"Implementation Matters in Deep RL: A Case Study on PPO and TRPO"

Engstrom et al.

2020

TLDR

Authors investigated a standard implementations of PPO and found out that optimization tricks that are used in it are even more influential than the algorithm itself.

PG vs TRPO vs PPO

The baseline model we are going to consider is an off-policy policy gradient (with importance sampling):

$$J_{\mathrm{PG}}(\theta) = \mathop{\mathbb{E}}_{(s_t,a_t)\sim\pi}\left[\frac{\pi_\theta\left(a_t|s_t\right)}{\pi\left(a_t|s_t\right)}\hat{A}_\pi\left(s_t,a_t\right)\right] \tag{1}$$

where $\hat{A}_\pi$ is the advantage function. It is a common part between TRPO and PPO objectives and is equivalent to PPO without clipping (or TRPO without KL constraint).

TRPO enforces the change in our policy method not to move too much from the current one. This is achieved by bounding KL distance at each iteration step:

$$\begin{array}{ll}\max_\theta & J_{\mathrm{PG}}(\theta) \\ \text{s.t.} & D_{KL}\left(\pi_\theta(\cdot|s)\|\pi(\cdot|s)\right) \le \delta\end{array} \tag{2}$$

TRPO uses a second-order approximation of KL with a natural gradient descent, which makes it expensive to compute. PPO was proposed to solve this problem by bounding policy update in a much simpler way: just by clipping the objective function:

$$\max_\theta \mathop{\mathbb{E}}_{(s_t,a_t)\sim\pi}\left[\min\left(\mathrm{clip}\left(\rho_t, 1-\varepsilon, 1+\varepsilon\right)\hat{A}_\pi\left(s_t,a_t\right), \rho_t\hat{A}_\pi\left(s_t,a_t\right)\right)\right] \tag{3}$$

where

$$\rho_t = \frac{\pi_\theta\left(a_t|s_t\right)}{\pi\left(a_t|s_t\right)}. \tag{4}$$

The motivation behind PPO is the following:

- If clip is not applied: gradient always flows through ρ_t.
- If clip is applied: gradient flows through ρ_t only if it makes things worse (i.e. decreases the objective) since we want to improve it.

PPO implementation tricks

In the original paper and in open-source implementations there is a myriad of tricks used to improve PPO performance:

1. Value function clipping: a loss function for the value function is clipped in a PPO-like manner:

$$L^V = \min\left[\left(V_{\theta_t} - V_{targ}\right)^2, \left(\mathrm{clip}\left(V_{\theta_t}, V_{\theta_{t-1}} - \varepsilon, V_{\theta_{t-1}} + \varepsilon\right) - V_{targ}\right)^2\right] \tag{5}$$

2. Reward scaling: divide the rewards by std of their rolling discounted sum;

3. Orthogonal init + layer scaling: initialize policy and value function layers orthogonally and scale each of them individually;
4. Learning rate annealing for Adam (i.e. gradually decrease LR during training)
5. Reward clipping and observation clipping;
6. Normalize observations before feeding them into the model;
7. Gradient clipping;
8. Use tanh instead of relu.

These changes may look like insignificant optimizations, but they change the picture dramatically. And if the PPO authors would use them for TRPO as well, it would considerably outperform PPO.

More fair benchmarks

Authors run several benchmarks for PPO and TRPO to understand what is the main driver of the improved performance for PPO. For each benchmark they perform a grid search over the best hyperparameters and tricks to use. For the best setup for each benchmark they run several experiments with different random seeds. The results are presented below.

	WALKER2D-V2	HOPPER-V2	HUMANOID-V2
PG +tricks	2867	2371	831
PG +PPO	2735	2142	674
PG +TRPO	2791	2043	586
PG +PPO +tricks	3292	2513	806
PG +TRPO+tricks	3050	2466	1030

From this table one can note that using tricks gives more boost than just substituting PPO objective for TRPO objective.

References

[1] Logan Engstrom et al. "Implementation Matters in Deep RL: A Case Study on PPO and TRPO". In: *International Conference on Learning Representations*. 2020. URL: https://openreview.net/forum?id=r1etN1rtPB.