Trust Region and Proximal policy optimization

Jan 11, 2019 6 mins read

Trust Region and Proximal policy optimization

Welcome to another journey towards unraveling the secrets behind Reinforcement Learning. This time, we going to take a step back and return to policy optimization in order to introduce two new methods: trust region policy optimization (TRPO) and proximal policy optimization (PPO). Remember that in policy gradients techniques, we try to optimize a policy objective function (the expected accumulative reward) using gradient descent. Policy gradients are great for continuous and large spaces but suffer from some problems.

- High variance (which we address with Actor-critic models)
- Delayed reward problem
- Sample inefficiency
- Learning rate highly affects training

Especially the last one troubled researchers for quite a long, because it is very hard to find a suitable learning rate for the whole optimization process. Small learning rate may cause vanishing gradients while large rate may cause exploding gradient. In general, we need a method to change the policy not too much but also not too little and even better to always improve our policy. One fundamental paper in this direction is :

Trust region policy optimization (TRPO)

To ensure that the policy won't move too far, we add a constraint to our optimization problem in terms of making sure that the updated policy lies within a trust region. Trust regions are defined as the region in which the local approximations of the function are accurate. Ok, but what does that mean? In trust regions, we determine the maximum step size and then we find the local maximum of the policy within the region. By continuing the same process iteratively, we find the global maximum. We can also expand or shrink the region based on how good the new approximation is. That way we are certain that the new policies can be trustworthy of not leading to dramatically bad policy degradation. We can express mathematically the above constraint using KL divergence(which you can think of as a distance between two probabilities distributions):

The KL divergence between the new and the old policy must be lower than the delta (**δ**), where delta is the size of the region. I could get into some math, but I think that this will only complicate things rather than clarify them. So essentially, we have just a constrained optimization problem.

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right] \\ \text{subject to} \quad & \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta. \end{aligned}$$

The question now is how we solve a constrained optimization problem? Using the Conjugate Gradient method. We can, of course, solve the problem analytically (natural gradient descent), but it is computational ineffective. If you dust of your knowledge on numerical mathematics, you might remember that the conjugate gradient method provides a numeric solution to a system of equations. That is way better from a computational perspective. So all we have to do is to approximate linearly the objective function and quadratically the constraint and let the conjugate gradient do its work.

To wrap it all up, the algorithm has the following steps:

- We run a set of trajectories and collect the policies
- Estimate the advantages using advantage estimation algorithm
- Solve the constrained optimization problem using conjugate gradient
- Repeat

Generally speaking, trust regions are considered pretty standard methods to approach optimization problems. The tricky part is to apply them in a reinforcement learning context in a way that provides an advantage over simple policy gradients.

Although TRPO is a very powerful algorithm, it suffers from a significant problem: that bloody constraint, which adds additional overhead to our optimization problem. I mean it forces us to use the conjugate gradient method and baffled us with linear and quadratic approximations. Wouldn't it be nice if the could somehow include the constraint directly into our optimization objective? As you might have guessed that is exactly what Proximal policy optimization does.

Proximal policy optimization (PPO)

This simple idea gave us a quite simpler and more intuitive algorithm than TRPO. And it turns out that it outperforms many of the existing techniques most of the time. So instead of adding a constraint separately, we incorporate it inside the objective function as a penalty (we subtract the KL divergence times a constant C from the function).

$$\underset{\theta}{\text{maximize}} \sum_{n=1}^{N} \frac{\pi_\theta(a_n \mid s_n)}{\pi_{\theta_{\text{old}}}(a_n \mid s_n)} \hat{A}_n - C \cdot \overline{\text{KL}}_{\pi_{\theta_{\text{old}}}}(\pi_\theta)$$

As soon as we do that there is no need to solve a constrained problem and we can instead use a simple stochastic gradient descent in the above function. And the algorithm is transformed as follows:

- We run a set of trajectories and collect the policies
- Estimate the advantages using an advantage estimation algorithm
- Perform stochastic gradient descent on the objective function for a certain number of epochs
- Repeat

A small caveat is that is hard to choose the coefficient *C* in a way that it works well over the whole course of optimization. To address that we update the coefficient based on how big or small the KL divergence is. If KL is too high, we increase it, or if it is too low, we decrease it.

So this is it? This is the famous proximal policy optimization? Actually no. Sorry about that. It turns out that the function described above is not the same as the original paper. The authors found a way to improve this penalized version into a

new, more robust objective function.

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

Hey hey wait. What happened here? Actually, not much. Let me explain. One thing I intentionally left out so far is that fraction of probabilities over there, that seems to appear in TRPO also. Well, that is called importance sampling. We essentially have a new policy that we want to estimate and an old one that we use to collect the samples. With importance sampling, we can evaluate a new policy with the samples from the old one and improve sample efficiency. The ratio infers how different the two policies are and we denote is as r(theta).

Using this ratio, we can construct a new objective function to clip the estimated advantage if the new policy is far away from the old one. And that's exactly what the above equation does. If an action is a lot more likely under the new policy than the old one, we do not want to overdo the action update, so we clipped the objective function. If it is much less likely under the new policy than the old, the objective action is flattened out to prevent from overdoing the update once again.

It may be just me, but I think I haven't come across a simpler and cleaner reinforcement learning algorithm in a while.

If you want to get into the rabbit hole, check out the baseline code from OpenAI, which will give you a crystal clear image of the whole algorithm and solve all your questions. Of course, it would be much better if you try to implement it from scratch all by yourself. It is not the easiest task, but why not.

I think that's it for now. Keep learning...

Previous
‹ The idea behind Actor-Criti...

Next
Semantic Segmentation in th... ›