Pratical session

Policy Gradient: Proximal policy optimization

This practical session is inspired from the blog post Huang et al. (2022a). This blog emphasizes the implementation details used in official PPO implementation by examining its historical revisions in the openai/baselines GitHub repository (the official repository for PPO).

Background

Proximal Policy Optimization (PPO) is an on-policy reinforcement learning algorithm Schulman et al. (2017b). PPO builds on Trust Region Policy Optimization (TRPO) Schulman et al. (2017a). TRPO improved the stability of Deep Q-Networks (DQN) by limiting how much the policy could change using a trust region constraint based on KL divergence Mnih et al. (2015). However, TRPO required computing the Hessian matrix (which contains second derivatives), making it inefficient for large-scale applications.
Introduced in 2017, PPO simplifies TRPO by removing the need for the Hessian matrix. TRPO enforces a strict KL divergence constraint, while PPO uses a clipped policy gradient update, making it easier to implement while maintaining stable learning.

Key Concepts

PPO introduces a clipped surrogate objective function to prevent large policy updates. The objective function is defined as:

$$L(\theta) = \mathbb{E}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \tag{0.1}$$

where:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the ratio of the probability under the new policy to the old policy.
- $\hat{A}_t$ is the estimated advantage at time step t.
- ϵ is a hyperparameter that controls the clipping range.

Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0, 1, 2, \ldots$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

end for

Figure 1: PPO pseudocode from Achiam (2018).

The pseudocode of PPO is given in figure 1.

In this practical session, we will start from the CleanRL implementation Huang et al. (2022b) for PPO. We will step by step include the implementation tricks presented in `https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/` to handle images as input. Finally, we will extend the code for continuous action space environments.

Start by looking at the implementation in ppo.py which already implements 13 implementations tricks presented in Huang et al. (2022a).

QUESTION 1

Create a copy of ppo.py and name it "ppo_atari.py". Following the blog Huang et al. (2022a), implement the 9 implementation tricks to play Atari games using images as input of your networks. Test your implementation on the BreakoutNoFrameskip-v4 environment `https://ale.farama.org/environments/breakout/`.

QUESTION 2

Create a new copy of ppo.py and name it "ppo_continuous_action.py". Following the blog Huang et al. (2022a), implement the 9 implementation tricks for continuous action domains. Test your implementation on the Inverted Double Pendulum environment `https://gymnasium.farama.org/environments/mujoco/inverted_double_pendulum/`.

BONUS: QUESTION 3

Proximal policy optimization is an algorithm that can be sensitive to hyperparameter tuning. If you want to learn more about it, read Andrychowicz et al. (2020) and Engstrom et al. (2020). Based on these papers, tune your ppo implementations.

REFERENCES

Achiam, J. (2018). Spinning Up in Deep Reinforcement Learning.

Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., Gelly, S., and Bachem, O. (2020). What matters in on-policy reinforcement learning? a large-scale empirical study.

Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., and Madry, A. (2020). Implementation matters in deep policy gradients: A case study on ppo and trpo.

Huang, S., Dossa, R. F. J., Raffin, A., Kanervisto, A., and Wang, W. (2022a). The 37 implementation details of proximal policy optimization. In *ICLR Blog Track*. https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/.

Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., and Araújo, J. G. (2022b). Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2017a). Trust region policy optimization.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017b). Proximal policy optimization algorithms.