WIKIPEDIA
The Free Encyclopedia

Proximal policy optimization

Proximal policy optimization (PPO) is an algorithm in the field of reinforcement learning that trains a computer agent's decision function to accomplish difficult tasks. PPO was developed by John Schulman in 2017,[1] and had become the default reinforcement learning algorithm at American artificial intelligence company OpenAI.[2] In 2018 PPO had received a wide variety of successes, such as controlling a robotic arm, beating professional players at Dota 2, and excelling in Atari games.[3] Many experts called PPO the state of the art because it seems to strike a balance between performance and comprehension. Compared with other algorithms, the three main advantages of PPO are simplicity, stability, and sample efficiency.[4]

PPO is classified as a policy gradient method for training an agent's policy network. The policy network is the function that the agent uses to make decisions. Essentially, to train the right policy network, PPO takes a small policy update (step size), so the agent can reliably reach the optimal solution. A too-big step may direct policy in the false direction, thus having little possibility of recovery; a too-small step lowers overall efficiency. Consequently, PPO implements a clip function that constrains the policy update of an agent from being too large or too small.[4]

Development

Reinforcement learning (RL), to which PPO belongs, has roots in psychology and neuroscience. Compared with other fields of machine learning, Reinforcement learning closely mimics the kind of learning that humans and other animals do. Many of the core algorithms, including PPO, were originally inspired by biological learning systems, like psychologist Edward Thorndike's learning by trial and error (1913).[5][6]

In 2015, John Schulman introduced Trust Region Policy Optimization (TRPO) as an earlier version of PPO. TRPO addressed the instability issue found in the previous algorithm, deep q-network (DQN), by using the trust region constraint to regulate the KL divergence between the old and new policy. However, TRPO is computationally complicated and inefficient due to its second-order optimization, leading to expensive and difficult implementation for large-scale problems.[7][8]

In 2017, John Schulman solved the complexity issue of TRPO by adopting first-order optimization in PPO. Schulman and his teams designed a clipping mechanism that forbids the new policy from deviating significantly from the old one when the likelihood ratio between them is out of clipping range.[1][8] In other words, PPO modifies TRPO's objective function with a punishment of too-large policy updates. Also, PPO deletes the complicated trust region constraints and utilizes the clipping function instead. As a result, PPO improves performance and implementation based on the framework of TRPO.

Theory

This section will first explore key components of the core algorithm in PPO, and then delve deep into the Main objective function in PPO.

Basic Concepts

To begin the PPO's training process, the agent is placed in an environment to perform actions based on its current input. In the early phase of training, the agent can freely explore solutions and keep track of the result. Later, with a certain amount of datasets and policy updates, the agent will select an action to take by randomly sampling from the probability distribution P(A|S) generated by the policy network.[9] The actions that are most likely to be beneficial will have the highest probability of being selected from the random sample. After an agent arrives at a different scenario known as a State by acting, it is rewarded with a positive reward or a negative reward. The objective of an agent is to maximize its total rewards across a series of States, also referred as episodes. Scientists reinforce the agent to learn to perform the best actions by experience, and this decision function is called Policy.[10]

Policy Gradient Laws: Advantage Function A

As PPO's essential part, the advantage function tries to answer the question of whether a specific action of the agent is better than the other possible action in a given state or worse than the other action. By definition, the advantage function is an estimate of the relative value for a selected action. The positive output of the advantage function means that the chosen action is better than the average return, so the possibilities of that specific action will increase, and vice versa.[8]

Advantage function calculation: A = discounted sum (Q) - baseline estimate (V). The first part, the discounted sum, is the total weighted reward for the completion of a current episode. More weight will be given to a specific action that brings easy and quick rewards. On the other hand, less weight will be credited to actions that need significant effort but offer disproportionate rewards.[11][8] Since the advantage function is calculated after the completion of an episode, the program records the outcome of the episode. Therefore, calculating advantage is essentially an unsupervised learning problem. The second part, the baseline estimate, is the value function that outputs the expected discounted sum of an episode starting from the current state. In the PPO algorithm, the baseline estimate will be noisy (with some variances) because it utilizes a neural network. With the two parts computed, the advantage function is calculated by subtracting the baseline estimate from the actual return (discounted sum).[12] $A > 0$ signifies how much better the actual return of the action is based on the expected return from experience; $A < 0$ implies how bad the actual return is based on the expected return.

Ratio Function

In PPO, the ratio function calculates the probability of taking action *a* at state *s* in the current policy network divided by the previous old version of policy.

In this function, $rt(\theta)$ denotes the probability ratio between the current and old policy:

- If $rt(\theta)>1$, the action *a* at state *s* is more likely based on the current policy than the old policy.
- If $rt(\theta)$ is between 0 and 1, the action a at state s is less likely based on the current policy than the old policy.

This ratio function can easily estimate the divergence between old and current policies.[13][4]

PPO's Objective Function

The central objective function of PPO takes the expectation operator (denoted as E) which means that this function will be computed over quantities of trajectories. The expectation operator takes the minimum of two terms:

1, R-theta * Advantage Function: this is the product of the ratio function and the advantage function that was introduced in TRPO, also known as normal policy gradient objective.[14]

2. Clipped (R-theta) * Advantage Function: The policy ratio is first clipped between 1- epsilon and 1 + epsilon; generally, epsilon is defined to be 0.20. Then, multiply the clipped version by the advantage.

The fundamental intuition behind PPO is the same as TRPO: conservatism. Clipping applies to make the "advantage estimate" of the new policy conservative. The reasoning behind conservatism is that if agents make significant changes due to high advantage estimates, the policy update will be large and unstable, and may "fall off the cliff" (little possibility of recovery).[15] There are two common applications of the clipping function. When an action under a new policy happens to be a really good action based on the advantage function, the clipping function limits how much credit can be given to a new policy for up-weighted good actions. On the other hand, when an action under the old policy is judged to be a bad action, the clipping function constrains how much the agent can cut the new policy slack for down-weighted bad actions.[16] Consequently, the clipping mechanism is designed to discourage the incentive of moving beyond the defined range by clipping both directions. The advantage of this method is that it can be optimized directly with gradient descent, as opposed to TRPO's strict KL divergence constraint, which makes the implementation faster and cleaner.

After computing the clipped surrogate objective function, the program has two probability ratios: one non-clipped and one clipped; then, by taking the minimum of the two objectives, the final objective becomes a lower bound (pessimistic bound) of what an agent knows is possible.[16] In other words, the minimum method makes sure that the agent is doing the safest possible update.

Advantages

Simplicity

PPO approximates what TRPO did without doing too much computation. It uses first-order optimization (clip function) to constrain the policy update, while TRPO uses KL divergence constraints outside the objective function (second-order optimization). Compared with the TRPO, the PPO method is relatively easy to implement and takes less computation time. Therefore, it is cheaper and more efficient to use PPO in large-scale problems.[17]

Stability

While other reinforcement learning algorithms require hyperparameter tuning, PPO does not necessarily require hyperparameter tuning (0.2 for epsilon can be used in most cases).[18] Also, PPO does not require sophisticated optimization techniques. It can be easily practiced with standard deep learning frameworks and generalized to a broad range of tasks.

Sample efficiency

Sample efficiency indicates whether the algorithms need more or less amount of data to train a good policy. On-policy algorithms, including PPO and TRPO, generally have a low level of sample efficiency.[19] However, PPO achieved sample efficiency because of its usage of surrogate objectives. The surrogate objectives enable PPO to avoid the new policy changing too far from the old policy; the clip function regularizes the policy update and reuses training data. Sample efficiency is especially useful for complicated and high-dimensional tasks, where data collection and computation can be costly.[20]

See also

- Reinforcement learning
- Temporal difference learning
- Game theory

References

1. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” arXiv.org, https://arxiv.org/abs/1707.06347 , arXiv:1707.06347 [cs.LG].
2. OpenAI, "*Proximal Policy Optimization" Available at:* https://openai.com/research/openai-baselines-ppo (Nov.1 2023 retrieved).
3. Arxiv Insights. "An introduction to Policy Gradient methods," *YouTube*, Oct 1st, 2018 [Video file]. Available: https://www.youtube.com/watch?v=5P7I-xPq8u8.
4. T. Simonini, “Proximal Policy Optimization (PPO),” Hugging Face – The AI community building the future., https://huggingface.co/blog/deep-rl-ppo .
5. R. Sutton and A. Barto, Reinforcement learning: An introduction, https://beiyulincs.github.io/teach/spring_21/behavior_modeling/reading/rl_reading.pdf (accessed Nov. 6, 2023).
6. C. Mahoney, “Reinforcement Learning: A review of historic, modern, and historic developments ... | towards Data Science,” Medium, Mar. 30, 2022. [Online]. Available: https://towardsdatascience.com/reinforcement-learning-fda8ff535bb6#5554
7. Wang, Y., He, H., Wen, C., & Tan, X. (2019). Truly Proximal Policy Optimization. *ArXiv*. /abs/1903.07940
8. Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2015). Trust Region Policy Optimization. *ArXiv*. /abs/1502.05477
9. “A Beginner’s Guide to deep Reinforcement learning,” *Pathmind*. https://wiki.pathmind.com/deep-reinforcement-learning#reward
10. Q. T. Luu, “Q-learning vs. deep Q-learning vs. Deep Q-Network,” Baeldung on Computer Science, https://www.baeldung.com/cs/q-learning-vs-deep-q-learning-vs-deep-q-network (accessed Nov. 1, 2023).
11. OpenAI, “Part 1: Key concepts in RL¶,” Part 1: Key Concepts in RL - Spinning Up documentation, https://spinningup.openai.com/en/latest/spinningup/rl_intro.html (accessed Nov. 4, 2023).
12. Rohitkumar, “PPO (Proximal Policy Optimization) explained with code examples in Pytorch and tensorflow,” PlainSwipe, https://plainswipe.com/ppo-proximal-policy-optimization-explained-with-code-examples-in-pytorch-and-tensorflow/ (accessed Nov. 5, 2023).
13. W. Heeswijk, “Proximal Policy Optimization (PPO) explained,” Medium, https://towardsdatascience.com/proximal-policy-optimization-ppo-explained-abed1952457b (accessed Nov. 4, 2023).

14. Edan Meyer. "Proximal Policy Optimization Explained," *YouTube*, May 20th, 2021 [Video file]. Available: https://www.youtube.com/watch?v=HrapVFNBN64 (Accessed Nov 4, 2023).
15. C. M. Wild. The Pursuit of (Robotic) Happiness: How TRPO and PPO Stabilize Policy Gradient Methods. 2018. URL: https://towardsdatascience.com/the-pursuit-of-robotic-happiness-how-trpoand-ppo-stabilize-policy-gradient-methods-545784094e3b (visited on 05/11/2023).
16. Zheng, R., Dou, S., Gao, S., Hua, Y., Shen, W., Wang, B.,(2023). Secrets of RLHF in Large Language Models Part I: PPO. *ArXiv*. /abs/2307.04964
17. J. Nocedal and Y. Nesterov., “Natural, trust region and proximal policy optimization,” TransferLab, https://transferlab.ai/blog/trpo-and-ppo/ (accessed Nov. 5, 2023).
18. J. Hui, “RL-reinforcement learning algorithms comparison,” Medium, https://jonathan-hui.medium.com/rl-reinforcement-learning-algorithms-comparison-76df90f180cf (accessed Nov. 4, 2023).
19. Huang, Shengyi, and Dossa, “The 37 implementation details of proximal policy optimization,” The 37 Implementation Details of Proximal Policy Optimization · The ICLR Blog Track, https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/ (accessed Nov. 5, 2023).
20. XiaoYang-ElegantRL, “ElegantRL: Mastering PPO Algorithms - towards Data Science,” *Medium*, Nov. 23, 2022. [Online]. Available: https://towardsdatascience.com/elegantrl-mastering-the-ppo-algorithm-part-i-9f36bc47b791

External links

- Announcement of Proximal Policy Optimization by OpenAI (https://openai.com/blog/openai-baselines-ppo/)
- GitHub repo (https://github.com/openai/baselines/tree/master/baselines/)

-