Addison Kalanther (addikala@berkeley.edu)

Lecture 5: Policy Gradients

This lecture goes over the basics of policy gradients.

Direct policy differentiation

The RL objective is taking the arg max of the expectation of the sum of rewards over the trajectory distribution of the policy. In policy differentiation, this is often estimated via sampling trajectories of a given policy and averaging their total rewards. This method of estimation is also known as *monte carlo*.

Defining $J(\theta) = E_{\tau \sim p_\theta(\tau)}[r(\tau)]$ as the RL objective, we can use the definition of expectation to evaluate this as $J(\theta) = \int p_\theta(\tau) r(\tau) d\tau$ and take it's gradient to get the default policy gradient.

$$\nabla * \theta J(\theta) = \int \nabla_\theta p_\theta(\tau) r(\tau) d\tau = \int p_\theta(\tau) \nabla_\theta \log p * \theta(\tau) r(\tau) d\tau = E_{\tau \sim p_\theta(\tau)}[\nabla \log p * \theta(\tau) r(\tau)]$$

In the above derivation of the policy gradient, we use the identity $p * \theta(\tau) \nabla_\theta \log p * \theta(\tau) = p * \theta(\tau) \frac{\nabla p * \theta(\tau)}{p * \theta(\tau)} = \nabla_\theta p * \theta(\tau)$.

Using the above policy gradient, we can rewrite it using by expanding the log over the trajectory to get

$$\nabla * \theta J(\theta) = E_{\tau \sim p_\theta(\tau)}[(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t))(\sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t))]$$

The only difference between the two representations of the policy gradient $\nabla * \theta J(\theta)$ is that $\nabla_\theta \log p * \theta(\tau)$ and $r(\tau)$ are expanded as sums over their individual states in a trajectory.

Evaluating the policy gradient

We can find an approximation of the policy gradient using monte carlo approximation. Using the definition of the policy gradient $\nabla * \theta J(\theta)$ found above, we can get its monte carlo approximation over N samples as

$$\nabla * \theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} (\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i \mid \mathbf{s}_t^i))(\sum_{t=1}^{T} r(\mathbf{s}_t^i, \mathbf{a}_t^i))$$

This is the version of $\nabla * \theta J(\theta)$ used in the REINFORCE algorithm, the first, most basic policy gradient.

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)$ (run the policy)
2. $\nabla * \theta J(\theta) \approx \sum_i (\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i \mid \mathbf{s}_t^i))(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i))$
3. $\theta \leftarrow \theta + \alpha \nabla * \theta J(\theta)$
4. repeat steps 1-3 until convergence

What the algorithm does is it first runs the policy to collect samples in step 1, calculates a monte carlo estimate of the policy gradient in step 2, and then uses that estimate to perform gradient ascent in step 3. It repeats this process until the policy converges.

Understanding policy gradients

The policy gradient is essentially a weighted maximum likelihood using the total reward as weights. This means that the rate at which the probabilities of good actions goes up is faster than those of bad actions. This is obvious when comparing the equations of the two methods.

Policy Gradient:

$$\nabla * \theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} (\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i \mid \mathbf{s}_t^i))(\sum_{t=1}^{T} r(\mathbf{s}_t^i, \mathbf{a}_t^i))$$

Maximum Likelihood:

$$\nabla_\theta J_{ML}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} (\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i \mid \mathbf{s}_t^i))$$

Example: gaussian policies

Assuming a gaussian policy,

$\pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) = \mathcal{N}(f_{\text{nn}}(\mathbf{s}_t); \Sigma)$,

we get $\log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) = -\frac{1}{2}||f(\mathbf{s}_t) - \mathbf{a}_t||_\Sigma^2 + \text{const}$

and $\nabla_\theta \log \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) = -\frac{1}{2}\Sigma^{-1}(f(\mathbf{s}_t) - \mathbf{a}_t)\frac{df}{d\theta}$.

Using this, we can use REINFORCE to perform gradient ascent.

What's going on here?

Optimizing the policy gradient just makes the good results more likely and the bad results less likely. This is ultimately a formalization of the notion of 'trial and error'. Try some actions and if it gives good results, make it more likely, if it does not, make it less likely than the better results.

Partial observability Notice that all of this is done without assuming the markov property, this means that policy gradients can be used on partially observed processes as well.

What is wrong with the policy gradient?

When using the raw total reward values in the policy gradient $\nabla * \theta J(\theta)$, we can greatly push and pull the distributions if the values from the reward function have a large range of values and a high absolute value.

This means the variance of our policy could be very high. Even worse, if the good samples have a $r(\tau) = 0$, this means that the distribution does not change at all for those samples.

Reducing Variance

Here are a few methods we can use to reduce variance.

Reward-to-go

We can reduce variance by finding a way to reduce the value of the 'weight' we are using. In default policy gradient, this is the reward. However, we can instead use a *reward-to-go* rather than a total reward.

This is because of *causality*, the idea that a policy at time t' cannot affect the reward at time t when $t < t'$. We can use causality as the expected gradient of the reward $r(t)$ with respect to a policy at time t' is 0, since they would be uncorrelated. This leaves the policy gradient unbiased.

By taking advantage of causality, we ensure that better policies continue to be more probabilistic while reducing the magnitude of the total rewards for a given state, reducing overall variance of the policy.

Baselines

We can further reduce the absolute value of reward by using a *baseline* b. Using a baseline, our policy gradient $\nabla * \theta J(\theta)$ becomes

$$\nabla * \theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log p * \theta(\tau)[r(\tau) - b]$$

A convenient baseline is the average of total rewards over N trajectories: $b = \frac{1}{N} \sum_{i=1}^{N} r(\tau)$.

One question we should ask if we are allowed to do this. The condition we are allowed to do this in is if it leaves our policy gradient unbiased, we can show it does by showing that $E[\nabla_\theta \log p * \theta(\tau) b] = 0$, since this would mean that our baselined policy is equal in expectation to the default policy by linearity of expectation.

We show that $E[\nabla_\theta \log p * \theta(\tau) b] = 0$ by

$$E[\nabla_\theta \log p * \theta(\tau) b] = \int p * \theta(\tau) \nabla_\theta \log p * \theta(\tau) b d\tau = \int \nabla_\theta p * \theta(\tau) b d\tau = b \nabla_\theta \int p * \theta(\tau) d\tau = b \nabla_\theta 1 = 0$$

In the derivation, we used the convenient identity $p * \theta(\tau) \nabla_\theta \log p * \theta(\tau) = \nabla_\theta p * \theta(\tau)$

Average reward isn't the best baseline, to minimize variance, we can minimize via differentiation.

Optimal baseline To minimize variance, we can take the derivative of the baselined variance with respect to b and set it to 0.

$$\text{Var} = E_{\tau \sim p_\theta(\tau)}[(\nabla_\theta \log p * \theta(\tau)(r(\tau) - b))^2] - E_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p * \theta(\tau)(r(\tau) - b)]^2$$

$$\begin{aligned} \frac{d\text{Var}}{db} = \frac{d}{db} E[g(\tau)^2 (r(\tau) - b)^2] &= \frac{d}{db}(E[g(\tau)^2 r(\tau)^2] - 2E[g(\tau)^2 r(\tau)] + b^2 E[g(\tau)^2]) \\ &= -2E[g(\tau)^2 r(\tau)] + 2bE[g(\tau)^2] = 0 \end{aligned}$$

Solving the equation, we get $b = \frac{E[g(\tau)^2 r(\tau)]}{E[g(\tau)^2]}$. This can be interpreted as the expected reward weighted by gradient magnitudes.

Off-policy policy gradients

Policy gradients according to the REINFORCE algorithm is on-policy. We have to resample after performing a gradient ascent step in order since the last samples don't come from the new policy's distribution, meaning the monte carlo expectation estimate of the new policy would not be correct unless we collect new samples.

This can be fixed using importance sampling.

Importance sampling

Definition Importance sampling allows us to calculate the expectation of one distribution using samples from the distribution of another.

Importance sampling allows us to rewrite the expectation of $f(x)$ over $p(x)$ as

$$E_{x \sim p(x)}[f(x)] = E_{x \sim q(x)} \left[\frac{p(x)}{q(x)} f(x) \right]$$

whose derivation follows simply from the definition of expectation.

Objective with importance sampling The RL objective with importance sampling becomes

$$J(\theta) = E_{\tau \sim \bar{p}(\tau)} \left[\frac{p * \theta(\tau)}{\bar{p}(\tau)} r(\tau) \right]$$

We can write the fractional term as

$$\begin{aligned} \frac{p * \theta(\tau)}{\bar{p}(\tau)} &= \frac{p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)}{p(\mathbf{s}_1) \prod_{t=1}^{T} \bar{\pi}(\mathbf{a}_t \mid \mathbf{s}_t) p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)} \\ &= \frac{\prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)}{\prod_{t=1}^{T} \bar{\pi}(\mathbf{a}_t \mid \mathbf{s}_t)} \end{aligned}$$

Deriving the policy gradient with importance sampling Using the importance sampled object, we get the policy gradient to be

$$\nabla_{\theta'} J(\theta') = E_{\tau \sim p_\theta(\tau)} \left[\frac{\nabla_{\theta'} p_{\theta'}(\tau)}{p * \theta(\tau)} r(\tau) \right] = E_{\tau \sim p_\theta(\tau)} \left[\frac{p_{\theta'}(\tau)}{p * \theta(\tau)} \nabla_{\theta'} \log p_{\theta'}(\tau) r(\tau) \right]$$

Estimated locally, at $\theta = \theta'$, we can see that it is equal to the policy gradient.

The off-policy policy gradient

Expanding policy gradient with importance sampling when $\theta \neq \theta'$, we get

$$\nabla_{\theta'} J(\theta') = E_{\tau \sim p_\theta(\tau)} \left[\frac{p_{\theta'}(\tau)}{p * \theta(\tau)} \nabla_{\theta'} \log \pi_{\theta'} r(\tau) \right]$$

Substituting the per-state expansions for all three terms,

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

Accounting for causality (reward-to-go),

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

The underlined term takes into account the probabilities up until term t, this enforces causality as the probabilities of future actions shouldn't affect the current weight.

The crossed-out term is the probability that we get that specific total reward from the current state. However, this is numerically unstable so it is canceled out (prone to growing exponentially large). While this means that the equation with the canceled term is no longer the gradient, it can still be used to get an algorithm that converges to the optimal policy.

Aside: policy iteration The equation with the crossed-out term is equivalent to policy iteration. This is because we take the probability that it gets to that state given its trajectory multiplied by the reward-to-go by that state.

If you are familiar with policy iteration, you will see that the sum over all trajectories of this value for a given state gives us the value function $V(\mathbf{s}_t)$ for that state. We use that value function to drive the rate at which the gradient moves, rather than just the reward of that specific rollout. The use of an implicit value function equates this equation to policy iteration.

A first-order approximation for IS Our IS policy gradient is now

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

The middle term is exponential in T, we can approximate the gradient using it's first order approximation. We approximate the middle term using the probability of getting to state $\mathbf{s}_t$ and taking $\mathbf{a}_t$.

$$\begin{aligned} \nabla_{\theta'} J(\theta') &\approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \frac{\pi_{\theta'}(\mathbf{s}_t^i, \mathbf{a}_t^i)}{\pi_\theta(\mathbf{s}_t^i, \mathbf{a}_t^i)} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t^i \mid \mathbf{s}_t^i) \hat{Q}_t^i \\ &= \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \frac{\cancel{\pi_{\theta'}(\mathbf{s}_t^i)}}{\cancel{\pi_\theta(\mathbf{s}_t^i)}} \frac{\pi_{\theta'}(\mathbf{a}_t^i \mid \mathbf{s}_t^i)}{\pi_\theta(\mathbf{a}_t^i \mid \mathbf{s}_t^i)} \nabla_{\theta'} \log \pi_{\theta'}(\mathbf{a}_t^i \mid \mathbf{s}_t^i) \hat{Q}_t^i \end{aligned}$$

We can cross out that term because the policy update is not dependent on the probability of reaching that state and is rather dependent on the probability of taking a certain action given that state (since the policy output an action given a state, not the state itself). This will become more clear in the policy gradient lecture.

Implementing policy gradients

We can implement policy gradients using automatic differentiation. Let's take a look at the policy gradient function.

$$\nabla * \theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i \mid \mathbf{s}_t^i) \hat{Q}_t^i$$

The first term looks like the maximum likelihood gradient ($\nabla_\theta J_{\text{ML}}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i \mid \mathbf{s}_t^i)$). The maximum likelihood is $J_{\text{ML}}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log \pi_\theta(\mathbf{a}_t^i \mid \mathbf{s}_t^i)$.

Maximum likelihood is a loss function commonly computed in autodif packages. We can calculate the per timestep maximum likelihood loss and multiply it by the individual state-action values. Then, we can sum the weighted timesteps and divide it by the number of trajectories to recover the policy gradient.

Policy gradient in practice

Remember policy gradient has a high variance, as this isn't the same as supervised learning. Gradients will be *very* noisy.

Consider using much larger batches to reduce variance.

Tweaking learning rates can be very hard. Adaptive step size rules like ADAM can be ok, but policy gradient-specific learning rate adjustments will be discussed later.

Advanced policy gradients

Covariant/natural policy gradient

An issue with multivariate policy gradients is the magnitude of independent variables influencing the rate at which gradients move in the component directions of individual independent variables.

This means some parameters change probabilities a lot more than others. To combat this, we have to rescale the gradient so this doesn't happen. We can control how 'far' the parameters go by performing the following gradient update step.

$$\theta' \leftarrow \arg\max_{\theta'}(\theta' - \theta)^T \nabla * \theta J(\theta) \text{ s.t. } ||\theta' - \theta||^2 \leq \epsilon$$

The constraint penalizes moving terms with large magnitudes and rewards moving terms with small magnitudes, resulting in a *rescaled* gradient that doesn't overcompensate in the direction of large values.

KL Divergence To control how far we go, we use a parameterization-independent divergence measure, usually the KL-divergence. The KL-divergence $D_{KL}(P||Q)$ can be described as the expected surprise when using Q as a model when the true distribution is P (the surprise can be thought of as the difference in information / probabilities over P). KL-divergence is equivalent to the following equation.

$$D_{KL}(\pi_{\theta'}||\pi_\theta) = E_{\pi_{\theta'}}[\log \pi_\theta - \log \pi_{\theta'}]$$

Note: KL Divergence is not symmetric

KL Divergence can be approximated using the Fisher information matrix, meaning it can be estimated with samples using the following.

$$D_{KL}(\pi_{\theta'}||\pi_\theta) \approx (\theta' - \theta)^T \mathbf{F} (\theta' - \theta)$$

$$\mathbf{F} = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(\mathbf{a} \mid \mathbf{s}) \nabla_\theta \log(\mathbf{a} \mid \mathbf{s})^T]$$

Trust Region Policy Optimization Using KL divergence, we can rewrite our gradient step as the following.

$$\theta' \leftarrow \arg\max_{\theta'}(\theta' - \theta)^T \nabla * \theta J(\theta) \text{ s.t. } D_{KL}(\pi_{\theta'}||\pi_\theta) \leq \epsilon$$

Natural Gradient We can achieve a similar effect by using the natural gradient update step

$$\theta \leftarrow \theta + \alpha \mathbf{F}^{-1} \nabla * \theta J(\theta)$$

We can solve for the optimal α while solving $\mathbf{F}^{-1} \nabla * \theta J(\theta)$. This allows us to take bigger gradient steps over TRPO at the cost of computation.

Policy gradients suggested readings

Classic Papers

- Williams (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning: introduces REINFORCE algorithm
- Baxter & Bartlett (2001). Infinite-horizon policy-gradient estimation: temporally decomposed policy gradient

- Peters & Schaal (2008). Reinforcement learning of motor skills with policy gradients: very accessible overview of optimal baselines and natural gradient

Deep reinforcement learning policy gradient papers

- Levine & Koltun (2013). Guided policy search: deep RL with importance sampled policy gradient
- Schulman, L., Moritz, Jordan, Abbeel (2015). Trust region policy optimization: deep RL with natural policy gradient and adaptive step size
- Schulman, Wolski, Dhariwal, Radford, Klimov (2017). Proximal policy optimization algorithm: deep RL with importance sampled policy gradient