Jonathan Hui Follow

Sep 17, 2018 · 9 min read · Listen

RL — Proximal Policy Optimization (PPO) Explained

Photo by Pietro De Grandi

A quote from OpenAI on PPO:

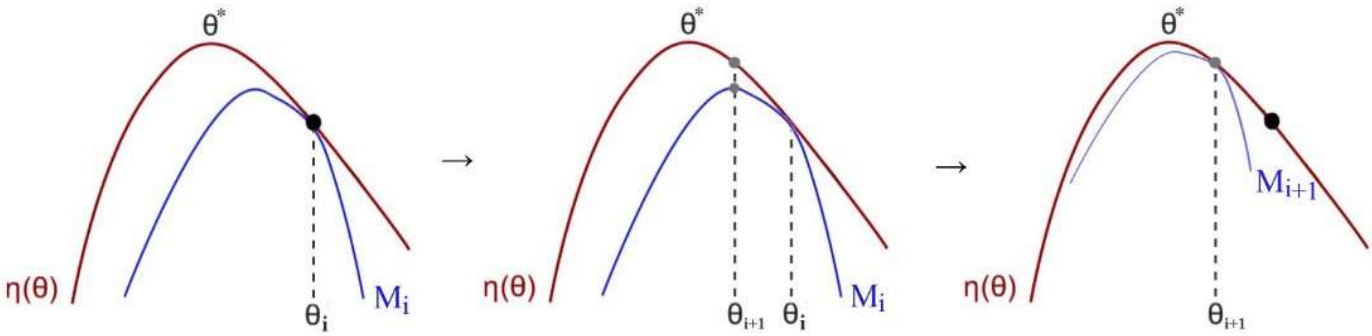
> *Proximal Policy Optimization (PPO), which perform comparably or better than state-of-the-*

gradient. However, in practice, natural policy gradient involves a second-order derivative matrix which makes it not scalable for large scale problems. The computational complexity is too high for real tasks. Intensive research is done to reduce the complexity by approximate the second-order method. PPO uses a slightly different approach. Instead of imposing a hard constraint, it formalizes the constraint as a penalty in the objective function. By not avoiding the constraint at all cost, we can use a first-order optimizer like the Gradient Descent method to optimize the objective. Even we may violate the constraint once a while, the damage is far less and the computation is much simple. Let's go through quickly on the basic concepts before explaining PPO in details.

Minorize-Maximization MM algorithm

How can we optimize a policy to maximize the rewards?

With the Minorize-Maximization **MM** algorithm, this is achieved **iteratively** by maximizing a lower bound function ***M*** (the blue line below) approximating the expected reward η locally.



Modified from source

First, we start with an initial policy guess and find a lower bound M for η at this policy. We optimize M and use the optimal policy for M as the next guess. We approximate a new lower bound again at the new guess and repeat the iterations until the policy converges. To make it works, we do need to find a lower bound M that is easier to optimize.

objective function. This is why it is so popular in deep learning even more accurate methods are available.

Line search first picks the steepest direction and then move forward by a step size. But how can this strategy go wrong in reinforcement learning RL? Let's take a robot to the Angels Landing for hiking. As shown below, we ascend the hill by determining the direction first. If the step size is too small, it will take forever to get to the peak. But if it is too big, we can fall down the cliff. Even if the robot survives the fall, it lands in areas with height much lower than where we were. Policy gradient is mainly an on-policy method. It searches actions from the current state. Hence, we resume the exploration from a bad state with a locally bad policy. This hurts performance badly.

Modified from Source

Trust region

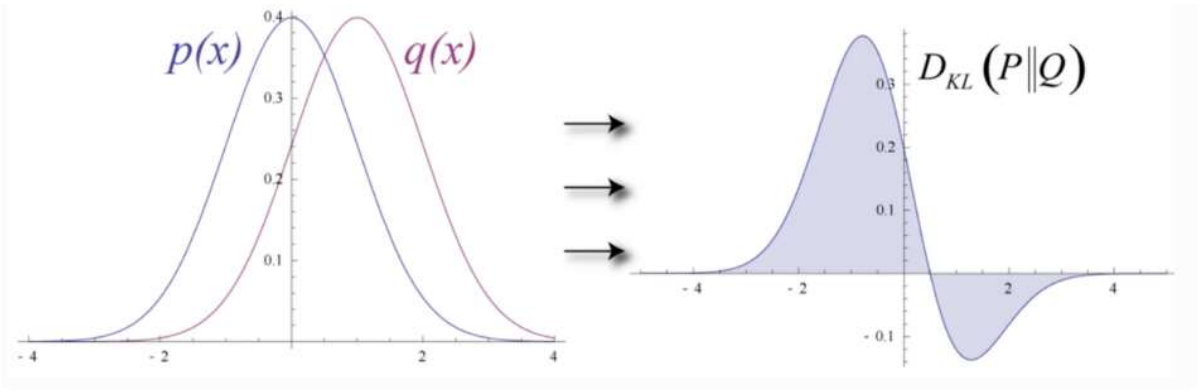
resume the search from there.

Modified from Source

What is the maximum step size in a trust region? In the trust region method, we start with an initial guess. Optionally, we can readjust the region size dynamically. For example, we can shrink the region if the divergence of the new and current policy is getting large (or vice versa). In order not to make bad decisions, we can shrink the trust region if the policy is changing too much.

In PPO, we limit how far we can change our policy in each iteration through the KL-divergence. KL-divergence measures the difference between two data distributions p and q.

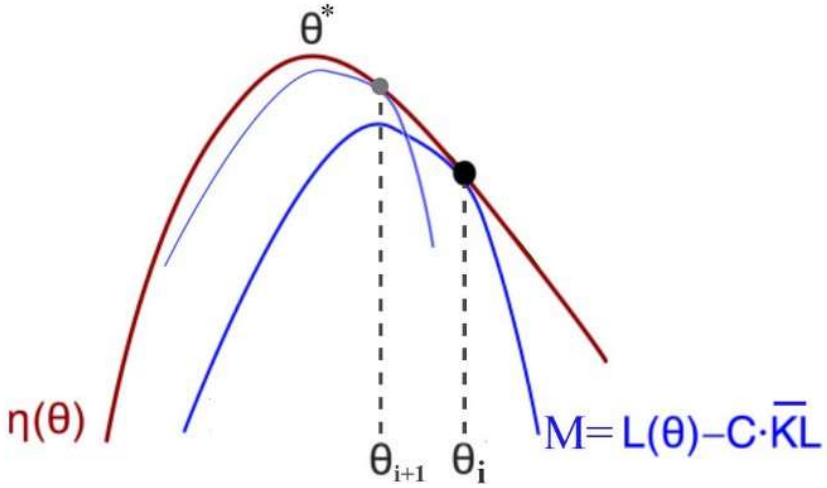
$$D_{KL}(P||Q) = \mathbb{E}_x \log \frac{P(x)}{}$$



Source: Wikipedia

So how can we limit policy change to make sure we don't make bad decisions? It turns out we can find a lower bound function ***M*** as

$$M = L(\theta) - C \cdot \overline{KL}$$



with ***L(θ)*** equals

$$\hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right]$$

using the probability ratio between the new and the old policy. We use the advantage function instead of the expected reward because it reduces the variance of the estimation. As long as the baseline does not dependent on our policy parameters, the optimal policy will be the same.

Let's look into more detail on the second term in ***M***. After two page of proof in the TRPO paper, we can establish the following lower bound.

$$\eta(\tilde{\pi}) \geq L_{\pi}(\tilde{\pi}) - CD_{\text{KL}}^{\max}(\pi, \tilde{\pi}), \text{ where } C = \frac{4\epsilon\gamma}{(1-\gamma)^2}.$$

$$D_{\text{KL}}^{\max}(\pi, \tilde{\pi}) = \max_s D_{\text{KL}}(\pi(\cdot|s) \parallel \tilde{\pi}(\cdot|s)).$$

$$\epsilon = \max_{s,a} |A_{\pi}(s,a)|$$

The second term in ***M*** is the maximum of KL-divergence underlined in red above. But it is too hard to find and therefore we relax the requirement a little bit by using the mean of the KL-divergence instead. Let's explain that intuitively.

Intuition

L approximates the advantage function locally at the current policy. But it gets less accurate as it moves away from the old policy. This inaccuracy has an upper bound. That is the second term in ***M***. After considering the upper bound of this error, we can guarantee that the calculated optimal policy within the trust region is always better than the old policy. If the policy is outside the trust region, even the calculated value may be better but the accuracy can be too off and cannot be trusted. So the bet is off. Let's summarize the objective below as:

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

or

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t\right] - \beta\hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]]$$

Mathematically, both equations above can be resolved to the same optimal policy. However, the theoretical threshold for δ is very small and is considered to be too conservative. So we relax the condition once more by setting them as tunable hyperparameters.

As mentioned before, M should be easy to optimize. So we further approximate it to a quadratic equation which is a convex function and heavily study on how to optimize it in high dimensional space.

We use Taylor's series to expand the terms up to the second-order. But the second-order of $\mathcal{L}$ is much smaller than the KL-divergence term and will be ignored.

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

$$\mathcal{L}_{\theta_k}(\theta) \approx \mathcal{L}_{\theta_k}(\theta_k)^{0} + g^T(\theta - \theta_k) + \ldots$$

$$\mathcal{L}_{\theta_k}(\theta) \approx g^T(\theta - \theta_k) \qquad g \doteq \nabla_\theta \mathcal{L}_{\theta_k}(\theta)\,|_{\theta_k}$$

$$\bar{D}_{KL}(\theta||\theta_k) \approx \frac{1}{2}(\theta - \theta_k)^T H(\theta - \theta_k) \qquad H \doteq \nabla^2_\theta \bar{D}_{KL}(\theta||\theta_k)\,|_{\theta_k}$$

Our objective becomes:

$$\theta_{k+1} = \arg\max_\theta \; g^T(\theta - \theta_k)$$
$$\text{s.t.} \;\; \frac{1}{2}(\theta - \theta_k)^T F\,(\theta - \theta_k) \le \delta$$

We can solve this quadratic equation analytical. The solution is:

$$\theta_{k+1} = \theta_k + \underbrace{\sqrt{\frac{2\delta}{g^T F^{-1} g}}}_{\text{natural policy gradient}} F^{-1} g$$

What are the challenges?

This solution involves the calculation of the second-order derivative and its inverse, a very expensive operation.

$$F = \nabla^2 f = \begin{pmatrix} \frac{\partial^2 f_1}{\partial x_1^2} & \frac{\partial^2 f_1}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f_1}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f_2}{\partial x_1^2} & \frac{\partial^2 f_2}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f_2}{\partial x_1 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f_m}{\partial x_1^2} & \frac{\partial^2 f_m}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f_m}{\partial x_1 \partial x_n} \end{pmatrix}$$

So there are two approaches to address this problem:

- Approximate some calculations involving the second order derivative and its

TRPO and ACKTR adopt the first approach. PPO is closer to the second one. We can still live with a bad policy decision once a while so we stick with the first-order solution like the stochastic gradient descent. But we are going to add a soft constraint to the objective function so the optimization will have better insurance that we are optimizing within a trust region. So the chance of bad decision is smaller.

PPO with Adaptive KL Penalty

One way to formulate our objective is changing the constraint to a penalty in the objective function:

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] - \beta\hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]]$$

β controls the weight of the penalty. It penalizes the objective if the new policy is different from the old policy. Borrow a page from the trust region, we can dynamically adjust β. d below is the KL-divergence between the old and the new policy. If it is higher than a target value, we shrink β. Similarly, if it falls below another target value, we expand the trust region.

$$d = \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]]$$

Here is the algorithm:

Input: Initial policy parameters θ_0, initial KL penalty β_0, target KL-divergence δ
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta||\theta_k)$$

  by taking K steps of minibatch SGD (via Adam)
  if $\bar{D}_{KL}(\theta_{k+1}||\theta_k) \geq 1.5\delta$ **then**
    $\beta_{k+1} = 2\beta_k$
  else if $\bar{D}_{KL}(\theta_{k+1}||\theta_k) \leq \delta/1.5$ **then**
    $\beta_{k+1} = \beta_k/2$
  end if
end for

Source

This gets us the performance of TRPO with speed closer to the gradient descent method. But can we do better?

PPO with Clipped Objective

PPO with clipped objective can even do better. In its implementation, we maintain two policy networks. The first one is the current policy that we want to refine.

$$\pi_\theta(a_t|s_t)$$

The second is the policy that we last used to collect samples.

$$\pi_{\theta_k}(a_t|s_t)$$

With the idea of importance sampling, we can evaluate a new policy with samples collected from an older policy. This improves sample efficiency.

$$\text{maximize} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{} \hat{A}_t \right]$$

bad decision because of the inaccuracy. So, say for every 4 iterations, we synchronize the second network with the refined policy again.

$$\pi_{\theta_{k+1}}(a_t|s_t) \longleftarrow \pi_\theta(a_t|s_t)$$

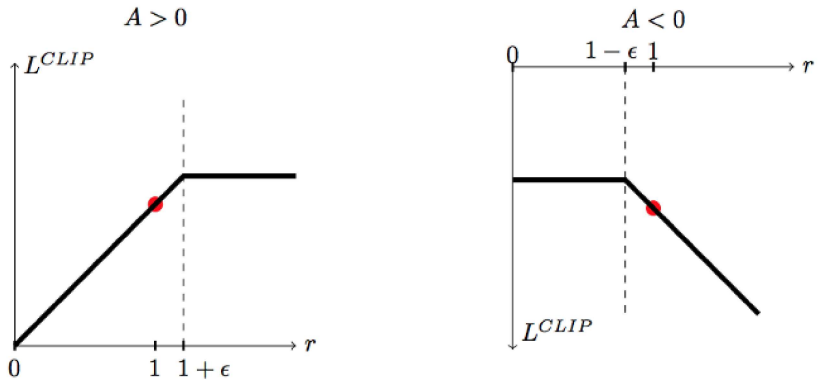
With clipped objective, we compute a ratio between the new policy and the old policy:

$$r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_k}(a_t|s_t)$$

This ratio measures how difference between two policies. We construct a new objective function to clip the estimated advantage function if the new policy is far away from the old policy. Our new objective function becomes:

$$\mathcal{L}^{CLIP}_{\theta_k}(\theta) = \mathop{\mathrm{E}}_{\tau\sim\pi_k}\left[\sum_{t=0}^{T}\left[\min(r_t(\theta)\hat{A}^{\pi_k}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}^{\pi_k}_t)\right]\right]$$

If the probability ratio between the new policy and the old policy falls outside the range (1 — ε) and (1 + ε), the advantage function will be clipped. ε is set to 0.2 for the experiments in the PPO paper.



Source

Effectively, this discourages large policy change if it is outside our comfortable zone.

Input: initial policy parameters θ_0, clipping threshold ϵ
for $k = 0, 1, 2, \ldots$ **do**
Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

by taking K steps of minibatch SGD (via Adam), where

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k} \left[\sum_{t=0}^{T} \left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k}) \right] \right]$$

end for

Source

This new method is simple and can use Gradient Descent like Adam to optimize it. In fact, it is anticlimax for taking so detail analysis on the issue but come up with such a simple solution.

Thoughts

A quote from the PPO paper:

> *Q-learning (with function approximation) fails on many simple problems and is poorly understood, vanilla policy gradient methods have poor data efficiency and robustness; and trust region policy optimization (TRPO) is relatively complicated, and is not compatible with architectures that include noise (such as dropout) or parameter sharing (between the policy and value function, or with auxiliary tasks).*

PPO adds a soft constraint that can be optimized by a first-order optimizer. We may make some bad decisions once a while but it strikes a good balance on the speed of the optimization. Experimental results prove that this kind of balance achieves the best performance with the most simplicity.

> *Simplicity rules in deep learning.*

[UC Berkeley RL course]

[UC Berkeley RL Bootcamp]