Solving Video Games with PPG

Team members: Yuwen Li, Sili Guo

Project Description

Deep Reinforcement Learning has achieved outstanding results in recent years. It combines Reinforcement Learning and Deep Learning. By doing this, DRL algorithms can take inputs with large sizes, and decide for the proper actions to gain highest rewards under different complex environments, such as gaming, computer vision, natural language processing and so on.

In our project, we focus on solving video games with a Deep Reinforcement Learning method called Phasic Policy Gradient (PPG) proposed by [2], which uses Proximal Policy Optimization (PPO) proposed by [1] in the policy phase. The first goal of our project is to have a better understanding towards the PPO and PPG algorithms through its reimplementation. Based on this, we will further explore the functionality of its clipping function and try to see if we can introduce new clipping functions into the original PPG algorithm to improve its performance on Procgen environments.

In particular, our project is separated into two phases. The first phase is to reimplement the PPO and PPG algorithm based on the original paper to make sure it shows similar results as the original paper proposed. The second part is to try new clipping functions in our PPO and PPG models. We will tune the hyperparameters, especially for the newly introduced clipping methods, and show the performance of our result compared to the one using original clipping methods.

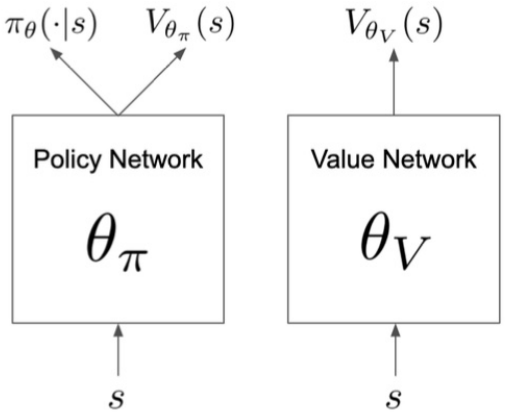
Approach and Solution Diagram

Our approach for solving video games is an algorithm called Phasic Policy Gradient (PPG). PPG is a deep reinforcement learning method that modifies traditional on-policy actor-critic methods by training the policy network and the value-function network separately.

Traditional on-policy actor-critic methods have advantages of policy and value-function networks can use shared features to help improve each other; however it also brings some problems, such as balancing the competing objectives of both networks is hard,

and training them need to base on the exactly same datasets with same conditions. PPG solves these problems by training two networks separately as policy training phase, and introduces a new phase called auxiliary phase to distill the feature trained in value-function network to improve the policy network.

The implementation of the PPG algorithm contains two phases, as mentioned earlier: policy phase and auxiliary phase. The model is shown in the figure below. From the solution diagram below, we can see the disjoint training processes for two networks and the auxiliary value head that is added to the policy training network.



The loss functions for policy networks are:

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

$$L^{value} = \hat{\mathbb{E}}_t \left[\frac{1}{2}(V_{\theta_V}(s_t) - \hat{V}_t^{\text{targ}})^2 \right]$$

The loss functions for auxiliary networks are:

$$L^{joint} = L^{aux} + \beta_{clone} \cdot \hat{\mathbb{E}}_t \left[KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)] \right]$$

$$L^{aux} = \frac{1}{2} \cdot \hat{\mathbb{E}}_t \left[(V_{\theta_\pi}(s_t) - \hat{V}_t^{\text{targ}})^2 \right]$$

In policy phase, the loss function for policy training is represented as L^{clip}, where $r_t(\theta)$is the ratio between new and old policy, and ϵ is the hyperparameter controls the range for

clipping; the loss function for value-function training is L^{value}, where V_t^{targ} is the value function targets. In auxiliary phase, L^{aux} preserves the feature from value function network, then it joins with the policy to calculate L^{joint}. The hyperparameter β_{clone} controls the trade-off of how much of the policy is preserved and how much feature it learns from value function.

The original clipping method used in PPG algorithm is exactly the same as introduced in PPO algorithm:

$$L^{\mathrm{CLIP}}(\pi) = \mathbb{E}\left[\min\left(r_{s,a}(\pi)A_{s,a}, \mathcal{F}\left(r_{s,a}(\pi), \epsilon\right)A_{s,a}\right)\right]$$

$$\mathcal{F}^{\mathrm{PPO}}\left(r_{s,a}(\pi), \epsilon, \alpha\right) = \begin{cases} 1-\epsilon & r_{s,a}(\pi) \leq 1-\epsilon \\ 1+\epsilon & r_{s,a}(\pi) \geq 1+\epsilon \\ r_{s,a}(\pi) & \text{otherwise} \end{cases}$$

In our first approach, we tried a functional clipping method, which obtains a smoothed effect. It is defined as following:

$$\mathcal{F}^{\mathrm{PPOS}}\left(r_{s,a}(\pi), \epsilon, \alpha\right) = \begin{cases} -\alpha tanh(r_{s,a}(\pi) - 1) + 1 + \epsilon + \alpha tanh(\epsilon) & r_{s,a}(\pi) \leq 1-\epsilon \\ -\alpha tanh(r_{s,a}(\pi) - 1) + 1 - \epsilon - \alpha tanh(\epsilon) & r_{s,a}(\pi) \geq 1+\epsilon \\ r_{s,a}(\pi) & \text{otherwise} \end{cases}$$

Another new clipping method we tried in our model is the linearly decaying clipping method in [3], which means the range hyperparameter for the clipping method is decaying linearly due to the increasing of time. The equation is defined as:

$$\epsilon_t^{lin} = \frac{T-t}{T}\epsilon_0$$

Implementation Details

The actual implementation of our project is separated into two phases: the first is PPO and PPG reimplementation, and the second is clipping functions implementation and comparison.

In the first phase, we follow the guideline paper and reference code to reimplement the PPO and PPG algorithm on 2 game environments in Procgen environments, which are CoinRun and BossFight. For PPG, we choose to perform a single-network PPG reimplementation, which detachs the value function gradient at the last layer shared between the policy and value heads, and takes the value function gradient with respect to all parameters including shared parameters during the auxiliary phase.

We compared our results with the results in the original paper to make sure the correctness of our reimplementation. To keep the same environments as defined in the original paper, we use torch == 1.4.0, gym3 == 0.0.3 and procgen == 0.10.4 in our reimplementation.

In the second phase, for each new clipping method, we first try on the PPO models for 350 iterations. This step is to filter out clipping methods that show significantly worse results than the original one, even if they are just applied to PPO as proposed. In another perspective, it is to identify where the failure comes from, if there is any. Next, if a clipping method shows similar or even better results in PPO, we then train the PPG model with this new clipping method for 1000 iterations. To make comparison, the environment used for the second phase is the same as in the first phase.

Notice that the original paper set total timesteps to be 100 millions; due to the computation and time limitations, for each game, we only train PPO models for the first 350 iterations (about 5 to 10 millions timesteps) since we do not need to get final results from these PPO models, and train PPG models for 1000 iterations (about 20 millions timesteps).

We tune hyperparameters of the model during the second phase and find out the hyperparameters that give the best result. In specific, we conducted a patterned hyperparameter search. For example, to tune the hyperparameter α for functional clipping method, we first decide the range [0, 0.3] derived from the corresponding paper, and then near-linearly increase α values for each try.

Experimental Results and Observations

1. PPO and PPG with original clipping

The implementation of our PPO and PPG models is successful. We compared the results with the original paper, and they show similar trends to converge. The detailed

results are included in figures showing the comparison between original clipping and linearly decaying clipping.

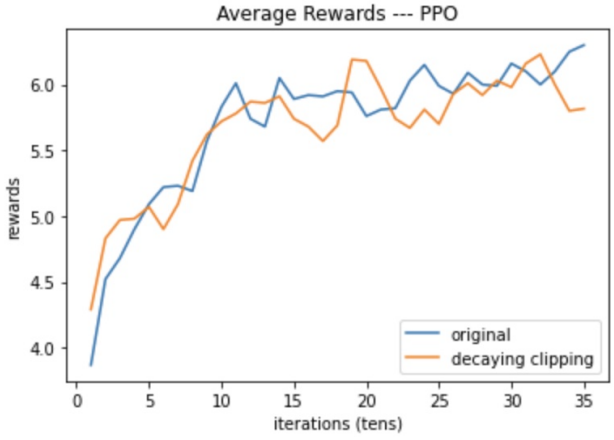
2. PPO with functional clipping

The functional clipping fails in our settlement in the first part of training with the PPO model for 350 iterations. Although we tried different hyperparameters, it always shows a significantly worse performance than the original one. For instance, the rewards it obtains for CoinRun quickly fall and remain in range (2, 3).

We think the possible reason is that video game environments from Procgen are very different from the tasks on which the author experimented, so more adjustment may be needed. Therefore, we conclude the functional clipping method fails in our experiment, and it is not necessary to try PPG with it.

3. PPO with linearly decaying clipping

The linearly decaying clipping method performs well in the first part of training with the PPO model for 350 iterations. Below is the figure showing rewards on CoinRun using PPO and PPO with linearly decaying clipping. From the figure, we can see that PPO with linearly decaying clipping has a similar trend compared to the one with the original clipping method. Thus, we think that introducing a linearly decaying clipping method into PPG is possible to improve its performance.

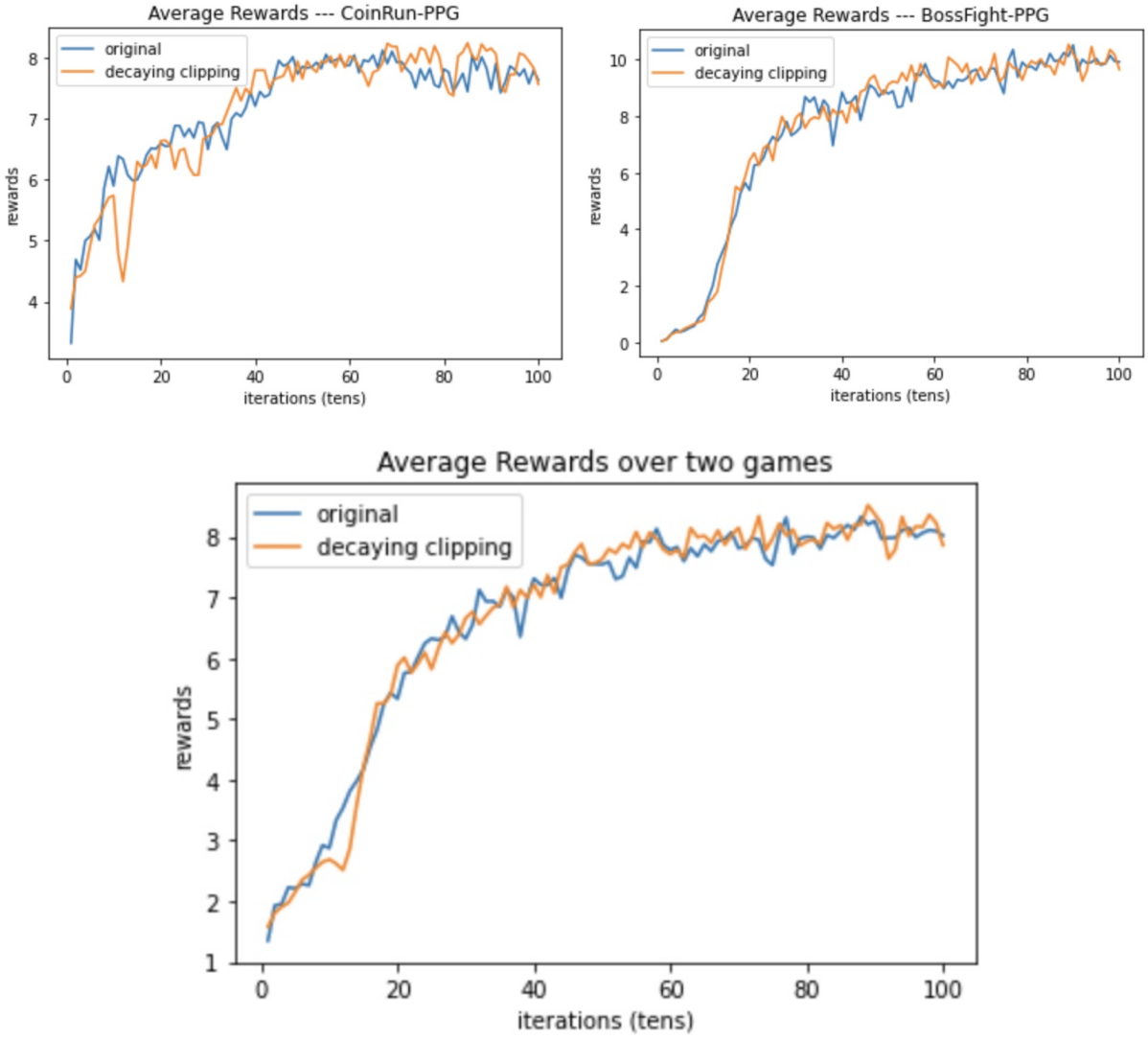


4. PPG with linearly decaying clipping

Based on the experiment of PPO with linearly decaying clipping, we then trained our PPG model with linearly decaying clipping for 1000 iterations on different game environments and observed the results. Because of the limited resource and time, we

only complete the whole 1000 iterations on CoinRun and BossFight, and perform the precise analysis on these two game environments.

We show the trend of the average rewards over every ten iterations for each of these. We also compute the mean rewards under all game environments to show the overall performance of the PPG model with the linearly decaying clipping method. The results are shown below.



As shown in figures, the linearly decaying clipping method can work together with PPG. Furthermore, PPG with linearly decaying clipping starts to get a slightly better performance as the training progresses. We have observed similar results on some other game environments from Procgen.

We then compute the average rewards over the last 500 iterations. For CoinRun, original PPG got 7.7526, and PPG with linearly decaying clipping got 7.8472. For BossFight, original PPG got 9.1181, and PPG with linearly decaying clipping got 9.2727. The average improvement for two games is 0.1246.

This small improvement can be intuitively explained that the more it has learned, the less change it needs to make. However, this improvement is rather subtle, maybe because the maximum number of timesteps we set is too large for our experiments, or the number of iterations we ran is not enough to show much difference.

Conclusion

To conclude, we combine the functional clipping method and the linearly decaying clipping method into PPG, and perform experiments on different game environments from Procgen.

From our experiments, the functional clipping method in our setting fails for solving video games from Procgen, maybe due to the big difference between the applied tasks. However, linearly decaying clipping is compatible with PPG and helps to improve the rewards of PPG for our tasks.

In future work, more experiments can be done to further adjust these different clipping methods to get more improvements.

Bibliography

[1] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov. Proximal Policy Optimization Algorithms. 2017.
[2] Karl Cobbe, Jacob Hilton, Oleg Klimov, John Schulman. Phasic Policy Gradient. 2020.
[3] Monika Farsang, Luca Szegletes. Decaying Clipping Range in Proximal Policy Optimization. 2021.
[4] Wangshu Zhu, Andre Rosendo. Proximal Policy Optimization Smoothed Algorithm. 2020.