Proximal Policy Optimisation (PPO)

Edward Young - ey245@cam.ac.uk

Quick facts

1. PPO is an *actor-critic* method. It learns a policy (an *actor*) and a value function (a *critic*).
2. PPO is *on-policy*.
3. PPO is *model-free*.
4. PPO is applicable to any kind of state space and action space.

High-level strategy

PPO can be seen as a successor to A2C. As in A2C, PPO learns a value network which is used for form **advantage estimates**, and uses these advantage estimates to critique action choices. Also in common with A2C, PPO is a **on-policy** method, it collects a batch of experiences, uses those experiences to train its networks, and then discards them. This is in contrast to DDPG and DQN, which *reuse* experiences through a **memory reply buffer**. However, PPO and A2C use different objectives to train the policy network. In A2C, we could only perform one gradient step on our policy objective per batch of experiences while remaining on-policy. PPO uses a different objective which allows us to take multiple gradient steps per batch, and is therefore much more **sample efficient**.

Training the value network

For concreteness, consider training a state-value network $V(s;\phi)$. Suppose we have just collected a new batch of experiences (s_i, a_i, r_i, s'_i) for $i = 1, \dots, M$. We use our standard procedure to train the value network - mini-batch gradient descent on a **mean squared error (MSE) loss**:

$$\mathcal{L}(\phi) = \frac{1}{N} \sum_{i \in B} [V(s_i;\phi) - y_i]^2 \tag{1}$$

The sum is over a **mini-batch** of N data points, B, and y_i is the regression target for transition i. We can form regression targets in any way applicable to A2C - for example, the one-step return:

$$y_i = r_i + \gamma V(s'_i;\phi) \tag{2}$$

For each batch we perform multiple mini-batch gradient steps, each time sampling a mini-batch B, forming the objective (1), finding the gradient, and updating ϕ. As in A2C, regression targets are formed immediately after collecting the batch, then remain fixed during mini-batch gradient descent.

Training the policy network

Training the policy network in PPO has a similar structure to training the value network. First, we iterate through the batch, this time forming **advantage estimates** for each transition. Then we sample mini-batches, and use the advantage estimates to form the **PPO policy objective**, which we perform gradient *ascent* on.

We already covered how to form advantage estimates. One method is the **TD error**,

$$\hat{A}_i = r_i + \gamma V(s'_i;\phi) - V(s_i;\phi) \tag{3}$$

The **PPO policy objective** formed through this mini-batch is

$$J(\theta) = \frac{1}{N} \sum_{i \in B} \min \left(\hat{A}_i \frac{\pi(a_i|s_i;\theta)}{\pi\left(a_i|s_i;\theta^{\text{old}}\right)}, \hat{A}_i \text{clip} \left(\frac{\pi(a_i|s_i;\theta)}{\pi\left(a_i|s_i;\theta^{\text{old}}\right)}, 1-\epsilon, 1+\epsilon \right) \right) \tag{4}$$

Where θ^{old} is the policy parameters at the start of training on this batch.

Conclusion: Putting it all together

The PPO algorithm loops through the following steps:

1. Gather a batch of experiences through interaction with the environment.
2. Update the value network:
   (a) Form the **regression targets** y_i, (2), for each transition.
   (b) For a number of value network training steps:
      i. Sample a minibatch of transitions, B
      ii. Form the **MSE loss** $\mathcal{L}(\phi)$, (1)
      iii. Perform gradient *descent* on this loss.
3. Update the policy network:
   (a) Form the **advantage estimates** $\hat{A}_i$, (3), for each transition.
   (b) For a number of policy network training steps:
      i. Sample a minibatch of transitions, B
      ii. Form the **PPO policy objective** $J(\theta)$, (4)
      iii. Perform gradient *ascent* on this objective.

Appendix: Where does the PPO objective come from?

In this section we will attempt to shed some light on the PPO objective and motivate where it comes from. What we really care about is the **value** of a policy, *i.e.*, the expected cumulative reward:

$$V(\theta) := \mathbb{E}\left[\sum_{t=0}^{T} r_t \middle| a_t \sim \pi(a|s_t;\theta)\right] \tag{5}$$

When θ is close to θ^{old}, the value difference is approximately the **surrogate advantage**:

$$V(\theta) - V\left(\theta^{\text{old}}\right) \approx \mathbb{E}\left[\sum_{t=0}^{T} \frac{\pi(a_t|s_t;\theta)}{\pi\left(a_t|s_t;\theta^{\text{old}}\right)} A\left(s_t, a_t|\theta^{\text{old}}\right) \middle| a_t \sim \pi\left(a|s_t;\theta^{\text{old}}\right)\right] \tag{6}$$

This means that, provided θ is close to θ^{old}, maximising the value of the policy can be done by performing mini-batch gradient ascent on the objective:

$$\frac{1}{N}\sum_{i\in B} \hat{A}_i \frac{\pi(a_i|s_i;\theta)}{\pi\left(a_i|s_i;\theta^{\text{old}}\right)} \tag{7}$$

There's only one issue. If we perform gradient ascent on this objective, the new policy becomes far away from the old policy, and the surrogate advantage ceases to approximate the value difference. We therefore remove the incentive for the policy to deviate from the old policy. Our new objective will be:

$$J(\theta) = \frac{1}{N}\left(\sum_{i\in B:\hat{A}_i>0} \hat{A}_i \min\left(\frac{\pi(a_i|s_i;\theta)}{\pi\left(a_i|s_i;\theta^{\text{old}}\right)}, 1+\epsilon\right) + \sum_{i\in B:\hat{A}_i<0} \hat{A}_i \max\left(\frac{\pi(a_i|s_i;\theta)}{\pi\left(a_i|s_i;\theta^{\text{old}}\right)}, 1-\epsilon\right)\right) \tag{8}$$

This turns out to be identical to the PPO policy objective (4).