Penalty PPO: A New Horizon on PPO Algorithm

Bowen Xu
xubw1@shanghaitech.edu.cn

Jun Zhang
zhangjun2@shanghaitech.edu.cn

Yuxiang Ying
yingyx@shanghaitech.edu.cn

Abstract

In this project, we mainly study and explore the PPO algorithm, and make new innovations in sub-optimization problem setting and network structure. Firstly, we review the Policy Gradient Methods represented by PPO and TRPO, explain how they solve the Model-Free RL problem, and optimize the algorithm through certain constraint functions. After that, we explore some possible improvements of the traditional PPO algorithm through theoretical analysis. Then, based on these analyses, we introduce the penalty function method, Monte Carlo Sampling and other techniques to propose our innovative algorithm: Penalty PPO Algorithm. Finally, we use the Pendulum environment in gym to test our algorithm and achieved good results, which proved the rationality of the innovation in practice.

1 Literature Review

As a policy-based method, PPO algorithm is widely used in solving Model-Free RL problems. Developed from Policy Gradient Methods, PPO algorithm solves the step length calculation problem of Policy Gradient Methods to some extent. Prior to this, the TRPO (Trust Region Policy Optimization) algorithm based on Policy Gradient Methods used the Trust Region method to limit a certain range for the selection of step size, but there are still some problems. Therefore, on the basis of TRPO, PPO algorithm introduces Clip function or Proximal method which is Lagrange dual with Trust Region to solve this problem. The following is a detailed review of the PPO algorithm.

1.1 Background

1.1.1 Policy Gradient Methods

In the Policy Gradient method, we first model the policy π as a function of the parameter θ. Then, we express the policy reward function (objective function) of strategy π as follows :

$$J(\pi_\theta) = \int_S \rho^{\pi_\theta}(s) \int_A \pi_\theta(a|s) r(s,a) da ds$$

where S is the state space, A is an action space and ρ^π is the discounted stationary distribution of states over policy π, $r(s,a)$ is the reward of state action pair (s,a).

However, in Model-Free RL problem, we cannot get the exact distribution of reward function $r(s,a)$. So we take the Monte Carlo sampling to estimate, and use the action value function $Q(s,a)$ to estimate the average reward function under strategy π instead of $r(s,a)$.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Therefore, we update the policy parameter θ by performing stochastic gradient ascent on the direction of policy reward function gradient estimator $\nabla_\theta \hat{J}(\pi_\theta)$:

$$\begin{aligned}\nabla_\theta \hat{J}(\pi_\theta) &= \int_S \rho^{\pi_\theta}(s) \int_A \nabla_\theta \pi_\theta(a|s) Q^{\pi_\theta}(s,a) da ds \\ &= E_{s\sim\rho^{\pi_\theta}, a\sim\pi_\theta}(\nabla_\theta log \pi_\theta(a|s) Q^{\pi_\theta}(s,a))\end{aligned}$$

In most cases, in order to reduce variance of estimation, we will use advantage function $\hat{A}_\pi(s,a)$ to replace the action value function $Q^\pi(s,a)$, where $\hat{A}_\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$.

1.1.2 TRPO (Trust Region Policy Optimization)

Because PPO is an extension to the TRPO, so we first discuss TRPO algorithm. In the process of using the Policy Gradient method to solve the Model-Free RL problem, a problem often faced is the choice of step size. An algorithm is often difficult to converge because the selected step size is too large. Therefore, one of the basic ideas of TRPO is to use the trust region method to limit the choice of step size, so as to prevent the problem of large difference between the new strategy and the old strategy.

First, we replace the policy over the actions by an importance sampling, using θ_{old} to denote the sampling distribution. We then use KL divergence to calculate the gap between the new strategy and the old strategy and limit it to a certain trust region δ. Specifically, in each step, the subproblem of TRPO policy update can be expressed as:

$$\begin{aligned} max_\theta \quad & E_{s\sim\rho_{\theta_{old}}, a\sim\pi_{\theta_{old}}}(\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} \hat{A}_{\pi_\theta}(s,a)) \\ s.t. \quad & E_{s\sim\rho_{\theta_{old}}}(D_{KL}(\pi_{\theta_{old}}(\cdot|s)||\pi_\theta(\cdot|s))) \leq \delta \end{aligned}$$

where we call $L(\theta) = E_{s\sim\rho_{\theta_{old}}, a\sim\pi_{\theta_{old}}}(\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} \hat{A}_{\pi_\theta}(s,a))$ as "surrogate" objective function on policy update.

However, if we apply the commonly used natural gradient method to solve this subproblem, we need to calculate the second-order gradient of $L(\theta)$ and its inverse matrix. The consumption of this calculation is very large, which makes TRPO not efficient in practical applications. Therefore, PPO algorithm is proposed to solve this problem.

1.2 PPO (Proximal Policy Optimization)

As an extension of TRPO, PPO algorithm inherits the data efficiency and reliable performance of TRPO, while using only first-order optimization. This greatly reduces the consumption of the algorithm, so that the algorithm can be widely used in practical scenarios.

1.2.1 Basic Idea

As we know, there is a Lagrange dual relationship between the Proximal Point method and the Trust Region method. Therefore, we can use the Lagrange dual method to transform a constrained Trust Region problem into an unconstrained Proximal Point problem. In solving an unconstrained optimization problem, we can use the commonly used Admm algorithm to achieve first-order optimization rather than second-order in TRPO. The improvement of PPO algorithmfor TRPO is mainly in this aspect.

1.2.2 Algorithm Analysis

There are two versions of the PPO algorithm, one is KL Penalty version and the other is Clipped version. Next, we analyze these two versions respectively.

1. KL Penalty Version

As mentioned above, we use the Lagrange dual method to add the KL divergence constraint in the TRPO suboptimization problem as a penalty term to the objective function and set a penalty

coefficient β. Thus, we generate the following "surrogate" objective function:

$$L^{KLPEN}(\theta) = E_t(\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}\hat{A}_t - \beta D_{KL}(\pi_{\theta_{old}}(\cdot|s)||\pi_\theta(\cdot|s)))$$

where $\hat{A}$ is the estimator of the advantage function at timestep t. Furthermore, We will also update this penalty coefficient appropriately according to whether the step size obtained in each step of policy update reaches a target value d_{targ} we set. If the step size is too large, this penalty factor β is increased in the next update to reduce the step size, and vice versa. Then, we perform the following steps in each policy update:

- Using several epochs of minibatch SGD, optimize the KL-penalized objective:

$$max_\theta \quad L^{KLPEN}(\theta).$$

- Compute stepsize $d_t = \hat{E}_t(D_{KL}(\pi_{\theta_{old}}(\cdot|s)||\pi_\theta(\cdot|s)))$
  - If $d < d_{targ}/1.5, \beta \leftarrow \beta/2$
  - If $d > d_{targ} \times 1.5, \beta \leftarrow \beta \times 2$

The updated β is used for the next policy update, and the parameters 1.5 and 2 above are chosen heuristically, but the algorithm is not very sensitive to them.

By theoretic analysis, the two surrogate objective functions $L^{KLPEN}(\theta)$ can be seen as the lower bound of the true reward function with a parametrized stochastic policy π_θ: $\eta(\theta) = J(\pi_\theta)$. Therefore, if we improve the lower bound function $L^{KLPEN}(\theta)$, then we can improve the policy reward function $\eta(\theta)$, which ensure the correctness of the algorithm.

2. Clipped Version

Unlike KL Penalty version, Clipped version uses the clip function in the surrogate objective function of the algorithm. The definition of the clip function is as follows:

$$clip(r_t(\theta), 1-\epsilon, 1+\epsilon) = \begin{cases} r_t(\theta) & 1-\epsilon \leq r_t(\theta) \leq 1+\epsilon \\ 1-\epsilon & r_t(\theta) < 1-\epsilon \\ 1+\epsilon & r_t(\theta) > 1+\epsilon \end{cases}$$

Then we define the surrogate objective function as below:

$$L^{CLIP}(\theta) = \hat{E}_t(min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon))).$$

where ε is a hyperparameter, $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ represents the probability ratio and $r_t(\theta_{old}) = 1$. Figure 1 shows the different images of the Clip function when the advantage function $\hat{A}_{\pi_\theta}$ is positive and negative. You can see that when $A > 0$, the clip function avoids $r_t(\theta)$ being too large, and when $A < 0$, the clip function avoids $r_t(\theta)$ being too small.

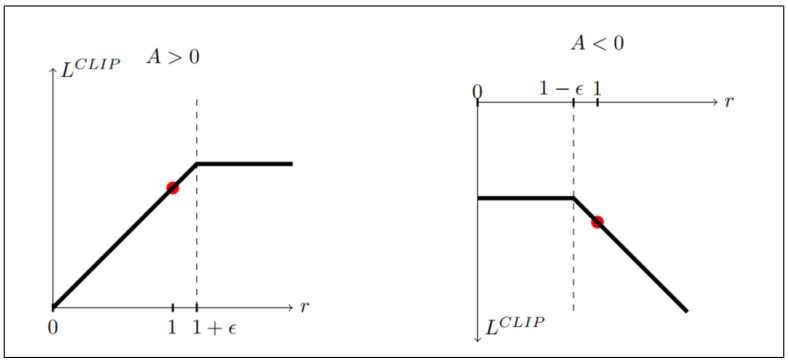

Figure 1: Surrogate objective function by clipped version

This surrogate objective function not only achieves the effect that limits the updated new policy not to differ too much from the old policy, but also avoids the problem that TRPO will lead to second-order gradient.

Then, after modeling each policy update as optimization subproblem of the surrogate objective function mentioned above, we can write the complete PPO algorithm that uses fixed-length trajectory segments as **Algorithm 1**:

Algorithm 1 PPO, Actor-Critic Style

for iteration$= 1, 2, \cdots$ **do**
  for actor$= 1, 2, \cdots, N$ **do**
    Run policy $\pi_{\theta_{old}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \cdots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{old} \leftarrow \theta$
end for

In each iteration process, Critic estimates the state value function through the network during the sampling process, and calculates the advantage function of the action, and feeds the result back to Actor for policy update.

1.2.3 Evaluation and Conclusion

In numerical experiments, the results of three surrogate objective and different hyperparameters are compared:

No clipping or penalty: $L_t(\theta) = r_t(\theta)\hat{A}_t$

Clipping: $L_t(\theta) = min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon))$

KL penalty: $L_t(\theta) = r_t(\theta)\hat{A}_t - \beta KL(\pi_{\theta_{old}}||\pi_{theta})$

From the numerical experiments we made, we can obtain the comparison of the performance of these settings:

- For surrogate objective function: Clipping > Adaptive KL $\approx$ Fixed KL > Without clipping or penalty
- For ε in Clipping, d_{targ} in Adaptive KL penalty, β in Fixed KL: small value may cause stepsize too small and difficult to converge, while large value may make stepsize too large and cross the optimal points.
- For Adaptive and Fixed KL: the difference is small.

2 Problem Setting

Actor-critic is a type of reinforcement learning algorithm that combines the ideas of both value-based and policy-based methods. In actor-critic algorithms, there are two main components: the actor and the critic. The actor uses a policy function π_θ to determine the action to take in a given state. The critic uses a value function V_w to evaluate the quality of the actions taken by the actor. The actor and critic work together to improve the overall performance of the system. The actor uses the feedback from the critic to improve its policy, while the critic uses the actions taken by the actor to improve its value function.

This policy function can be updated by policy gradient. However, performance of policy is very sensitive to step-size selection. When the step-size is not appropriate, the strategy corresponding to the updated parameter is a worse strategy. When the worse strategy is used for sampling and learning, the parameters updated again will be worse, then it is easy to lead to worse learning and finally can not converge. Therefore, appropriate step size is very important for reinforcement learning. In PPO, the actor is updated by a proposed sub-problem, which tends to give monotonic improvement. One common method to update critic is temporal difference (TD) learning. In TD learning, the critic estimates the value of a given state by the value of the next state and the immediate reward. To update the value function, the critic uses samples of states, actions, and rewards to estimate the value of each state. Then the value function is adjusted based on the difference between the estimated value and the actual value.

- Update actor

$$\underset{\theta}{\text{maximize}} \quad \mathbb{E}_t \left[\min \left(r_t(\theta) \mathcal{A}_{\theta_{old}}(t), \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon \right) \mathcal{A}_{\theta_{old}}(t) \right) \right],$$

  where $\mathcal{A}_{\theta_{old}}(t) = \mathcal{Q}_{\theta_{old}}(s_t, a_t) - V_{w_{old}}(s_t)$.

- Update critic

$$\underset{w}{\text{minimize}} \quad \mathbb{E}_{s \sim \rho_{\theta_{old}}} \left[\left(V_{\theta_{old}}(s) - V_w(s) \right)^2 \right].$$

However, in the process of actor update, actual state value $V_{\theta_{old}}(s)$ is estimated by the value function $V_{w_{old}}(s)$, which makes it sensitive to the bias of $V_{w_{old}}(s)$. Another important consideration is that the actor and critic are updated separately, actor update only utilizes w_{old} but not w^*, which is the estimation of state value on policy θ_{old} of the last step, but not the current step. Thus, we can improve the problem by combining these two questions:

$$\begin{aligned} \underset{\theta, w}{\text{maximize}} \quad & \mathbb{E}_t \left[\min \left(r_t(\theta) \mathcal{A}_{\theta_{old}}(t), \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon \right) \mathcal{A}_{\theta_{old}}(t) \right) \right] \\ \text{subject to} \quad & \mathbb{E}_{s \sim \rho_{\theta_{old}}} \left[\left(V_{\theta_{old}}(s) - V_w(s) \right)^2 \right] = 0, \end{aligned} \tag{1}$$

where $\mathcal{A}_{\theta_{old}}(t) = \mathcal{Q}_{\theta_{old}}(s_t, a_t) - V_w(s_t)$.

3 Sample-Based Estimation of the Objective and Constraint

This section describes how the objective and constraint functions can be approximated using Monte Carlo simulation. In this estimation procedure, we collect a sequence of states by sampling $s_0 \sim \rho_0$ and then simulating the policy $\pi_{\theta_{old}}(a|s)$ for some number of timesteps to generate a trajectory $s_0, a_0, s_1, a_1, \ldots, s_{T-1}, a_{T-1}, s_T$. Hence, $\mathcal{Q}_{\theta_{old}}(s_t, a_t)$ is computed at each state-action pair (s_t, a_t) by taking the discounted sum of future rewards along the trajectory. $V_{\theta_{old}}(s)$ is also computed at each state s_t by taking the discounted sum of future rewards along the trajectory.

$$\mathcal{Q}_{\theta_{old}}(s_t, a_t) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}$$

$$V_{\theta_{old}}(s_t) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}.$$

In this way, we estimate $\mathcal{Q}_{\theta_{old}}(s_t, a_t)$ and $V_{\theta_{old}}(s_t)$ with the same value, which leads to the bias on $V_{\theta_{old}}(s_t)$ and transmits the error to V_w at the next step. To overcome this defect, the Q-value critic $\mathcal{Q}_w(s, a)$ can be substituted for value critic $V_w(s)$. $\mathcal{Q}_{\theta_{old}}(s_t, a_t)$ is computed the same, while $V_{\theta_{old}}(s)$ can be computed by $\mathbb{E}_{a \sim q(\cdot|s)} [\mathcal{Q}_w(s, a)]$. We seek to solve the the following optimization problem, obtained by improve the problem in 1:

$$\begin{aligned} \underset{\theta, w}{\text{maximize}} \quad & \mathbb{E}_t \left[\min \left(r_t(\theta) \mathcal{A}_{\theta_{old}}(t), \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon \right) \mathcal{A}_{\theta_{old}}(t) \right) \right] \\ \text{subject to} \quad & \mathbb{E}_{s \sim \rho_{\theta_{old}}, a \sim q(\cdot|s)} \left[\left(\mathcal{Q}_{\theta_{old}}(s_t, a_t) - \mathcal{Q}_w(s_t, a_t) \right)^2 \right] = 0, \end{aligned} \tag{2}$$

where $\mathcal{A}_{\theta_{old}}(t) = \mathcal{Q}_{\theta_{old}}(s_t, a_t) - \mathbb{E}_{a_t \sim q(\cdot|s_t)} [\mathcal{Q}_w(s_t, a_t)]$.

4 Algorithm Design

4.1 Introduction: Main Problem

Next, we will explain the design details of our algorithm and the corresponding principles.

First of all, based on the design of PPO algorithm, our algorithm also adopts the update method of policy iteration, that is, in each iteration step, the policy and network are updated by solving the optimization subproblem. Through the above analysis of the problem, we can write the optimization subproblem in each iteration as Equations **(2)** above,

However, we can find that this optimization subproblem is a constrained optimization subproblem. But solving such a sub-problem also has many shortcomings:

- Firstly, generally constrained optimization problems are more difficult to solve, while unconstrained optimization problems are relatively easier to solved;
- moreover, it is also more important that if the loss of the objective function and the loss of the constraint function are too different in value, it may bring a large error to the overall solution. For example, in the environment we set, the Critic loss (the loss of the constraint function) will be much larger than the Actor loss (the loss of the objective function), which may cause the Actor 's policy update to become very slow.

Therefore, we need to strike a balance between the objective function and the constraint function in the sub-optimization problem. We determine to use Penalty Function Methods to solve this problem.

4.2 Idea: Penalty Function Methods

First, we make a brief introduction to the basic idea of penalty function method:

For a general constrained optimization problem in the following form :

$$\begin{aligned} max_x \quad & f(x) \\ s.t. \quad & c(x) = 0 \end{aligned}$$

We can reformulate the original problem into the following unconstrained optimization problem by introducing the penalty parameter λ :

$$max_x \quad f(x) - \lambda ||c(x)||_2^2$$

Then, by adjusting the value of the penalty parameter λ, we can achieve the purpose of balancing the loss of the objective function $f(x)$ and the loss of $c(x)$ In the overall optimization problem. If we need to optimize the objective function more, then we can increase the penalty parameter λ ; if we need to satisfy the constraint function more, we need to reduce the penalty parameter λ.

4.3 Optimization Subproblem

Through the above analysis, we can use the penalty function method to transform the above constrained optimization subproblem into the following unconstrained optimization subproblem :

$$L^{CLIP}(\theta, w) - \lambda \mathbb{E}_{s\sim\rho_{\theta_{old}}, a\sim q}\left[(\mathcal{Q}_{\theta_{old}}(s,a) - \mathcal{Q}_w(s,a))^2\right] \tag{3}$$

where $L^{CLIP}(\theta, w) = \mathbb{E}_t\left[\min(r_t(\theta)\mathcal{A}_{\theta_{\text{old}}}(t), \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon), \mathcal{A}_{\theta_{\text{old}}}(t))\right]$,

$\mathcal{A}_{\theta_{old}}(t) = \mathcal{Q}_{\theta_{old}}(s_t, a_t) - \mathbb{E}_{a\sim q}[\mathcal{Q}_w(s_t, a_t)]$ and λ is the penalty parameter.

In each iteration update, by solving the optimal solution of such an unconstrained sub-problem, we can update the policy parameter θ and the action value estimation parameter w, and propagate backward through the network to achieve the effect of updating the network parameters.

If we find that the Critic loss in the optimization problem is much larger than the Actor loss, we can increase the penalty parameter λ and increase the weight of the Actor loss in the overall optimization problem to increase the update amount of the action parameter θ to prevent the action parameter from changing too little. Otherwise, the penalty parameter λ should be reduced.

4.4 Algorithm

Similar to the PPO algorithm, we will use the Actor-Critic update mode, that is, in each update, there will be the following process :

- we use the old strategy $\pi_{\theta_{old}}$ for N times of Monte Carlo sampling to estimate the state value function $\hat{V}(s)$, and combine the action value function under the old strategy $\pi_{\theta_{old}}$ to obtain the corresponding advantage function $A_{\theta_{old}}(t)$.
- Solve the optimization subproblem obtained above, update the policy function θ and the value estimation parameter w, and update the network parameters using backward propagation.
- The penalty parameter λ is updated appropriately according to the size relationship between Critic loss and Actor loss.

Based on the above ideas, we get the final pseudocode, as **Algorithm 2: Penalty PPO**:

Algorithm 2 Penalty PPO

for iteration $= 1, 2, \ldots$ **do**
  for iteration $= 1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute state value estimates $\hat{V}(s_1), \ldots, \hat{V}(s_{\text{T}})$
    Compute advantage estimates $\hat{\mathcal{A}}_1, \ldots, \hat{\mathcal{A}}_{\text{T}}$
  end for
  Optimize objective function wrt θ, w, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
  $w_{\text{old}} \leftarrow w$
  Update penalty parameter λ by some strategy.
end for

4.5 Innovations and Comparison with PPO

Since our Penalty PPO algorithm is an innovation based on the PPO algorithm, we will compare our algorithm with the PPO algorithm :

- Firstly, we use the Penalty Function Method to combine the update of Actor with the update of Critic, and use the penalty parameter λ to adjust the update weight of the two. This can prevent one of the losses from being too large and causing another update to be too small, resulting in a slow global update.
- Secondly, our Critic network no longer evaluates the action performance of Actor indirectly by estimating the state value function V like PPO, but directly by estimating the action value function Q of each action. Through Monte Carlo sampling (importance sampling), the state value function V of the next state is obtained, and finally the important part of the optimization subproblem: advantage function A_t is obtained. The advantage of this kind of Critic network is that we can directly estimate the action value function Q that best reflects the performance of the action through the network, rather than indirect estimation through the state value function V of the next state. This innovation will greatly improve the accuracy of Critic's evaluation of Actor which will be reflected in our experiments.

5 Experiment

In this section, we present an experimental result to explore the capability of our new PPO algorithm in detail. We implement our algorithm on a problem having continuous action space.

5.1 Experiment Setting

The problem we are experimenting on is the inverted pendulum swingup problem. It is based on the classic problem in control theory. The system consists of a pendulum attached at one end to a fixed point, and the other end being free. The pendulum starts in a random position and the goal is to apply torque on the free end to swing it into an upright position, with its center of gravity right above the fixed point.

Table 1: Game State

(a) Action Space

Num	Action	Min	Max
0	Torque	-2.0	2.0

(b) Observation Space

Num	Observation	Min	Max
0	$x = cos(\theta)$	-1.0	1.0
1	$y = sin(\theta)$	-1.0	1.0
2	Angular Velocity	-8.0	8.0

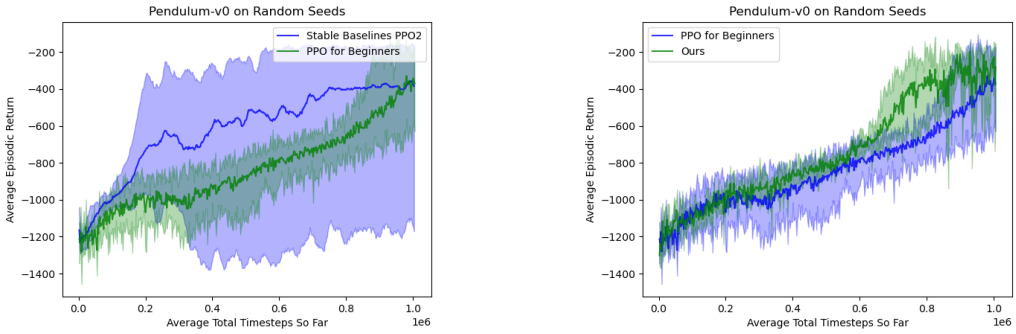

Figure 2: The graph on the left shows the performance of our baseline PPO for Beginners and the state-of-the-art PPO algorithm. The graph on the right shows the performance of our algorithm comparing to the code we made modification on, where green line is the data of our algorithm. The thick lines are the averages over all the seeds and highlighted regions represent the variance.

The action space in this problem is in Table1a, and the observation in this problem is in Table1b.

The reward function is defined as: $r = -(\theta^2 + 0.1 \times \theta_{\text{dt}}^2 + 0.001 \times \text{torque}^2)$, where θ is the pendulum's angle normalized between $[-\pi, \pi]$, θ_{dt} is the angular velocity, whose range is $[-8.0, 8.0]$. Hence, the minimum reward that can be obtained is about -16.2736044, while the maximum reward is zero (pendulum is upright with zero velocity and no torque applied). Episode truncates at 200 time steps.

5.2 Result

To establish a baseline for pendulum performance, we use PPO for Beginners to learn the pendulum problem. Figure 2 illustrates the progress of our algorithm. As noted earlier, our new designed Critic network greatly improve the accuracy of Critic's evaluation of Actor, hence having a steady and fast convergence rate. We can see that, it reaches a larger best rewards, even than the state-of-the-art PPO algorithm. Also, our algorithm converges to a sub-optimal level rewards of -400 per episode about 200000 steps earlier than the original algorithm.

6 Conclusion

In this project, we gain a deeper understanding of the Proximal Policy Optimization algorithm. First of all, we have a detailed review of the PPO algorithm, including the principle and structure of the algorithm. After that, we theoretically analyze some problems existing in the traditional PPO algorithm, and propose several improvements, including: (1) Using the network to estimate Q value, and using Monte Carlo tree sampling to estimate V value; (2) Penalty function method is used to improve the sub-optimization problem. Based on these two innovations, we propose our Penalty PPO algorithm and achieve good results in simulation. However, in the simulation process, we found that the setting of hyperparameters (such as λ) may have a certain impact on the results, but the time is limited, and we have not been able to explore the mechanism. Although we have made some achievements, if we continue our project, there are still some innovations to try: (1) when the magnitude of TD error is below or above a certain threshold, the learning of actor or critic is frozen; (2) Solve a similar Exploitation-Exploration Trade-off problem by adding a penalty to the actor, thereby encouraging the behavior of higher-entropy exploration strategies. In addition, due to the similarity between GAN and our project, we can also learn some optimization techniques from GAN. If the experiments are successful, they will bring some improvement to our algorithm.

References

[1] John Schulman, Filip Wolski, et al. Proximal Policy Optimization Algorithms, 2017.

[2] John Schulman, Sergey Levine, et al. Trust Region Policy Optimization, 2017.

[3] Fan-Ming Luo, Tian Xu, Hang Lai, et al. A Survey on Model-Based Reinforcement Learning, 2022.

[4] Yoshiharu Sato, Model-Free Reinforcement Learning for Financial Portfolios: A Brief Survey, 2019.

[5] S. Boyd and L. Vandenberghe. Convex Optimization. Cambridge university press, 2004.

7 Division of Work

Bowen Xu: code writing, algorithm design, ppo review, data visualization, presentation slides making
Jun Zhang: code writing, problem setting, sample based estimation, presentation slides making
Yuxiang Ying: report writing, presentation slides making and dataset preparing