Proximal Policy Optimization with Future rewards

Chengcheng Yu¹, Lijun Zhang², Dawei Yin³, Dezhong Peng⁴, Haixiao Huang*¹,⁴
¹,⁴College of Computer Science, Sichuan University, Chengdu, 610065, China
²Chengdu Ruibeiy Yingte Information Technology Limited Company, Chengdu, China
³Sichuan Zhiqian Technology Co., Ltd, Chengdu, China
* Sichuan Provincial Commission of Politics and Law, Chengdu, China
¹ First author’s e-mail: 156025065@qq.com
*Corresponding author’s e-mail: hhx.97@163.com

Abstract. Among the current reinforcement learning algorithms, the Policy Gradient algorithm (PG)[7] is one of the traditional and most widely used algorithms, but it has the disadvantage of unstable gradient estimation, and the newly Proximal Policy Optimization algorithm (PPO) [8] solves the problem. It solves the stability problem, but the update policy is slow, and it is easy to produce over-fitting when the training times are too many. In this article, a new method is proposed, referring to Asynchronous Advantage Actor-Critic (A3C)[9], the basic PPO algorithm is trained in parallel, and a method that considers future rewards is introduced, and the future rewards are also calculated to the current. In the reward, through the OPEN GYM platform and the experimental results of capturing at any position of the robotic arm, our actions can ensure a faster training speed, while also avoiding overfitting during long-term training.

1. Introduction
In recent years, reinforcement learning has developed rapidly, and many new algorithms have been proposed, including DQN[2], DDPG and other algorithms used in discrete space, and PG, A3C and other algorithms used in continuous space. Earlier reinforcement learning algorithms did not perform well in high-dimensional action environments. As Google combined deep learning and reinforcement learning, they brought a new deep reinforcement learning. Experimental results show that the deep learning and reinforcement learning algorithm combination, as DQN, in many experiments have achieved good results, and these results can even surpass human players. And on the basis of DQN, double-DQN, dueling-DQN and PG[4], AC[5], A3C and other algorithms applied in continuous space are derived. And it has been applied in more fields. However, these algorithms have some shortcomings. They can only be applied to discrete or non-discrete spaces, and policy gradient algorithms like PG are less robust. The Trust Region Policy Optimization algorithm (TRPO) is relatively complex, but it is not compatible with architectures that include noise or parameter sharing[4]. The newly Proximal Policy Optimization algorithm reduces the difference between the new and old strategies by cutting the objective function.

Proximal Policy Optimization algorithm has become the preferred algorithm of OPEN AI, but the Proximal Policy Optimization can only be used for online strategies, because it is based on the policy gradient method like A3C, so you can refer to the parallel calculation method of A3C. Collect training data in a single environment, and then package the data for offline training of the model, thereby transforming the online policy into an offline policy. This method can greatly shorten the training time...
and make the model converge to a better effect quickly and they can cut the correlation between experiences. But in model training, due to different batch of data, resulting in the training focus only on the current model of incentives, leading to poor results caused by fitting the case appeared in the latter part of the training. In order to address this situation, this article presents a Proximal Policy Optimization based on future rewards. By re-calculate the rewards of each batch of data, the experience of each round is relevant, so as to eliminate the over-fitting situation in the later stage.

Experimentally, this article first compares with the traditional PPO algorithm, and proves that the multi-threaded PPO with reference to the A3C algorithm can shorten the training time and achieve better results in a shorter period of time. Then, the method based on future rewards proposed in this article is compared with the current method. The comparison of the proposed multi-threaded PPO algorithm proves that the proposed algorithm is better in resisting over-fitting in the later stage, and at the same time, it can achieve the expected result faster.

2. Basic knowledge

2.1. AC algorithm
In order to solve the problem in the continuous action space, the Actor-Critic (AC) framework algorithm was born.\[5\] The AC algorithm has two neural networks, the Actor network, which is used to interact with the current environment. The other Critic network is responsible for scoring the actions and rewards made by the current Actor network and returning them in the form of time difference errors. Because the method based on the value function is combined with the method based on the policy gradient, the AC algorithm solves the problems of the value function being difficult to apply and the training variance of the policy gradient method is large.\[10\] However, the training based on the value function fluctuates greatly, so the A3C training algorithm is proposed. A3C defines an advantage function, which gives the advantage value of the average action compared with the current state.\[11\] Thereby suppressing training fluctuations.

2.2. Policy gradient (PG) \[7\]
The main feature of the policy gradient method is to directly model and optimize the policy.\[7\] The policy is usually as a function $\pi_\theta(a|s)$ parameterized by $\theta$. The reward of the objective function is affected by the current policy. So the definition of the objective function is:

$$J(\theta) = \sum_{s \in S} d^\pi(s)V^\pi(s) = \sum_{s \in S} d^\pi(s)\sum_{a \in A} \pi_\theta(a|s)Q^\pi(s,a)$$  \hspace{1cm} (1)

Algorithm 1: Policy Gradient(PG)

Random initialization

For $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$,  
for $t = 1$ to $T - 1$ do :  
$\theta \leftarrow \theta + \alpha V_\theta \log \pi_\theta(s_t, a_t)v_t$

End for

Policy gradient method is calculated by estimating the amount of the policy gradient and inserted random gradient ascent algorithm work, policy gradient algorithm can learn random policy, design policy objective function, decreased by gradient algorithm parameters are optimized, the final reward. But the key is that the policy gradient algorithm update step, when the step size is not appropriate, update the parameters corresponding to the policy might be a worse policy, when using bad tactics sampling study, again to update the parameters more difference.\[10\]

2.3. Trust Region Policy Optimization (TRPO)\[4\]
In the front section we mentioned policy gradient algorithm disadvantage is the inability to choose an appropriate step, the so-called right step is when the policy update, the return value of the function can not be worse. TRPO is proposed to solve how to find new strategies to make the value of the new return function monotonously increase or monotonously undecrease. It gives a new policy to improve the
monotonous method. TRPO decomposes the corresponding reward function into the reward function corresponding to the old policy + other items. As long as the other items corresponding to the new policy are greater than or equal to zero, the new policy can ensure that the reward function is monotonous.[4]

So there is this equation proposed by Sham Kakade in 2002:

\[ \eta(\pi_{\text{new}}) = \eta(\pi) + E_{s_0 a_0 \ldots s_n} \sum_{t=0}^{\infty} y^t A_{\pi}(s_t, a_t) \]

Which \( \pi_{\text{new}} \) said the new policy, \( \pi \) represents the old policy. And:

\[ A_{\pi}(s, a) = Q_{\pi}(s, a) - V_{\pi}(s) \]

So TRPO:

\[ \max_{\pi_{\text{new}}} E_t \left[ \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)} A_t \right] \]

subject to \( \hat{E}_t KL[\pi_{\text{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)] \leq \delta \)

2.4. Proximal Policy Optimization (PPO)[8]

PPO can be regarded as an approximation of TRPO, but unlike TRPO which uses the second-order Taylor expansion, PPO only uses the first-order approximation, which makes PPO have a better effect on the problem of wider distribution space[8]. In TRPO, The new policy can’t change more from the old policy are we hope, if the same state is input, the probability distribution obtained by the network cannot be too different, so we can use the KL divergence to calculate, which is also proposed The first PPO algorithm:

\[ L_{\text{KLP}}(\theta) = E_t \left[ \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)} A_t - \beta KL[\pi_{\text{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)] \right] \]

The idea of the second PPO algorithm is to update by limiting the step length of each update:

\[ L_{\text{CLIP}}(\theta) = E_t \left[ \min \left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)} A_t, \text{clip} \left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)} A_t, 1 - \epsilon, 1 + \epsilon \right) A_t \right) \right] \]

These two PPO algorithms are based on the AC network, and their training steps are as follows:[8]

| Algorithm 2 PPO, Actor-Critic Style |
|-------------------------------------|
| For iteration = 1, 2, ..., do       |
|   For actor = 1, 2, ..., N do       |
|     Run policy \( \pi_{\text{old}} \) in environment for T timesteps |
|     Compute advantage estimates \( \hat{A}_1, ..., \hat{A}_T \) |
|   End for                           |
|   Optimize surrogate L. wrt \( \theta \), with K epochs and minibatch size M \leq NT |
| End for                             |

In this article we use the second PPO algorithm to optimize.

3. PPO with Future rewards

Although the PPO algorithm has achieved good results in model training compared to the previously introduced algorithm, it is easy to fall into overfitting in the later stage. When the training times are too many, the effect of the model is not ideal. Observation shows that every time we use the reward update, we only consider the current reward, resulting in the subsequent steps being irrelevant to the previous steps. Therefore, this paper proposes a near-end policy optimization algorithm based on future rewards, and introduces the multi-threaded parallel structure of A3C. Each thread explores in several different environments and collects the data of each thread. And recalculate the reward value of each thread.
through the proposed future reward formula. So as to achieve the effect of anti-over-fitting in the later stage without shortening the time.

3.1. Distributed PPO (DPPO)

The DPPO algorithm used in this article can be regarded as a multi-threaded PPO algorithm. It uses a network structure similar to the A3C algorithm. Unlike A3C, the secondary network of DDPO does not have to have the same structure as the primary network. Each secondary network has its own independent environment, the secondary network collects environments in different environments, and then the main network performs parameter updates. First, multiple threads are used to collect data in different environments, and then each worker sends the collected data to Global PPO. Global PPO gets the batch data of multiple workers and updates it, and then pushes the latest policy to each Workers, workers then use the new policy to collect data.

3.2. Future rewards

In the DPPO algorithm, because the data collected in batches by workers only considers the current reward, and then puts into training, these batches of collected data are one-sided when handed over to Global PPO for updating, and it is easy to cause a local optimum after updating a certain number of times. This makes the trained model appear over-fitting, so this paper proposes a new method of calculating reward. This method refers to the idea of Q-learning[1], introduces the concept of future rewards, and calculates the rewards of each step in the future into the current rewards, so that our model has the ability to consider the future. The specific calculation formula is as follows:

$$R_t = R_t + \alpha R_{t+1} + \cdots + \alpha^{k-1} R_{t+k}$$

Where $\alpha$ is the discount factor for future rewards.

Based on the above, the algorithm flow we propose is:

| Algorithm 3 PPO with future rewards |
|-------------------------------------|
| Initialization parameters          |
| repeat                             |
| repeat                             |
| Initialize the environment         |
| Each PPO-worker explores in the environment |
| Collect reward and state           |
| Reach epochs                       |
| Stop                               |
| Push data to global PPO            |
| Recalculate the reward using the future reward formula |
| Training global PPO                |
| Distribute the trained PPO parameters to each worker |
| Until $T > T_{\text{max}}$          |

4. Experiment

4.1. Experimental environment

This experimental environment uses the pendulum game on the OPEN gym platform as shown in Figure 1. The goal of the game is to keep the pole upright. Its highest reward is 0. When the pole is not kept upright, the reward is less than 0, and the game has no termination state. In another experimental environment, we use the pyglot visualization library in python to build a two-dimensional robotic arm environment to carry out the capture experiment of the robotic arm as shown in Figure 2. This environment refers to Morvan Zhou’s 2D robotic arm simulation model and modified on this basis. The robotic arm Take the center as the point, and grab the randomly generated target position.
4.2. Experimental parameters

Experiment 1 parameters are as follows:
Among them, the single-threaded PPO and multi-threaded PPO and the network structure are consistent with the reference. In the proposed PPO algorithm based on future rewards, the learning rate and other parameters are consistent with the multi-threaded PPO parameters, and the discount factor $\alpha=0.9$, each batches = 64, workers = 4.

Experiment two parameters are as follows:
The formula for setting the reward value of the reward function is:

\[ R = R_1 + R_2 \]

\[ R_1 = -\sqrt{(x_x - x_0)^2 + (y_y - y_0)^2} \]

\[ R_2 = \begin{cases} 0, & \text{The end of the robotic arm did not reach the target position} \\ 1, & \text{The end reaches the target position} \end{cases} \]

\[(x_2, y_2)\text{ represents the relative distance between the end of the robot arm[13] and the target position } (x_0, y_0).\]

4.3. Results and analysis

The results are shown in the follow figures:

As shown in Figure 3, as can be seen from the experimental results that in the early stage of training, our proposed method shows better robustness in the later stage of model training than the existing PPO algorithm. At the same time, it can maintain good training in the early stage of training.

As shown in Figure 4, as can be seen from the experimental results of the self-built simulated manipulator that our proposed method exhibits better robustness compared to the original PPO algorithm under the condition of the same training time.
5. Result
In this paper, a near-end policy optimization algorithm based on future rewards is proposed. By comparing with the existing PPO algorithm and A3C algorithm, it is proved that the proposed improvement can indeed speed up the PPO algorithm training speed and can avoid it under certain circumstances. An overfitting situation has occurred. In theory, the traditional PPO algorithm is online training and the reward is only based on the current step, which leads to overfitting in the later stage. The proposed algorithm collects multiple continuous data and associates the data to avoid the overfitting of the model. At the same time, our model also achieved better performance in the newly introduced test environment.

References
[1] Watkins C J C H, Dayan P. Q-learning[J]. Machine learning, 1992, 8(3-4): 279-292.
[2] Hester T, Vecerik M, Pietquin O, et al. Deep q-learning from demonstrations[C]//Thirty-second AAAI conference on artificial intelligence. 2018.
[3] de Lacerda M G P, de Lima Neto F B, Neto H A A, et al. On the Learning Properties of Dueling DDQN in Parameter Control for Evolutionary and Swarm-based Algorithms[C]//2019 IEEE Latin American Conference on Computational Intelligence (LA-CCI). IEEE, 2019: 1-6.
[4] Lapan M. Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more[M]. Packt Publishing Ltd, 2018.
[5] Peters J, Schaal S. Natural actor-critic[J]. Neurocomputing, 2008, 71(7-9): 1180-1190.
[6] Silver D, Lever G, Heess N, et al. Deterministic policy gradient algorithms[C]//International conference on machine learning. PMLR, 2014: 387-395.
[7] Kakade S M. A natural policy gradient[J]. Advances in neural information processing systems, 2001, 14.
[8] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms[J]. arXiv preprint arXiv:1707.06347, 2017.
[9] Babaeizadeh M, Frosio I, Tyree S, et al. Reinforcement learning through asynchronous advantage actor-critic on a gpu[J]. arXiv preprint arXiv:1611.06256, 2016.
[10] Liu Y, Yang J, Chen L, et al. Overview of Reinforcement Learning Based on Value and Policy[C]//2020 Chinese Control And Decision Conference (CCDC). IEEE, 2020: 598-603.
[11] Wu Y, Mansimov E, Grosse R B, et al. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation[J]. Advances in neural information processing systems, 2017, 30: 5279-5288.
[12] Xiao Z, Xie N, Yang G, et al. Fast-PPO: Proximal Policy Optimization with Optimal Baseline Method[C]//2020 IEEE International Conference on Progress in Informatics and Computing (PIC). IEEE, 2020: 22-29.
[13] Hesselroth T, Sarkar K, Van Der Smagt P P, et al. Neural network control of a pneumatic robot arm[J]. IEEE Transactions on Systems, Man, and Cybernetics, 1994, 24(1): 28-38.