Deep Reinforcement Learning

Jubayer Ibn Hamid

Preface

This document is a combination of notes from CS 224R taught by Prof. Chelsea Finn at Stanford University, CS 234 taught by Prof. Emma Brunskill at Stanford University, and CS 285 taught by Prof. Sergey Levine at UC Berkeley, as well as reading notes from various papers (in particular, Richard Sutton and Andrew Barto's "Reinforcement Learning: An Introduction" [1]).

Although these are reading notes, there may be various errors throughout, both minor and major, which almost certaintly did not appear in the original works that I was reading. If you find any, please let me know by sending me an email at `jubayer@stanford.edu`. I have also not been able to stick to the same notation throughout these notes and I apologize for that - in my defence, reinforcement learning is notorious for a lack of universal notation. Hopefully, the inconsistencies are not too troublesome.

These notes are still under construction - especially the chapters on offline reinforcement learning and preference fine-tuning.

I would also like to thank Ifdita, Andy and Sheryl for helping improve these notes.

Contents

1 Markov Decision Processes

1.1 Framework

For the majority of these notes, we will consider the framework of a Markov Decision Process (MDP), represented as $\langle \mathcal{S}, \mathcal{A}, p, r, \rho_0, \gamma, H \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, p is the transition dynamics, $r(s,a) \in \mathcal{R}$ is the reward function, ρ_0 is the initial state distribution, $\gamma \in [0,1]$ is the discount factor and H is the task horizon (which could be positive infinity).

In this setting, the agent interacts with the environments continually which results in observed rewards. We say that the agent interacts with the environment at time steps $t \in \mathbb{Z}^+$ (the set of positive integers).

The reward function is $r(s_t, a_t) = \mathbb{E}[r|S_t = s_t, A_t = a_t]$ i.e., the expected reward from taking an action from the current state. Here we are assuming that the reward after taking action a_t from state s_t has a distribution and $r(s_t, a_t)$ is the mean of that distribution. However, the reward could also be deterministic in which case the mean would collapse to the deterministic value. Eitherways, we will use the shorthand $r_t := r(s_t, a_t)$.

A finite MDP is an MDP such that $\mathcal{S}, \mathcal{A}$ and $\mathcal{R}$ are finite sets. In finite MDPs, the transition probabilities and the distribution over rewards are discrete distributions.

The MDP framework is a neat abstraction of sequential decision-making. As [1] describes, the action can be low-level (e.g. voltages applies to the motors of a robot arm) or high-level decisions (e.g. spend more money on a book or not). The states can be low-level sensations (e.g. direct sensory readings) or high-level information (e.g. symbolic descriptions of objects in a room or even images).

Definition 1. (Policy). A policy refers to an agent's distribution over actions at each state i.e.,

$$\pi(a|s) := P(A_t = a|S_t = s).$$

Note that a policy can be deterministic too, in which case $\pi(a|s)$ can be written as a Dirac-delta distribution.

An episode refers to "one play of the game" i.e., our agent ends up in the terminal state or the agent has taken H actions after which the environment is reset. For example, a game of Chess is an episodic task - even if the agent plays multiple games, the next episode begins independently of how the previous episode went. We sometimes denote the set of all terminal states as $\mathcal{S}^+$.

Goal of reinforcement learning: Our agent seeks to maximize the expected discounted sum of rewards i.e.

$$\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{H} \gamma^t r_t \mid \pi\right].$$

Note, the expectation is taken over the distribution of the random variables $s_t, a_t, s_{t+1}, a_{t+1}, \ldots$ induced by transition dynamics $p(s_{t'+1} \mid s_{t'}, a_{t'})$ and the policy $\pi(a_{t'}|s_{t'})$ for all $t' \in [0, H]$. Also note that the goal is not to maximize immediate reward but the cumulative reward in the long run.

1.2 Values

Definition 2. (Return). The total return from a trajectory from time t ownards is

$$G_t = r_t + r_{t+1} + \cdots + r_H$$

if H is finite. When we use the *discounted* sum of rewards (especially for non-episodic/continuing tasks), one can define the total discounted return even for infinite horizon tasks

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k}.$$

Then, we see that the goal of reinforcement learning is to maximize the expected total return from a trajectory. For most of these notes, I will try to be consistent and assume that H is infinite for the sake of simplicity.

Note: if $\gamma < 1$, then G_t is finite as long as the rewards $r_{t'}, \forall t'$, are boounded. We either require horizon H to be finite (episodic tasks) or, if $H = \infty$ (continuing tasks), we require $\gamma < 1$. Otherwise, the value $V_\pi(s)$ could blow up.

Note that we also have the following: $G_t = r_t + \gamma G_{t+1}$.

Definition 3. (Optimal Policy). The optimal policy is defined as:

$$\pi^* = \arg\max_\pi \mathbb{E}_\pi \left[\sum_t \gamma^t r_t \mid \pi \right]$$

Now, we introduce some more constructions that help express the goal of reinforcement learning - maximizing the expected (discounted) sum of rewards. To do so, we define the following:

Definition 4. Value function. Given a policy π, the value function of π refers to the expected sum of (discounted) rewards when starting from a given state s and acting according to π. For generality, suppose $H = \infty$, then this can be written as:

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R_t \middle| \pi, S_0 = s \right].$$

Note that the expectation is taken over trajectories $s_0, a_0, r_0, s_1, a_1, r_1, \cdots$ where $a_{t'} \sim \pi(\cdot \mid s_{t'})$ and $s_{t'+1} \sim p(\cdot \mid s_{t'}, a_{t'})$ for all t'. Furthermore, note that the value of the terminal state is always zero (since no action is taken at the terminal state and, therefore, no reward is observed).

(This notation is not universal; sometimes people write $V^{\pi}(s), v_{\pi}(s)$ or $V(s;\pi)$ to mean the same thing.)

Lemma 1. We have the following recursive relationship for the value function:

$$V_{\pi}(s) = \mathbb{E}_{\substack{a\sim\pi(a|s) \\ s',r\sim P(s',r|s,a)}} \left[r + \gamma V_{\pi}(s') \mid S_0 = s\right].$$

This is called the Bellman equation for V^{π}.

Proof. The derivation is as follows:

$$\begin{aligned}
V_{\pi}(s) &= \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\right]. \\
&= \mathbb{E}_{\pi}\left[r_0 + \sum_{t=1}^{\infty} \gamma^t r_t | s_0 = s\right]. \\
&= \mathbb{E}_{\pi}\left[r_0\right] + \mathbb{E}_{\pi}\left[\gamma \cdot \sum_{t=1}^{\infty} \gamma^{t-1} r_t | s_0 = s\right]. \\
&= \mathbb{E}_{\pi}\left[r_0\right] + \gamma\mathbb{E}_{\pi}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t \mid s_0 = s\right] \\
&= \mathbb{E}_{\pi}\left[r_0\right] + \gamma\mathbb{E}_{s_1\sim P(\cdot|s_0=s,a), a\sim\pi(\cdot|s_0=s)}\left[\sum_{t=1}^{\infty} \gamma^{t-1} r_t\right] \\
&= \mathbb{E}_{\pi}\left[r_0\right] + \gamma\mathbb{E}_{s'\sim P(\cdot|s_0=s,a), a\sim\pi(\cdot|s_0=s)}\left[\sum_{t=0}^{\infty} \gamma^{t} r_t \mid s_0 = s'\right] \\
&= \mathbb{E}_{\pi}\left[r_0\right] + \gamma\mathbb{E}_{s'\sim P(\cdot|s_0=s,a), a\sim\pi(\cdot|s_0=s)}\left[V_{\pi}(s')\right] \\
&= \mathbb{E}_{\substack{a\sim\pi(a|s) \\ s',r\sim P(s',r|s,a)}}\left[r + \gamma V_{\pi}(s')\right]
\end{aligned}$$

□

We then see that the goal of RL is to find the optimal policy defined by $\pi(a \mid s) = \arg\max_{\pi} V^{\pi}(s)$.

Intuitively, $V^{\pi}(s)$ is telling us, on average, how "good" we are when we start from state s and follow policy π (here "good" refers to the expected total return).

However, sometimes we might be interestd in the following question - instead of following policy π from state s, what if we take a specific action a and *then* follow policy π. In other words, we are trying to understand the value of taking a specific action a over the trajectory induced by our policy π.

Definition 5. (State-action value function/action-value function/Q function.) The Q-value of an action from a state or the state-action value function is defined to be

$$Q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| S_0 = s, A_0 = a\right].$$

Intuitively, $Q_\pi(s,a)$ refers to the expected sum of discounted rewards from taking action a from state s and then following policy π from then on.

Lemma 2. We have the following alternative expression of the action-value function in terms of the state-value function:

$$Q_\pi(s,a) = \mathbb{E}_{s',r\sim P(s',r|s,a)}\left[r + \gamma V_\pi(s') | S_0 = s, A_0 = a\right].$$

Lemma 3. We can express the state-value function in terms of the action-value function as follows:

$$V_\pi(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}\left[Q_\pi(s,a)|s\right].$$

1.3 Optimal Values and Policies

Definition 6. Optimal value function and optimal action-value function. The optimal value function is the expected sum of discounted rewards when starting from a given state s and acting optimally:

$$\begin{aligned} V^*(s) &= \max_\pi \mathbb{E}_\pi\left[\sum_{t=0}^{H} \gamma^t r_t | \pi, s_0 = s\right] \\ &= \max_\pi V_\pi(s). \end{aligned}$$

Similarly, we define

$$Q_*(s,a) = \max_\pi Q_\pi(s,a).$$

In other words, $Q_*(s,a)$ is the value of taking action a and then acting optimally. We have the following relations

$$V_*(s) = \max_a Q_*(s,a)$$

and

$$Q_*(s_t,a_t) = \mathbb{E}[r_t + \gamma V_*(s_{t+1}) \mid s_t, a_t].$$

Definition 7. Optimal policy. We say $\pi \geq \pi'$ if and only if $V_\pi(s) \geq V_{\pi'}(s)$ for all $s \in \mathcal{S}$. The optimal policy $\pi_*(s)$ is

$$\pi_*(s) = \arg\max_\pi V_\pi(s).$$

Note that π_* need not be unique.

Lemma 4. For an infinite horizon problem i.e $H = \infty$, the optimal policy is deterministic, stationary (i.e action distribution at a given state does not depend on the time) and not necessarily unique.

Why do we care about value functions and action-value functions? This is because knowing these would help us find the optimal policy fairly easily. Here are two ways you can do it:

1. If we have a policy π and we know $q_\pi(s, a)$, we can set the optimal policy to be
$$\pi^*(a \mid s) = 1\{a = \arg\max_{a'} q_\pi(s, a')\}. \tag{1}$$
In other words, the optimal policy could just take the optimal action in terms of $q_\pi(s, a)$. If we define this for every state, then we could get an optimal policy regardless of what π we use in equation 1. Note that this would make a deterministic policy.

2. if policy $\pi_\theta(a \mid s)$ is parametrized by parameters θ, we could update these parameters such that $\pi_\theta(a|s)$ for any good action a is maximized. In other words, if action a is such that $q_\pi(s, a) > V_\pi(s)$, we maximize $\pi_\theta(a \mid s)$ (increase probability of that action being taken).

1.4 State Distributions

We will often require some notion of the distribution of states visited or distribution of states we might start from.

- $\mathbb{P}(s_0 \to s, k, \pi_\theta)$ is the probability of reaching state s from s_0 in k time-steps under policy π. We can calculate this as follows:
$$\mathbb{P}(s_0 \to s, k, \pi) = P(s_0) \prod_{t=0}^{k-2} \left(\sum_{a_t, s_{t+1}} \pi(a_t \mid s_t) P(s_{t+1}|s_t, a_t) \right) \sum_{a_{k-1}} \pi(a_{k-1} \mid s_{k-1}) P(s_k = s|s_{k-1}, a_{k-1}).$$

- $p^\pi(s) = \sum_{k=0}^{\infty} \gamma^k \mathbb{P}(s_0 \to s, k, \pi)$ is the improper discounted state distribution. This is called improper because this does not sum to 1.

- $d^\pi(s) := (1 - \gamma) \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(s_0 \to s, t, \pi)$ is the distribution over all states induced by π. This is called the stationary distribution of Markov chain for π. The factor $(1 - \gamma)$ is a normalization constant and this distribution simply discounts states visited later in time.

1.5 Partially Observed MDPs

We frequently find ourselves working with partially observed MDPs (POMDPs). In this setting, the agent has access to *observations* and not the states. These observations contain noisy and/or incomplete information about the state. In these settings, the agent's action depends on a *history* of observations and not just the most recent observation. In other words, $a_t \sim \pi(\cdot \mid h_t)$ where $h_t = (o_0, a_0, o_1, a_1, \cdots, a_{t-1}, o_t)$ (here $o_{t'}$ is the observation at time step t').

1.6 Oversimplified Overview

With this, we can now give an overview of various methods discussed in these notes:

1. value-based methods: here we first try to estimate either the value function or the action-value function of a policy by interacting with the environment and collecting rewards. Once we have estimated the value of the policy, we update the policy and then we repeat. While this works for relatively small state and action spaces, for larger ones, we would requite value function approximation methods.
2. policy gradient methods: directly learn the optimal policy π_θ parametrized by θ. We learn this by updating θ so as to maximize the expected value. For these methods, we would still require some estimation of the value of the policy or, at the very least, returns from a policy.
3. model-based reinforcement learning: learn the transition model i.e. the transition dynamics of the environment. Once we have this transition model, we can use it for planning or improving a policy.

Generally, the algorithms we consider will be characterized as one of two things:

1. off-policy: these algorithms can collect experiences (experiences are lists like $s_t, a_t, r_t, s_{t+1}, a_{t+1}, r_{t+1}, \ldots, s_{t+H}$) using a policy π_β and use that to update a *different* policy π_θ.
2. on-policy: to update any policy π, we require experiences collected using policy π. In other words, we cannot use experiences from other policies to update this policy.

While on-policy methods can give us more reliable signals to help us update our policy, off-policy methods are more sample efficient.

2 Policy Gradient Methods

In policy gradient methods, we directly parametrise the policy to be $\pi_\theta(a|s)$. Our goal is then to find π_θ that maximizes expected returns. In the other algorithms we have seen so far, we tried to estimate the value of a policy and *then* improved it, whereas, in the case of policy gradients, we will directly aim to learn the optimal policy. We may still learn a value function in order to help learn our parametrized optimal policy, but the value function will not be required for our action selection (in the sense that, when selecting which action to take, we will not query the value function - the value function is only used to help us update our policy π_θ). 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.

Policy gradient methods are useful for generating stochastic policies. A useful example of a parametrized policy to keep in mind is a Gaussian policy (especially for continuous action spaces) that is parametrized as $a \sim \mathcal{N}(\mu_\theta(s), \Sigma) =: \pi_\theta(\cdot \mid s)$ (we usually set Σ to be a diagonal matrix). The policy can be parametrized in any way as long as it is differentiable with respect to its parameters. More complex parameterizations can be used; the choice of policy parameterization is sometimes a good way to inject prior knowledge about the desired form of the policy into the reinforcement learning system. In any case, we generally want to encourage the policy to explore and, to do so, we usually require that the policy never becomes deterministic i.e. $\pi_\theta(a \mid s) \in (0,1), \forall s, a$.

Policy gradient methods are usually suitable for continuous action spaces and for high-dimensional action spaces. However, a common phenomenon with policy gradient methods is that these policies tend to be hard to evaluate and can have a high variance.

Recall: we will require the following notions of distributions over states that we introduced before in section 1.4:

- $\mathbb{P}(s_0 \to s, k, \pi_\theta)$ 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.
- $d^{\pi_\theta}(s) := (1-\gamma) \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(s_0 \to s, t, \pi_\theta)$ is the distribution over all states induced by π_θ be. The factor $(1-\gamma)$ is a normalization constant and this distribution simply discounts states visited later in time.

Overview:

- **On-policy algorithms:** REINFORCE, REINFORCE with baseline and actor-critic algorithms. These algorithms require sampling trajectories with the parameters at each iteration and *then* making updates to the policy.
- **Off-policy algorithms:** Off-policy actor-critic, SAC, etc. These algorithms collect trajectories using a behavior policy which are then used to update potentially different policies.

2.1 Deriving the Policy Gradient

In this note, we will cover episodic tasks. Let $V(\theta) = V_{\pi_\theta}(s_0)$, 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 $V(\theta)$:

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

where $Q_{\pi_\theta}(s_0, a)$ 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 5. $\nabla_\theta V(\theta) = \sum_\tau R(\tau) P_\theta(\tau) \nabla_\theta \log(P_\theta(\tau))$

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 5, we see that

$$\nabla_\theta V(\theta) = \mathbb{E}_{P_\theta}[R(\tau) \nabla_\theta \log(P_\theta(\tau))].$$

We can also 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 lemma:

Lemma 6. $\nabla_\theta \log(P_\theta(\tau)) = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t)$

Proof.

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

□

With Lemma 6, we get the following tractable update rule:

$$\boxed{\nabla_\theta V(\theta) = \mathbb{E}_{\tau\sim P_\theta(\tau)}\left[\left(\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)\right)\left(\sum_{t=0}^{T} r(s_t,a_t)\right)\right].}$$

We can easily compute this as long as we can take the derivative of $\log \pi_\theta(a \mid s)$.

2.2 Policy Gradient Theorem

We generalize this approach through the policy gradient theorem [1]. First, we consider an episodic task.

Theorem 7. (Policy Gradient Theorem) Suppose either that $\gamma < 1$ or, if $\gamma = 1$, that our task is episodic. Furthermore, suppose, our start state is s_0 for all trajectories. Then,

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

Proof.

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

□

2.3 REINFORCE

Using our derivation, we have our first policy gradient algorithm. So far, we have:

$$\nabla_\theta V(\theta) = \mathbb{E}_{\tau \sim P_\theta(\tau)} \left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \left(\sum_{t'=0}^{T} r(s_{t'}, a_{t'}) \right) \right].$$

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

$$\boxed{\nabla_\theta V(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \right) \left(\sum_{t'=0}^{T} r(s_{i,t'}, a_{i,t'}) \right)}$$

Intuitive interpretation: One way to interpret this update rule is to note that we are maximizing the log-likelihood of each trajectory sampled - except we are weighing them by their returns. In other words, if a trajectory yields higher returns, we are increasing the log-likelihood of that with a larger weight than one that yields lower returns.

Problems with REINFORCE:

1. Since this is an on-policy method, we require sampling multiple trajectories for each update to the policy.
2. The algorithm has high variance since the trajectory samples are often quite noisy (especially in most real-world environments). More precisely, $\sum_{t'=1}^{T} r(s_{i,t'}, a_{i,t'})$ has high variance. As such, REINFORCE is a quite slow algorithm in most cases.

We will next introduce a couple of algorithms that are aimed towards solving these issues.

2.4 REINFORCE using causality

The update rule we derived so far is

$$\nabla_\theta V(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \right) \left(\sum_{t'=0}^{T} r(s_{i,t'}, a_{i,t'}) \right).$$

Recall the intuition we provided in the previous section: we are maximizing log probability of each action weighted by the "goodness" of the action i.e. the returns achieved by executing the action. Now, notice that the action at time t cannot affect the rewards at time $t' < t$. In other words, when we maximize the log-likelihood of taking a particular action $a_{i,t}$ from a state $s_{i,t}$, should we really weigh it by the reward attained from the entire trajectory? Intuitively, it makes more sense to weigh it by the rewards attained from that time t onward in the trajectory i since only the rewards attained *after* executing $a_{i,t}$ gives us a signal for how good the action is. With this in mind, we have the following modification:

$$\boxed{\nabla_\theta V(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \right) \left(\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}) \right).}$$

The term $\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})$ is called reward-to-go. In other words, we are using the "reward to go" from time t.

```
Algorithm: REINFORCE (with causality)
1: Initialize policy parameters $\theta$
2: for iteration = 1, 2, ... do
3:     Collect a set of trajectories $\{\tau^i\}$ by running the policy $\pi_\theta(a_t|s_t)$
4:     for each trajectory $\tau^i$ do
5:         for each timestep $t$ in $\tau^i$ do
6:             Compute return: $G_t^i = \sum_{t'=t}^{T} r(s_{t'}^i, a_{t'}^i)$
7:         end for
8:     end for
9:     Update the policy parameters:
           $\nabla_\theta V(\theta) \approx \sum_i \sum_t \nabla_\theta \log \pi_\theta(a_t^i|s_t^i) G_t^i$
           $\theta \leftarrow \theta + \alpha \nabla_\theta V(\theta)$
10: end for
```

Intuitive interpretation: we are updating the policy parameters by taking a step in the direction of $\nabla_\theta V(\theta) \approx \sum_i \sum_t (\nabla_\theta \log \pi_\theta) G_t^i$. Focus on the term $\sum_i \sum_t (\nabla_\theta \log \pi_\theta)$. A step in this direction is essentially maximizing the probability of $A = a \mid s$ under π_θ, which is what maximum likelihood estimation does! Since this is scaled by G_t^i, it takes a step in a direction that is closed aligned with high G_t^i actions i.e., actions that we expect to give higher returns.

Here is a more formal derivation of this causality idea:

So far, in the derivation of the policy gradient, we assumed we wanted to maximize $\mathbb{E}_{\pi_\theta}[R(\tau)]$. What if we instead choose to maximize the reward at one time-step, r_t, only? Following the same steps of derivation, we would arrive at

$$\nabla_\theta \mathbb{E}_{\pi_\theta}[r_t] = \mathbb{E}_{\pi_\theta}\left[\sum_{t'=0}^{t} \nabla_\theta \log \pi_\theta(a_{t'} \mid s_{t'}) r_t\right].$$

Now, we can take the sum of rewards over time $t' = 0, \cdots, T$ to get

$$\begin{aligned}\nabla_\theta \mathbb{E}_{\pi_\theta}[r_t] &= \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} r_t \sum_{t'=0}^{t} \nabla_\theta \log \pi_\theta(a_{t'} \mid s_{t'})\right] \\ &= \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) \sum_{t'=t}^{T} r_{t'}\right].\end{aligned}$$

2.5 REINFORCE with baseline

To further reduce variance, we introduce a baseline:

$$\nabla_\theta V(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \right) \left(\sum_{t'=0}^{T} r(s_{i,t'}, a_{i,t'}) - b(s_{i,t'}) \right).$$

where b is any arbitrary function as long as it does not depend on a_t.

Intuitive explanation: in REINFORCE, we sample trajectories and maximize the log probability of actions based on how good they were. Subtracting an appropriate baseline (we will discuss what are appropriate baselines soon) is equivalent to asking "Are the returns from this action better than expected?" We choose a baseline that gives us a sense of $V_{\pi_\theta}(s_{i,t'})$ because that is the expected returns from the state.

What baseline should we choose? We usually choose baselines that are either constant or depends only on $s_{i,t'}$ but not on $a_{i,a'}$.

1. We usually select $b(s) = \mathbb{E}_{\tau \sim \pi_\theta}[r(\tau) \mid s_0 = s]$.
2. We could also use a learned state value function parametrized by ϕ,
$$v_\phi(s_t) =: b(s_t).$$

Note that either choice gives is an unbiased modification of the original update rule derived from the policy gradient theorem:

Proposition 8. Subtracting the baseline b (that does not depend on action) is unbiased in expectation i.e.,

$$\nabla_\theta V(\theta) = \mathbb{E}_{s \sim p^{\pi_\theta}, a \sim \pi_\theta(\cdot|s)} \left[(\nabla_\theta \log \pi_\theta(a|s)) \left(Q_{\pi_\theta}(s,a) - b(s) \right) \right]$$

where $\nabla_\theta V(\theta)$ is computed as in the policy gradient theorem (theorem 7).

Proof. We show

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

$$
\begin{aligned}
&\mathbb{E}_{\tau\sim P_\theta(\tau)}\left[\left(\sum_{t=0}^{T}(\nabla_\theta \log \pi_\theta(a_t|s_t))V^{\pi_\theta}(s_t)\right)\right] \\
&= \sum_{t=0}^{T}\mathbb{E}_{\tau\sim P_\theta(\tau)}\left[(\nabla_\theta \log \pi_\theta(a_t|s_t))V^{\pi_\theta}(s_t)\right] \\
&= \sum_{t=0}^{T}\sum_{s_t}\mathbb{P}(s_0 \to s_t, t, \pi_\theta)\sum_{a_t}\pi_\theta(a_t \mid s_t)(\nabla_\theta \log \pi_\theta(a_t|s_t))V^{\pi_\theta}(s_t) \\
&= \sum_{t=0}^{T}\sum_{s_t}\mathbb{P}(s_0 \to s_t, t, \pi_\theta)V^{\pi_\theta}(s_t)\sum_{a_t}\pi_\theta(a_t \mid s_t)(\nabla_\theta \log \pi_\theta(a_t|s_t)) \\
&= \sum_{t=0}^{T}\sum_{s_t}\mathbb{P}(s_0 \to s_t, t, \pi_\theta)V^{\pi_\theta}(s_t)\sum_{a_t}\nabla_\theta\pi_\theta(a_t \mid s_t) \\
&= \sum_{t=0}^{T}\sum_{s_t}\mathbb{P}(s_0 \to s_t, t, \pi_\theta)V^{\pi_\theta}(s_t)\nabla_\theta\left(\sum_{a_t}\pi_\theta(a_t \mid s_t)\right) \\
&= \sum_{t=0}^{T}\sum_{s_t}\mathbb{P}(s_0 \to s_t, t, \pi_\theta)V^{\pi_\theta}(s_t)\nabla_\theta\,(1) \\
&= \sum_{t=0}^{T}\sum_{s_t}\mathbb{P}(s_0 \to s_t, t, \pi_\theta)V^{\pi_\theta}(s_t)\cdot 0 \\
&= 0.
\end{aligned}
$$

□

We can actually find the baseline that reduces the variance the most in a principled manner:

Proposition 9. $b = \frac{\mathbb{E}[g(\tau)^2 r(\tau)]}{\mathbb{E}[g(\tau)^2]}$ minimizes the variance:

$$
\mathrm{Var} := \mathbb{E}_{\tau\sim P_\theta(\tau)}\left[(\nabla_\theta \log P_\theta(\tau)(r(\tau)-b))^2\right] - \mathbb{E}_{\tau\sim P_\theta(\tau)}\left[\nabla_\theta \log P_\theta(\tau)(r(\tau)-b)\right]^2 .
$$

Here $g(\tau)$ is the gradient $\nabla_\theta \log P_\theta(\tau)$.

Proof. Take the derivative with respect to the baseline b and, after a little bit of algebra, you get the desired expression. □

In other words, the optimal baseline is the expected reward but weighted by the magnitude of our gradients. In practice, we end up just using the expected reward as the baseline.

Algorithm: REINFORCE with baseline

```
1:  **Initialize** policy parameters $\theta$
2:  for iteration = 1, 2, ... do
3:      Collect a set of trajectories $\{\tau^i\}_{i=1}^N$ by running the policy $\pi_\theta(a_t|s_t)$
4:      for each trajectory $\tau^i$ do
5:          for each timestep $t$ in $\tau^i$ do
6:              Compute return: $G_t^i = \sum_{t'=t}^T r(s_{t'}^i, a_{t'}^i)$
7:          end for
8:      end for
9:      Compute $b = \frac{1}{N} \sum_{i=1}^N r(\tau^i)$
10:     Update the policy parameters:
            $\nabla_\theta V(\theta) \approx \sum_i \sum_t \nabla_\theta \log \pi_\theta(a_t^i|s_t^i)(G_t^i - b)$
            $\theta \leftarrow \theta + \alpha \nabla_\theta V(\theta)$
11: end for
```

REINFORCE with Baseline using learned value-function baseline

Input: a differentiable policy parameterization $\pi(a \mid s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$
Algorithm parameters: step sizes $\alpha^\theta > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):

- Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot \mid \cdot, \boldsymbol{\theta})$
- **Loop for each step of the episode** $t = 0, 1, \ldots, T-1$:
  - $G \leftarrow \sum_{k=t+1}^T \gamma^{k-t-1} R_k \qquad (G_t)$
  - $\delta \leftarrow G - \hat{v}(S_t, \mathbf{w})$
  - $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}} \delta \nabla \hat{v}(S_t, \mathbf{w})$
  - $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^\theta \gamma^t \delta \nabla \ln \pi(A_t \mid S_t, \boldsymbol{\theta})$

2.6 On-Policy Actor-Critic Methods

Let us consider the baseline $b(s_{i,t'}) = V_{\pi_\theta}(s_{i,t'})$ i.e. the true state-value function under policy π_θ. We will see how to get this value function soon. We also replace the reward to go with the state-action value function for taking $a_{i,t'}$ from $s_{i,t'}$ (as seen in the sampled trajectory) i.e. $Q_{\pi_\theta}(s_{i,t'})$. Then, note that the factor multiplying the gradient of the log of our policy becomes $Q_{\pi_\theta}(s_{i,t'}) - V_{\pi_\theta}(s_{i,t'}) = r(s_{i,t'}) + \gamma \cdot \mathbb{E}_{s_{t'+1} \sim P(\cdot|s_{i,t'}, a_{i,t'})}[V_{\pi_\theta}(s_{t'+1})] = A^{\pi_\theta}(s_{i,t'}, a_{i,t'})$. This is the main idea in this section.

Actor-Critic methods replace the reward to go $\sum_{t'=t}^T r(s_{i,t'}, a_{i,t'})$ with $r(s_{i,t}, a_{i,t}) + \gamma V_\phi^{\pi_\theta}(s_{i,t+1})$.

As for the baseline, we use $V_\theta(s_{i,t})$. Then, we can replace $\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})$ in REINFORCE with the advantage function $A^{\pi_\theta}(s_{i,t'}, a_{i,t'}) \approx r(s_{i,t}, a_{i,t}) + \gamma V_\phi^{\pi_\theta}(s_{i,t+1}) - V_\phi^{\pi_\theta}(s_{i,t})$. Note that here we removed the expectation over the next state $s_{i,t+1}$ with just a single sample estimate from the trajectory i. This can be a poor estimate. In general, the better the estimate of the advantage, the lower the variance.

Actor-critic methods use a learned value function $V_\phi^{\pi_\theta}(s_t)$. This is trained using supervised regression. Suppose that 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:

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

One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

```
Input: a differentiable policy parameterization $\pi(a \mid s, \boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s, \mathbf{w})$
Parameters: step sizes $\alpha^\theta > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
    • Initialize $S$ (first state of episode)
    • $I \leftarrow 1$
    • Loop while $S$ is not terminal (for each time step):
        – $A \sim \pi(\cdot \mid S, \boldsymbol{\theta})$
        – Take action $A$, observe $S'$, $R$
        – $\delta \leftarrow R + \gamma\hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$        (if $S'$ is terminal, then $\hat{v}(S', \mathbf{w}) \doteq 0$)
        – $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta\nabla\hat{v}(S, \mathbf{w})$
        – $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^\theta I\delta\nabla \ln \pi(A \mid S, \boldsymbol{\theta})$
        – $I \leftarrow \gamma I$
        – $S \leftarrow S'$
```

2.7 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 $P_\theta(s_t = s) = \sum_{a_0} \sum_{s_1} ... \sum_{s_{t-1}} \sum_{a_{t-1}} \left(\prod_{i=1}^{T-2} \pi_\theta(a_i|s_i)P(s_{i+1}|s_i, a_i)\right) \pi_\theta(a_{t-1} \mid s_{t-1})P(s_t = s \mid s_{t-1}, a_{t-1})$.

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

Proof.

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

$$
\begin{aligned}
&= \sum_{t=0}^{\infty} \gamma^t \sum_{s_t} \sum_{a_t} P_\theta(s_t) \pi_\theta(a_t|s_t) f(s_t, a_t) \\
&= \frac{1}{1-\gamma}(1-\gamma) \sum_{t=0}^{\infty} \gamma^t \sum_{s_t} P_\theta(s_t) \sum_{a_t} \pi_\theta(a_t|s_t) f(s_t, a_t) \\
&= \frac{1}{1-\gamma}(1-\gamma) \sum_{t=0}^{\infty} \gamma^t \sum_{s_t} P_\theta(s_t) \mathbb{E}_{a\sim\pi_\theta}\left[f(s_t, a)\right] \\
&= \frac{1}{1-\gamma} \mathbb{E}_{s\sim d^{\pi_\theta}}\left[\mathbb{E}_{a\sim\pi_\theta}\left[f(s_t, a)\right]\right]
\end{aligned}
$$

□

Theorem 11. (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:

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

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

Therefore, we get

$$\begin{aligned}
V^{\pi}(s_0) - V^{\pi'}(s_0) &= \mathbb{E}_{\tau\sim P^{\pi}}\left[\sum_{t=0}^{\infty}\gamma^t\left(Q^{\pi'}(s_t,a_t) - V^{\pi'}(s_t)\right)\right] \\
&= \mathbb{E}_{\tau\sim P^{\pi}}\left[\sum_{t=0}^{\infty}\gamma^t A^{\pi'}(s_t,a_t)\right] \\
&= \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\pi}}\left[\mathbb{E}_{a\sim\pi}\left[A^{\pi'}(s,a)\right]\right]
\end{aligned}$$

□

Sometimes, as in [2], this lemma has the following equivalent expression:

Proposition 12. (Performance Difference Lemma (2)) Given two policies π and π',

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

Proof. We write:

$$A_{\pi}(s,a) = \mathbb{E}_{s'\in P(s'|s,a)}\left[r(s,a) + \gamma V_{\pi}(s') - V_{\pi}(s)\right].$$

Then,

$$\begin{aligned}
\mathbb{E}_{\pi'}\left[\sum_{t=0}^{\infty}\gamma^t A_{\pi}(s_t,a_t)\right] &= \mathbb{E}_{\pi'}\left[\sum_{t=0}^{\infty}\gamma^t\left(r(s_t,a_t) + \gamma V_{\pi}(s_{t+1}) - V_{\pi}(s_t)\right)\right] \\
&= \mathbb{E}_{\pi'}\left[-V_{\pi}(s_0) + \sum_{t=0}^{\infty}\gamma^t\left(r(s_t,a_t)\right)\right] \\
&= -\mathbb{E}_{s_0\sim\rho_0}[V_{\pi}(s_0)] + \mathbb{E}_{\pi'}[\sum_{t=0}^{\infty}\gamma^t r_t] \\
&= -V(\pi) + V(\pi').
\end{aligned}$$

□

2.8 Covariant/natural policy gradient

So far, our update rules were aimed towards computing

$$\theta \leftarrow \theta + \alpha\nabla_{\theta}V(\theta)$$

to update the policy $\pi_\theta(a \mid s)$. However, controlling the learning rate to maximize $V(\theta)$ is non-trivial. Some parameters of your policy ultimately end up affecting $V(\theta)$ more than others and choosing one constant α that controls the learning rate for all the parameters is difficult. We would want to have higher learning rates for parameters that do not change the policy very much and smaller learning rates for those that do.

Now, notice that using a first-order Taylor expansion, we can write:

$$\arg\max_{\theta'} V(\theta') \approx \arg\max_{\theta'} V(\theta) + (\theta' - \theta)^T \nabla_\theta V(\theta).$$

So, we aim to solve $\arg\max_{\theta'} (\theta' - \theta)^T \nabla_\theta V(\theta)$ such that $||\theta' - \theta|| \leq \epsilon$ (so that the Taylor approximation is valid). We can reframe this problem in the policy space: we aim to solve $\arg\max_{\theta'} (\theta' - \theta)^T \nabla_\theta V(\theta)$ such that $D(\pi_{\theta'} \mid\mid \pi_\theta) \leq \epsilon$ where D is a divergence-measure [3].

We can choose D to be the KL-divergence. In this case, we can approximate (using the second-order Taylor approximation) $D_{KL}(\pi_{\theta'} \mid\mid \pi_\theta) \approx (\theta' - \theta)^T F (\theta' - \theta)$ where F is the Fisher information matrix, i.e. $F = \mathbb{E}_{\pi_\theta}[(\nabla_\theta \log \pi_\theta(a \mid s))\, (\nabla_\theta \log \pi_\theta(a \mid s))^T]$. Then, the problem becomes: $\arg\max_{\theta'} (\theta' - \theta)^T \nabla_\theta V(\theta)$ such that $||\theta' - \theta||_F^2 \leq \epsilon$. Then, the update rule becomes

$$\theta \leftarrow \theta + \alpha F^{-1} \nabla_\theta V(\theta)$$

2.9 Trust Region Policy Optimization (TRPO)

In this section, we discuss the building blocks of TRPO.

Let $V(\pi) = \mathbb{E}_\pi[\sum_{t=0}^\infty \gamma^t r_t]$. Recall: $Q_\pi(s_t, a_t) = \mathbb{E}_\pi[\sum_{l=0}^\infty \gamma^l r_{t+l} \mid s_t, a_t]$, $V_\pi(s_t) = \mathbb{E}_\pi[\sum_{l=0}^\infty \gamma^l r_{t+l} \mid s_t]$ and $A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$ where $a_t \sim \pi(a_t \mid s_t)$.

Recall the improper discounted state distribution $p^\pi(s) = \sum_{k=0}^\infty \gamma^k \mathbb{P}(s_0 \to s, k, \pi)$. Using this, we can write the performance difference lemma as:

$$\begin{aligned}
V(\pi') - V(\pi) &= \sum_{t=0}^{\infty} \sum_s P(s_t = s \mid \pi') \sum_a \pi'(a \mid s) \gamma^t A_\pi(s, a) \\
&= \sum_s \sum_{t=0}^{\infty} \gamma^t P(s_t = s \mid \pi') \sum_a \pi'(a \mid s) A_\pi(s, a) \\
&= \sum_s p^{\pi'}(s) \sum_a \pi'(a \mid s) A_\pi(s, a)
\end{aligned}$$

Note: $V(\pi') - V(\pi) \geq 0$ if, at every state s, we have that $\sum_a \pi'(a \mid s) A_\pi(s, a) \geq 0$.

Now, the first building block of TRPO is the local approximation of this as

$$L_\pi(\pi') = V(\pi) + \sum_s p^\pi(s) \sum_a \pi'(a \mid s) A_\pi(s,a)$$

where we have replaced $p^{\pi'}$ with $p^{\pi'}$. Note that when our policy $\pi_\theta(a \mid s)$ is differentiable, then, for the parameters θ_0, we have that $L_{\pi_{\theta_0}}(\pi_{\theta_0}) = V(\pi_{\theta_0})$ and $\nabla_\theta L_{\pi_{\theta_0}}(\pi_\theta)|_{\theta=\theta_0} = \nabla_\theta V(\pi_\theta)|_{\theta=\theta_0}$. Therefore, if the update $\pi_{\theta_0} \to \pi'$ is small enough such that $L_{\pi_{\theta_0}}(\pi_{\theta_0})$ improves, then we see an improvement in the value V as well. However, controlling the learning rate that ensures this is difficult. TRPO aims to solve this issue.

The guiding principal comes from the following theorem [2]:

Theorem 13. Let $\alpha = D_{\mathrm{TV}}^{\max}(\pi_{\mathrm{old}}, \pi_{\mathrm{new}}) = \max_s D_{\mathrm{TV}}(\pi_{\mathrm{old}}(\cdot \mid s) || \pi_{\mathrm{new}}(\cdot \mid s))$. Then,

$$V(\pi_{\mathrm{new}}) \geq L_{\pi_{\mathrm{old}}}(\pi_{\mathrm{new}}) - \frac{4\epsilon\gamma}{(1-\gamma)^2}\alpha^2$$

where $\epsilon = \max_{s,a} |A_\pi(s,a)|$.

The main building blocks are the following two definitions and lemma:

Definition 8. (α-coupled policy pair). We call (π, π') an α-coupled policy pair if the joint distribution $(a, a') \mid s$ is such that $\mathbb{P}(a \neq a' \mid s) \leq \alpha$ for all s.

We also define

$$\bar{A}(s) = \mathbb{E}_{a \sim \pi'(\cdot|s)}\left[A_\pi(s,a)\right].$$

Then, we have the following lemma:

Lemma 14. Let (π, π') be an α-coupled policy pair. Then, for all states s,

$$|\bar{A}(s)| \leq 2\alpha \max_{s,a} |A_\pi(s,a)|.$$

Proof.

$$\begin{aligned}\bar{A}(s) &= \mathbb{E}_{a' \sim \pi'}[A_\pi(s,a')] \\ &= \mathbb{E}_{(a,a') \sim (\pi,\pi')}[A_\pi(s,a') - A_\pi(s,a)]\end{aligned}$$

as $\mathbb{E}_{a\sim\pi}[A_\pi(s,a)] = 0$. Then, continuing:

$$\begin{aligned}
\bar{A}(s) &= \mathbb{E}_{(a,a')\sim(\pi,\pi')}[A_\pi(s,a') - A_\pi(s,a)] \\
&= \sum_{a'} \pi'(a' \mid s) \sum_{a \neq a'} \pi(a \mid s)(A_\pi(s,a') - A_\pi(s,a)) \\
|\bar{A}(s)| &= |\sum_{a'} \pi'(a' \mid s) \sum_{a \neq a'} \pi(a \mid s)(A_\pi(s,a') - A_\pi(s,a))| \\
&\leq |\sum_{a'} \pi'(a' \mid s) \sum_{a \neq a'} \pi(a \mid s)(A_\pi(s,a'))| + |\sum_{a'} \pi'(a' \mid s) \sum_{a \neq a'} \pi(a \mid s)(A_\pi(s,a))| \\
&\leq |\sum_{a'} \pi'(a' \mid s) \sum_{a \neq a'} \pi(a \mid s)(\max_{s,a} A_\pi(s,a))| + |\sum_{a'} \pi'(a' \mid s) \sum_{a \neq a'} \pi(a \mid s)(\max_{s,a} A_\pi(s,a))| \\
&= |\max_{s,a} A_\pi(s,a)| 2 \sum_{a'} \pi'(a' \mid s) \sum_{a \neq a'} \pi(a \mid s)| \\
&= 2\alpha |\max_{s,a} A_\pi(s,a)| \\
&\leq 2\alpha \max_{s,a} |A_\pi(s,a)|
\end{aligned}$$

□

Lemma 15. Let (π, π') be an α-coupled policy pair. Then,

$$|\mathbb{E}_{s_t\sim\pi'}\left[\bar{A}(s_t)\right] - \mathbb{E}_{s_t\sim\pi}\left[\bar{A}(s_t)\right]| \leq 2\alpha \max_s |\bar{A}(s)| \leq 4\alpha(1-(1-\alpha)^t) \max_s |A_\pi(s,a)|.$$

Proof. Let n_t be the number of time steps such that $a_i \neq a'_i$ for $i < t$ and $a_i \sim \pi, a'_i \sim \pi'$. This denotes the number of times π and π' take different actions before time step t. Then,

$$\mathbb{E}_{s_t\sim\pi'}\left[\bar{A}(s_t)\right] = P(n_t = 0)\mathbb{E}_{s_t\sim\pi'|n_t=0}\left[\bar{A}(s_t)\right] + P(n_t > 0)\mathbb{E}_{s_t\sim\pi'|n_t>0}\left[\bar{A}(s_t)\right].$$

Similarly,

$$\mathbb{E}_{s_t\sim\pi}\left[\bar{A}(s_t)\right] = P(n_t = 0)\mathbb{E}_{s_t\sim\pi|n_t=0}\left[\bar{A}(s_t)\right] + P(n_t > 0)\mathbb{E}_{s_t\sim\pi|n_t>0}\left[\bar{A}(s_t)\right].$$

Now,

$$\mathbb{E}_{s_t\sim\pi'|n_t=0}\left[\bar{A}(s_t)\right] = \mathbb{E}_{s_t\sim\pi|n_t=0}\left[\bar{A}(s_t)\right]$$

since $n_t = 0 \implies \pi$ and π' executed the same actions on all time steps less than t. Then,

$$\begin{aligned}
&\mathbb{E}_{s_t\sim\pi'}\left[\bar{A}(s_t)\right] - \mathbb{E}_{s_t\sim\pi}\left[\bar{A}(s_t)\right] \\
&= P(n_t > 0)\left(\mathbb{E}_{s_t\sim\pi'|n_t>0}\left[\bar{A}(s_t)\right] - \mathbb{E}_{s_t\sim\pi|n_t>0}\left[\bar{A}(s_t)\right]\right).
\end{aligned}$$

Now, since π and π' are α-coupled, then, $\mathbb{P}(a = a' \mid s) \geq 1 - \alpha$, so $P(n_t = 0) \geq (1-\alpha)^t$ and

$$P(n_t > 0) \leq 1 - (1-\alpha)^t.$$

On the other hand, using triangle inequality,

$$\begin{aligned}
&|\mathbb{E}_{s_t \sim \pi' | n_t > 0}\left[\bar{A}(s_t)\right] - \mathbb{E}_{s_t \sim \pi | n_t > 0}\left[\bar{A}(s_t)\right]| && (2)\\
&\leq |\mathbb{E}_{s_t \sim \pi' | n_t > 0}\left[\bar{A}(s_t)\right]| + |\mathbb{E}_{s_t \sim \pi | n_t > 0}\left[\bar{A}(s_t)\right]| && (3)\\
&\leq 2 \max_s |\bar{A}(s)| && (4)\\
&\leq 4\alpha \max_{s,a} |A_\pi(s,a)| && (5)\\
& && (6)
\end{aligned}$$

Then,

$$\begin{aligned}
&|\mathbb{E}_{s_t \sim \pi'}\left[\bar{A}(s_t)\right] - \mathbb{E}_{s_t \sim \pi}\left[\bar{A}(s_t)\right]| \\
&= |P(n_t > 0)\left(\mathbb{E}_{s_t \sim \pi' | n_t > 0}\left[\bar{A}(s_t)\right] - \mathbb{E}_{s_t \sim \pi | n_t > 0}\left[\bar{A}(s_t)\right]\right)| \\
&\leq 4\alpha(1 - (1-\alpha)^t) \max_{s,a} |A_\pi(s,a)|.
\end{aligned}$$

□

Now we prove theorem 13:

Proof. Denote $\pi = \pi_{\text{old}}$ and $\pi' = \pi_{\text{new}}$. Let $\epsilon = \max_{s,a} |A_\pi(s,a)|$. Then, using performance difference lemma and the definition of $\bar{A}(s)$:

$$\begin{aligned}
V(\pi') - V(\pi) &= \mathbb{E}_{\tau \sim \pi'}\left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t)\right] \\
&= \mathbb{E}_{\tau \sim \pi'}\left[\sum_{t=0}^{\infty} \gamma^t \bar{A}(s_t)\right]
\end{aligned}$$

On the other hand,

$$\begin{aligned}
L_\pi(\pi') &= V(\pi) + \mathbb{E}_{s \sim p^\pi, a \sim \pi'}[A_\pi(s,a)] \\
&= V(\pi) + \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t \bar{A}(s_t)\right].
\end{aligned}$$

Combining:

$$\begin{aligned}
|V(\pi') - L_\pi(\pi')| &\leq \sum_{t=0}^{\infty} \gamma^t \left|\mathbb{E}_{\tau\sim\pi'}[\bar{A}(s_t)] - \mathbb{E}_{\tau\sim\pi}[\bar{A}(s_t)]\right| \\
&\sum_{t=0}^{\infty} \gamma^t 4\epsilon\alpha(1-(1-\alpha)^t) \\
&= 4\epsilon\alpha\left(\frac{1}{1-\gamma} - \frac{1}{1-\gamma(1-\alpha)}\right) \\
&= \frac{4\alpha^2\gamma\epsilon}{(1-\gamma)(1-\gamma(1-\alpha))} \\
&= \frac{4\alpha^2\gamma\epsilon}{(1-\gamma)(1-\gamma(1-\alpha))} \\
&\leq \frac{4\alpha^2\gamma\epsilon}{(1-\gamma)^2}.
\end{aligned}$$

Now, if we have two policies π and π' such that $\max_s D_{\mathrm{TV}}(\pi(\cdot \mid s)||\pi'(\cdot \mid s)) \leq \alpha$, then we can define an α-coupled policy with the appropriate marginals by Theorem 32. Therefore, take $\alpha = \max_s D_{\mathrm{TV}}(\pi(\cdot \mid s)||\pi'(\cdot \mid s))$, plug this into $\frac{4\alpha^2\gamma\epsilon}{(1-\gamma)^2}$ to conclude. □

From theorem 13, by noting that $D_{\mathrm{TV}}(p||q)^2 \leq D_{\mathrm{KL}}(p||q)$, we get that

$$V(\pi') \geq L_\pi(\pi') - \frac{4\epsilon\gamma}{(1-\gamma)^2} D_{\mathrm{KL}}^{\max}(\pi, \pi').$$

Using this, we can find a preliminary algorithm:

Algorithm: Policy iteration guaranteeing non-decreasing expected return V

```
1: **Initialize** policy $\pi_0$
2: **for** $i = 0, 1, 2, \ldots$ until convergence **do**
3:     Compute all advantage values $A_{\pi_i}(s,a)$
4:     Solve the constrained optimization problem:
         $\pi_{i+1} = \arg\max_\pi [L_{\pi_i}(\pi) - C D_{\mathrm{KL}}^{\max}(\pi_i, \pi)]$
         where $C = \frac{4\epsilon\gamma}{(1-\gamma)^2}$
         and $L_{\pi_i}(\pi) = V(\pi_i) + \sum_s p^{\pi_i}(s) \sum_a \pi(a|s) A_{\pi_i}(s,a)$
5: **end for**
```

This algorithm has guaranteed monotonic improvement. We can now use it to derive TRPO. Let $V(\theta) := V(\pi_\theta)$, $L_\theta(\theta') := L_{\pi_\theta}(\pi_{\theta'})$, and $D_{\mathrm{KL}}(\theta \,\|\, \theta') := D_{\mathrm{KL}}(\pi_\theta \,\|\, \pi_{\theta'})$. We saw that:

$$V(\theta) \geq L_{\theta_{\mathrm{old}}}(\theta) - C D_{\mathrm{KL}}^{\max}(\theta_{\mathrm{old}}, \theta)$$

with equality at $\theta = \theta_{\text{old}}$. Thus, we can improve $V(\theta)$ by solving the following optimization problem:

$$\max_\theta \; L_{\theta_{\text{old}}}(\theta) - C D_{\text{KL}}^{\max}(\theta_{\text{old}}, \theta)$$

However, choosing the step size here is tricky - if we use $C = \frac{4\epsilon\gamma}{(1-\gamma)^2}$, then the step sizes become small. Instead, we solve the following with a trust region constraint:

$$\begin{aligned} &\max_\theta \; L_{\theta_{\text{old}}}(\theta) && (7)\\ &\text{subject to } D_{\text{KL}}^{\max}(\theta_{\text{old}}, \theta) \le \delta && (8) \end{aligned}$$

In practice, we use the average KL divergence: $\bar{D}_{\text{KL}}^{p^{\theta_{\text{old}}}}(\theta_{\text{old}}, \theta) = \mathbb{E}_{s \sim p^{\theta_{\text{old}}}}[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot \mid s) || \pi_\theta(\cdot \mid s))]$

Lastly, we show how to estimate the objective and constraint functions using Monte Carlo simulation. We replace the objective $\sum_s p^{\theta_{\text{old}}}(s) \sum_a \pi_\theta(a \mid s) A_{\theta_{\text{old}}}(s,a)$ with $\frac{1}{1-\gamma}\mathbb{E}_{s\sim\theta_{\text{old}}, a\sim\pi_\theta(\cdot|s)}[A_{\theta_{\text{old}}}(s,a)]]$. Lastly, we use importance sampling. Altogether, we have:

$$\begin{aligned} &\max_\theta \mathbb{E}_{s\sim p^{\theta_{\text{old}}},\, a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} A_{\theta_{\text{old}}}(s,a)\right] && (9)\\ &\text{subject to } \mathbb{E}_{s\sim p^{\theta_{\text{old}}}}\left[D_{\text{KL}}\left(\pi_{\theta_{\text{old}}}(\cdot \mid s) \,\|\, \pi_\theta(\cdot \mid s)\right)\right] \le \delta && (10) \end{aligned}$$

The alternative version is the optimize the following:

$$\max_\theta \mathbb{E}_{s\sim p^{\theta_{\text{old}}},\, a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} A_{\theta_{\text{old}}}(s,a) - \beta \cdot D_{\text{KL}}\left(\pi_{\theta_{\text{old}}}(\cdot \mid s) \,\|\, \pi_\theta(\cdot \mid s)\right)\right] \quad (11)$$
$$(12)$$

The second version comes from the first via the method of Lagrange multipliers so these are two equivalent formulations. However, comparing these two versions, tuning δ (in the first version) is generally easier than tuning β in the second version. Also, for the second verison in practice, TRPO uses the *maximum* over KL divergences conditioned on each state instead of the mean. In other words, optimize:

$$\max_\theta \mathbb{E}_{s\sim p^{\theta_{\text{old}}},\, a\sim\pi_{\theta_{\text{old}}}}\left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} A_{\theta_{\text{old}}}(s,a)\right] - \beta \cdot \max_s \left[D_{\text{KL}}\left(\pi_{\theta_{\text{old}}}(\cdot \mid s) \,\|\, \pi_\theta(\cdot \mid s)\right)\right] \quad (13)$$
$$(14)$$

TRPO solves this optimization problem by making a linear approximation to the objective $L_{\pi_{\theta_{\text{old}}}}(\pi_\theta)$, a quadratic approximation to the constraint and then using a conjugate gradient algorithm.

Then, the problem becomes:

$$\max_\theta \quad g \cdot (\theta - \theta_{\text{old}}) - \frac{\beta}{2}(\theta - \theta_{\text{old}})^T F(\theta - \theta_{\text{old}})$$

where $g = \frac{\partial}{\partial\theta} L_{\pi_{\theta_{\text{old}}}}(\pi_\theta)\Big|_{\theta=\theta_{\text{old}}}$ and $F = \frac{\partial^2}{\partial\theta^2} \text{KL}_{\pi_{\theta_{\text{old}}}}(\pi_\theta)\Big|_{\theta=\theta_{\text{old}}}$.

This problem can be solved efficiently using the conjugate gradient algorithm.

2.10 Proximal Policy Optimization (PPO)

(This section is written independently of the previous section on trust regional policy optimization so that it is self-complete, which means there are some repetitions here)

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

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

With this, TRPO maximises $\mathcal{L}_\pi(\pi')$ subject to $\mathbb{E}_{s\sim\pi}\left[D_{KL}(\pi(\cdot|s)||\pi'(\cdot|s))\right] \le \delta$. 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 [4] 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

$$\mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1}\left[\min(r_t(\theta)A^{\pi_\theta}(s_t,a_t), \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A^{\pi_\theta}(s_t,a_t))\right]\right]$$

.

The second variant is more commonly used. In practical implementations, we often add an entropy bonus (similar to SAC) to encourage exploration and prevent collapsing to a local optimal.

Intuition: let's see what this algorithm really does. Consider the vanilla algorithm that maximizes $\mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_t r_t(\theta)A^{\pi_\theta}(s_t,a_t)\right]$. If we take the gradient of this with respect to θ', then, we get $\mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_t r_t(\theta)\nabla_{\theta'}(\log \pi_{\theta'}(a_t \mid s_t))A^{\pi_\theta}(s_t,a_t)\right]$. Now, if $A^{\pi_\theta}(s_t,a_t)$ is positive, this means we take a gradient step in the direction given by $\nabla_{\theta'}(\log \pi_{\theta'}(a_t \mid s_t))$. If this step is too large, then we will have left the "trust region" in TRPO where $\pi_{\theta'}$ is close to π_θ. If we compute the gradient of the PPO objective, we can see that when $A^{\pi_\theta}(s_t,a_t)$ and whenever $r_t(\theta) > 1+\epsilon$, the gradient becomes zero i.e. whenver we have left the trust region, we do not update policy anymore suggesting that we need new data and cannot use our old data from previous policy anymore.

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

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:

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

Algorithm: PPO (Actor-Critic Style)

```
1: for iteration = 1, 2, ... do
2:     for actor = 1, 2, ..., N do
3:         Run policy $\pi_{\theta_{\text{old}}}$ in environment for $T$ timesteps
4:         Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
5:     end for
6:     Optimize surrogate objective $L$ with respect to $\theta$, using $K$ epochs and minibatch
   size $M \leq NT$
7:     $\theta_{\text{old}} \leftarrow \theta$
8: end for
```

Algorithm: PPO with GAE

```
1: Initialize parameters $\phi$, environment state $s$
2: for iteration $k = 1, 2, \ldots$ do
3:     $\phi_{\text{old}} \leftarrow \phi$
4:     $(\tau, s) = \text{rollout}(s, \pi_{\phi_{\text{old}}})$
5:     $(s_1, a_1, r_1, \ldots, s_T) = \tau$
6:     $v_t = V_\phi(s_t)$ for $t = 1 : T$
7:     $(A_{1:T}, y_{1:T}) = \text{GAE}(r_{1:T}, v_{1:T}, \gamma, \lambda)$
8:     for $m = 1 : M$ do
9:         $\rho_t = \frac{\pi_\phi(a_t|s_t)}{\pi_{\phi_{\text{old}}}(a_t|s_t)}$ for $t = 1 : T$
10:        $\tilde{\rho}_t = \text{clip}(\rho_t)$ for $t = 1 : T$
11:        $\mathcal{L}(\phi) = \frac{1}{T}\sum_{t=1}^{T}\left[\lambda_{TD}(V_\phi(s_t) - y_t)^2 - \lambda_{PG}\min(\rho_t A_t, \tilde{\rho}_t A_t) - \lambda_{\text{ent}}\,\mathbb{H}(\pi_\phi(\cdot|s_t))\right]$
12:        $\phi \leftarrow \phi - \eta\nabla_\phi\mathcal{L}(\phi)$
13:    end for
14: end for
```

2.11 Soft Actor-Critic (SAC)

Soft Actor-Critic [5] is an off-policy actor-critic algorithm that aims to learn a policy that maximizes rewards while also acting as stochastically as possible. In other words, if there are two actions that both achieve the same maximum rewards, our goal would be to assign nearly equal probability mass to both of them. This falls under a general framework called maximum entropy reinforcement learning where the goal is to maximize (assume a finite horizon task with discount factor $\gamma = 0$):

$$J(\theta) = \sum_{t=0}^{T} \mathbb{E}_{s_t \sim p^{\pi_\theta}, a_t \sim \pi_\theta}\left[r(s_t, a_t) + \alpha H(\pi(\cdot \mid s_t))\right] \tag{15}$$

where $H(p)$ is the entropy of the distribution p.

There are many benefits to learning such policies. Firstly, sometimes we would want our agents to learn multiple strategies for solving a task. Imagine building an agent that plays chess - it would be a problem if the agent acts deterministically (or close to deterministically) because then its opponent could very well predict its moves. Secondly, a highly stochastic policy would also end up doing more exploration allowing us to potentially uncover even more optimal strategies as training progresses, instead of collapsing to suboptimal ones.

First, we define the modified Bellman backup operator that gets us the soft Q-value corresponding to $J(\theta)$ as in equation 15:

Definition 9. (Bellman backup for soft Q-value). The soft Q-value of a fixed policy π can be computed by repeatedly applying the modified Bellman backup operator:

$$\tau^{\pi} Q_{\pi}(s_t, a_t) := r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p(\cdot|s_t, a_t)} \left[V(s_{t+1}) \right]$$

where

$$V(s_t) = \mathbb{E}_{a_t \sim \pi} \left[Q_{\pi}(s_t, a_t) - \log \pi(a_t \mid s_t) \right].$$

The fact that this converges comes from the following result:

Lemma 16. For any mapping $Q_{\pi}^0 : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ with $|\mathcal{A}| < \infty$ and $Q_{\pi}^{k+1} = \tau^{\pi} Q_{\pi}^k$. Then, the sequence Q_{π}^k converges to the soft Q-value of π as $k \to \infty$.

Proof. Define $r_{\pi}(s_t, a_t) = r(s_t, a_t) + \mathbb{E}_{s_{t+1} \sim P(\cdot|s_t, a_t)} \left[H(\pi(\cdot|s_{t+1})) \right]$. Then, $Q_{\pi}(s_t, a_t) \leftarrow r_{\pi}(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim P, a_{t+1} \sim \pi} \left[Q(s_{t+1}, a_{t+1}) \right]$ is the update rule. Given $|\mathcal{A}| < \infty$, the entropy augment reward r_{π} is bounded. Using this, we can show the convergence as required following similar steps as for classical policy evaluation. $\square$

Given this evaluation, we can then do policy improvement as follows:

$$\pi_{\text{new}} = \arg\min_{\pi' \in \Pi} D_{\text{KL}} \left(\pi'(\cdot \mid s_t) \;||\; \frac{\exp(Q_{\pi_{\text{old}}}(s_t, \cdot))}{Z^{\pi_{\text{old}}}(s_t)} \right)$$

where $Z^{\pi_{\text{old}}}$ is the partition function that normalizes the distribution. We can prove that this results in an improved policy as follows:

Lemma 17. Let $\pi_{\text{old}} \in \Pi$ and let π_{new} be the resulting policy from the soft-policy improvement step. Then, $Q^{\pi_{\text{new}}}(s_t, a_t) \geq Q^{\pi_{\text{old}}}(s_t, a_t)$ for all $(s_t, a_) \in \mathcal{S} \times \mathcal{A}$ with $|\mathcal{A}| < \infty$.

Proof. Given π_{old} with corresponding $Q^{\pi_{\text{old}}}$ and $V^{\pi_{\text{old}}}$, we can define

$$\begin{aligned} \pi_{\text{new}}(\cdot \mid s_t) &= \arg\min_{\pi' \in \Pi} D_{\text{KL}}(\pi'(\cdot|s_t) || \exp(Q^{\pi_{\text{old}}}(s_t, \cdot) - \log Z^{\pi_{\text{old}}}(s_t))) \\ &=: \arg\min_{\pi' \in \Pi} J_{\pi_{\text{old}}}(\pi'(\cdot|s_t)). \end{aligned}$$

Now, $J_{\pi_{\text{old}}}(\pi_{\text{new}}(\cdot|s_t)) \leq J_{\pi_{\text{old}}}(\pi_{\text{old}}(\cdot|s_t))$ since we could choose $\pi_{\text{new}} = \pi_{\text{old}}$ in our minimization process. Therefore, expanding this line, we get:

$$
\begin{aligned}
&\mathbb{E}_{a_t \sim \pi_{\text{new}}}[\log \pi_{\text{new}}(a_t \mid s_t) - Q^{\pi_{\text{old}}}(s_t, a_t) + \log Z^{\pi_{\text{old}}}(s_t)] \\
&\quad \le \mathbb{E}_{a_t \sim \pi_{\text{old}}}[\log \pi_{\text{old}}(a_t \mid s_t) - Q^{\pi_{\text{old}}}(s_t, a_t) + \log Z^{\pi_{\text{old}}}(s_t)]
\end{aligned}
$$

Since $Z^{\pi_{\text{old}}}$ is independent of a_t, this reduces to

$$
\mathbb{E}_{a_t \sim \pi_{\text{new}}}[Q^{\pi_{\text{old}}}(s_t, a_t) - \log \pi_{\text{new}}(a_t \mid s_t)] \ge V^{\pi_{\text{old}}}(s_t).
$$

Then,

$$
\begin{aligned}
Q^{\pi_{\text{old}}}(s_t, a_t) &= r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim P}\left[V^{\pi_{\text{old}}}(s_{t+1})\right] \\
&\le r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim P}\left[\mathbb{E}_{a_{t+1} \sim \pi_{\text{new}}}[Q^{\pi_{\text{old}}}(s_{t+1}, a_{t+1}) - \log \pi_{\text{new}}(a_{t+1} \mid s_{t+1})]\right]. \\
&\dots \\
&\dots \\
&\le Q^{\pi_{\text{new}}}(s_t, a_t)
\end{aligned}
$$

where the expansion uses $\mathbb{E}_{a_t \sim \pi_{\text{new}}}[Q^{\pi_{\text{old}}}(s_t, a_t) - \log \pi_{\text{new}}(a_t \mid s_t)] \ge V^{\pi_{\text{old}}}(s_t)$. The convergence follows from the previous lemma. □

The soft-policy iteration algorithm essentially works like policy iteration except we alternate between soft policy evaluation and soft policy improvement as described above. This algorithm will, in fact, converge to the optimal policy as proven in the following theorem:

Theorem 18. The soft policy iteration algorithm over $\pi \in \Pi$ converges to a policy π^* such that $Q^{\pi^*}(s_t, a_t) \ge Q^{\pi}(s_t, a_t)$ for all $\pi \in \Pi$ and $(s_t, a_t) \in \mathcal{S} \times \mathcal{A}$, assuming $|\mathcal{A}| < \infty$.

Proof. We know by the previous lemma that Q^{π_i} is monotonically increasing. However, Q^{π} is bounded above since the reward and entropy (given $|\mathcal{A}| < \infty$) are bounded above. Therefore, the policy iteration will converge to some π^*. Now, we claim that this policy is optimal. Since π^* is the policy at convergence, we have that $J_{\pi^*}(\pi^*(\cdot|s_t)) < J_{\pi^*}(\pi(\cdot|s_t))$ for all $\pi \in \Pi$ such that $\pi \ne \pi^*$. From this, we can easily show that $Q^{\pi^*}(s_t, a_t) > Q^{\pi}(s_t, a_t)$ for all s_t and a_t. □

The catch, however, is that performing the exact optimization in each of these steps may not be tractable. We now describe the practical algorithm we get from SAC.

In SAC, we parametrize three function: $V_\psi(s_t)$ (to estimate the soft state-value function), $Q_\phi(s_t, a_t)$ (to estimate the soft state-action value function) 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. In practice, $V_\psi(s_t)$ and $Q_\phi(s_t, a_t)$ could be neural networks where as the policy could be Gaussian with mean and covariance given by neural networks.

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 the following unbiased estimator:

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

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 computed as the estimator:

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

Finally, we can find the policy by minimizing the expected KL-divergence:

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}}\left[D_{\mathrm{KL}}\left(\pi_\phi(\cdot \mid s_t) \mid\mid \frac{\exp\left(Q_\theta(s_t, \cdot)\right)}{Z_\theta(s_t)}\right)\right].$$

To minimize this, we use the reparametrization trick: let $a_t = f_\phi(\epsilon_t; s_t)$ where ϵ_t is an input noise vector sampled from a fixed distribution (say, normal). Then,

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}, \boldsymbol{\epsilon}_t \sim \mathcal{N}}\left[\log \pi_\phi\left(f_\phi(\boldsymbol{\epsilon}_t; \mathbf{s}_t) \mid \mathbf{s}_t\right) - Q_\theta\left(\mathbf{s}_t, f_\phi(\boldsymbol{\epsilon}_t; \mathbf{s}_t)\right)\right].$$

Here we ignored the partition function since it does not depend on ϕ.

Then,

$$\hat{\nabla}_\phi J_\pi(\phi) = \nabla_\phi \log \pi_\phi(\mathbf{a}_t \mid \mathbf{s}_t) + \left(\nabla_{\mathbf{a}_t} \log \pi_\phi(\mathbf{a}_t \mid \mathbf{s}_t) - \nabla_{\mathbf{a}_t} Q_\theta(\mathbf{s}_t, \mathbf{a}_t)\right) \nabla_\phi f_\phi(\boldsymbol{\epsilon}_t; \mathbf{s}_t),$$

Algorithm 1: Soft Actor-Critic

```
1:  Initialize parameter vectors $\psi$, $\bar{\psi}$, $\theta$, $\phi$
2:  for each iteration do
3:      for each environment step do
4:          $a_t \sim \pi_\phi(a_t \mid s_t)$
5:          $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$
6:          $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
7:      end for
8:      for each gradient step do
9:          $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
10:         for $i \in \{1, 2\}$ do
11:             $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$
12:         end for
13:         $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
14:         $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$
15:     end for
16: end for
```

2.12 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:

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

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,

$$\nabla_\theta J(\mu_\theta) = \int_{\mathcal{S}} p^{\mu_\theta}(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu_\theta}(s,a)|_{a=\mu_\theta(s)} ds = \mathbb{E}_{s\sim p^{\mu_\theta}}\left[\nabla_\theta \mu_\theta(s) \nabla_a Q^{\mu_\theta}(s,a)|_{a=\mu_\theta(s)}\right] \tag{16}$$

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 16). 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 $Q^w(s,a)$:

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

3 Tabular MDP

3.1 Policy Evaluation

In this subsection, we will look at policy evaluation methods for when we know the underlying environment dynamics $P(s_{t+1} \mid s_t, a_t)$ and the environment's reward model.

First, why do we care about policy evaluation? Even if we solely care about finding the optimal policy, policy evaluation can be very useful. Generally, if we can evaluate a policy, we should be able to update the policy. Suppose we have a policy π. If we know $Q_\pi(s, a)$, then we can improve the policy. For example, $\pi_{new}(a \mid s) = 1\{a = \arg\max_{a'} Q^\pi(s, a')\}$.

The question is - how do we evaluate a given policy π? We can do this via the Bellman backup. We initialize $V_\pi(s) = 0$ for all states and then iteratively update V_π at every state using the reward model and dynamics model - both of which we are assuming we know.

Algorithm: Policy Evaluation of policy π

```
1: Initialize $V_\pi(s) = 0$ for all states $s$
2: loop
3:     for all $s \in \mathcal{S}$ do
4:         $V_\pi(s) \leftarrow \sum_a \pi(a|s) \left[r(s, a) + \gamma \sum_{s'} P(s'|s, a) V_\pi(s')\right]$
5:     end for
6:     Until Convergence
7: end loop
```

This update is called the **Bellman backup** for a particular policy: $V_\pi^{k+1}(s) = B_\pi V_\pi^k(s)$ where $V_\pi^k(s)$ is the value function estimation at iteration k of our algorithm. To do policy evaluation, we repeatedly apply the Bellman operator B_π to $V(s) = 0$ i.e $V_\pi = B_\pi B_\pi \cdots B_\pi V$ where $B_\pi V(s) = \mathbb{E}_{a \sim \pi(a|s)} \left[r(s, a) + \gamma \sum_{s'} P(s'|s, a) V(s')\right]$.

Lemma 19. (Bellman Backup is a contraction). For $\gamma < 1$ or for finite horizon tasks,

$$||B^\pi V - B^\pi V'||_\infty \leq \gamma ||V - V'||_\infty.$$

Proof. For any state s, we have that:

$$\begin{aligned}
|B^\pi V(s) - B^\pi V'(s)| &= \left|\mathbb{E}_{a\sim\pi}\left[r(s,a) + \gamma\sum_{s'} P(s'|s,a)V(s')\right] - \mathbb{E}_{a'\sim\pi}\left[r(s,a') + \gamma\sum_{s''} P(s''|s,a')V'(s'')\right]\right| \\
&\leq \left|\gamma\sum_{a,s'} \pi(a \mid s)P(s'|s,a)(V(s') - V'(s'))\right| \\
&\leq |\gamma|\left|\sum_{a,s'} \pi(a \mid s)P(s'|s,a)(V(s') - V'(s'))\right| \\
&= \gamma\,|V - V'|_\infty
\end{aligned}$$

Since this is true for any state s, we conclude the desired result. ☐

Proposition 20. (Convergence of Policy Evaluation). Suppose $\gamma < 1$ and suppose $||V - V'|| < \infty$. Then,

$$\lim_{n\to\infty} (B^\pi)^n V = V_\pi$$

Proof. First, note that $B^\pi V_\pi = V_\pi$. Therefore, $V_\pi = (B^\pi)^n V_\pi$. Now, we write:

$$\begin{aligned}
\lim_{n\to\infty} ||(B^\pi)^n V - V_\pi||_\infty &= \lim_{n\to\infty} ||(B^\pi)^n V - (B^\pi)^n V_\pi||_\infty \\
&\leq \lim_{n\to\infty} \gamma||(B^\pi)^{n-1} V - (B^\pi)^{n-1} V_\pi||_\infty \\
&\leq \lim_{n\to\infty} \gamma^n ||V - V_\pi||_\infty \\
&= 0.
\end{aligned}$$

☐

3.2 Policy Iteration

In policy iteration algorithms, we alternate between evaluating a policy and then improving the policy using the evaluation. We usually do so by estimating $q_\pi(s,a)$.

Assuming we know the model dynamics $P(s_{t+1} \mid s_t, a_t)$, we could easily do this using the policy iteration algorithm:

```
Algorithm: Policy Iteration
1: Set $i = 0$
2: Initialize $\pi_0(s)$ randomly for all $s \in \mathcal{S}$
3: while $i = 0$ or $|\pi_i - \pi_{i-1}| > 0$ do
4:     $V_{\pi_i} \leftarrow$ Policy Evaluation of $\pi_i$:
5:     for all $s, a$ do
6:         $q_{\pi_i}(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) V_{\pi_i}(s')$
7:     end for
8:     $\pi_{i+1} \leftarrow$ Policy Improvement:
9:     for all $s$ do
10:        $\pi_{i+1}(s) = \arg\max_a q_{\pi_i}(s,a)$
11:    end for
12:    $i = i + 1$
13: end while
```

Note: This method leads to deterministic policies since we define $\pi(s) := \arg\max_a q_\pi(s,a)$.

Now we analyse this algorithm and show the monotonic improvement of policy learned via policy iteration.

We say $V_{\pi_a} \geq V_{\pi_b}$ if and only if $V_{\pi_a}(s) \geq V_{\pi_b}(s)$ for all $s \in \mathcal{S}$.

Lemma 21. (Monotonic improvement). $V_{\pi_{i+1}} \geq V_{\pi_i}$ with strict inequality if π_i is suboptimal, where π_{i+1} is the new policy we get after policy improvement on π_i.

Proof.

$$\begin{aligned}
V_{\pi_i}(s) &\leq \max_a q_{\pi_i}(s,a) \\
&= \max_a r(s,a) + \gamma \sum_{s'} P(s'|s,a) V_{\pi_i}(s') \\
&= r(s, \pi_{i+1}(s)) + \gamma \sum_{s'} P(s'|s, \pi_{i+1}(s)) V_{\pi_i}(s') \\
&\leq r(s, \pi_{i+1}(s)) + \gamma \sum_{s'} P(s'|s, \pi_{i+1}(s)) \max_{a'} q_{\pi_i}(s', a') \\
&= r(s, \pi_{i+1}(s)) + \gamma \sum_{s'} P(s'|s, \pi_{i+1}(s)) \left[r(s', \pi_{i+1}(s')) + \gamma \sum_{s"} P(s"|s', \pi_{i+1}(s')) V_\pi(s") \right] \\
&\dots \\
&= V_{\pi_{i+1}}(s)
\end{aligned}$$

□

The maximum number of iterations that the policy iteration algorithm can run for is $|\mathcal{A}|^{|\mathcal{S}|}$.

This is because this is the maximum number of policies possible for the MDP (since we get deterministic policies) and since we have monotonic improvement, no two policies will be the same (unless it is the optimal policy).

3.3 Value Iteration

So far, we evaluated a particular policy π, then improved the policy, and evaluated the new policy again in a loop. In particular, we take one policy and evaluate it; to do so, we require looking at trajectories that the policy would take.

The next algorithm will aim towards approximating the value of the optimal policy directly. These algorithms are called ***off-policy***, whereas policy iteration is called an ***on-policy*** method.

Algorithm: Value Iteration

```
1: Set $k = 1$
2: Initialize $V_0(s) = 0, \forall s$
3: **repeat**
4:     **for all** $s \in \mathcal{S}$ **do**
5:         $V_{k+1}(s) = \max_a \left(r(s,a) + \gamma \sum_{s'} P(s'|s,a)V_k(s')\right)$
6:     **end for**
7:     **for all** $s \in \mathcal{S}$ **do**
8:         $\pi_{k+1}(s) = \arg\max_a \left(r(s,a) + \gamma \sum_{s'} P(s'|s,a)V_k(s')\right)$
9:     **end for**
10:    $k = k + 1$
11: **until** convergence
```

The value function update can be written using the **Bellman backup operator** which, when applied to a value function, returns a new value function that improves on the input value function if possible:

$$BV_k(s) := \max_a r(s,a) + \gamma \sum_{s'} P(s'|s,a)V_k(s')$$

and then

$$V_{k+1}(s) = BV_k(s).$$

Furthermore, for a greedy policy π,

$$B_\pi V = BV.$$

Lemma 22. (Bellman Backup is a contraction). For $\gamma < 1$ or for finite horizon tasks,

$$||BV - BV'||_\infty \leq \gamma ||V - V'||_\infty.$$

Proof. For any state s, we have that

$$\begin{aligned}
|BV(s) - BV'(s)| &= \left|\max_a \left(r(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s')\right) - \max_{a'} \left(r(s,a') + \gamma \sum_{s''} P(s''|s,a')V'(s'')\right)\right| \\
&\leq \max_a \left|\left(r(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s')\right) - \left(r(s,a) + \gamma \sum_{s'} P(s'|s,a)V'(s')\right)\right| \\
&\leq \max_a \left|\gamma \sum_{s'} P(s'|s,a)(V(s') - V'(s'))\right| \\
&\leq \max_a |\gamma| \left|\sum_{s'} P(s'|s,a)(V(s') - V'(s'))\right| \\
&= \gamma \left|V - V'\right|_\infty
\end{aligned}$$

□

Proposition 23. (Convergence of Value Iteration). Suppose $\gamma < 1$ and suppose $||V - V_*|| < \infty$. Then,

$$\lim_{n\to\infty} B^n V = V_*$$

Proof. First, note that $BV_* = V_*$. Therefore, $V_* = B^n V_*$. Now, we write:

$$\begin{aligned}
\lim_{n\to\infty} ||B^n V - V_*|| &= \lim_{n\to\infty} ||B^n V - B^n V_*|| \\
&\leq \lim_{n\to\infty} \gamma ||B^{n-1} V - B^{n-1} V_*|| \\
&\leq \lim_{n\to\infty} \gamma^n ||V - V_*|| \\
&= 0.
\end{aligned}$$

□

There are two main limitations of exact solutions methods like policy and value iteration.

1. They require access to dynamics, or the transition model $P(s'|s,a)$.
   - We can use *sampling-based approximations* to collect experiences and learn the dynamics if they're not provided.
2. Looping requires iteration over all states and actions, which is impractical for large MDPs.
   - We can use *Q-value or state-value-function fitting* over tabular approaches. Instead of storing a table with all values, we have a function that takes in states and outputs corresponding value.

3.4 Useful Results on Bellman Operators

Proposition 24. For any policy π and value function V, we have

$$||V - V_*|| \le \frac{||V - BV||}{1-\gamma}$$

and

$$||V - V_\pi|| \le \frac{||V - B_\pi V||}{1-\gamma}.$$

Proof.

$$\begin{aligned} ||V - V_*|| &= ||V - BV + BV - V_*|| \\ &\le ||V - BV|| + ||BV - V_*|| \\ &= ||V - BV|| + ||BV - BV_*|| \\ &\le ||V - BV|| + \gamma\,||V - V_*|| \\ ||V - V_*|| &\le \frac{||V - BV||}{1-\gamma}. \end{aligned}$$

The second equation can be proven in the same way. □

Proposition 25. If π is greedy, $V_\pi(s) \ge V_*(s) - \frac{2\epsilon}{1-\gamma}$ where $\epsilon = ||BV - V||$.

Proof. Recall $B^\pi V = BV$ for a greedy policy π.

$$\begin{aligned} V_*(s) - V_\pi(s) &\le ||V_*(s) - V_\pi(s)|| \\ &= ||V_*(s) - V(s) + V(s) - V_\pi(s)|| \\ &\le ||V_*(s) - V(s)|| + ||V(s) - V_\pi(s)|| \\ &\le \frac{||V - BV||}{1-\gamma} + \frac{||V - B^\pi V||}{1-\gamma} \\ &= \frac{2\epsilon}{1-\gamma}. \end{aligned}$$

□

Proposition 26. If $V \ge V_*$, then $V_\pi(s) \ge V_*(s) - \frac{\epsilon}{1-\gamma}$.

Proof.

$$V_*(s) - V_\pi(s) \le V(s) - V_\pi(s) \le ||V(s) - V_\pi(s)|| \le \frac{\epsilon}{1-\gamma}.$$

□

Proposition 27. For π a greedy policy,

$$V_\pi(s) \geq V_*(s) - \frac{2\epsilon\gamma}{1-\gamma}.$$

Proof. We know

$$||V - V_\pi|| \leq \frac{\epsilon}{1-\gamma}$$

and

$$||V - V_*|| \leq \frac{\epsilon}{1-\gamma}.$$

Rewrite this as $V(s) \leq V_\pi(s) + \frac{\epsilon}{1-\gamma}$ (since $V(s) - V_\pi(s) \leq ||V(s) - V_\pi(s)|| \leq \frac{\epsilon}{1-\gamma}$) and similarly $V_*(s) \leq V(s) + \frac{\epsilon}{1-\gamma}$.

Then, we write

$$\begin{aligned}
V_*(s) &= r(s,a) + \gamma \sum_{s'} P(s'|s,a)V_*(s') \\
&\leq r(s,a) + \gamma \sum_{s'} P(s'|s,a)(V(s') + \frac{\epsilon}{1-\gamma}) \\
&= r(s,a) + \gamma \sum_{s'} P(s'|s,a)V(s') + \frac{\gamma\epsilon}{1-\gamma} \\
&\leq r(s,a) + \gamma \sum_{s'} P(s'|s,a)(V_\pi(s') + \frac{\epsilon}{1-\gamma}) + \frac{\gamma\epsilon}{1-\gamma} \\
&= r(s,a) + \gamma \sum_{s'} P(s'|s,a)V_\pi(s') + \frac{\gamma\epsilon}{1-\gamma} + \frac{\gamma\epsilon}{1-\gamma} \\
&= V_\pi(s) + \frac{2\gamma\epsilon}{1-\gamma}.
\end{aligned}$$

□

4 Value-based Methods

Now we look at how to evaluate algorithms when we do not know the underlying dynamics models or reward model. In this case, in order to evaluate the policy, we need to rollout the policy i.e. get trajectories under policy π, observe rewards and *then* evaluate the policy π. Having evaluated the policy, we seek to improve the policy using some sort of policy improvement step. Such methods are *on-policy* methods - we require trajectories sampled using a specific policy π to be able to evaluate π which we then improve to get π_{new}.

There are also methods that are called *off-policy* - these methods can improve a policy π to π_{new} using trajectories sampled by a host of other policies, not necessarily just π. As such, these methods get greater sample efficiency.

4.1 Monte Carlo Policy Evaluation

The idea of Monte Carlo policy evaluation is simple - sample multiple trajectories using a policy, observe the rewards gotten from a state s to compute $V_\pi(s)$.

Algorithm: First-Visit Monte Carlo (MC) Policy Evaluation

```
1: Initialize $N(s) = 0$, $G(s) = 0$ for all $s \in \mathcal{S}$
2: loop
3:     Sample episode $i = s_{i,1}, a_{i,1}, r_{i,1}, s_{i,2}, a_{i,2}, r_{i,2}, \dots, s_{i,T_i}, a_{i,T_i}, r_{i,T_i}$
4:     for each time step $t$ until $T_i$ (the end of episode $i$) do
5:         Define $G_{i,t} = r_{i,t} + \gamma r_{i,t+1} + \gamma^2 r_{i,t+2} + \cdots + \gamma^{T_i - t} r_{i,T_i}$
6:         if this is the first time $t$ that state $s$ is visited in episode $i$ then
7:             Increment counter: $N(s) \leftarrow N(s) + 1$
8:             Increment total return: $G(s) \leftarrow G(s) + G_{i,t}$
9:             Update estimate: $V_\pi(s) \leftarrow \frac{G(s)}{N(s)}$
10:        end if
11:    end for
12:    Until Convergence
13: end loop
```

In this algorithm, the learned $\hat{V}_\pi(s)$ becomes an unbiased estimator of the true value function at state s, $V_\pi(s)$. However, this also has high variance.

There are multiple variations of this idea that aim to make the learning more stable. For example, here is another algorithm:

Algorithm: Incremental Monte Carlo (MC) Policy Evaluation

```
1: loop
2:     Sample episode $i = s_{i,1}, a_{i,1}, r_{i,1}, s_{i,2}, a_{i,2}, r_{i,2}, \ldots, s_{i,T_i}, a_{i,T_i}, r_{i,T_i}$
3:     for $t = 1$ to $T_i$ do
4:         Compute return: $G_{i,t} = r_{i,t} + \gamma r_{i,t+1} + \gamma^2 r_{i,t+2} + \cdots + \gamma^{T_i - t} r_{i,T_i}$
5:         Update estimate: $V^\pi(s_{i,t}) \leftarrow V^\pi(s_{i,t}) + \alpha \left(G_{i,t} - V^\pi(s_{i,t})\right)$
6:     end for
7:     Until Convergence
8: end loop
```

Note that generally Monte Carlo approaches require episodic settings - you need a full trajectory to be able to make an update to your value estimate.

4.2 TD Learning

Now we take a look at a model-free policy evaluation algorithm that does not require function approximation. TD Learning is an algorithm for learning the value function corresponding to a policy when we do not know the dynamics of the environment. The main idea is to use bootstrapping: update $V_\pi(s_t) \leftarrow V_\pi(s_t) + \alpha((r_t + \gamma V_\pi(s_{t+1})) - V_\pi(s_t))$. With this, we have the algorithm:

```
Algorithm: TD(0) Learning to evaluate policy $\pi$

1: Initialize $V_\pi(s) = 0$ for all states $s$
2: loop
3:     for all $s \in \mathcal{S}$ do
4:         $V_\pi(s_t) \leftarrow V_\pi(s_t) + \alpha((r_t + \gamma V_\pi(s_{t+1})) - V_\pi(s_t))$
5:     end for
6:     Until Convergence
7: end loop
```

When analysing this family of algorithms, we often require the **TD(0) error**:

$$\delta_t = r_t + \gamma V_\pi(s_{t+1}) - V_\pi(s_t).$$

The first major benefit of TD(0) learning is that we can update our value function immediately after observing (s_t, a_t, r_t, s_{t+1}) - we do not need to wait until the end of the trajectory. This also means that TD(0) learning can easily be used for non-episodic/continuing settings (i.e. the task horizon goes to infinity).

4.3 Q-learning

The rough idea of our algorithm will be as follows. We want to initialize a policy π and then (1) evaluate π i.e. find $q_\pi(s, a)$ and then (2) improve π i.e. find π_{new} given $q_\pi(s, a)$.

However, the policy improvement step here is not trivial. To update a policy, we would be increasing the probability of taking actions that have a higher expected sum of returns than what the current policy's actions get. However, how do we know the expected sum of returns from an action that our existing policy does not take? In other words, for policy improvement, we require *exploration* - trying out actions that our existing policy would not take and seeing how good they are so that we can guide the policy improvement step. This brings us to ϵ-greedy policies - a simple method for exploration.

Definition 10. (ϵ-greedy policies). Given $q_\pi(s, a)$ for a policy π, we define the ϵ-greedy policy to be $\pi_\epsilon(a \mid s) = \left(1 - \epsilon + \frac{\epsilon}{|\mathcal{A}|}\right) \cdot 1\{a = \arg\max_{a'} q_\pi(s, a')\} + \left(\frac{\epsilon}{|\mathcal{A}|}\right) \cdot 1\{a \neq \arg\max_{a'} q_\pi(s, a')\}$. In other words, with probability $\frac{\epsilon}{|\mathcal{A}|}$, our policy will take a completely random suboptimal action that we will uniformly sample.

Now we turn our attention to Q-learning.

Goal: we aim to learn q_* (the state-action value function of the optimal policy) given trajectories rolled out using a different behavior policy π_β. In most cases, we would be using π_β to be an ϵ-greedy policy so as to enable exploration (which is necessary to do policy improvement).

The key idea is to have an estimate of q_*, which we will write to be just q and we will update this q using both our observes rewards and bootstrapping. We do this via the following update rule:

$$q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha((r_t + \gamma \max_{a'} q(s_{t+1}, a') - q(s_t, a_t)).$$

Note that we are updating the q value by taking the max over subsequent action a_{t+1} i.e. pushing our current estimate q to an larger estimate. This is how we attempt to get towards estimating $q_*(s, a)$.

Algorithm: Q-Learning with ϵ-greedy Exploration

```
1: Initialize $q(s,a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}$, set $t = 0$, initial state $s_t = s_0$
2: Set behavior policy $\pi_b$ to be $\epsilon$-greedy w.r.t. $q$
3: loop
4:     Take action $a_t \sim \pi_b(s_t)$                              ▷ Sample action from policy
5:     Observe reward $r_t$ and next state $s_{t+1}$
6:     $q(s_t, a_t) \leftarrow q(s_t, a_t) + \alpha\,(r_t + \gamma \max_a q(s_{t+1}, a) - q(s_t, a_t))$
7:     $\pi(s_t) = \arg\max_a q(s_t, a)$ with probability $1 - \epsilon$, else choose action randomly
8:     $t \leftarrow t + 1$
9: end loop
```

In most real-world settings, we actually update the α parameter as well based on the number of iterations of udpates we have done. For example, letting iteration number (for the outermost loop) be i. Then, we update $\alpha_i = \frac{1}{i}$.

Q-learning converges to optimal policy even if actions are suboptimal. The caveats are that there must be enough exploration and α must decay over time. Formally, this means:

1. All states and actions are visited infinitely often. In the limit, it doesn't matter how you select actions.

2. There is a learning rate schedule such that for all (s, a):

$$\sum_{t=0}^{\infty} \alpha_t(s,a) = \infty \qquad \text{and} \qquad \sum_{t=0}^{\infty} \alpha_t^2(s,a) < \infty.$$

4.4 Value Function Approximation

Given a policy π, we want to query $q_\pi(s, a)$. But how do we do so? The Q-learning algorithm we saw above requires taking the max over all actions in the action space. But as action space gets larger, this algorithm requires a very large number of iterations for convergence. What if we instead *learn* the function? For example, we may parametrize $q_\pi(s, a; \phi)$ and update the parameters ϕ as to best approximate $q_\pi(s, a)$ (or parametrize $q(s, a; \phi)$ in order to approximate $q_*(s, a)$.

Now, fitting such a function would be easy if we knew the true $q_\pi(s, a)$. We would just be minimizing the squared error

$$\mathbb{E}_{\tau \sim P_\pi(\cdot)} \left[(q_\pi(s_t, a_t) - q_\pi(s_t, a_t; \phi))^2 \right].$$

However, we do not have the true $q_\pi(s_t, a_t)$ available to us. So we will estimate the target that we are fitting through various techniques.

4.4.1 Monte Carlo Value Function Approximation

In this algorithm, we want to learn $q_\pi(s, a)$ i.e. the q-value of a particular policy π. We we will use a Monte Carlo estimate of the target that we want to fit.

Algorithm: MC Value Function Approximation for Policy Evaluation

```
1: Initialize $\phi$, $k = 1$
2: **loop**
3:     Sample $k$-th episode $(s_{k,1}, a_{k,1}, r_{k,1}, s_{k,2}, \ldots, s_{k,L_k})$ given policy $\pi$
4:     **for** $t = 1, \ldots, L_k$ **do**
5:         **if** first visit to $(s_t, a_t)$ in episode $k$ **then**
6:             $G_t(s, a) = \sum_{j=t}^{L_k} r_{k,j}$
7:             $\nabla_\phi J(\phi) = -2 \left[ G_t(s, a) - \hat{q}(s_t, a_t; \phi) \right] \nabla_\phi \hat{q}(s_t, a_t; \phi)$
8:             Update weights: $\phi \leftarrow \phi - \alpha \nabla_\phi J(\phi)$
9:         **end if**
10:     **end for**
11:     $k \leftarrow k + 1$
12: **end loop**
```

4.4.2 TD Learning for Value Function Approximation

In this algorithm, again, we want to learn $V_\pi(s)$ using $\hat{V}_\pi(s; \phi)$. However, this time, we will compute the target using a TD estimate. In other words, our target will be $r_t + \gamma \hat{V}_\pi(s_{t+1}; \phi)$. This uses three kinds of approximation in computing the target: sampling (sampling the next state s_{t+1}), bootstrapping and value function approximation.

Algorithm: TD(0) Value Function Approximation for Policy Evaluation

```
1: Initialize $\phi, s$
2: loop
3:     Given $s$, sample $a \sim \pi(s)$, observe $r(s,a)$ and $s' \sim p(s' \mid s,a)$
4:     $\nabla_\phi J(\phi) = -2\left[r + \gamma \hat{V}_\pi(s';\phi) - \hat{V}_\pi(s;\phi)\right] \nabla_\phi \hat{V}_\pi(s;\phi)$
5:     Update weights: $\phi \leftarrow \phi - \alpha \nabla_\phi J(\phi)$
6:     if $s'$ is not a terminal state then
7:         Set $s \leftarrow s'$
8:     else
9:         Restart episode, sample new initial state $s$
10:    end if
11: end loop
```

4.4.3 Deep q-learning

In this algorithm, we aim to directly learn $q_*(s_t, a_t)$. To do so, we will set the target to be $r(s_t, a_t) + \gamma \max_{a_{t+1}} \hat{q}(s_{t+1}, a_{t+1}; \phi)$ and minimize the squared error between this target and our estimate $\hat{q}(s_t, a_t; \phi)$.

This algorithm is off-policy - our trajectories could be sampled using any behavior policy. However, the issue with this algorithm is that this suffers from the "deadly triad" that often leads to lack of convergence due to rapid oscillations in the target. The deadly triad is characterized as a learning algorithm that incorporates (1) bootstrapping (2) function approximation and (3) off-policy learning.

In the case of q-learning with function approximation, the two major sources of instability come from (1) correlation between samples (i.e. different tuples of the form (s, a, r, s') within a trajectory) and (2) non-stationary targets (since we are doing bootstrapping with a function approximation).

To deal with these issues, deep q-learning uses a number of techniques.

Replay buffer: To remove correlations between samples, the replay buffer, $\mathcal{D}$, is used. The replay buffer is a dataset containing all tuples of the form (s, a, r, s') from prior experience. Then, during learning $\hat{q}(s, a, \phi)$, we sample an experience (a tuple $(s, a, r, s') \sim \mathcal{D}$) and then perform a gradient step.

Fixed q-targets: We use a different set of weights ϕ' for our target. In other words, we are training $\hat{q}(s, a; \phi)$ but when computing the target, we will use $r(s_t, a_t) + \gamma \max_{a_{t+1}} \hat{q}(s_{t+1}, a_{t+1}; \phi')$. What does this ϕ' look like? Say, we are updating ϕ. For a number of iterations/epochs, we will fix ϕ' and only update ϕ. Then, we will set ϕ' to be equal to ϕ for the next interval.

A preliminary deep q-learning algorithm is provided below:

Algorithm: DQN Pseudocode

```
1:  Input: $C$, $\alpha$, $\mathcal{D} = \{\}$; Initialize $\phi$, $\phi' = \phi$, $t = 0$
2:  Get initial state $s_0$
3:  loop
4:      Sample action $a_t$ using $\epsilon$-greedy policy for current $\hat{q}(s_t, a; \phi)$
5:      Observe reward $r_t$ and next state $s_{t+1}$
6:      Store transition $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $D$
7:      Sample random minibatch of tuples $(s_i, a_i, r_i, s_{i+1})$ from $\mathcal{D}$
8:      for iter in $\{1, \cdots, K\}$ do
9:          if episode terminated at step $i + 1$ then
10:             $y_i = r_i$
11:         else
12:             $y_i = r_i + \gamma \max_{a'} \hat{q}(s_{i+1}, a'; \phi')$
13:         end if
14:         Do gradient descent step on $(y_i - \hat{q}(s_i, a_i; \phi))^2$:
15:            $\Delta \mathbf{w} = \alpha(y_i - \hat{q}(s_i, a_i; \phi))\nabla_\phi \hat{q}(s_i, a_i; \phi)$
16:     end for
17:     $t \leftarrow t + 1$
18:     if $\mathrm{mod}(t, C) == 0$ then
19:         $\phi' \leftarrow \phi$
20:     end if
21: end loop
```

We usually set $K = 1$ although larger K provides more reliable signal.

Note that we need not only use ϵ-greedy policies for exploration. For example, especially in continuous action spaces, one can use Boltzmann exploration: $\pi(a \mid s) \propto \exp(\hat{q}(s, a; \phi))$.

Another modification we often need is what is often called double q-learning. Notice that the target $y_i = r_i + \gamma \max_{a'} \hat{q}(s_{i+1}, a'; \phi')$ computes a max over actions. However, if there are some actions a' for which we had few datapoints in our dataset such that $\hat{q}(s_{i+1}, a'; \phi')$ is inaccurate, we might end up electing the wrong target.

Now, note that $\max_{a'} q(s, a'; \phi') = q(s, \arg\max_{a'} q(s, a'; \phi'))$. Now, note that, on the right hand side, we are computing the value using $q(\cdot, \cdot; \phi')$ and the action we are using is found by taking the arg max over $q(s, a'; \phi')$. We want to try and decorrelate the noise between these two estimates. Double q-learning trains two networks:

$$q(s, a; \phi'_A) \xleftarrow{\text{target for}} r(s, a) + \gamma q(s', \arg\max_{a'} q(s', a'; \phi'_A); \phi_B)$$

and

$$q(s, a; \phi'_B) \xleftarrow{\text{target for}} r(s, a) + \gamma q(s', \arg\max_{a'} q(s', a'; \phi'_B); \phi_A)$$

The main idea is that we want to select the ”optimal actions” using one target network and we want to evaluate the action using a different target network. In practice, we use ϕ to evaluate action and use ϕ' to evaluate the value.

Algorithm: Double DQN Pseudocode

```
1: **Input:** $C$, $\alpha$, $\mathcal{D} = \{\}$; Initialize $\phi$, $\phi' = \phi$, $t = 0$
2: Get initial state $s_0$
3: **loop**
4:     Sample action $a_t$ using $\epsilon$-greedy policy for current $\hat{q}(s_t, a; \phi)$
5:     Observe reward $r_t$ and next state $s_{t+1}$
6:     Store transition $(s_t, a_t, r_t, s_{t+1})$ in replay buffer $\mathcal{D}$
7:     Sample random minibatch of tuples $(s_i, a_i, r_i, s_{i+1})$ from $\mathcal{D}$
8:     **for** iter in $\{1, \cdots, K\}$ **do**
9:         **if** episode terminated at step $i + 1$ **then**
10:            $y_i = r_i$
11:        **else**
12:            $a^* = \arg\max_{a'} \hat{q}(s_{i+1}, a'; \phi)$        ▷ action selection using online net
13:            $y_i = r_i + \gamma \hat{q}(s_{i+1}, a^*; \phi')$        ▷ action evaluation using target net
14:        **end if**
15:        Do gradient descent step on $(y_i - \hat{q}(s_i, a_i; \phi))^2$:
16:            $\Delta \mathbf{w} = \alpha(y_i - \hat{q}(s_i, a_i; \phi))\nabla_{\phi}\hat{q}(s_i, a_i; \phi)$
17:    **end for**
18:    $t \leftarrow t + 1$
19:    **if** mod$(t, C) == 0$ **then**
20:        $\phi' \leftarrow \phi$
21:    **end if**
22: **end loop**
```

5 Offline Reinforcement Learning

Online reinforcement learning requires expensive data collection (often times for each gradient step unless we are completely off-policy). Offline reinforcement learning allows us to use data collected by humans, agents or systems. This is important if data collection is expensive or unsafe. This can also act as a method for warmstarting online RL.

5.1 Overestimation Issue

Let us take a slight detour. Recall that in Q-learning, we aim to learn the optimal Q-function i.e. $Q_*(s,a)$ corresponding to the optimal policy. This means, in Q-learning, we collect data using a behavior policy π_β and then push our q-function iteratively towards trying to approximate the optimal Q-function (hence the max over a'):

$$q(s_t,a_t) \leftarrow q(s_t,a_t) + \alpha(r(s_t,a_t) + \gamma \max_{a'} q(s_{t+1},a') - q(s_t,a_t)).$$

Recall that this works if your π_β has sufficient coverage (meaning, it sufficiently explored), which is why we used ϵ-greedy policies or Boltzmann-sampling policies. If π_β does not have sufficient coverage, then your estimated Q-function will be inaccurate at actions a' that was not sufficiently explored by π_β.

In offline RL, there is no guarantee that your behavior policy π_β will actually have sufficient coverage. As such, we cannot just learn a q-function as above. In other words, if we tried to learn a Q-function $Q_\theta(s,a)$ by minimizing $\mathbb{E}_{(s,a,s')\sim\mathcal{D}_{\pi_\beta}}\left[(r(s,a)+\gamma\max_{a'}Q_\theta(s',a')-Q_\theta(s,a))^2\right]$, we would suffer from overestimation - if there is an action a' that was not sufficiently taken by π_β from s', then $Q_\theta(s',a')$ would be inaccurate and if it is too large (i.e. overestimated by our learned q-function), then we would use that in our target.

5.2 Implicit Q-learning

Implicit Q-learning [6] attempts to learn a q-function by only considering actions that are in the support of the data distribution or the behavior policy π_β. As such, the goal is to learn $Q_\theta(s,a)$ by minimizing

$$\mathcal{L}(\theta) = \mathbb{E}_{s,a,s'}\left[\left(r(s,a) + \gamma \max_{a'\in\mathcal{A},\pi_\beta(a'|s')>0} Q_{\hat{\theta}}(s',a') - Q_\theta(s,a)\right)^2\right].$$

To do so, IQL uses expectile regression.

Let $\tau \in [0,1]$. Then, the τ-expectile of a random variable X is defined as

$$\arg\min_{m_\tau} \mathbb{E}_{x\sim X}[L_2^\tau(x-m_\tau)]$$

where $L_2^\tau(u) = |\tau - 1\{u < 0\}|u^2$. Note that when $\tau > 0.5$, the loss function downweights the contribution of any m_τ that is larger than x. Suppose $\tau = 0.6$. Then, if $x > m_\tau$, we have that

$$L_2^{0.6}(x - m_\tau) = |0.6|(x - m_\tau)^2.$$

On the other hand, if $x < m_\tau$, we have that

$$L_2^{0.6}(x - m_\tau) = |0.6 - 1|(x - m_\tau)^2 = 0.4(x - m_\tau)^2.$$

As such, we penalize more when $x > m_\tau$.

Now, the main idea is to learn $Q_\theta(s, a)$ by minimizing:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,s',a')\sim\mathcal{D}}\left[L_2^\tau(r(s, a) + \gamma Q_{\hat{\theta}}(s', a') - Q_\theta(s, a))\right].$$

Note that the loss is greater if $Q_\theta(s, a) < r(s, a) + \gamma Q_{\hat{\theta}}(s', a')$. The loss is smaller if $Q_\theta(s, a) > r(s, a) + \gamma Q_{\hat{\theta}}(s', a')$. This is how we push the Q-function towards the upper expectile of the TD targets (i.e. pushing $Q_\theta(s, a)$ closer and closer towards $r(s, a) + \gamma \max_{a', \pi_\beta(a'|s')>0} Q_\theta(s', a')$.

This objective suffers from instability. This is because, we have an expectation over $s' \sim P(\cdot|s, a)$. If the next state s' is a lucky sample where we have a very high $Q_{\hat{\theta}}(s', a')$ for some a' in the dataset, we would be pushing our $Q_\theta(s, a)$ towards that (or even greater than that) even though that was merely a lucky sample.

To fix this, we first learn a value function $V_\psi(s)$ by minimizing

$$L_V(\psi) := \mathbb{E}_{s,a\sim\mathcal{D}}\left[L_2^\tau(Q_{\hat{\theta}}(s, a) - V_\phi(s))\right]$$

where, note that, the only samples we are taking from the dataset are s and a, *not* the next state s'. Because we are using expectile regression, this is still pushing $V_\psi(s)$ towards the upper expectile of Q values. Then, we learn the Q function by minimizing:

$$L_Q(\theta) = \mathbb{E}_{s,a,s'\sim\mathcal{D}}\left[(r(s, a) + \gamma V_\psi(s') - Q_\theta(s, a))^2\right]$$

where we can now just do normal MSE.

Next, we extract the policy from this learned Q-function using the AWR objective:

$$L_\pi(\phi) = \mathbb{E}_{s,a\sim\mathcal{D}}\left[e^{\beta\left(Q_{\hat{\theta}}(s,a) - V_\psi(s)\right)} \log \pi_\phi(a \mid s)\right]$$

Algorithm 1: Implicit Q-learning[6]

```
1:  Initialize: parameters $\psi$, $\theta$, $\hat{\theta}$, $\phi$
2:
3:  TD learning (IQL):
4:  for each gradient step do
5:      $\psi \leftarrow \psi - \lambda_V \nabla_\psi L_V(\psi)$
6:      $\theta \leftarrow \theta - \lambda_Q \nabla_\theta L_Q(\theta)$
7:      $\hat{\theta} \leftarrow (1-\alpha)\hat{\theta} + \alpha\theta$
8:  end for
9:
10: Policy extraction (AWR):
11: for each gradient step do
12:     $\phi \leftarrow \phi - \lambda_\pi \nabla_\phi L_\pi(\phi)$
13: end for
```

5.3 Conservative Q-learning

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

Recall that $\pi_\beta(a|s)$ represents the behavior policy and $\mathcal{D}$ is the dataset. 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:

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

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

Note that $\hat{\mathcal{B}}^\pi \hat{Q}^k(s,a) = r(s,a) + \gamma \mathbb{E}_{a' \sim \hat{\pi}^k(a'|s')}[\hat{Q}^k(s',a')]$.

In the policy evaluation step, the target uses actions sampled from the learned policy π^k. However, on OOD actions (i.e. actions a' that were never taken from $\pi_\beta(\cdot \mid s')$), the Q-function would give erroneous values since the Q-function is trained only on actions sampled from the behavior policy.

To fix this, we first do conservative off-policy evaluation.

5.3.1 Conservative Off-Policy Evaluation

Our goal is to estimate $V^\pi(s)$ of a target policy π using a dataset $\mathcal{D}$, generated by the behavior policy $\pi_\beta(a|s)$.

We want to learn a $Q(s,a)$ that correctly estimates the q-values of states and actions seen in the dataset and minimizies the the q-values of actions that were not seen in the dataset. In other words, we aim 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 $\mu(s,a) = d^{\pi_\beta}(s)\mu(a|s)$. One intuitive choice could be to set $\mu(a \mid s) = \pi(a \mid s)$ so as to minimize the Q-values of actions taken by our current learned policy - although we will later find a larger set of algorithms by considering various choices of $\mu(a \mid s)$.

This gives the following update, with α as the tradeoff factor:

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

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 $\pi_\beta(a|s)$:

$$\begin{aligned}\hat{Q}^{k+1} \leftarrow \underset{Q}{\operatorname{argmin}}\ \alpha\cdot&\left(\mathbb{E}_{s\sim\mathcal{D},a\sim\mu(a|s)}[Q(s,a)] - \mathbb{E}_{s\sim\mathcal{D},a\sim\hat{\pi}_\beta(a|s)}[Q(s,a)]\right)\\ &+\frac{1}{2}\ \mathbb{E}_{s,a,s'\sim\mathcal{D}}\left[\left(Q(s,a) - \ \hat{\mathcal{B}}^\pi \hat{Q}^k(s,a)\right)^2\right].\end{aligned}$$

5.3.2 CQL for Offline RL

As discussed before, we could set $\mu(a \mid s) = \pi(a \mid s)$ i.e. the learned policy in the offline RL set up. But, since we aim to learn a policy that outperforms the behavior policy and since we do so by using the Q-function (i.e. placing more probability mass on actions where the q-value is larger), we can set μ to be distribution that does so:

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

where $\mathcal{R}(\mu)$ is a regularizer.

One choice of the regularizes is to choose

$$\mathcal{R}(\mu) \ = \ -D_{\mathrm{KL}}(\mu \| \rho),$$

where $\rho(a \mid s)$ is a prior distribution. Then one can show that

$$\mu(a \mid s) \;\propto\; \rho(a \mid s) \exp\big(Q(s,a)\big).$$

As a popular instantiation, we could choose $\rho(a \mid s) = \hat{\pi}^{k-1}(a \mid s)$ i.e. the latest policy at hand.

6 Preference Fine-tuning

6.1 RLHF and Direct Preference Optimization

6.1.1 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:

$$\max_{\pi_\theta} \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi_\theta(y|x)}[r_\phi(x, y)] - \beta D_{KL}(\pi_\theta(y \mid x) \mid\mid \pi_{\text{ref}}(y \mid x)). \tag{17}$$

However, this is not differentiable. To see this, note that

$$\frac{\partial r_\phi(x, y)}{\partial \theta} = \frac{\partial r_\phi(x, y)}{\partial y} \frac{\partial y}{\partial \theta}$$

but y is not differentiable (as these are language tokens). 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}.

6.1.2 DPO

Direct Preference Optimization (DPO) starts by observing that there is a theoretical solution to the objective 17 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 28. 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.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.

[2] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," 2017. [Online]. Available: https://arxiv.org/abs/1502.05477

[3] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomput.*, vol. 71, no. 7–9, p. 1180–1190, Mar. 2008. [Online]. Available: https://doi.org/10.1016/j.neucom.2007.11.026

[4] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[5] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018. [Online]. Available: https://arxiv.org/abs/1801.01290

[6] I. Kostrikov, A. Nair, and S. Levine, "Offline reinforcement learning with implicit q-learning," 2021. [Online]. Available: https://arxiv.org/abs/2110.06169

[7] D. A. Levin and Y. Peres, *Markov Chains and Mixing Times*, 2nd ed. American Mathematical Society, 2009.

[8] D. Silver, G. Lever, N. M. O. Heess, T. Degris, D. Wierstra, and M. A. Riedmiller, "Deterministic policy gradient algorithms," in *International Conference on Machine Learning*, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:13928442

[9] L. Weng, "A (long) peek into reinforcement learning," *lilianweng.github.io*, 2018. [Online]. Available: https://lilianweng.github.io/posts/2018-02-19-rl-overview/

A Probability

A.1 Total Variation Distance

This section is reading notes from [7].

A.1.1 Defintions and Properties

Total variation distance is a metric for measuring the distance between two distributions.

Definition 11. (Total Variation Distance). The total variation distance between two probability distributions μ and ν on $\mathcal{X}$ is defined by

$$||\mu - \nu||_{\mathrm{TV}} = \max_{A \subseteq \mathcal{X}} |\mu(A) - \nu(A)|.$$

We have the equivalent formulation:

Proposition 29. For μ and ν distributions on $\mathcal{X}$, we have:

$$||\mu - \nu||_{\mathrm{TV}} = \frac{1}{2} \sum_{x \in \mathcal{X}} |\mu(x) - \nu(x)||$$

Proof. Let $B = \{x : \mu(x) \geq \nu(x)\}$ and let $A \subset \mathcal{X}$ be any event. Then,

$$\begin{aligned} \mu(A) - \nu(A) &\leq \mu(A \cap B) - \nu(A \cap B) \\ &\leq \mu(B) - \nu(B). \end{aligned}$$

Similarly,

$$\nu(A) - \mu(A) \leq \nu(B^C) - \mu(B^C).$$

Note that $\mu(B) - \nu(B) = \nu(B^C) - \mu(B^C)$ since probabilities sum to 1.

$$\mu(B) - \nu(B) + \nu(B^C) - \mu(B^C) = \sum_{x \in B} |\mu(x) - \nu(x)| + \sum_{x \in B^C} |\nu(x) - \mu(x)| = \sum_{x} |\mu(x) - \nu(x)|.$$

Now, consider A as in the definition of the TV distance - this A is either B or B^C i.e. $|\mu(A) - \nu(A)|$ is maximize when either $A = B$ or $A = B^C$. Then,

$$||\mu - \nu||_{\mathrm{TV}} = \frac{1}{2}\left(\mu(B) - \nu(B) + \nu(B^C) - \mu(B^C)\right) = \frac{1}{2} \sum_{x} |\mu(x) - \nu(x)|.$$

□

Corollary 30. We also have that (using the proof above)

$$||\mu - \nu||_{\mathrm{TV}} = \sum_{x \in \mathcal{X}, \mu(x) \geq \nu(x)} \mu(x) - \nu(x).$$

Corollary 31. Total variation distance satisfies the triangle inequality:

$$||\mu - \nu||_{\mathrm{TV}} \leq ||\mu - \eta||_{\mathrm{TV}} + ||\eta - \nu||_{\mathrm{TV}}$$

A.1.2 Coupling

A coupling of two probability distributions is a way of defining a joint distribution over two random variables such that their marginals match the original distributions. More formally:

Definition 12. (Coupling). A coupling of two probability distributions μ and ν is a pair of random variables (X, Y) defined on a single probability space such that the marginal distribution of X is μ and the marginal distribution Y is ν. So, the coupling (X, Y) satisfies $\mathbb{P}(X = x) = \mu(x)$ and $\mathbb{P}(Y = y) = \nu(y)$.

Example: Consider μ and ν both be Bernoulli random variables with parameter 0.5. One way to couple μ and ν is to define the joint distribution (X, Y) such that $\mathbb{P}(X = x, Y = y) = \frac{1}{4}$ for all x, y. Another way is to define X to be the independent Bernoulli variable and let $Y = X$.

Given a coupling (X, Y) with joint distribution $\pi(X, Y) = P(X = x, Y = y)$, by the law of total probability $\sum_x \pi(X = x, Y = y) = \nu(y)$ (and the same for the marginal of X). Conversely, if we have a probability distribution π on $\mathcal{X} \times \mathcal{X}$ such that $\sum_y \pi(x, y) = \mu(x)$ and $\sum_x \pi(x, y) = \nu(y)$, we can find a pair of random variables (X, Y) that have π as their joint distribution, and so (X, Y) is a coupling of μ and ν.

Proposition 32. Let p and q be two probability distributions on $\mathcal{X}$. Then,

$$||p - q||_{\text{TV}} = \inf\{\mathbb{P}(X \neq Y) \mid (X, Y) \text{ is a coupling of } p \text{ and } q\}.$$

In fact, there is a coupling where this infimum is achieved which we call the optimal coupling.

Proof. For any coupling of p and q and any event $A \subset \mathcal{X}$, we have that:

$$\begin{aligned} p(A) - q(A) &= \mathbb{P}(X \in A) - \mathbb{P}(Y \in A) \\ &\leq \mathbb{P}(X \in A, Y \notin A) \\ &\leq \mathbb{P}(X \neq Y). \end{aligned}$$

where the second inequality comes from noting:

$$\mathbb{P}\{X \in A\} - \mathbb{P}\{Y \in A\} = \mathbb{P}\{X \in A, Y \in A\} + \mathbb{P}\{X \in A, Y \notin A\} - \mathbb{P}\{X \in A, Y \in A\} - \mathbb{P}\{X \notin A, Y \in A\}.$$

As such,

$$||p - q||_{\text{TV}} \leq \inf\{\mathbb{P}(X \neq Y) \mid (X, Y) \text{ is a coupling of } p \text{ and } q\}.$$

Now, we construct a coupling for which this is an equality.

Let $m = \sum_{x \in \mathcal{X}} \min(p(x), q(x))$. Then,

$$\begin{aligned} m &= \sum_{x \in \mathcal{X}, p(x) \leq q(x)} p(x) + \sum_{x \in \mathcal{X}, p(x) > q(x)} q(x) \\ &= 1 - \sum_{x \in \mathcal{X}, p(x) > q(x)} (p(x) - q(x)) \\ &= 1 - ||p - q||_{\text{TV}} \end{aligned}$$

where the second inequality comes from adding and subtracting $\sum_{x\in\mathcal{X},p(x)>q(x)} p(x)$.

So

$$m = 1 - ||p - q||_{\mathrm{TV}}.$$

Now, flip a coin with probability of heads equal to $m = 1 - ||p - q||_{\mathrm{TV}}$.

1. If we get heads, then define Z as follows: sample x from the probability distribution $\psi(x) = \frac{\min(p(x),q(x))}{m}$ and set $X = Y = Z$.
2. If we get tails, then choose X from the probability distribution

$$\varphi_1(x) = \begin{cases} \dfrac{p(x) - q(x)}{\|p - q\|_{\mathrm{TV}}} & \text{if } p(x) > q(x), \\ 0 & \text{otherwise,} \end{cases}$$

   and choose Y independently from the distribution:

$$\varphi_2(y) = \begin{cases} \dfrac{p(y) - q(y)}{\|\mu - \nu\|_{\mathrm{TV}}} & \text{if } p(y) > q(y), \\ 0 & \text{otherwise.} \end{cases}$$

   (Note: when the coin gives us tails, $X \neq Y$ as φ_1 and φ_2 are positive on disjoint subsets of $\mathcal{X}$)

Then,

$$m\psi + (1 - m)\varphi_1 = p$$
$$m\psi + (1 - m)\varphi_2 = q$$

so we get the appropriate marginals. Also, $X = Y$ if and only if the coin toss gives us heads which happens with probability m, so $P(X \neq Y) = m = ||p - q||_{\mathrm{TV}}$.

$\square$