{a'} e^{\phi(s,a')^T\theta} \right] \\
&= \phi(s,a) - \frac{1}{\sum_{a'} e^{\phi(s,a')^T\theta}} \nabla_\theta \sum_{a'} e^{\phi(s,a')^T\theta} \\
&= \phi(s,a) - \frac{1}{\sum_{a'} e^{\phi(s,a')^T\theta}} \sum_{a'} \phi(s,a') e^{\phi(s,a')^T\theta} \\
&= \phi(s,a) - \sum_{a'} \phi(s,a') \frac{e^{\phi(s,a')^T\theta}}{\sum_{a''} e^{\phi(s,a'')^T\theta}} \\
&= \phi(s,a) - \sum_{a'} \phi(s,a') \pi_\theta(a'|s) \\
&= \phi(s,a) - \mathbb{E}_{a' \sim \pi_\theta(a'|s)}[\phi(s,a')]
\end{aligned}$$

6.2 Continuous action space: Gaussian policy

For continuous action spaces, a common choice is a Gaussian policy $a \sim \mathcal{N}(\mu(s), \sigma^2)$.

- The mean action is a linear combination of state features: $\mu(s) = \phi(s)^T\theta$
- The variance σ^2 can be fixed, or also parameterized

The score function is

$$\nabla_\theta \log \pi_\theta(a|s) = \frac{(a - \mu(s))\phi(s)}{\sigma^2}$$

7 Variance reduction with a baseline

A weakness of Monte-Carlo policy gradient algorithms is that the returns $G_t^{(i)}$ often have high variance across multiple episodes. One way to address this is to subtract a **baseline** $b(s)$ from each $G_t^{(i)}$. The baseline can be any function, as long as it does not vary with a.

$$\nabla_\theta V(\theta) = \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] = \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{T-1} (G_t - b(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

First, why do we want to do this? Intuitively, we can think of $(G_t - b(s_t))$ as an estimate of how much better we did after time step t than is expected by the baseline $b(s_t)$. So, if the baseline is approximately equal to the expected return $b(s_t) \approx \mathbb{E}[r_t + r_{t+1} + \cdots + r_{T-1}]$, then we will be increasing the log-probability of action a_t proportionally to how much better the return G_t is than expected. Previously, we were increasing the log-probability proportionally to the magnitude of G_t, so even if the policy always achieved exactly its expected returns, we would still be applying gradient updates that could cause it to diverge. The quantity $(G_t - b(s_t))$ is usually called the *advantage*, A_t. We can estimate the true advantage from a sampled trajectory $\tau^{(i)}$ with

$$\hat{A}_t = (G_t^{(i)} - b(s_t))$$

Secondly, why *can* we do this? It turns out that subtracting a baseline in this manner does not introduce any bias into the gradient calculation. $\mathbb{E}_\tau[b(s_t)\nabla_\theta \log \pi_\theta(a_t|s_t)]$ evaluates to zero, and hence

skip proof

has no effect on the gradient update.

$$
\begin{aligned}
&\mathbb{E}_{\tau\sim\pi_\theta}[b(s_t)\nabla_\theta \log \pi_\theta(a_t|s_t)] \\
&= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[\mathbb{E}_{s_{(t+1):T},a_{t:(T-1)}}[\nabla_\theta \log \pi_\theta(a_t|s_t)b(s_t)]\right] \text{ (break up expectation)} \\
&= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[b(s_t)\mathbb{E}_{s_{(t+1):T},a_{t:(T-1)}}[\nabla_\theta \log \pi_\theta(a_t|s_t)]\right] \text{ (pull baseline term out)} \\
&= \mathbb{E}_{s_{0:t},a_{0:(t-1)}}\left[b(s_t)\mathbb{E}_{a_t}[\nabla_\theta \log \pi_\theta(a_t|s_t)]\right] \text{ (remove irrelevant variables)} \\
&= \mathbb{E}_{s_0:t,a_0:(t-1)}\left[b(s_t)\sum_a \pi_\theta(a_t|s_t)\frac{\nabla_\theta \pi_\theta(a_t|s_t)}{\pi_\theta(a_t|s_t)}\right] \text{ (expand expectation + take derivative of logarithm)} \\
&= \mathbb{E}_{s_0:t,a_0:(t-1)}\left[b(s_t)\sum_{a_t} \nabla_\theta \pi_\theta(a_t|s_t)\right] \\
&= \mathbb{E}_{s_0:t,a_0:(t-1)}\left[b(s_t)\nabla_\theta \sum_{a_t} \pi_\theta(a_t|s_t)\right] \\
&= \mathbb{E}_{s_0:t,a_0:(t-1)}\left[b(s_t)\nabla_\theta 1\right] \\
&= \mathbb{E}_{s_0:t,a_0:(t-1)}\left[b(s_t)\cdot 0\right] \\
&= 0
\end{aligned}
$$

7.1 Vanilla policy gradient

Using the baseline as described above, we introduce the "vanilla" policy gradient algorithm. Suppose that the baseline function has parameters $\mathbf{w}$.

Algorithm 2 Vanilla Policy Gradient Algorithm

procedure POLICY GRADIENT(α)
  Initialize policy parameters θ and baseline values $b(s)$ for all s, e.g. to 0
  for iteration $= 1, 2, \ldots$ **do**
    Collect a set of m trajectories by executing the current policy π_θ
    for each time step t of each trajectory $\tau^{(i)}$ **do**
      Compute the *return* $G_t^{(i)} = \sum_{t'=t}^{T-1} r_{t'}$
      Compute the *advantage estimate* $\hat{A}_t^{(i)} = G_t^{(i)} - b(s_t)$
    Re-fit the baseline to the empirical returns by updating $\mathbf{w}$ to minimize

why norm ?
b(st) and Gt are scalars →

$$\sum_{i=1}^{m}\sum_{t=0}^{T-1} \|b(s_t) - G_t^{(i)}\|^2$$

gradient evaluation

    Update policy parameters θ using the policy gradient estimate $\hat{g}$

$$\hat{g} = \sum_{i=1}^{m}\sum_{t=0}^{T-1} \hat{A}_t^{(i)} \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})$$

    with an optimizer like SGD ($\theta \leftarrow \theta + \alpha \cdot \hat{g}$) or Adam
  return θ and baseline values $b(s)$

One natural choice for the baseline is the state value function, $b(s_t) = V(s_t)$. Under this formulation, we can define the advantage function as $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$. However, since we do not know the true state values, we instead use an estimate $\hat{V}(s_t;\mathbf{w})$ for some weight vector $\mathbf{w}$. We can simultaneously learn the weight vector $\mathbf{w}$ for the baseline (state-value) function and policy parameters θ using the Monte-Carlo trajectory samples.

Note that in the above algorithm, we usually do not compute the gradients $\sum_t \hat{A}_t \nabla_\theta \log \pi_\theta(a_t|s_t)$ individually. Rather, we accumulate data from a batch into a loss function

$$L(\theta) = \sum_t \hat{A}_t \log \pi_\theta(a_t|s_t)$$

and then apply the gradients all at once by computing $\nabla_\theta L(\theta)$. We can also introduce a component into this loss to fit the baseline function:

$$L(\theta, \mathbf{w}) = \sum_t \left(\hat{A}_t \log \pi_\theta(a_t|s_t) - \|b(s_t) - G_t^{(i)}\|^2 \right)$$

We can then compute the gradients of $L(\theta, \mathbf{w})$ w.r.t. θ and $\mathbf{w}$ to perform SGD updates.

7.2 N-step estimators

In the above derivations, we have used the Monte-Carlo estimates of the reward in the policy gradient approximation. However, if we have access to a value function (for example, the baseline), then we can also us TD methods for the policy gradient update, or any intermediate blend between TD and MC methods:

$$\begin{aligned}
\hat{G}_t^{(1)} &= r_t + \gamma V(s_{t+1}) \\
\hat{G}_t^{(2)} &= r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) \\
&\cdots \\
\hat{G}_t^{(\text{inf})} &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots
\end{aligned}$$

0-step look ahead

1-step look ahead

which we can also use to compute advantages:

$$\begin{aligned}
\hat{A}_t^{(1)} &= r_t + \gamma V(s_{t+1}) - V(s_t) \\
\hat{A}_t^{(2)} &= r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t) \\
&\cdots \\
\hat{A}_t^{(\text{inf})} &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots - V(s_t)
\end{aligned}$$

$A_t^{(1)}$ is a purely TD estimate, and has low variance, but high bias. $A_t^{(\text{inf})}$ is a purely MC estimate, and has zero bias, but high variance. If we choose an intermediate value of k for $A_t^{(k)}$, we can get an intermediate amount of bias and an intermediate amount of variance.

hyperparameter.

References

[1] https://blog.openai.com/evolution-strategies/

[2] Kohl and Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. ICRA 2004. http://www.cs.utexas.edu/ ai-lab/pubs/icra04.pdf Emma Brunskill (CS234 Reinforcement Learning.) Lecture 8: Policy Gradient I 28 Winter 201