Proximal Policy Optimization (PPO)
Explanation

Salar Mokhtari Laleh

salarmokhtari0@gmail.com

I. Introduction

Reinforcement learning (RL) has gained significant attention due to its ability to solve complex sequential decision-making problems. Among various RL algorithms, Proximal Policy Optimization (PPO), introduced by Schulman et al. [**?**], stands out for its simplicity, stability, and efficient training. PPO improves upon earlier policy gradient methods by introducing a clipped surrogate objective, which prevents large policy updates and ensures training stability.

PPO is an on-policy, model-free, policy gradient algorithm that provides a balance between the stability of off-policy methods like Q-learning and the sample efficiency of on-policy methods. PPO is widely used due to its simplicity and computational efficiency, making it one of the most popular algorithms in reinforcement learning.

II. Background and Related Work

A. Policy Gradient Methods

Policy gradient methods aim to optimize a policy directly by maximizing the expected return. The basic policy gradient objective is formulated as:

$$L^{\text{PG}}(\theta) = \mathbb{E}_t \left[\log \pi_\theta(a_t|s_t) \cdot \hat{A}_t \right], \tag{1}$$

where $\pi_\theta(a_t|s_t)$ is the probability of taking action a_t in state s_t under the policy π_θ, and $\hat{A}_t$ is the advantage function, which estimates the relative benefit of taking action a_t at state s_t compared to the average action in that state.

While policy gradient methods are effective, they can be prone to instability due to large updates in the policy parameters.

B. Trust Region Policy Optimization (TRPO)

Trust Region Policy Optimization (TRPO) [**?**] is an algorithm designed to improve stability by constraining the policy update to a trust region, preventing large deviations between the new and old policies. The TRPO objective is:

$$L^{\text{TRPO}}(\theta) = \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \cdot \hat{A}_t \right], \tag{2}$$

subject to the constraint:

$$\mathbb{E}_t \left[\text{KL} \left(\pi_{\theta_{\text{old}}}, \pi_\theta \right) \right] \leq \delta. \tag{3}$$

While TRPO effectively stabilizes training, it requires second-order optimization and is computationally expensive, making it impractical for large-scale problems.

III. Proximal Policy Optimization (PPO)

PPO improves upon TRPO by introducing a simpler approach to prevent large policy updates through a clipped objective. The core idea of PPO is to limit the change in the policy at each update, ensuring that the new policy does not deviate too far from the old one, avoiding instability.

The PPO objective function is a clipped surrogate of the policy gradient, which is defined as:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) \cdot \hat{A}_t, \, \text{clip} \left(r_t(\theta), 1-\epsilon, 1+\epsilon \right) \cdot \hat{A}_t \right) \right], \tag{4}$$

where $r_t(\theta)$ is the probability ratio:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}. \tag{5}$$

The clipping mechanism ensures that the ratio stays within a specified range, typically between 1 - ϵ and 1 + ϵ, to prevent excessively large updates.

This clipping helps prevent the policy from making large, unstable updates while maintaining the advantages of policy gradient methods.

A. Generalized Advantage Estimation (GAE)

To reduce the variance in the policy gradient estimate, PPO uses Generalized Advantage Estimation (GAE) [**?**]. GAE introduces a trade-off between bias and variance by smoothing the advantage estimates. The advantage $\hat{A}_t$ is computed as:

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}, \tag{6}$$

where γ is the discount factor and λ is a hyperparameter controlling the trade-off. Smaller values of λ result in higher bias and lower variance, while larger values reduce bias but increase variance.

GAE allows PPO to improve sample efficiency by providing more stable estimates of the value function, making it particularly useful for environments with high variance in the rewards.

IV. Policy Update Algorithm

The PPO algorithm follows a two-step process for policy optimization:

1) **Data Collection**: Collect experience by running the current policy π_θ in the environment. This experience is stored in a replay buffer.
2) **Policy Optimization**: Maximize the clipped objective $L^{\text{CLIP}}(\theta)$ using minibatch stochastic gradient descent (SGD) over several epochs. The value function is also updated to minimize the value loss.

The main advantage of PPO over TRPO is that PPO does not require second-order optimization, which makes it easier to implement and computationally cheaper.

V. Pseudocode for PPO

The following pseudocode summarizes the key steps of the PPO algorithm:

Algorithm 1 Proximal Policy Optimization (PPO)

```
Initialize policy parameters θ and value function parameters φ
for each iteration do
    for actor=1 to N do
        Collect data by running policy π_θ for T timesteps
        Compute advantages Â_t using GAE
    end for
    for epoch=1 to K do
        Sample a minibatch of data from collected experience
        Update policy by maximizing the clipped objective L^CLIP(θ)
        Update value function by minimizing the value loss
    end for
    Update old policy: θ_old ← θ
end for
```

VI. Experimental Results

PPO has been evaluated in various environments, including continuous control tasks such as HalfCheetah, Walker2D, and Ant, as well as in Atari games. In these evaluations, PPO has shown superior performance compared to other policy gradient algorithms, including TRPO and A3C, in terms of both stability and sample efficiency.

In continuous control tasks, PPO has been shown to converge faster than TRPO while maintaining high stability. In environments with sparse rewards, PPO also outperforms other algorithms by efficiently balancing exploration and exploitation. Moreover, PPO is particularly well-suited for tasks with large action spaces, where algorithms like Q-learning are computationally expensive.

VII. Results Comparison

Table **??** shows a comparison between PPO, TRPO, and A3C in terms of sample efficiency and final performance in various environments. PPO consistently performs well across different benchmarks, with better sample efficiency than TRPO and superior stability compared to A3C.

TABLE I
Performance Comparison of PPO, TRPO, and A3C

Algorithm	HalfCheetah	Walker2D	Ant
PPO	**2000**	**2200**	**2500**
TRPO	1900	2100	2400
A3C	1700	1900	2200

VIII. Conclusion

Proximal Policy Optimization (PPO) provides a robust, efficient, and easy-to-implement solution for reinforcement learning. By introducing the clipped objective function, PPO strikes an optimal balance between the stability of off-policy methods and the efficiency of on-policy methods. The algorithm's simplicity and performance make it a popular choice for various RL applications, from robotics to games.

Acknowledgments

We thank the authors of the original PPO paper and the community for their contributions and insights. Special thanks to the ERC - NTNU for providing computational resources.