Summary of the paper:

Proximal Policy Optimization Algorithms

Ignacio Monardes

February 6, 2026

Referencia

Authors: John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov
Title: *Proximal Policy Optimization Algorithms*
Conference/Journal:
Link: `https://arxiv.org/pdf/1707.06347`

1 Motivation

Current Problem

TRPO is very complex to implement in many scenarios and bad with noisy environments. Deep Q-Learning is difficult is not very good on many simple and continuous tasks.

There is room for a robust, scalable and efficient model.

Relevance of the problem

An scalable algorithm for many environments and easy to implement.

Hole in the literature / Previous limitations

2 Main Idea

Reinforce Gradient

First of all, the expected value to maximize is:

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] = \sum_{\tau} p_\theta(\tau) R(\tau)$$

We can notice that:

$$\nabla_\theta p(\tau) = p(\tau) \nabla_\theta \log p(\tau)$$

Demonstration:

$$\nabla_\theta \log p(\tau) = \frac{\nabla_\theta p(\tau)}{p(\tau)} \Rightarrow \nabla_\theta p(\tau) = p(\tau) \nabla_\theta \log p(\tau)$$

The probability of a trajectory τ is defined by:

$$
\begin{aligned}
p_\theta(\tau) &= p(s_0) \prod_t \pi_\theta(a_t \mid s_t) p(s_{t+1} \mid s_t, a_t) \\
\log p_\theta(\tau) &= \log p(s_0) + \sum_t \log \pi_\theta(a_t \mid s_t) + \sum_t \log p(s_{t+1} \mid s_t, a_t) \\
\nabla_\theta \log p_\theta(\tau) &= \sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)
\end{aligned}
$$

Then:

$$
\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t) G_t\right]
$$

We can make rest a bias and define the advantage $A(s_t, a_t) := Q(s_t, a_t) - V(s_t)$, resulting in:

$$
\hat{g} = \hat{\mathbb{E}}_t\left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t\right]
$$

TRPO

TRPO maximizes this problem:

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

with a hard constraint over the restriction.

Theorically we could use the restriction on the function with a coefficient β:

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

this would make a lower bound of the value.

The problem here is that β shouldn't be fixed and with additional modfications to Stochastic Gradient Descent.

Innovation: CLIP, Adaptative KL

We define the ratio $r_t(\theta) := \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$.

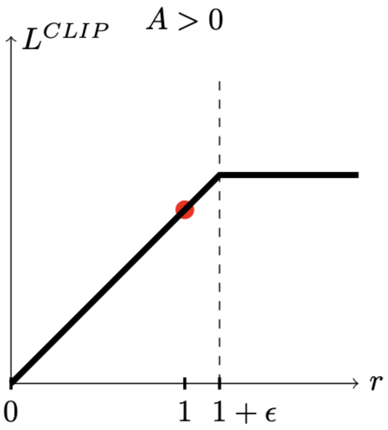
TRPO maximizes a surrogate objective:

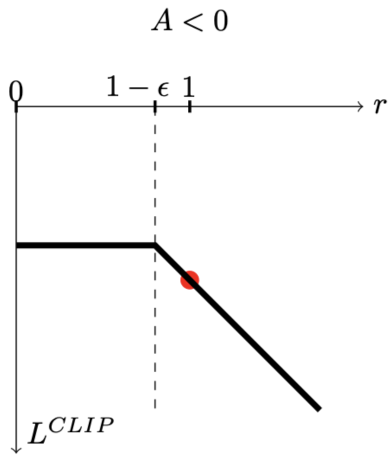
$$
L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t\right] = \hat{\mathbb{E}}_t[r_t(\theta)\hat{A}_t]
$$

where CPI refers to *Conservative Policy Iteration*. But the gradient could be very large. So we clipped it.

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

This allows to move the gradient in no more than the range $[1-\varepsilon, 1+\varepsilon]$, and as we take the minnimum we create a lower bound.





KL Penalty coeff

Another option to the clipped objective is to use a penalty on KL divergence. This performed worse than the clipped version.

3 Methodology

The methodology is just to adapt the Loss function to the desired loss with clipped or other thing and use stochastci gradient ascent. We could use a learned state value $V(s)$ or the finite-horizon estiamters. If our neural network shares parameters between the policy and value function, we must use a loss function that uses both of them. We can add exploration by using entropy bonus.

We finally have this loss function:

$$L_t^{CLIP+VF+S}(\theta) := \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) + c_1 L_t^{VF}(\theta) + c_2 S[\phi_\theta](s_t) \right]$$

where S is an entropy bonus and L_t^{VF} is a squared error loss.

It is very used an estimator of the advantage as:

$$\begin{aligned} \hat{A}_t &:= \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \\ \text{where} \quad \delta_t &= r_t + \gamma V(s_{t+1}) - V(s_t) \end{aligned}$$

with $T <<$ Length of the episode.

Modeling

Algorithm 1: Pseudo code PPO

```
for i in range(n_iters):
    for j in range(n_envs):
        buffer, next_reward_AC = run_policy(steps=T)
        advantages = compute_advantages(buffer, next_reward_AC)
    # M <= NT
    L = loss(epochs=K, batch_size=M)
```

```
    weights_old = weights
```

Assumptions

Architecture

The policy is made up of a fully connected MLP with two hidden layers of 64 units, and tanh nonlinearities, outputting the mean of a Gaussian distribution. The parameters of the policy and the value function were not shared. There is no entropy bonus.

4 Experiments

Baselines

The baselines are:

- TRPO.
- Cross-entropy method (CEM).
- Vanilla policy gradient with adaptative stepsize.
- A2C (Advantage Actor Critic, which is synchronous A3C).

Environments or datasets

They tested the problem with 7 simulated robotics tasks in MuJoCo physics engine. For each task they do 1M timesteps of training. The tasks were runned with 4 different seeds. The score is the average reward of the last 100 episodes. The scores are normalized to the scale [0, 1] and averaged over 21 runs.

algorithm	avg. normalized score
No clipping or penalty	-0.39
Clipping, $\epsilon = 0.1$	0.76
Clipping, $\epsilon = 0.2$	**0.82**
Clipping, $\epsilon = 0.3$	0.70
Adaptive KL $d_{\text{targ}} = 0.003$	0.68
Adaptive KL $d_{\text{targ}} = 0.01$	0.74
Adaptive KL $d_{\text{targ}} = 0.03$	0.71
Fixed KL, $\beta = 0.3$	0.62
Fixed KL, $\beta = 1.$	0.71
Fixed KL, $\beta = 3.$	0.72
Fixed KL, $\beta = 10.$	0.69

PPO outperforms almos all the continuos control environments compared to the others algorithms.

PPO works well on humanoid robotics tasks:

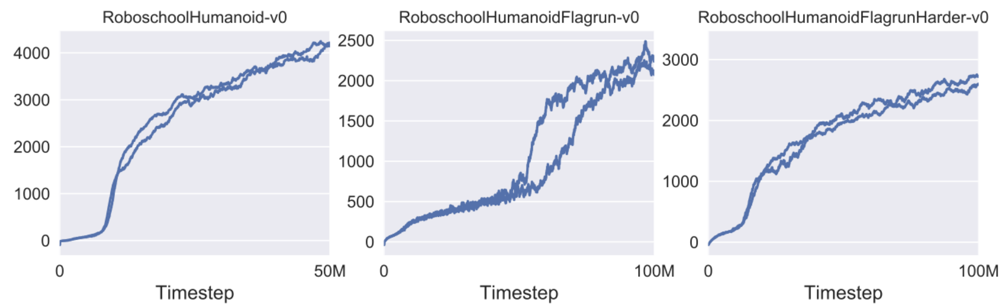


Figure 4: Learning curves from PPO on 3D humanoid control tasks, using Roboschool.

Settings

New Scenarios

5 Results

¿En qué escenarios funciona mejor / peor?

6 Discusión Crítica

Fortalezas

Debilidades

Supuestos cuestionables

Qué no queda claro

7 Conclusiones

Comentario Personal

¿Es una buena contribución?

¿Lo usaría en mi investigación?