policy gradient methods

- we've seen value iteration and policy iteration. what if we have a large/continuous state space and action space? the method we are about to discuss works when we don't even know the transition probabilities, dynamics and have large/continuous state and action space! what it does assume though, is that we some kind of a simulator and can sample a bunch of trajectories and see the total reward at the end of a trajectory.
- there are a few ways we can formulate the optimization problem in the type of RL we've seen till now. we've only talked about a subset of this but the algorithms end up being similar for the others.

- Fixed-horizon episodic: $\sum_{t=0}^{T-1} r_t$
- Average-cost: $\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} r_t$
- Infinite-horizon discounted: $\sum_{t=0}^{\infty} \gamma^t r_t$
- Variable-length undiscounted: $\sum_{t=0}^{T_{\text{terminal}}-1} r_t$
- Infinite-horizon undiscounted: $\sum_{t=0}^{\infty} r_t$

- for now we will stick to the finite horizon episodic formulation (first one) → fixed number of time steps.
- Formally, we will deal with this problem:

$$
\begin{aligned}
s_0 &\sim \mu(s_0) \\
a_0 &\sim \pi(a_0 \mid s_0) \\
s_1, r_0 &\sim P(s_1, r_0 \mid s_0, a_0) \\
a_1 &\sim \pi(a_1 \mid s_1) \\
s_2, r_1 &\sim P(s_2, r_1 \mid s_1, a_1) \\
&\dots \\
a_{T-1} &\sim \pi(a_{T-1} \mid s_{T-1}) \\
s_T, r_{T-1} &\sim P(s_T \mid s_{T-1}, a_{T-1})
\end{aligned}
$$

Objective:

$$
\begin{aligned}
&\text{maximize } \eta(\pi), \text{ where} \\
&\eta(\pi) = E[r_0 + r_1 + \cdots + r_{T-1} \mid \pi]
\end{aligned}
$$

- "Policies are parametrized by a vector of real valued numbers, which we call θ". → fancy way of saying we will have a neural net.
- in the discrete action space case, if we do argmax, its a deterministic policy. if we want stochastic, we will have it output probabilities eg. using softmax.
- in the continuous case, we could have the network output mean and diagonal covariance of gaussian and we will sample from that to take an action

- largely the policy gradient methods we will look at fall into one of these methods:
  collect a bunch of trajectories and then do one of:
  - make the good trajectories more probable
  - make the good actions more probable (we kinda have to solve the credit assignment problem for this?)
  - push the actions towards good actions
- consider an expectation $E[f(x)]$ where $x \sim p(x|\theta)$. the idea is to compute the gradient wrt θ. this idea is called the score function gradient estimator or the likelihood gradient estimator.
- understand the following derivation:

► Consider an expectation $E_{x\sim p(x\mid\theta)}[f(x)]$. Want to compute gradient wrt θ

$$\begin{aligned}\nabla_\theta E_x[f(x)] &= \nabla_\theta \int \mathrm{d}x\ p(x\mid\theta) f(x)\\ &= \int \mathrm{d}x\ \nabla_\theta p(x\mid\theta) f(x)\\ &= \int \mathrm{d}x\ p(x\mid\theta)\frac{\nabla_\theta p(x\mid\theta)}{p(x\mid\theta)} f(x)\\ &= \int \mathrm{d}x\ p(x\mid\theta)\nabla_\theta \log p(x\mid\theta) f(x)\\ &= E_x[f(x)\nabla_\theta \log p(x\mid\theta)].\end{aligned}$$

- Above, we derived an unbiased estimator for $\nabla_\theta E_x[f(x)]$ *assuming we know the distribution* $p(x|\theta)$ that x is sampled from. this is called the score function gradient trick or the likelihood function gradient trick. We sample a few $f(x)$ values and we have an estimate (unbiased) of the gradient of $\nabla_\theta E_x[f(x)]$ using the above formula!
- Here in our initial approach, x is going to be a sample trajectory that we choose. now, if $f(x)$ measures how good the sample x is (the total reward for trajectory x), then by following the direction the of the gradient, we are making it more likely to pick better x's → those that are likely to give a higher $f(x)$ → this is the policy gradient!!
- "this is true even if x is discrete and not continuous" → does not prove this.
- We can sample trajectories just by using our current policy and using chain rule to pick actions.
- Now think about the fact that we are maximizing the log probability here! remind you of something? its kind of like our classification and regression problems.
- Some math:
  - once we apply chain rule and find trajectories, the gradient of expected reward with respect to parameters of policy turns out to be
    $\nabla_\theta E[R] = E[R\nabla_\theta \sum log\ \pi(a_t|s_t,\theta)]$
    $= E[\sum r_t \nabla_\theta \sum log\ \pi(a_t|s_t,\theta)]$
- In general, solutions and algorithms that better solve the credit assignment problem have better sample efficiency in reinforcement learning. this can be huge for certain domains where you can't train for crazy amounts.
- we will now rewrite the math in terms of the expected reward at time t' and then compute the sum over these to find the expected reward over time:

- Previous slide:

$$\nabla_\theta \mathbb{E}_\tau [R] = \mathbb{E}_\tau \left[\left(\sum_{t=0}^{T-1} r_t \right) \left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta) \right) \right]$$

- We can repeat the same argument to derive the gradient estimator for a single reward term $r_{t'}$.

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

- Sum this formula over t, we obtain

$$\begin{aligned} \nabla_\theta \mathbb{E} [R] &= \mathbb{E} \left[\sum_{t=0}^{T-1} r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi(a_t \mid s_t, \theta) \right] \\ &= \mathbb{E} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta) \sum_{t'=t}^{T-1} r_{t'} \right] \end{aligned}$$

- note that the above sampling still computes the same number, $\nabla_\theta E[R]$, with using the additional fact that current actions only affect current and future rewards so we can disregard summing over all possible time and just sum over future actions.
- thus, the above way to write it still computes the same number $\nabla_\theta E[R]$, just with lesser variance → the reason the above has lesser variance is because (some intuition below):
  - lets say you are trying to find $E[x_1 x_2 x_3 ... x_n .. x_{n+1} .. x_{n+k}]$ by sampling. Think of each term as a layer in a tree. The tree is $n + k$ deep. You'd love it if there were a property such that you can sample the first n layers and that gives you the true expectation of going down all the other subtrees that start at $n + 1$ under that sample. This is the case in our problem here with ignoring rewards before the current action! this will also show up in other scenarios where we do sampling where this property is true. The property that we are taking advantage of here is **causality:** current actions can only affect future rewards.
- in general, we'd like estimates with lesser variance. we can further reduce the variance of this estimate of the gradient by using baselines. To get an estimate with lesser variance, note that each grad log term above is weighted by a "sum of future rewards" term. Say the future rewards were 2 million, 1.96 million, 2.03 million etc, versus 0, -4, 3 etc. Intuition wise, see how the second way of weighting is a better way to guide the gradient to to the right value? this is why weights with mean 0 are a good thing in terms of getting better sampling with lesser variance.
- So, we will subtract a bias (which is actually the expected sum of rewards → value function) from each r_t term. this is like when we make a set of observations have mean 0. It is straightforward to show that this baseline does not introduce bias in the gradient. hence, this leaves the gradient unchanged, but we get estimates with more precision (lesser variance), which is a good thing. It will make SGD algorithm move in more consistently good directions, because gradient estimates are more precise.
- note how baselines are basically value functions given a policy, starting at that state.
- note that the above mentioned baseline is not an optimal baseline, but works well in practice. to find the optimal baseline (one that reduces variance the most), we can write down the term for variance and set the derivative of that w.r.t the baseline to 0 and solve for the baseline, but the above works well in practice so we won't do that.
- additionally, we can introduce the idea of discounting rewards that are delayed. this is empirical and tends to make the policy gradient method more sample efficient ¯_(ツ)_/¯. note that if we use this, we should update our baseline to be expected sum of discounted rewards.
- Incorporating baselines for lower variance and discounts for sample efficiency:
  the formula for gradient of expected reward.

- Introduce discount factor γ, which ignores delayed effects between actions and rewards

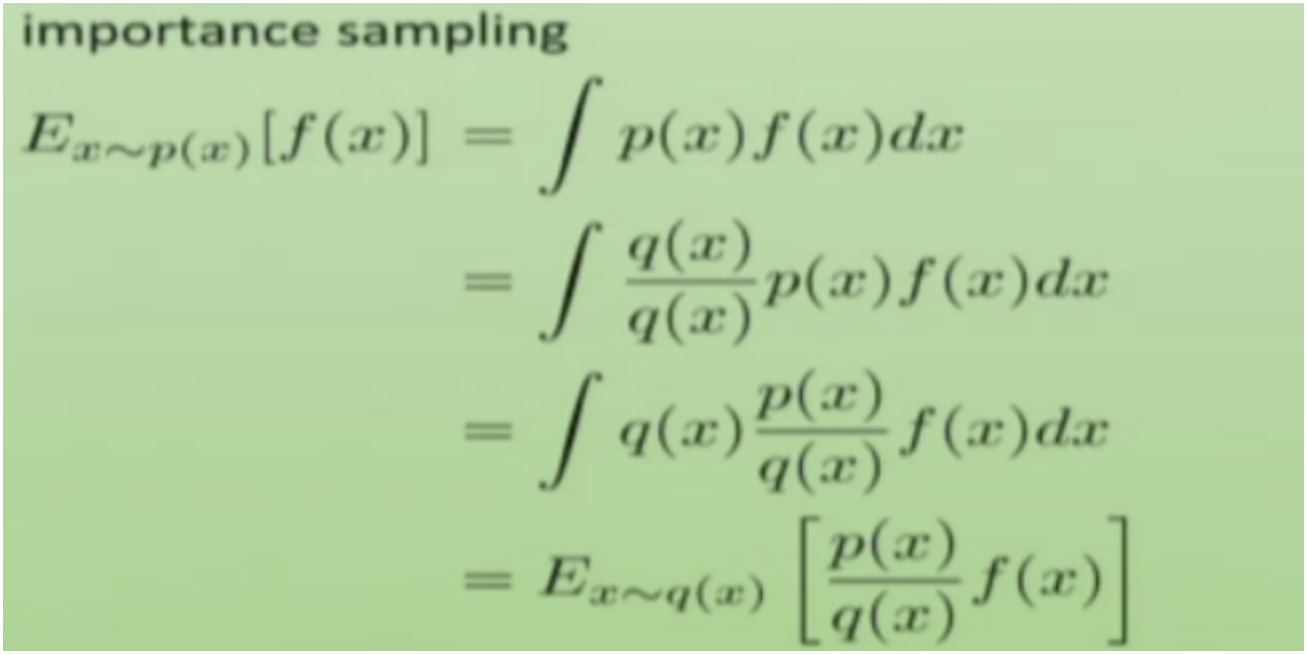
$$\nabla_\theta \mathbb{E}_\tau [R] \approx \mathbb{E}_\tau \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta) \left(\sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} - b(s_t) \right) \right]$$

- Now, we want $b(s_t) \approx \mathbb{E} \left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-1-t} r_{T-1} \right]$

- Now, the algorithm (also called the REINFORCE algorithm)

Initialize policy parameter θ, baseline b
for iteration=$1, 2, \ldots$ **do**
  Collect a set of trajectories by executing the current policy
  At each timestep in each trajectory, compute
   the *return* $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and
   the *advantage estimate* $\hat{A}_t = R_t - b(s_t)$.
  Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,
   summed over all trajectories and timesteps.
  Update the policy, using a policy gradient estimate $\hat{g}$,
   which is a sum of terms $\nabla_\theta \log \pi(a_t \mid s_t, \theta) \hat{A}_t$.
   (Plug $\hat{g}$ into SGD or ADAM)
end for

- Now, in the above algorithm, we have to sample trajectories at every iteration → this can be inefficient. anytime we want to sample something to estimate some quantity but we only have access to samples from another distribution, we can use something called importance sampling.
- Importance sampling → we can estimate the expectation of a function $f(x)$ over $p(x)$ even if we have samples from a different distribution $q(x)$ as follows:



- In the above example of the REINFORCE algorithm, we estimate expectation of gradient of reward over new policy with trajectories sampled from the old policy.
- This is called the off-policy policy gradient and can be summarized as:

$$\theta^\star = \arg\max_\theta J(\theta)$$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)]$$

$$\frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)} = \frac{\prod_{t=1}^T \pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\prod_{t=1}^T \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}$$

$$\nabla_{\theta'} J(\theta') = E_{\tau\sim\pi_\theta(\tau)}\left[\frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)}\nabla_{\theta'}\log\pi_{\theta'}(\tau)r(\tau)\right] \quad \text{when } \theta \neq \theta'$$

$$= E_{\tau\sim\pi_\theta(\tau)}\left[\left(\prod_{t=1}^T \frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}\right)\left(\sum_{t=1}^T \nabla_{\theta'}\log\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)\right)\left(\sum_{t=1}^T r(\mathbf{s}_t,\mathbf{a}_t)\right)\right] \quad \text{what about causality?}$$

$$= E_{\tau\sim\pi_\theta(\tau)}\left[\sum_{t=1}^T \nabla_{\theta'}\log\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)\left(\prod_{t'=1}^t \frac{\pi_{\theta'}(\mathbf{a}_{t'}|\mathbf{s}_{t'})}{\pi_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'})}\right)\left(\sum_{t'=t}^T r(\mathbf{s}_{t'},\mathbf{a}_{t'})\right)\right]$$

The last step comes from incorporating causality: for action probabilities, we only need weight probabilities up until this action just like for rewards we do everything going forward and leave out rewards that came before we took an action. These fewer multiplications reduce the variance of the estimate.

- Note the product term in the above. For long trajectories it is possible (depending on policy) that the product is long and less than 0 and thus rapidly goes to 0. This can cause bad variance while sampling even with all our causality tricks to reduce variance.
- There is a way to get around this by rewriting the above product of ratios in terms of just one product of ratios as below. this is just a sneak peak into a trick that will get us around this problem at a later time:

$$\theta^\star = \arg\max_\theta \sum_{t=1}^T \underbrace{E_{(\mathbf{s}_t,\mathbf{a}_t)\sim p_\theta(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t,\mathbf{a}_t)]}_{\text{expectation under state-action marginal}}$$

$$J(\theta) = \sum_{t=1}^T E_{(\mathbf{s}_t,\mathbf{a}_t)\sim p_\theta(\mathbf{s}_t,\mathbf{a}_t)}[r(\mathbf{s}_t,\mathbf{a}_t)] = \sum_{t=1}^T E_{\mathbf{s}_t\sim p_\theta(\mathbf{s}_t)}\left[E_{\mathbf{a}_t\sim\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}[r(\mathbf{s}_t,\mathbf{a}_t)]\right]$$

$$J(\theta') = \sum_{t=1}^T E_{\mathbf{s}_t\sim p_\theta(\mathbf{s}_t)}\left[\frac{p_{\theta'}(\mathbf{s}_t)}{p_\theta(\mathbf{s}_t)} E_{\mathbf{a}_t\sim\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}\left[\frac{\pi_{\theta'}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)} r(\mathbf{s}_t,\mathbf{a}_t)\right]\right]$$

ignore this part

We'll see why this is reasonable later in the course!

when the policies are close enough, in practice we can scratch out the above probability ratio so as to avoid the product that could vanish.

- the above is a preview of how we can make policy gradient work on problems even with long horizon problems.
- "A lot of people doing research in policy gradients spend time thinking about how to reduce variance of samples that generate gradient estimates."