Deep Reinforcement Learning

This document is a combination of notes from CS 234 taught by Prof. Emma Brunskill at Stanford University, CS 224R taught by Prof. Chelsea Finn at Stanford University, CS 285 taught by Prof. Sergey Levine at UC Berkeley as well as reading notes from various papers. There may be various errors throughout, both minor and major.

Contents

1 Policy Gradient Methods

In policy gradient methods, we directly parametrise the policy to be $\pi_\theta(a|s) = \mathbb{P}_\theta(a|s)$. Our goal is then to find π_θ that maximises returns. We may still learn a value function in order to help learn our parametrized policy, but the value function will not be required for our action selection. Methods that incorporate both a parametrized policy and a parametrized, learned value function are often called *actor-critic methods*. For policy-based methods, we have no value function but only a learned policy. For actor-critic methods, we have both a learned value function and a learned policy.

Policy gradient methods are useful for generating stochastic policies but they are often hard to evaluate and can be high variance. We often use Gaussian policies (especially for continuous action spaces) which is parametrised as $a \sim \mathcal{N}(\mu_\theta(s), \sigma^2)$.

Some notes:

- To ensure exploration, in practice, we *require* that the parametrized policy never becomes deterministic i.e $\pi_\theta(a|s) \in (0, 1), \forall s, a, \theta$.
- The choice of policy parameterization is sometimes a good way of injecting prior knowledge about the desired form of the policy into the reinforcement learning system.

Notation:

- $\mathbb{P}(s_0 \to s, k, \pi_\theta)$ is the probability of reaching state s from s_0 in k time-steps under policy π_θ.
- $p^\pi(s) = \sum_{k=0}^{\infty} \gamma^k \mathbb{P}(s_0 \to s, k, \pi_\theta)$ is the improper discounted state distribution.
- Let the distribution over all states induced by π_θ be:

$$d^{\pi_\theta}(s) := (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(s_0 \to s, t, \pi_\theta).$$

  The factor $(1 - \gamma)$ is a normalization constant and this distribution simply discounts states visited later in time.

Note:

- The first few algorithms are going to be ***on-poligy*** : REINFORCE, REINFORCE with baseline and actor-critic. These algorithms require sampling trajectories with the parameters at each iteration and *then* making updates to the policy.
- Next, we will look at ***off-policy*** algorithms: off-policy actor-critic

1.1 Vanilla Policy Gradient Methods

In this note, we will cover episodic tasks. Let $V(\theta) = V(s_0, \theta)$, where s_0 is the starting state (which we consider to be fixed for simplicity) and the dependency on θ specifies that the value depends on the policy π_θ. Policy π_θ can be any distribution. For example, we may use a Gaussian policy and write $\pi_\theta(a|s) = \mathcal{N}(f^\theta_{\text{neural network}}(s), \Sigma)$

Now, we want to maximize $V(\theta)$:

$$\begin{aligned} V(\theta) &= V(s_0, \theta) \\ &= \sum_a \pi_\theta(a|s_0) Q(s_0, a; \theta) \\ &= \sum_\tau P_\theta(\tau) R(\tau) \end{aligned}$$

where $Q(s_0, a; \theta)$ is the expected reward attained under the policy π_θ after taking action a from state s_0. $P_\theta(\tau)$ is the probability of trajectory $\tau = (s_0, a_0, r_0, s_1, ..., s_{T-1}, a_{T-1}, r_{T-1}, s_T)$ under policy π_θ and $R(\tau)$ is the total reward from trajectory τ.

Lemma 1. $\nabla_\theta V(\theta) = \sum_\tau R(\tau) P_\theta(\tau) \nabla_\theta \log(P_\theta(\tau))$

Proof.

$$\begin{aligned} \nabla_\theta V(\theta) &= \nabla_\theta \sum_\tau P_\theta(\tau) R(\tau) \\ &= \sum_\tau R(\tau) \nabla_\theta P_\theta(\tau) \\ &= \sum_\tau R(\tau) \frac{P_\theta(\tau)}{P_\theta(\tau)} \nabla_\theta P_\theta(\tau) \\ &= \sum_\tau R(\tau) \frac{P_\theta(\tau)}{P_\theta(\tau)} \nabla_\theta P_\theta(\tau) \\ &= \sum_\tau R(\tau) P_\theta(\tau) \nabla_\theta \log(P_\theta(\tau)) \end{aligned}$$

□

Now, using Lemma 1, we can get an approximation for our update rule:

$$\nabla_\theta V(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} R(\tau^{(i)}) \nabla_\theta \log(P_\theta(\tau^{(i)})).$$

However, we still don't know how to compute the gradient of the log-probability. For that, we need the following lemme:

Lemma 2. $\nabla_\theta \log(P_\theta(\tau^{(i)})) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)$

Proof.

$$\begin{aligned}
\nabla_\theta \log(P_\theta(\tau^{(i)}) &= \nabla_\theta \log(\mu(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t|s_t) P(s_{t+1}|s_t, a_t)) \\
&= \nabla_\theta \log(\mu(s_0)) + \sum_{t=0}^{T-1} \log \pi_\theta(a_t|s_t) + \log P(s_{t+1}|s_t, a_t) \\
&= \sum_{t=0}^{T-1} \nabla_\theta \log(\pi_\theta(a_t|s_t))
\end{aligned}$$

□

With Lemma 2, we get the following update rule:

$$\nabla_\theta V(\theta) = \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \left(\sum_{t=1}^{T} r(s_t, a_t) \right) \right].$$

We generalize this approach through the policy gradient theorem [1]. For an episodic task, let $\eta(s)$ be the number of time-steps spent, on average, in state s within a single episode. If $\mu_0(s)$ is the initial state distribution, then $\eta(s) = \mu_0(s) + \sum_{s'} \eta(s') \sum_a \gamma \cdot \pi_\theta(a|s') p(s|s', a)$. To turn this into a probability distribution, which we call the on-policy distribution i.e the fraction of time-steps spent in a state s, we get $\mu(s) = \frac{\eta(s)}{\sum_{s'} \eta(s')}$

Theorem 3. Suppose, our task is episodic. Furthermore, suppose, our start state is s_0 for all trajectories. Then, $\nabla_\theta V(\theta) \propto \sum_s \mu(s) \sum_a Q_{\pi_\theta}(s, a) \nabla_\theta \pi_\theta(a|s)$

or more simply:

$$\begin{aligned}
\nabla_\theta V(\theta) &= \sum_s p^\pi(s) \sum_a \nabla_\theta \pi_\theta(a|s) q_{\pi_\theta}(s, a) \\
&= \mathbb{E}_{s \sim p^{\pi_\theta}(s), a \sim \pi_\theta(a|s)} \left[\nabla_\theta \log \pi_\theta(a|s) q_{\pi_\theta}(s, a) \right]
\end{aligned}$$

where $p^\pi(s) = \sum_{k=0}^{\infty} \gamma^k \mathbb{P}(s_0 \to s, k, \pi_\theta)$ and $\mathbb{P}(s_0 \to s, k, \pi_\theta)$ is the probability of reaching state s from s_0 in k time-steps under policy π_θ.

Proof.

$$
\begin{aligned}
\nabla_\theta V(\theta) &= \nabla_\theta \left(\sum_a \pi_\theta(a|s) q_{\pi_\theta}(s,a) \right) \\
&= \sum_a \left(\nabla_\theta \pi_\theta(a|s) \right) q_{\pi_\theta}(s,a) + \pi_\theta(a|s) \nabla_\theta q_{\pi_\theta}(a,s) \\
&= \sum_a \left(\nabla_\theta \pi_\theta(a|s) \right) q_{\pi_\theta}(s,a) + \pi_\theta(a|s) \nabla_\theta \left(\sum_{s',r} p(s',r|s,a) \left(r + \gamma V_{\pi_\theta}(s') \right) \right) \\
&= \sum_a \left(\nabla_\theta \pi_\theta(a|s) \right) q_{\pi_\theta}(s,a) + \pi_\theta(a|s) \left(\sum_{s',r} p(s',r|s,a) \gamma \cdot \nabla_\theta \left(V_{\pi_\theta}(s') \right) \right) \\
&= \sum_a \nabla_\theta \pi_\theta(a|s) q_{\pi_\theta}(s,a) + \\
&\pi_\theta(a|s) \sum_{s'} p(s'|s,a) \cdot \gamma \cdot \left(\sum_{a'} \left(\nabla_\theta \pi_\theta(a'|s') q_{\pi_\theta}(s',a') \right) + \pi_\theta(a'|s') \sum_{s''} P(s''|s',a') \gamma \cdot \nabla_\theta V_{\pi_\theta}(s'') \right) \\
&= \sum_{x \in S} \sum_{k=0}^{\infty} \mathbb{P}(s_0 \to x, k, \pi_\theta) \gamma^k \sum_a \nabla_\theta \pi_\theta(a|x) q_{\pi_\theta}(x,a)
\end{aligned}
$$

where $\mathbb{P}(s \to x, k, \pi_\theta)$ is the probability of reaching state x from x in k steps under policy π_θ.

$$
\begin{aligned}
\nabla_\theta V(\theta) &= \sum_s \sum_{k=0}^{\infty} \mathbb{P}(s_0 \to s, k, \pi_\theta) \gamma^k \sum_a \nabla_\theta \pi_\theta(a|s) q_{\pi_\theta}(s,a) \\
&= \sum_s \eta(s) \sum_a \nabla_\theta \pi_\theta(a|s) q_{\pi_\theta}(s,a) \\
&= \sum_{s'} \eta(s') \sum_s \frac{\eta(s)}{\sum_{s'} \eta(s')} \sum_a \nabla_\theta \pi_\theta(a|s) q_{\pi_\theta}(s,a) \\
&= \sum_{s'} \eta(s') \sum_s \mu(s) \sum_a \nabla_\theta \pi_\theta(a|s) q_{\pi_\theta}(s,a) \\
&\propto \sum_s \mu(s) \sum_a \nabla_\theta \pi_\theta(a|s) q_{\pi_\theta}(s,a)
\end{aligned}
$$

or more simply

$$\begin{aligned}\nabla_\theta V(\theta) &= \sum_s \sum_{k=0}^{\infty} \mathbb{P}(s_0 \to s, k, \pi_\theta)\gamma^k \sum_a \nabla_\theta \pi_\theta(a|s) q_{\pi_\theta}(s,a) \\ &= \sum_s p^\pi(s) \sum_a \nabla_\theta \pi_\theta(a|s) q_{\pi_\theta}(s,a)\end{aligned}$$

where $p^\pi(s) = \sum_{k=0}^{\infty} \gamma^k \mathbb{P}(s_0 \to s, k, \pi_\theta)$ □

1.2 REINFORCE

So far, we have:

$$\nabla_\theta V(\theta) = \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \left(\sum_{t'=1}^{T} r(s_{t'}, a_{t'}) \right) \right].$$

REINFORCE is an on-policy algorithm that approximates this by sampling multiple trajectories rolled out by policy π_θ and then letting:

$$\nabla_\theta V(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \right) \left(\sum_{t'=1}^{T} r(s_{i,t'}, a_{i,t'}) \right)$$

Algorithm: REINFORCE

Initialize policy parameters θ
for iteration = 1, 2, ... **do**
  Collect a set of trajectories $\{\tau^i\}$ by running the policy $\pi_\theta(a_t|s_t)$
  for each trajectory τ^i **do**
    for each timestep t in τ^i **do**
      Compute return: $G_t^i = \sum_{t'=t}^{T} r(s_{t'}^i, a_{t'}^i)$
  Update the policy parameters:
    $\nabla_\theta V(\theta) \approx \sum_i \sum_t \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) G_t^i$
    $\theta \leftarrow \theta + \alpha \nabla_\theta V(\theta)$

This algorithm is, however, very noisy since our trajectory samples are often quite noisy. In other words G_t^i is a noise estimate and we generally require a large number of samples to get a good estimate of the value of the policy.

To solve this, we use several modifications.

1.3 REINFORCE using causaility

Our current approach is

$$\nabla_\theta V(\theta) \approx \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\right)\left(\sum_{t'=1}^{T} r(s_{i,t'}, a_{i,t'})\right).$$

Now, notice that the policy at time t cannot affect rewards at time $t' < t$. With this in mind, our first modification is as follows:

$$\nabla_\theta V(\theta) \approx \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\right)\left(\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})\right).$$

In other words, we are using the "reward to go" from time t onwards.

1.4 REINFORCE with baseline

We introduce a baseline to further reduce variance:

$$\nabla_\theta V(\theta) \approx \frac{1}{m}\sum_{i=1}^{m}\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\right)\left(\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}) - b(s_{i,t'})\right).$$

where b is any arbitrary function as long as it does not depend on a_t. We usually select $b(s) = \mathbb{E}_{\tau\sim\pi_\theta}[r(\tau) \mid s_0 = s]$. We could also use a learned state-action value function parametrized by ϕ,

$$v_\phi(s_t) =: b(s_t).$$

Note that in the policy gradient theorem, this amounts to modifying the update to be :

$$\nabla_\theta V(\theta) \propto \sum_s \mu(s)\sum_a (q_{\pi_\theta(s,a)} - b(s_t))\nabla_\theta \pi_\theta(a|s)$$

1.5 On-Policy Actor-Critic Methods

Actor-Critic methods replace the reward to go $\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})$ with $Q_\theta(s_{i,t'}, a_{i,t'})$. As for the baseline, we use $V_\theta(s_{i,t'})$. Then, we can replace $Q_\theta(s_{i,t'}, a_{i,t'}) - V_\theta(s_{i,t'})$ with the advantage

function $A^{\pi_\theta}(s_{i,t'}, a_{i,t'})$. The better the estimate of this advantage, the lower the variance.

Actor-critic methods use a learned value function $V_\phi^{\pi_\theta}(s_t)$ to approximate $V_\theta(s_t)$. This is trained using supervised regression. Suppose our training set (using the rollouts by policy π_θ) is of the form $\{(s_{i,t}, \sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}))\}$. Then, we train via minimizing

$$\frac{1}{2}\sum_i \sum_t \left|\left| V_\phi^{\pi_\theta}(s_{i,t}) - \left(\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})\right)\right|\right|^2.$$

Alternatively, we can also train this by using a boostrapped estimate of the target:

$$\frac{1}{2}\sum_i \sum_t \left|\left| V_\phi^{\pi_\theta}(s_{i,t}) - \left(r(s_{i,t}, a_{i,t}) + V_\phi^{\pi_\theta}(s_{i,t+1})\right)\right|\right|^2.$$

1.6 Off-Policy Actor-Critic

We want to estimate the policy gradient *off-policy* from trajectories sampled from a distinct behaviour policy $\beta(a|s) \neq \pi_\theta(a|s)$. In this setting, the performance objective is the value function of the target policy, π_θ, averaged over the state distribution of the behaviour policy $\beta(a|s)$:

$$J_\beta(\pi_\theta) = \int_{\mathcal{S}} p^\beta(s) V^{\pi_\theta}(s) = \int_{\mathcal{S}} \int_{\mathcal{A}} p^\beta(s) \pi_\theta(a|s) Q^{\pi_\theta}(s,a) da ds.$$

The policy gradient becomes (after approximating by dropping $\nabla_\theta Q^{\pi_\theta}(s,a)$):

$$\nabla_\theta J_\beta(\pi_\theta) = \mathbb{E}_{s\sim p^\beta, a\sim\beta}\left[\frac{\pi_\theta(a|s)}{\beta(a|s)} \nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a)\right].$$

OffPAC (Off-Policy Actor Critic) algorithm uses the behavior policy $\beta(a|s)$ to generate trajectories. A critic estimates $V^\phi(s) \approx V^{\pi_\theta}(s)$ off-policy by gradient temporal-difference learning. Instead of the unknown $Q^{\pi_\theta}(s,a)$, the temporal-difference error $\delta_t = r_{t+1} + \gamma V^\phi(s_{t+1}) - V^\phi(s_t)$ is used.

1.7 Soft Actor-Critic

We parametrize three function: $V_\psi(s_t)$, $Q_\phi(s_t, a_t)$ and $\pi_\theta(a_t|s_t)$. Although technically we do not need to have separate function approximators for V and Q, doing this improves training stability.

The soft value function is trained to minimize the following loss:

$$J_V(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\frac{1}{2} \left(V_\psi(s_t) - \mathbb{E}_{a_t \sim \pi_\theta(\cdot|s_t)} \left[Q_\phi(s_t, a_t) - \log \pi_\theta(a_t|s_t) \right] \right)^2 \right].$$

Here $\mathcal{D}$ is the replay buffer. The gradient is computed as

$$\nabla_\psi J_V(\psi) = \nabla_\psi V_\psi(s_t)(V_\psi(s_t) - Q_\phi(s_t, a_t) + \log \pi_\theta(a_t|s_t))$$

where the actions are sampled $a_t \sim \pi_\theta(\cdot|s_t)$.

The soft Q-function is trained to minimize the soft Bellman residual:

$$J_Q(\phi) = \mathbb{E}_{s_t, a_t \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\phi(s_t, a_t) - \hat{Q}(s_t, a_t) \right)^2 \right]$$

where $\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p} \left[V_{\bar{\psi}}(s_{t+1}) \right]$. Here $V_{\bar{\psi}}$ is our target network where $\bar{\psi}$ is an exponentially moving average of the value network weights. The gradient of this is:

$$\nabla_\phi (J_Q(\phi)) = \nabla_\phi Q_\phi(s_t, a_t)(Q_\phi(s_t, a_t) - r(s_t, a_t) - \gamma V_{\bar{\psi}}(s_{t+1})).$$

Now, to parametrize the policy $\pi_\theta(a_t|s_t)$, represent the action as $a_t = f_\theta(\epsilon_t : s_t)$ where ϵ_t is an input random noise vector sampled from a fixed distribution (like spherical Gaussian). Then, the policy is optimized by minimizing:

$$J_\pi(\theta) = \mathbb{E}_{s_t \sim \mathcal{D}, \epsilon_t \sim \mathcal{S}} \left[\log \pi_\theta(f_\theta(\epsilon_t; s_t) \mid s_t) - Q_\phi(s_t, f_\theta(\epsilon_t; s_t)) \right].$$

The unbiased gradient of this is

$$\nabla_\theta J_\pi(\theta) = \nabla_\theta \pi_\theta(a_t|s_t) + (\nabla_{a_t} \log \pi_\theta(a_t|s_t) - \nabla_{a_t} Q(s_t, a_t)) \nabla_\theta f_\theta(\epsilon_t; s_t).$$

1.8 Notes on Continuing Problems

Suppose, our task is no longer episodic. In this case,

$$V(\theta) = \lim_{h \to \infty} \frac{1}{h} \sum_{t=1}^{h} \mathbb{E}_{\pi_\theta}[R_t] = \lim_{t \to \infty} \mathbb{E}_{s,a \sim \pi_\theta}[R_t]$$

where $\mu(s) := \lim_{t\to\infty} \mathbb{P}(s_t = s|a_{0:t} \sim \pi_\theta)$. In the continuing case, we define

$$G_t := R_{t+1} - V(\theta) + R_{t+2} - V(\theta) + \cdots$$

and using this, we define, $V_{\pi_\theta}(s) := \mathbb{E}_{\pi_\theta}[G_t|s_t = s]$, $q_{\pi_\theta}(s,a) = \mathbb{E}_{\pi_\theta}[G_t|s_t = s, a_t = a]$.

We prove the following theorem, which is analogous to the episodic setting:

Theorem 4. $\nabla_\theta V(\theta) \propto \sum_s \mu(s) \sum_a q_{\pi_\theta}(s,a) \nabla_\theta \pi_\theta(a|s)$

Proof.

$$\begin{aligned}
\nabla_\theta V_{\pi_\theta}(s) &= \nabla \left[\sum_a \pi_\theta(a|s) q_{\pi_\theta}(s,a) \right] \\
&= \sum_a \left[\nabla_\theta \pi_\theta(a|s) q_{\pi_\theta}(s,a) + \pi_\theta(a|s) \nabla_\theta q_{\pi_\theta}(s,a) \right] \\
&= \sum_a \left[\nabla_\theta \pi_\theta(a|s) q_{\pi_\theta}(s,a) + \pi_\theta(a|s) \nabla_\theta \sum_{s',r} P(s',r|s,a)(r - V(\theta) + V_{\pi_\theta}(s')) \right] \\
&= \sum_a \left[\nabla_\theta \pi_\theta(a|s) q_{\pi_\theta}(s,a) + \pi_\theta(a|s) [-\nabla_\theta V(\theta) + \sum_{s'} P(s'|s,a) \nabla_\theta V_{\pi_\theta}(s')] \right] \\
\nabla_\theta V(\theta) &= \sum_a \left[\nabla_\theta \pi_\theta(a|s) q_{\pi_\theta}(s,a) + \pi_\theta(a|s) \sum_{s'} P(s'|s,a) \nabla_\theta V_{\pi_\theta}(s') \right] - \nabla_\theta V_{\pi_\theta}(s)
\end{aligned}$$

Now, the left hand side is independent of s, so we can do a weighted sum over s (weighted

by $\mu(s)$:

$$\sum_s \mu(s)\nabla_\theta V(\theta) = \sum_s \mu(s)\left(\sum_a \left[\nabla_\theta \pi_\theta(a|s) q_{\pi_\theta}(s,a) + \pi_\theta(a|s)\sum_{s'} P(s'|s,a)\nabla_\theta V_{\pi_\theta}(s')\right] - \nabla_\theta V_{\pi_\theta}(s)\right)$$

$$\begin{aligned}
\nabla_\theta V(\theta) &= \sum_s \mu(s)\sum_a \nabla_\theta \pi_\theta(a|s) q_{\pi_\theta}(s,a) + \\
&\sum_s \mu(s)\sum_a \pi_\theta(a|s)\sum_{s'} P(s'|s,a)\nabla_\theta V_{\pi_\theta}(s') - \sum_s \mu(s)\nabla_\theta V_{\pi_\theta}(s) \\
&= \sum_s \mu(s)\sum_a \nabla_\theta \pi_\theta(a|s) q_{\pi_\theta}(s,a) + \\
&\sum_{s'}\sum_s \mu(s)\sum_a \pi_\theta(a|s) P(s'|s,a)\nabla_\theta V_{\pi_\theta}(s') - \sum_s \mu(s)\nabla_\theta V_{\pi_\theta}(s) \\
&= \sum_s \mu(s)\sum_a \nabla_\theta \pi_\theta(a|s) q_{\pi_\theta}(s,a) + \sum_{s'} \mu(s')\nabla_\theta V_{\pi_\theta}(s') - \sum_s \mu(s)\nabla_\theta V_{\pi_\theta}(s) \\
&= \sum_s \mu(s)\sum_a q_{\pi_\theta}(s,a)\nabla_\theta \pi_\theta(a|s)
\end{aligned}$$

where in the second last line we used the fact that $\mu(s') = \sum_s \mu(s)\sum_a \pi_\theta(a|s)P(s'|s,a)$. $\square$

I find this to be a very interesting result - the fact that the same result holds for both episoding and non-episodic tasks after the simple, understandable re-definition of G_t. In fact, the new way of seeing G_t is akin to the introduction of baselines in methods like REINFORCE.

Now, we move on to the more advanced policy gradient methods.

1.9 Performance Difference Lemma

We want to prove a few results that are building blocks for trust regional policy optimisation.

Firstly, note that the probability of sampling any particular trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, ...)$ is $P_\theta(\tau) = \prod_{i=0}^{\infty} \pi_\theta(a_i|s_i)P(s_{i+1}|s_i,a_i)$. The probability of sampling a particular trajectory τ such that $s_t = s$ is $P_\theta(s_t = s) = \sum_{a_0}\sum_{s_1} ... \sum_{s_{t-1}}\sum_{a_{t-1}} \left(\prod_{i=1}^{T-1} \pi_\theta(a_i|s_i)P(s_{i+1}|s_i,a_i)\right)$.

Let the distribution over all states induced by π_θ be:

$$d^{\pi_\theta}(s) := (1-\gamma)\sum_{t=0}^{\infty} \gamma^t P_\theta(s_t = s).$$

The factor $(1-\gamma)$ is a normalization constant and this distribution simply discounts states visited later in time.

Lemma 5. $\mathbb{E}_{\tau\sim P_\theta}\left[\sum_{t=0}^{\infty}\gamma^t f(s_t,a_t)\right] = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\pi_\theta}}\left[\mathbb{E}_{a\sim\pi_\theta(\cdot|s)}\left[f(s,a)\right]\right]$

Proof.

$$
\begin{aligned}
\mathbb{E}_{\tau\sim P_\theta}\left[\sum_{t=0}^{\infty}\gamma^t f(s_t,a_t)\right] &= \sum_{\tau} P_\theta(\tau)\sum_{t=0}^{\infty}\gamma^t f(s_t,a_t) \\
&= \sum_{\tau}\prod_{i=0}^{\infty}\pi_\theta(a_i|s_i)P(s_{i+1}|s_i,a_i)\sum_{t=0}^{\infty}\gamma^t f(s_t,a_t) \\
&= \sum_{a_0}\sum_{s_1}\sum_{a_1}\cdots\prod_{i=0}^{\infty}\pi(a_i|s_i)P(s_{i+1}|s_i,a_i)\sum_{t=0}^{\infty}\gamma^t f(s_t,a_t) \\
&= \sum_{a_0}\pi_\theta(a_0|s_0)f(s_0,a_0) + \gamma\sum_{a_0}\sum_{s_1}\sum_{a_1}\pi_\theta(a_0|s_0)P(s_1|s_0,a_0)\pi_\theta(a_1|s_1)f(s_1,a_1)+\cdots \\
&= \sum_{t=0}^{\infty}\gamma^t\sum_{a_0}\sum_{s_1}\sum_{a_1}\cdots\sum_{s_t}\sum_{a_t}\sum_{s_{t+1}}\prod_{i=0}^{t}\pi_\theta(a_i|s_i)P(s_{i+1}|s_i,a_i)f(s_t,a_t) \\
&= \sum_{t=0}^{\infty}\gamma^t\sum_{s_{t+1}}(\cdots)\,f(s_t,a_t) \\
&= \sum_{t=0}^{\infty}\gamma^t(\cdots)\,f(s_t,a_t) \\
&= \sum_{t=0}^{\infty}\gamma^t\sum_{s_t}\sum_{a_t}\left(\sum_{a_0}\cdots\sum_{s_{t-1}}\sum_{a_{t-1}}\prod_{i=0}^{t-1}\pi_\theta(a_i|s_i)P(s_{i+1}|s_i,a_i)\right)\pi_\theta(a_t|s_t)f(s_t,a_t) \\
&= \sum_{t=0}^{\infty}\gamma^t\sum_{s_t}\sum_{a_t}P_\theta(s_t)\pi_\theta(a_t|s_t)f(s_t,a_t) \\
&= \frac{1}{1-\gamma}(1-\gamma)\sum_{t=0}^{\infty}\gamma^t\sum_{s_t}P_\theta(s_t)\sum_{a_t}\pi_\theta(a_t|s_t)f(s_t,a_t) \\
&= \frac{1}{1-\gamma}(1-\gamma)\sum_{t=0}^{\infty}\gamma^t\sum_{s_t}P_\theta(s_t)\mathbb{E}_{a\sim\pi_\theta}\left[f(s_t,a)\right] \\
&= \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\pi_\theta}}\left[\mathbb{E}_{a\sim\pi_\theta}\left[f(s_t,a)\right]\right]
\end{aligned}
$$

□

Theorem 6. (Performance Difference Lemma)

$$V_\pi(s_0) - V_{\pi'}(s_0) = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^\pi}\left[\mathbb{E}_{a\sim\pi(s)}\left[A^{\pi'}(s,a)\right]\right]$$

where the advantage function is $A^\pi(s,a) := q^\pi(s,a) - V^\pi(s)$.

Proof.

$$\begin{aligned}
&V_\pi(s_0) - V_{\pi'}(s_0) \\
&= \mathbb{E}_{\tau\sim P^\pi}\left[\sum_{t=0}^{\infty}\gamma^t R(s_t,a_t)\right] + \mathbb{E}_{\tau\sim P^\pi}\left[\sum_{t=0}^{\infty}\gamma^{t+1}V_{\pi'}(s_{t+1})\right] - \mathbb{E}_{\tau\sim P^\pi}\left[\sum_{t=0}^{\infty}\gamma^{t+1}V_{\pi'}(s_{t+1})\right] - V_{\pi'}(s_0) \\
&= \mathbb{E}_{\tau\sim P^\pi}\left[\sum_{t=0}^{\infty}\gamma^t(R(s_t,a_t) + \gamma V_{\pi'}(s_{t+1})) - \sum_{t=0}^{\infty}\gamma^{t+1}V_{\pi'}(s_{t+1}) - V_{\pi'}(s_0)\right]
\end{aligned}$$

Now, we expand the first term:

$$\begin{aligned}
&\mathbb{E}_{\tau\sim P^\pi}\left[\sum_{t=0}^{\infty}\gamma^t\left(R(s_t,a_t) + \gamma V_{\pi'}(s_{t+1})\right)\right] \\
&= \sum_{t=0}^{\infty}\gamma^t\mathbb{E}_{\tau\sim P^\pi}\left[R(s_t,a_t) + \gamma V^{\pi'}(s_{t+1}|s_t,a_t)\right] \\
&= \sum_{t=0}^{\infty}\gamma^t\mathbb{E}_{s_t\sim P^\pi}\left[\mathbb{E}_{a_t\sim P^\pi}\left[\mathbb{E}_{s_{t+1}\sim P^\pi}\left[R(s,a) + \gamma V^{\pi'}(s_{t+1})|s=s_t,a=a_t\right]|a=a_t\right]|s=s_t\right] \\
&= \sum_{t=0}^{\infty}\gamma^t\mathbb{E}_{s_t,a_t\sim P^\pi}\left[R(s,a) + \gamma\sum_{s_{t+1}}P(s_{t+1}|s_t,a_t)V^{\pi'}(s_{t+1})|s=s_t,a=a_t\right] \\
&= \sum_{t=0}^{\infty}\gamma^t\mathbb{E}_{\tau\sim P^\pi}\left[Q^{\pi'}(s,a)|s=s_t,a=a_t\right] \\
&= \mathbb{E}_{\tau\sim P^\pi}\left[\sum_{t=0}^{\infty}\gamma^t Q^{\pi'}(s_t,a_t)\right]
\end{aligned}$$

Therefore, we get

$$V^\pi(s_0) - V^{\pi'}(s_0) = \mathbb{E}_{\tau\sim P^\pi}\left[\sum_{t=0}^{\infty}\gamma^t\left(Q^{\pi'}(s_t,a_t) - V^{\pi'}(s_t)\right)\right]$$

$$= \mathbb{E}_{\tau \sim P^{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi'}(s_t, a_t) \right]$$
$$= \frac{1}{1-\gamma} \mathbb{E}_{s \sim d^{\pi}} \left[\mathbb{E}_{a \sim \pi} \left[A^{\pi'}(s, a) \right] \right]$$

□

1.10 TRPO and PPO

There are two major issues with vanilla policy gradient methods. Firstly, it is difficult to optimize in the sense that it is difficult to find the right step size to use in gradient descent. The input data distribution is non-stationary - you sample trajectories using a learned policy, then you use those samples to update your policy, then you use this updated policy to sample new trajectories. However, if, at any point in time, you use a set of bad samples and therefore, your optimisation step is wrong, this could lead to performance collapse - with the bad samples, you take a "wrong step" to get a poor policy, with which you sample new trajectories which are also poor which you then use to optimise again. The second issue is that the algorithm is sample inefficient - with any particular set of sampled trajectories, we carry out one step of gradient descent and then throw those samples out. For future optimisation steps, we sample *new* trajectories. Although we have made some modifications to the basic vanilla PG algorithm like we found the actor-critic methods, they are still insufficient in completely curbing these issues.

We now derive the building blocks of Trust Region Policy Optimisation (TRPO): We already saw the performance difference lemme:

$$V_{\pi'} - V_{\pi} = \frac{1}{1-\gamma} \mathbb{E}_{s \sim d^{\pi'}} \left[\mathbb{E}_{a \sim \pi'} \left[A^{\pi}(s, a) \right] \right]$$

Now, suppose, our current policy is π. In our next step, we essentially want to maximize the difference between $V_{\pi'} - V_{\pi}$. Therefore,

$$\begin{aligned}
\arg\max_{\pi'} V_{\pi'} - V_{\pi} &= \arg\max_{\pi'} \frac{1}{1-\gamma} \mathbb{E}_{s \sim d^{\pi'}} \left[\mathbb{E}_{a \sim \pi'} \left[A^{\pi}(s, a) \right] \right] \\
&= \arg\max_{\pi'} \frac{1}{1-\gamma} \mathbb{E}_{s \sim d^{\pi'}} \left[\mathbb{E}_{a \sim \pi} \left[\frac{\pi'(a|s)}{\pi(a|s)} A^{\pi}(s, a) \right] \right] \\
&\approx \arg\max_{\pi'} \frac{1}{1-\gamma} \mathbb{E}_{s \sim d^{\pi}} \left[\mathbb{E}_{a \sim \pi} \left[\frac{\pi'(a|s)}{\pi(a|s)} A^{\pi}(s, a) \right] \right] \\
&=: \arg\max_{\pi'} \mathcal{L}_{\pi}(\pi')
\end{aligned}$$

where, in the second last line, we made the approximation $d^{\pi'} \approx d^{\pi}$. This approximation only holds true if

$$||V_{\pi'} - V_{\pi} - \mathcal{L}_{\pi}(\pi')|| \leq C\sqrt{\mathbb{E}_{s_t \sim \pi}\left[D_{KL}(\pi(\cdot|s_t)||\pi'(\cdot|s_t))\right]}$$

With this, TRPO maximises $\mathcal{L}_{\pi}(\pi')$ subject to $\mathbb{E}_{s\sim\pi}\left[D_{KL}(\pi(\cdot|s)||\pi'(\cdot|s))\right] \leq \delta$. Note that, in actual implementation, we use a learned approximation for the advantage function.

Also note that we get monotonic improvement since the KL divergence is zero when $\pi' = \pi$ whereas $\mathcal{L}_{\pi}(\pi) = 0$ too, therefore, the performance of π' is at least as good as π.

PPO slightly modifies this - instead of placing a harsh constraint in the optimization process (which requires conjugate gradient descent otherwise), instead PPO brings in 2 variants. The first is to maximize $\mathbb{E}_{s_t\sim d^{\pi}, a_t\sim\pi}\left[\frac{\pi'(a_t|s_t)}{\pi(a_t|s_t)}A^{\pi}(s_t,a_t) - \beta \cdot D_{KL}(\pi'(\cdot|s_t)||\pi(\cdot|s_t))\right]$. If the KL-divergence is too high, we adaptively increase β and if it is small, then we decrease β. The other variant is as follows - define $r_t(\theta) := \frac{\pi_{\theta'}(a_t|s_t)}{\pi_{\theta}(a_t|s_t)}$. Then, maximize $\mathbb{E}_{\tau\sim\pi_{\theta}}\left[\sum_{t=0}^{T-1}\left[\min(r_t(\theta)A^{\pi_{\theta}}(s_t,a_t), \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A^{\pi_{\theta}}(s_t,a_t))\right]\right]$.

In both variants, the algorithm uses an advantage estimator $\hat{A}^{\pi}(s_t, a_t)$. PPO uses Generalized Advantage Estimator (GAE).

First, we define N-step advantage estimators:

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

If we define

$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t),$$

then, these become:

$$\begin{aligned}
\hat{A}_t^{(1)} &= \delta_t^V, \\
\hat{A}_t^{(2)} &= \delta_t^V + \gamma\delta_{t+1}^V, \\
\hat{A}_t^{(k)} &= \sum_{l=0}^{k-1}\gamma^l\delta_{t+l}^V.
\end{aligned}$$

Thus, generally,

$$\hat{A}_t^{(k)} = \sum_{l=0}^{k-1}\gamma^l r_{t+l} + \gamma^k V(s_{t+k}) - V(s_t).$$

GAE is an exponentially-weighted average of k-step estimators:

$$\begin{aligned}
\hat{A}_t^{GAE(\gamma,\lambda)} &= (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + \dots\right) \\
&= (1-\lambda)\left(\delta_t^V + \lambda(\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V) + \dots\right) \\
&= (1-\lambda)\left(\delta_t^V(1+\lambda+\lambda^2+\dots) + \gamma\delta_{t+1}^V(\lambda+\lambda^2+\dots) + \dots\right) \\
&= (1-\lambda)\left(\delta_t^V\frac{1}{1-\lambda} + \gamma\delta_{t+1}^V\frac{\lambda}{1-\lambda} + \gamma^2\delta_{t+2}^V\frac{\lambda^2}{1-\lambda} + \dots\right) \\
&= \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l}^V.
\end{aligned}$$

PPO uses a truncated version of a GAE:

$$\hat{A}_t = \sum_{l=0}^{T-t-1}(\gamma\lambda)^l\delta_{t+l}^V$$

1.11 Deterministic Policy Gradient Methods (DPG)

Notation: We denote $r_t^\gamma = \sum_{k=t}^{\infty}\gamma^{k-t}r(s_k,a_k)$. Then, $V^\pi(s) = \mathbb{E}[r_1^\gamma|S_1 = s;\pi]$ and $Q^\pi(s,a) = \mathbb{E}[r_1^\gamma|S_1 = s, A_1 = a;\pi]$. The density at state s' after transitioning for t timesteps from state s is $p(s \to s', t, \pi)$. The improper, discounted state distribution is $p^\pi(s') := \int_{\mathcal{S}}\sum_{t=1}^{\infty}\gamma^{t-1}p_1(s)p(s\to s',t,\pi)ds$.

Suppose, $\mathcal{A} = \mathbb{R}^m$ and $\mathcal{S} = \mathbb{R}^d$.

Goal: Learn a policy which maximizes $J(\pi) := \mathbb{E}[r_1^\gamma|\pi]$. With our notation, this becomes:

$$J(\pi_\theta) = \int_{\mathcal{S}}p^\pi(s)\int_{\mathcal{A}}\pi_\theta(s,a)r(s,a)dads = \mathbb{E}_{s\sim p^\pi, a\sim\pi_\theta}[r(s,a)].$$

Intuition behind the deterministic policy gradient theorem:

Most model-free RL algorithms use policy evaluation and policy improvement together. Evaluation approximates $Q^\pi(s,a)$ and then improvement updates the policy, most often through $\pi^{k+1}(s) = \arg\max_a Q^{\pi^k}(s,a)$. However, in continuous action spaces, this is difficult since the $\arg\max_a$ requires a global maximisation at each step and our action space is very large (because it is continuous). Instead, the idea is to move the policy in the direction of the gradient of Q^{π^k} (instead of maximizing it altogether). Then

$$\theta^{k+1} = \theta^k + \alpha\mathbb{E}_{s\sim p^{\theta_k}}[\nabla_\theta Q^{\pi^{\theta^k}}(s,\pi^{\theta^k}(s)].$$

By chain rule this becomes

$$\theta^{k+1} = \theta^k + \alpha \mathbb{E}_{s\sim p^{\theta_k}}[\nabla_\theta \pi^\theta(s) \nabla_a Q^{\pi^{\theta^k}}(s,a)|_{a=\pi^\theta(s)}].$$

Notation: To distinguish between stochastic and deterministic policy, we will use $\mu_\theta(s)$ as our deterministic policy.

More formally, our performance objective is

$$J(\mu_\theta) = \int_{\mathcal{S}} p^{\mu_\theta}(s) r(s, \mu_\theta(s)) ds = \mathbb{E}_{s\sim p^{\mu_\theta}}\left[r(s, \mu_\theta(s))\right]$$

.

Then,

$$\nabla_\theta J(\mu_\theta) = \int_{\mathcal{S}} p^{\mu_\theta}(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu_\theta}(s,a)|_{a=\mu_\theta(s)} ds = \mathbb{E}_{s\sim p^{\mu_\theta}}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu_\theta}(s,a)|_{a=\mu_\theta(s)}\right] \tag{1}$$

On-policy algorithm: We have a critic that estimates the action-value function while the actor updates the policy by ascending the gradient of the action-value function (using equation 1). The critic, $Q^w(s,a)$ approximates $Q^\mu(s,a)$. The update rules are :

$$\begin{aligned} \delta_t &= r_t + \gamma Q^w(s_{t+1}, a_{t+1}) - Q^w(s_t, a_t) \\ w_{t+1} &= w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t) \\ \theta_{t+1} &= \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s_t) \nabla_a Q^w(s_t, a_t)|_{a=\mu_\theta(s)} \end{aligned}$$

Off-policy algorithm: Suppose we have trajectories generated by behavior policy $\pi(s,a)$. The new objective becomes:

$$J_\pi(\mu_\theta) = \int_{\mathcal{S}} p^\pi(s) V^\mu(s) ds = \int_{\mathcal{S}} p^\pi(s) Q^\mu(s, \mu_\theta(s)) ds$$

and the update rule becomes

$$\nabla_\theta J_\pi(\mu(\theta)) \approx \int_{\mathcal{S}} p^\pi(s) \nabla_\theta \mu_\theta(a|s) Q^\mu(s,a) ds = \mathbb{E}_{s\sim p^\pi}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}\right].$$

Now we can develop an actor-critic algorithm similar to the on-policy case:our critic is $Q^w(s,a)$:

$$
\begin{aligned}
\delta_t &= r_t + \gamma Q^w(s_{t+1}, \mu_\theta(s_{t+1})) - Q^w(s_t, a_t) \\
w_{t+1} &= w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t) \\
\theta_{t+1} &= \theta_t + \alpha_\theta \nabla_\theta \mu_\theta(s_t) \nabla_a Q^w(s_t, a_t)|_{a=\mu_\theta(s)}
\end{aligned}
$$

2 Offline Reinforcement Learning

2.1 Conservative Q-learning

Offline RL algorithms learn from large, previously collected datasets, without interaction. This in principle can make it possible to leverage large datasets, but in practice fully offline RL methods pose major technical difficulties, stemming from the distributional shift between the policy that collected the data and the learned policy.

High level: CQL learns a conservative Q-function such that the expected value of a policy under this Q-function lower bounds its true value.

- It does so by augmenting the standard Bellman error objective with a Q-value regularizer during training.
- The key idea is to minimize values under an appropriately chosen distribution over state-action tuples, and then further tighten this bound by also incorporating a maximization term over the data distribution.

Problem of overestimating value functions: Directly utilizing existing value-based off-policy RL algorithms in an offline setting generally results in poor performance, due to issues with bootstrapping from out-of-distribution actions and overfitting.

We can learn a less conservative lower bound Q-function, such that only the expected value of Q-function under the policy is lower-bounded, as opposed to a point-wise lower bound.

$\pi_\beta(a|s)$ represents the behavior policy, $\mathcal{D}$ is the dataset, and $d^{\pi_\beta}(s)$ is the discounted marginal state-distribution of $\pi_\beta(a|s)$.

Since $\mathcal{D}$ does not typically contain all possible transitions (s, a, s'), the policy evaluation step uses an *empirical* Bellman operator that only backs up a single sample. We denote this $\hat{\mathcal{B}}^\pi$. Given a datset $\mathcal{D} = \{s, a, r, s')\}$ of tuples from trajectories collected under behavior policy π_β:

$$\hat{Q}^{k+1} \leftarrow \operatorname*{argmin}_Q \mathbb{E}_{\underline{s,a,s' \sim \mathcal{D}}} \left[\left(\left(r(s,a) + \gamma \mathbb{E}_{\underline{a' \sim \hat{\pi}^k(a'|s')}} [\hat{Q}^k(s', a')] \right) - Q(s,a) \right)^2 \right] \qquad \text{Policy evaluation}$$

$$\hat{\pi}^{k+1} \leftarrow \operatorname*{argmax}_\pi \mathbb{E}_{s \sim \mathcal{D}, a \sim \pi^k(a|s)} \left[\hat{Q}^{k+1}(s,a) \right] \qquad \text{Policy improvement}$$

Action distribution shift: The target values for Bellman backups in policy evaluation

use actions sample from the learned policy π^k, but the Q-function is trained only on actions sampled from the behavior policy that produced $\mathcal{D}$, π_β.

- Since π is trained to maximize Q-values, it may be biased towards OOD actions (OOD relative to learning environment) with erroneously high Q-values.
- Since we can't sample online and observe the action's actual value in offline RL, we can't correct for overestimation of OOD actions.
- ***Note:*** Offline RL does not suffer from state distribution shift during *training*, as the Bellman backup never queries the Q-function on OOD states. However, the policy may suffer from state distribution shift at test time.

2.1.1 Conservative Off-Policy Evaluation

We want to estimate $V^\pi(s)$ of a target policy π given access to $\mathcal{D}$, generated by following behavior policy $\pi_\beta(a|s)$. CQL's choice of penalty is to minimize the expected Q-value under a particular distribution of state-action pairs, $\mu(s,a)$. We restrict μ to match the state-marginal in the dataset, such that $\mu(s,a) = d^{\pi_\beta}(s)\mu(a|s)$. This gives the following update, with α as the tradeoff factor:

$$\hat{Q}^{k+1} \leftarrow \operatorname*{argmin}_Q \alpha\, \mathbb{E}_{s\sim\mathcal{D},a\sim\mu(a|s)}[Q(s,a)] + \frac{1}{2}\mathbb{E}_{s,a\sim\mathcal{D}}\Big[\big(Q(s,a) - \hat{\mathcal{B}}^\pi\hat{Q}^k(s,a)\big)^2\Big].$$

If we only require that the expected value of $\hat{Q}^\pi$ under $\pi(a|s)$ lower-bound V^π, we can improve the bound by introducing an additional Q-value maximization term under $\pi_\beta(a|s)$:

$$\hat{Q}^{k+1} \leftarrow \operatorname*{argmin}_Q \alpha\cdot\big(\mathbb{E}_{s\sim\mathcal{D},a\sim\mu(a|s)}[Q(s,a)] - \mathbb{E}_{s\sim\mathcal{D},a\sim\hat{\pi}_\beta(a|s)}[Q(s,a)]\big) + \frac{1}{2}\,\mathbb{E}_{s,a,s'\sim\mathcal{D}}\Big[\big(Q(s,a)\;\hat{\mathcal{B}}^\pi\hat{Q}^k(s,a)\big)^2\Big].$$

Note: As $|\mathcal{D}(s,a)|$ increases, the theoretical value of α needed to guarantee a lower bound decreases, which indicates that in the limit of infinite data, a lower bound can be obtained by using extremely small α.

2.1.2 CQL for Offline RL

Since the policy $\hat{\pi}^k$ is typically derived from the Q-function, we could choose $\mu(a|s)$ to approximate the policy that would maximize the current Q-function iterate, thus giving rise to an online algorithm.

We can capture such online algorithms by defining a family of optimization problems over $\mu(a|s)$. An instance of this family is denoted by CQL($\mathcal{R}$) and characterized by a choice of regularizer $\mathcal{R}(\mu)$:

$$\min_{Q} \max_{\mu} \alpha \left(\mathbb{E}_{s\sim\mathcal{D},a\sim\mu(a|s)}[Q(s,a)] - \mathbb{E}_{s\sim\mathcal{D},a\sim\hat{\pi}_\beta(a|s)}[Q(s,a)]\right) + \frac{1}{2}\,\mathbb{E}_{s,a,s'\sim\mathcal{D}}\left[\left(Q(s,a) - \hat{\mathcal{B}}^{\pi_k}\hat{Q}^k(s,a)\right)^2\right] + \mathcal{R}(\mu).$$

2.2 RLHF and Direct Preference Optimization

2.2.1 RLHF (overview)

There are three stages:

Supervised Fine-tuning (SFT): We have a pre-trained language model that we fine-tune using data for downstream tasks (such as problem solving, dialogue, etc.). This gives us the model π^{SFT}.

Reward Modelling: First we create a dataset of human preferences. We prompt the language model π^{SFT} with prompts x to get pairs of responses $(y_1, y_2) \sim \pi^{\text{SFT}}$. Then, human labeller(s) rank these responses as $y_w \succ y_l \mid x$ where y_w is the preferred response and y_l is the dispreferred response in $\{y_1, y_2\}$. This gives us a dataset $\mathcal{D} = \{x^{(i)}, y_w^{(i)}, y_l^{(i)}\}_{i=1}^N$.

Now, we make the following assumption: *these human preferences come from some reward model $r^*(y,x)$ that we do not have access to.* However, *if* we had access to this reward function $r^*(y,x)$, we could model preferences using the Bradley-Terry model:

$$p^*(y_1 \succ y_2 \mid s) := \frac{\exp(r^*(x,y_1))}{\exp(r^*(x,y_1)) + \exp(r^*(x,y_2))}.$$

This allows us to say that $\mathcal{D} \sim p^*$.

Now we model this reward function using the dataset $\mathcal{D}$: we train $r_\phi(x,y)$. We can frame this as a binary classification problem and train r_ϕ by minimizing the following negative log-likelihood loss:

$$\mathcal{L}_R(r_\phi) := -\mathbb{E}_{x,y_w,y_l\sim\mathcal{D}}\left[\log\sigma(r_\phi(x,y_w) - r_\phi(x,y_l))\right]$$

where σ is the sigmoid function. The network r_ϕ is initialized with π^{SFT} with an addition of a linear layer on top of the SFT model.

RL Fine-tuning: Now that we have a reward model, we can maximize the following:

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\theta(y|x)}[r_\phi(x,y)] - \beta D_{KL}(\pi_\theta(y \mid x) \mid\mid \pi_{\text{ref}}(y \mid x)). \tag{2}$$

However, this is not differentiable so we instead do online reinforcement learning. We do this by modeling $r(x,y) = r_\phi(x,y) - \beta\left(\log\pi_\theta(y \mid x) - \log\pi_{\text{ref}}(y \mid x)\right)$ where π_{ref} is the base policy π_{SFT}.

2.2.2 DPO

Direct Preference Optimization (DPO) starts by observing that there is a theoretical solution to the objective 2 in RLHF. This is given by

$$\pi_r(y \mid x) = \frac{1}{Z(x)}\pi_{\text{ref}}(y \mid x)\exp\left(\frac{1}{\beta}r(x,y)\right).$$

where $Z(x) = \sum_y \pi_{\text{ref}}(y \mid x)\exp(\frac{1}{\beta}r(x,y))$ is the partition function . We summarize this as follows:

Lemma 7. The optimal π_θ of the following problem $\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\theta(y|x)}[r_\phi(x,y)] - \beta D_{KL}(\pi_\theta(y \mid x) \mid\mid \pi_{\text{ref}}(y \mid x))$ is given by

$$\pi_r(y \mid x) = \frac{1}{Z(x)}\pi_{\text{ref}}(y \mid x)\exp\left(\frac{1}{\beta}r(x,y)\right).$$

where $Z(x) = \sum_y \pi_{\text{ref}}(y \mid x)\exp(\frac{1}{\beta}r(x,y))$.

Proof. We want to solve: $\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\theta(y|x)}[r_\phi(x,y)] - \beta D_{KL}(\pi_\theta(y \mid x) \mid\mid \pi_{\text{ref}}(y \mid x))$. Rewriting:

$$
\begin{aligned}
&\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\theta(y|x)}[r_\phi(x,y)] - \beta D_{KL}(\pi_\theta(y \mid x) \mid\mid \pi_{\text{ref}}(y \mid x)) \\
&= \max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\theta(y|x)} \left[r_\phi(x,y) - \beta \log \left(\frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)} \right) \right] \\
&= \min_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\theta(y|x)} \left[\beta \log \left(\frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)} \right) - r_\phi(x,y) \right] \\
&= \min_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\theta(y|x)} \left[\log \left(\frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)} \right) - \frac{1}{\beta} r_\phi(x,y) \right] \\
&= \min_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\theta(y|x)} \left[\log \left(\frac{\pi_\theta(y \mid x)}{\frac{1}{Z(x)} \pi_{\text{ref}}(y \mid x) \exp\left(\frac{1}{\beta} r(x,y) \right)} \right) - \log\left(Z(x)\right) \right]
\end{aligned}
$$

where $Z(x) = \sum_y \pi_{\text{ref}}(y \mid x) \exp\left(\frac{1}{\beta} r(x,y) \right)$.

Now, we define $\pi^*(y \mid x) := \frac{1}{Z(x)} \pi_{\text{ref}}(y \mid x) \exp\left(\frac{1}{\beta} r(x,y) \right)$. It is straightforward to check that this is a valid probability distribution. This allows us to continue:

$$
\begin{aligned}
&\min_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\theta(y|x)} \left[\log \left(\frac{\pi_\theta(y \mid x)}{\frac{1}{Z(x)} \pi_{\text{ref}}(y \mid x) \exp\left(\frac{1}{\beta} r(x,y) \right)} \right) - \log\left(Z(x)\right) \right] \\
&= \min_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}} \left[D_{\text{KL}}(\pi_\theta(y \mid x) \mid\mid \pi^*(y \mid x)) - \log\left(Z(x)\right) \right] \\
&= \min_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}} \left[D_{\text{KL}}(\pi_\theta(y \mid x) \mid\mid \pi^*(y \mid x)) \right].
\end{aligned}
$$

This is minimized if and only if $\pi_\theta = \pi^*$. □

This, however, is not useful in practice since we do not know $Z(x)$ and it is difficult to compute $Z(x)$ in practice because of the large support of the variable y (i.e. the entire language).

So far, we have assumed that we have trained the reward model and then we tried computing the optimal policy π^*, where we ran into the trouble with $Z(x)$. What if we do the opposite? Given the optimal policy, we can write the write the optimal reward function in terms of it by simple algebra:

$$
r(x,y) = \beta \log \left(\frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)} \right) + \beta \log Z(x).
$$

Since this is the parametrization of the reward model defining the optimal policy, we can impose this parametrization to the optimal reward function:

$$r^*(x, y) = \beta \log \left(\frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)} \right) + \beta \log Z(x).$$

Then, the Bradley-Terry model for the preference distribution becomes:

$$p^*(y_1 \succ y_2 \mid x) := \frac{1}{1 + \exp\left(\beta \log \frac{\pi^*(y_2|x)}{\pi_{\text{ref}}(y_2|x)} - \beta \log \frac{\pi^*(y_1|x)}{\pi_{\text{ref}}(y_1|x)}\right)}$$

This means, we can write the distribution of human preferences in terms of the optimal policy and not the optimal reward model. Since we already have the dataset where for each x, we have $y_w \succ y_l$, we now learn the optimal policy by maximizing the log probability of human preferences, modelled as above. This gives us the DPO loss function:

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) := -\mathbb{E}_{x, y_w, y_l \sim \mathcal{D}} \left[\log \left(\sigma \left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)} \right) \right) \right].$$

In summary, given access to a dataset where for each x, we have $y_w \succ y_l$, DPO models the probability of human preferences using the learned policy and then finds the policy that maximizes the probability of the human preferences.