Lecture 13: Proximal Policy Optimization Algorithms

1 Introduction

Admin:

- Midterm: if you have questions or are concerned, please come to office hours
- Project proposal due on Monday Mar 24. Check-in with me/TAs due on Mar 31. Do this early, perhaps even before the Mon deadline. If you need a group, check on Ed.

1.1 Review: Policy Gradient Methods

Define the RL objective in terms of the policy parameters θ as

$$\mathcal{J}(\theta) \triangleq \mathbb{E}_{\pi_\theta(\tau)}\left[\sum_t \gamma^t r_t\right]. \tag{1}$$

We would like to do gradient descent on this objective:

$$\theta \leftarrow \theta + \eta \nabla_\theta \mathcal{J}(\theta). \tag{2}$$

In deep learning, we can compute the gradient of the loss w.r.t. the model parameters directly via backprop. In the RL setting, the objective $\mathcal{J}$ depends on both the model parameters and the environment dyanmics. The environment dynamics are not known, so we can't *solely* use backprop. Instead, we use the likelihood ratio trick:

$$\nabla_\theta \mathcal{J} = \mathbb{E}_{\pi_\theta}\left[\Bigg(\underbrace{\sum_{t'=t}^{\infty} \gamma^{t'=t} r_{t'}}_{Q(s_t,a_t)}\Bigg) \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right]. \tag{3}$$

In class on Tuesday, we saw that we can compute the same gradient by subtracting a *baseline* from the Q function. A common choice of baseline is a constant (average discounted sum of returns) or the value function:

$$\nabla_\theta \mathcal{J} = \mathbb{E}_{\pi_\theta}\Bigg[(\underbrace{Q^\pi(s_t,a_t) - V^\pi(s_t)}_{A^\pi(s_t,a_t)}) \nabla_\theta \log \pi_\theta(a_t \mid s_t)\Bigg], \tag{4}$$

where this difference is known as the *advantage function*. Note that the Q function, value function, and advantage function all depend on the policy π.

Algorithm summary. Vanilla actor-critic methods repeat the following steps

1. Collect trajectories from π_{θ_t}
2. Update policy with one policy gradient step.
3. Update Q function and value function (e.g., with MC regression).
4. Discard data.

It is hopefully clear that they are horribly wasteful: they discard data after taking a single gradient step. Today we're going to look at ways that we can retain slightly old data and learn from it; this will improve sample efficiency and stability.

1.2 Goals for today

In today's class, we're going to walk through the PPO paper [5], from John Schulman in 2017. It remains one of the most widely used RL algorithms today, with applications ranging from robotics to games to LLMs. If you have a simulator (i.e., something that implements the Gym API), PPO is a good place to start.

2 Surrogate Objective Trick

One thing that's strange about implementing REINFORCE and other policy gradient methods is their interaction with deep learning libraries. Deep learning libraries (e.g., JAX, PyTorch) typically expect a certain loss function, and then use automatic differentiation to compute the gradient of that loss. It is unclear immediately how to apply these frameworks to the policy gradient:

1. When we studied the policy gradient before, we were looking at a *gradient*, not a particular objective. Now you can roll your own deep learning training script to use gradients, but it's often easier if you can provide a loss function.
2. The RL loss function (negative expected returns) can't be pass directly to a deep learning framework because it's not differentiable (the dynamics are unknown).

So, what's done in practice is to implement a *surrogate loss*: the value of this loss won't have a particular meaning, but it's *gradient* will be the policy gradient:

$$\mathcal{L}(\theta) = \mathbb{E}_{\rho^{\pi_\theta}(s_t)\pi_\theta(a_t|s_t)}\left[A^{\pi_\theta}(s_t, a_t)\log \pi_\theta(a_t \mid s_t)\right]. \tag{5}$$

These surrogate objectives will provide us with a convenient way about thinking about different policy gradient methods.

3 Trust Regions [3]

Let's start by recalling importance weighting from last time. The key idea was to use data sampled from one policy $\pi_{\theta_{\text{old}}}$ to compute the policy gradient of a new policy, $\nabla_\theta J(\theta)$. We'll write this out in terms of surrogate objectives:

$$\mathcal{L}(\theta) = \mathbb{E}_{\rho^{\pi_{\theta_{\text{old}}}}(s_t)\pi_{\theta_{\text{old}}}(a_t|s_t)}\left[\left(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\right) A^{\pi_\theta}(s_t, a_t)\log \pi_\theta(a_t \mid s_t)\right] \tag{6}$$

$$= \mathbb{E}_{\rho^{\pi_{\theta_{\text{old}}}}(s_t)\pi_\theta(a_t|s_t)}\left[A^{\pi_\theta}(s_t, a_t)\log \pi_\theta(a_t \mid s_t)\right]. \tag{7}$$

If you simplify this, you'll see that it almost, but not quite, gives us the original surrogate objective. Hence, it almost (but not quite) gives us the correct policy gradients. The missing part is that we're looking at the distribution over states that the *old* policy visited. When we're computing the policy gradient, we're telling the policy how to increase/decrease its actions at the states that it visits, so that its overall returns increase.

If the policy isn't sufficiently expressive, then there will be a tradeoff between predicting the correct actions at some states versus other states. Intuitively, it seems like we care most about the policy making the correct decisions on the states that it actually visits. This is exactly what the policy gradient does!

However, the importance weighted version about is incorrect because it might provide big updates at states that our current policy π_θ rarely visits and small updates at states that are visited frequently.

3.1 TRPO [3]

The key challenge here is that the new policy may visit different states than the old policy, and therefore may get different returns and have a different policy gradient. If π_θ and $\pi_{\theta_{\text{old}}}$

visit similar states, then the importance weighted surrogate objective will be approximately correct. And, if the two policies take similar actions everywhere ($\pi_\theta(a \mid s) \approx \pi_{\theta_{\text{old}}}(a \mid s)$), then they'll visit similar states. Thus, if we can gaurantee that $\pi_\theta(a \mid s)$ doesn't differ from $\pi_{\theta_{\text{old}}}(a \mid s)$ too much, then we can gaurantee that the importance weighted surrogate objective will be a good estimate of the policy gradient.

Formally, we have the following theorem [3]:[1]

$$\mathbb{E}_{\pi_\theta}\left[\sum_t \gamma^t r_t\right] \geq \mathbb{E}_{\pi_{\theta_{\text{old}}}}\left[\sum_t \gamma^t r_t\right] \tag{8}$$

$$+ L_{\pi_{\theta_{\text{old}}}(a|s)}(\pi_\theta) - C \max_s D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot \mid s) \| \pi_\theta(\cdot \mid s)), \tag{9}$$

where $C = \frac{4\epsilon\gamma}{(1-\gamma)^2}\alpha^2$.

Intuition:

- The LHS is the expected return under the new policy. The first term on the RHS is the expected return under the old policy. The second term on the RHS is the surrogate objective.
- Taken together, the equation is saying that our new policy will get return that is at least as good as the old policy plus the surrogate objective, minus a term that tells us how much the two policies differ.
- We will think about maximizing the LHS. The first term on the LHS is a constant w.r.t. θ, so we can ignore that. The second term is the surrogate objective, so we'll try to maximize that; this is the same as the importance-weighted policy gradient introduced above. The third term is the discrepancy between the old and new policies, and we're trying to minimize this.
- So, taken together, we're maximizing the importance weighted surrogate objective and minimizing the deviation from the old policy.

Practical considerations. In practice, we can't compute the maximum KL divergence over all states, so we approximate it with an expectation over visited states. This leads to the final surrogate objective used by TRPO [3]:

$$\max_\theta \mathbb{E}_{\rho^{\pi_{\theta_{\text{old}}}}(s_t)\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} A(s_t, a_t) \log \pi_\theta(a_t \mid s_t)\right] \tag{10}$$

$$\text{s.t.} \quad \mathbb{E}_{\rho^{\pi_{\theta_{\text{old}}}}(s_t)\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \underbrace{\left[\log \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\right]}_{D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot|s_t) \| \pi_\theta(\cdot|s_t))} \leq \delta. \tag{11}$$

This method, Trust Region Policy Optimization (TRPO), is called a trust region method because is says that you only "trust" your surrogate objective to produce accurate policy gradients when your two policies are close to one another.

The key challenge with TRPO is that handling this KL divergence can be challenging in practice, and that the δ coefficient has to be carefully chosen.

4 Clipped Surrogate Objective

One of the key ideas in PPO is that, rather than constraining the old policy to be close to the new policy, we can directly *clip* the policy gradient (i.e., make it smaller) when the old and new policies are far apart from one another. Define the importance weight $r(s, a; \theta) \triangleq \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$. For an action that the old policy took much more frequently, we have

[1] Technically, we are using the "vine" surrogate objective used in Eq. 16 of [3].

$r(\theta) \approx 0$, so the importance weight means that effectively no gradient will be applied; good. For an action that the new policy takes much more frequently than the old policy, we'll have $r(\theta) = \frac{\text{large}}{\text{small}} = \text{very large}$; this is where the problem occurs. While the solution in TRPO was the regularize the policies to be close to one another, the solution in PPO is simply to clip this ratio:

$$\text{clip}(r(\theta), 1-\epsilon, 1+\epsilon), \tag{12}$$

where ϵ is a small positive number. Plugging this into the surrogate objective, we get:

$$\mathbb{E}_{\rho^{\pi_{\theta_{\text{old}}}}(s_t)\pi_{\theta_{\text{old}}}(a_t|s_t)}\left[\text{clip}(r(s_t,a_t;\theta), 1-\epsilon, 1+\epsilon)A(s_t,a_t)\log\pi_\theta(a_t \mid s_t)\right]. \tag{13}$$

Intuitively, what this means that that the optimization will increase the likelihood of actions with positive advantage, but only increasing the liklihoods by up to $(1+\epsilon)\times$.

One final trick. When the advantage is negative, we want to decrease the likelihood of those actions. However, the clipping may mean that we don't decrease the likelihood enough. So, PPO doesn't apply the clipping when the advantage is negative. The final PPO surrogate objective is

$$\mathbb{E}_{\rho^{\pi_{\theta_{\text{old}}}}(s_t)\pi_{\theta_{\text{old}}}(a_t|s_t)}\left[\min(r(s_t,a_t;\theta)A(s_t,a_t), \text{clip}(r(s_t,a_t;\theta), 1-\epsilon, 1+\epsilon)A(s_t,a_t))\log\pi_\theta(a_t \mid s_t)\right]. \tag{14}$$

5 Advantage Estimation

Learning the value function. We will learn a value function by regression to the empirical returns:

$$\min_\theta \frac{1}{2}(V_\theta(s_t) - \sum_{t'=t}^{\infty}\gamma^{t'-t}r_{t'})^2. \tag{15}$$

While the value function and policy are often implemented as sharing many of the layers, this seems to degrade performance [1].

Estimating the advantage. We will estimate the advantage by taking a Q value estimate and subtracting a value function, estimated above. We will estimate the Q values using the empirically observed returns up to time step $t+k$, whereupon we substitute the learned value function. Note that we can get several different Q value estimates by varying k:

$$\begin{aligned} Q(s_t,a_t) &= \mathbb{E}[r_t + \gamma V(s_{t+1})] && (16)\\ &= \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2})] && (17)\\ &= \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V(s_{t+3})] && (18)\\ &= \cdots. && (19)\end{aligned}$$

Each of these can be turned into an advantage estimator by subtracting the baseline (the value function at time t:

$$\begin{aligned} A(s_t,a_t) &= Q(s_t,a_t) - V(s_t) && ((1-\lambda))\\ &= \mathbb{E}[r_t + \gamma V(s_{t+1})] - V(s_t) && ((1-\lambda)\lambda)\\ &= \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2})] - V(s_t) && ((1-\lambda)\lambda^2)\\ &= \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V(s_{t+3})] - V(s_t) && ((1-\lambda)\lambda^3)\\ &= \cdots && (20)\end{aligned}$$

Which of these advantage estimators should we use? If we wait longer to substitute the terminal value function, then we have a method that relies more on Monte Carlo returns (low bias, high variance); if we immediately plug in the value function at $t+1$, then we are relying more on the learned value function (high variance, low bias).

One effective approach is to use all the estimators, taking a certain weighted combination. The math works out especially nicely if you combine these estimators with geometrically-decreasing weights (shown above):

$$A(s_t, a_t) = -V_t + r_t + (1-\lambda)\gamma V_{t+1} + (1-\lambda)(\underbrace{\lambda + \lambda^2 + \cdots}_{=\frac{\lambda}{1-\lambda}})\gamma r_{t+1} + \lambda(1-\lambda)V_{t+2}\gamma^2 + (1-\lambda)(\underbrace{\lambda^2 + \lambda^3 + \cdots}_{=\frac{\lambda^2}{1-\lambda}}\gamma^2 r_{t+2} + \cdots \tag{21}$$

$$= -V_t + r_t + (1-\lambda)\gamma V_{t+1} + \lambda\gamma r_{t+1} + \lambda(1-\lambda)V_{t+2}\gamma^2 + \lambda^2\gamma^2 r_{t+2} + \cdots \tag{22}$$

$$= (r_t + \gamma V_{t+1} - V_t) + \lambda\gamma(r_{t+1} + \gamma V_{t+2} - V_{t+1}) + \lambda^2\gamma^2(r_{t+2} + \gamma V_{t+3} - V_{t+2}) + \cdots \tag{23}$$

$$= \sum_{t'=t}^{\infty} (\lambda\gamma)^{t'-t}(r_{t'} + \gamma V_{t'+1} - V_{t'}). \tag{24}$$

This technique is known as generalized advantage estimation (GAE) [4]. It is applicable in many settings outside of PPO.

- Note that this looks like a discounted sum of rewards, where the rewards are *shaped* by the value function. The discount is $\tilde{\gamma} = \lambda\gamma$.
- Note that the term above looks like a TD error.
- In practice, we only do this up to $t' = t + T$ time steps. This allows us to collect T steps of experience with a bunch of actors in parallel; this can be much less than the horizon length
- Some implementation normalize the advantages for all transitions before computing the policy gradient.

5.1 (Optional) Invariant Reward Shaping [2]

At the start of this class, we looked at how some reward functions can give the same optimal policy (e.g., adding a constant). There's a similar idea of augmenting rewards that keeps the optimal policy the same. For an arbitrary *potential* function $\phi(s_t)$, define the augmented rewards:

$$\tilde{r}(s_t, a_t) = r(s_t, a_t) + \gamma\phi(s_{t+1}) - \phi(s_t). \tag{25}$$

The discounted sum of returns for the new and old rewards are the same, up to a term that depends only on the initial state. As the initial state is not controlled by the agent, the optimal policies for these two rewards are the same:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \tilde{r}(s_t, a_t)\right] = \mathbb{E}\left[\sum_{t=0}^{\infty} r(s_t, a_t) + \gamma\phi(s_{t+1}) - \phi(s_t)\right] \tag{26}$$

$$= \mathbb{E}\left[\sum_{t=0}^{\infty} r(s_t, a_t)\right] + \mathbb{E}[\gamma\phi(s_1) + \gamma^2\phi(s_2) + \gamma^3\phi(s_3) + \cdots] \tag{27}$$

$$- \mathbb{E}[\phi(s_0) + \gamma\phi(s_1) + \gamma^2\phi(s_2) + \gamma^3\phi(s_3) + \cdots] \tag{28}$$

$$= \mathbb{E}\left[\sum_{t=0}^{\infty} r(s_t, a_t)\right] + \cancel{\gamma^{\infty}\phi(s_{t+\infty}} - \phi(s_0). \tag{29}$$

The first step plugs in the definition of $\tilde{r}$. The second step applies linearity of expectation. We then recognize the telescoping summation. Finally, we note that $\gamma^{\infty} = 0$

6 PPO: Algorithm Summary

Algorithm 1 PPO [5]

Initialize policy $\pi_\theta(a \mid s)$, value function $V_\theta(s)$.
while not converged **do**
  Collect T environment steps from N actors (in parallel).
  Compute advantages for all $N \cdot T$ transitions using GAE.
  Take gradient of surrogate objective w.r.t. θ, doing a total of K passes over the data (corresponding to many minibatches). ▷ Uses off-policy data!
  Update value function with gradient descent.

Comparison to alternative methods:

- More sample efficient than REINFORCE (multiple gradients with same data) and more stable (more data for each gradient allows for lower variance gradients.
- More simple than TRPO (no trust regions), but more complex than REINFORCE.
- Less sample efficient than DQN, but can handle continuous actions.

References

[1] Huang, S., Dossa, R. F. J., Raffin, A., Kanervisto, A., and Wang, W. (2022). The 37 implementation details of proximal policy optimization. *The ICLR Blog Track 2023*.

[2] Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287. Citeseer.

[3] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015a). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR.

[4] Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

[5] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.