Policy gradients

This is a set of notes to accompany the code in `vpg.py` - these essentially outline a set of variations on vanilla policy gradient (which is derived below) as well as explain why they are required (primarily, because the naive Monte Carlo estimator of the policy gradient has high variance).

The reinforcement learning problem

A Markov decision process is given by the graphical model below.

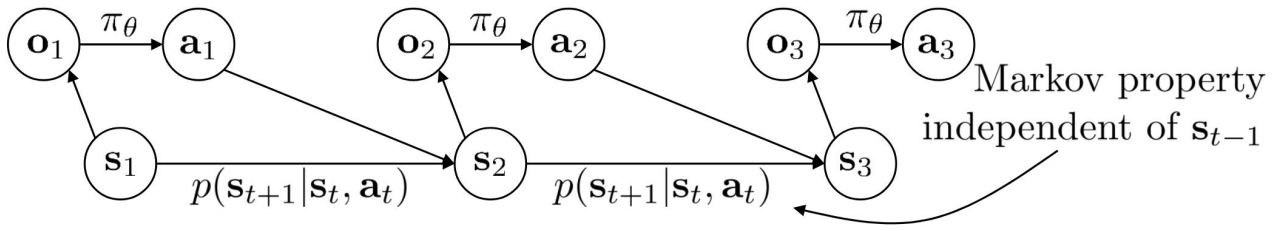


Figure 1: Markov decision process as a graphical model. From Sergey Levine's slides on deep RL (lecture *Introduction to deep RL*

The framework for reinforcement learning is a Markov decision process equipped with a scalar field $R : S \times A \to \mathbb{R}$. The goal is to select a policy model $\pi_\theta(a_t|o_t)$ which maximizes the reinforcement learning objective:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)}\big[r(\tau)\big] \tag{1}$$

where $\tau = (s_0, o_0, a_0, s_1 ..., s_T)$ is a sample trajectory (also called a trace). The intuition here is that this framework resembles the process of sequential decision making in a very general way. The reward function $r(\tau)$ is $r(\tau) = \sum_{t=0}^{T} r(a_t, s_t)$ and the distribution $\pi_\theta(\tau)$ for the entire graphical model is given by

$$\pi_\theta(\tau) = p(s_o) \prod_{t=1}^{T} \underbrace{\pi_\theta(a_t|o_t)}_{\text{policy}} \quad \underbrace{p(o_t|s_t)}_{\text{emission model}} \quad \underbrace{p(s_{t+1}|s_t, a_t)}_{\text{dynamics}} \tag{2}$$

The reinforcement learning objective is the expected reward accrued over a trajectory τ drawn from the graphical model distribution. Note that the agent may not know the exact dynamics (although modeling the dynamics is the purview of model-based RL) or the exact state s_t. The agent only recieves observations o_t (which may alias the state) and must select an action according to a policy model $\pi_\theta(a_t|o_t)$. To get a good agent, the goal is to identify a policy from a class of policies parametrized by θ which maximizes the reinforcement learning objective. The reward function is assumed to be provided by the environment (or the programmer, in most cases).

Vanilla policy gradient

What's the naive thing to do? Let's take the gradient of the reinforcement learning objective.

$$\nabla_\theta J(\theta) = \nabla_\theta \int r(\tau)\pi_\theta(\tau)d\tau = \int r_\tau \nabla_\theta \pi_\theta(\tau) d\tau \tag{3}$$

Using the following identity

$$\nabla_x \log f(x) = \frac{1}{f(x)} \nabla_x f(x) \tag{4}$$

Produces the following expectation

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)}\big[r(\tau)\nabla_\theta \log \pi_\theta(\tau)\big] \tag{5}$$

which is really nice because we can estimate the expectation (which is the exact objective gradient with respect to the policy parameters θ) by Monte Carlo sampling of trajectories (i.e. running the agent with the policy in the environment). First, let's simplify the objective gradient.

In $\pi_\theta(\tau)$, note that applying the log produces a sum

$$\log \pi_\theta(\tau) = \log p(s_0) + \sum_{t=1}^{T} \big[\log \pi_\theta(a_t|o_t) + \log p(o_t|s_t) + \log p(s_{t+1}|a_t, s_t) \big] \tag{6}$$

In the gradient $\nabla_\theta \pi_\theta(\tau)$, the only term which is dependent on θ is the policy $\pi_\theta(a_t|o_t)$. Fully expanding in the expectation, this produces

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\tau)} \big[\big(\sum_{t=0}^{T} r(a_t, s_t) \big) \big(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|o_t) \big) \big] \tag{7}$$

and, taking Monte Carlo estimates, produces

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=0}^{N} \big[\big(\sum_{t=0}^{T} r(a_{t,i}, s_{t,i}) \big) \big(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_{t,i}|o_{t,i}) \big) \big] \tag{8}$$

With the gradient in hand, we can use any gradient-based optimizer we want

$$\theta \to \theta + g(\nabla_\theta J(\theta)) \tag{9}$$

This set of computations produces the following algorithm (called *REINFORCE* or *vanilla policy gradient*) for training up good policies

Algorithm 1: Vanilla policy gradient

Result: A trained policy $\pi_\theta(a_t|o_t)$
Input : MDP with a policy $\pi_\theta(a_t|o_t)$.
Number k of training epochs.
Gradient optimizer (e.g. ADAM).

for $e \leftarrow 1$ **to** k **do**
  Initialize trajectory buffer to store sample experience trajectories.
  for $i \leftarrow 1$ **to** N **do**
    Collect trajectory samples τ_i by sampling actions from current policy $\pi_\theta(a_t|o_t)$
  end
  Compute the loss with the naive Monte Carlo estimate $\frac{1}{N} \sum_{i=0}^{N} \log \pi_\theta(\tau_i) r(\tau_i)$ and backpropagate.
end

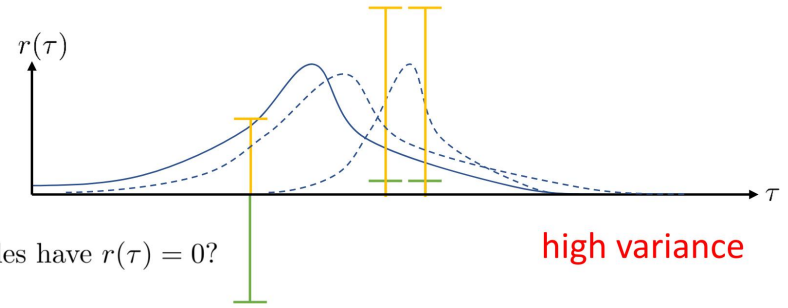
Baselines

The problem with this algorithm is the variance in the estimator. This is beautifully illustrated in the following set of slides by Sergey Levine, which I expand on with notes and calculations taken from Peter and Schaal, 2008 and Grahlwohl et al, 2018.

Basically, what this plot illustrates is that if you adjust all the $r(\tau)$ by constant $r(\tau) \to r(\tau) + b$, this will adjust how the policy distribution is shifted by the gradient step. In particular, the loss tries to push the distribution up where the reward is high but if you choose a "naive" constant to add to the reward, then you'll mess up the distribution shift. The issue is the high variance in the estimator. This is easy to see by assuming that $r(a_t, s_t) = c$ for some constant c. The variance of the estimator grows cubically in T

$$Var_{\tau \sim \pi_\theta(\tau)} \big[r(\tau) \nabla_\theta \log \pi_\theta(\tau) \big] = c^2 T^2 \sum_{t=0}^{T} Var \big[\nabla_\theta \log \pi_\theta(a_t|o_t) \big] \tag{10}$$

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_\theta \log \pi_\theta(\tau) r(\tau)$$



even worse: what if the two "good" samples have $r(\tau) = 0$?

Figure 2: The y-axis is overloaded - it also corresponds to the policy distribution. The goal is to shift the policy distribution to emphasize high-reward trajectories.

Of course, if you choose $b = -c$ then the algorithm does the right thing - the expectation is 0 and the variance is also 0 which makes sense. If every state gives the same reward, then you shouldn't shift the distribution at all, you're already at an optimal policy.

The above example gives rise to an interesting proposal: let's reduce the variance by shifting by a smart b. We call such a smart b a *baseline*. First, we will prove that the expectation is unaffected by the constant shift. Express the gradient with the inclusion of the constant b

$$\nabla_\theta J(\theta) = \nabla_\theta \int \pi_\theta(\tau)(r(\tau) + b) d\tau \tag{11}$$

$$= \nabla_\theta \int \pi_\theta(\tau) r(\tau) d\tau + b \underbrace{\int \pi_\theta(\tau) d\tau}_{1} \tag{12}$$

$$= \nabla_\theta \int \pi_\theta(\tau) r(\tau) d\tau \tag{13}$$

This proves that the expectation is unaffected

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[r(\tau)\nabla_\theta \log \pi_\theta(\tau)\right] = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[(r(\tau) + b)\nabla_\theta \log \pi_\theta(\tau)\right] \tag{14}$$

The variance, however, is affected. Let $q(\theta) = \nabla_\theta \log \pi_\theta(\tau)$. The variance is

$$Var_{\tau\sim\pi_\theta(\tau)}\left[(r(\tau) + b)q(\theta)\right] = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[(r(\tau) + b)^2 q(\theta)^2\right] - \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[(r(\tau) + b)q(\theta)\right]^2 \tag{15}$$

$$= \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[(2br(\tau) + b^2)q(\theta)^2\right] \tag{16}$$

and we can solve for the b which minimizes the variance.

$$\frac{d}{db} Var_{\tau\sim\pi_\theta(\tau)}\left[(r(\tau) + b)q(\theta)\right] = \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[(2r(\tau) + 2b)q(\theta)^2\right] = 0 \tag{17}$$

$$\implies \underbrace{b = \frac{-\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[r(\tau)q(\theta)^2\right]}{\mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[q(\theta)^2\right]}}_{\text{optimal } b} \tag{18}$$

One thing to note here: the gradient is vectorial, so we really have a vector of expectations

$$\nabla_\theta J(\theta) = \begin{bmatrix} \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[r(\tau)\nabla_{\theta_1} \log \pi_\theta(\tau)\right] \\ \mathbb{E}_{\tau\sim\pi_\theta(\tau)}\left[r(\tau)\nabla_{\theta_2} \log \pi_\theta(\tau)\right] \\ \vdots \end{bmatrix} \tag{19}$$

and each component of this vector can have a different constant b_i. The optimal b above is thus also vectorial. In practice, a popular, simple to implement selection for b is

$$b = \frac{1}{N}\sum_{i=0}^{N} r(\tau_i) \tag{20}$$

which is the average reward over the all sampled trajectories. The intuition for this is simple - trajectories which have total reward greater than average should contribute to pushing the distribution up. Note that Grahlwohl et al, 2018 illustrates more advanced techniques for more general stochastic gradient estimators.

This introduces our first theme on vanilla policy gradient.

Algorithm 2: Vanilla policy gradient with baselines

Result: A trained policy $\pi_\theta(a_t|o_t)$
Input : MDP with a policy $\pi_\theta(a_t|o_t)$.
Number k of training epochs.
Gradient optimizer (e.g. ADAM).

for $e \leftarrow 1$ **to** k **do**
  Initialize trajectory buffer to store sample experience trajectories.
  for $i \leftarrow 1$ **to** N **do**
    Collect trajectory samples τ_i *by sampling actions from current policy* $\pi_\theta(a_t|o_t)$
  end
  Compute the loss with the Monte Carlo estimate $\frac{1}{N}\sum_{i=0}^{N} \log \pi_\theta(\tau_i)(r(\tau_i) - b)$ *with* $b = \sum_{i=0}^{N} r(\tau_i)$ *and backpropagate.*
end

Rewards-to-go