Comparative Analysis of Reinforcement Learning Algorithm based on Tennis Environment

Yu Bai [1, †], Haoyu Dong [2, †, *], Qiwei Lian [3, †]

[1] Computer and information institute, Hefei University of Technology (Xuancheng), Xuancheng, China

[2] School of Automation Science and Engineering, Xi'an Jiaotong University, Xi'an, China

[3] Civil engineering institute, Zhejiang University of Technology, Hangzhou, China

* Corresponding author email: 2204214976@stu.xjtu.edu.cn

[†]All authors contributed equally.

Abstract. Reinforcement learning and deep reinforcement learning, as a research hotspot in the field of machine learning, have been widely used in our daily life. In this field, game is playing an extremely important role in the developing of reinforcement learning algorithms. Based on the Tennis environment built by Unity ML Agents, this paper used three algorithms, Proximal Policy Optimization (PPO), Multi-Agent Deep Deterministic Policy Gradients (MADDPG) and Soft Actor-Critic (SAC), combined with PyTorch framework, solved the continuous control problem of this environment. Meanwhile, a group of optimal parameters are obtained through multiple trainings, so that Agents can achieve a perfect effect of solving the continuous control problem in Tennis environment. At the end, this paper compared and analyzed the difference among these three algorithms, summarized the application and properties of each algorithm. For different parameters of the algorithm, this paper also maked a comparison and explained the reasons for some special cases as well which can be used for the future work.

Keywords: PPO; MADDPG; SAC; PyTorch; Tennis Environment.

1. Introduction

PPO is a model-free algorithm which is aiming to enhance the stability and sample efficiency of policy gradient methods [1]. PPO uses a "trust region" optimization method to ensure that the updates to the policy are bounded, which helps to prevent the policy from making large, unexpected changes. PPO has been used in a huge range of RL problems, including control of robotic systems, game playing, and recommendation systems. Its combination of stability and sample efficiency make it a popular choice for many RL tasks [2-3]. MADDPG is an extension of the popular DDPG algorithm to the multi-agent setting. MADDPG is a model-based RL algorithm that uses deep neural networks to learn the optimal policies for multiple agents interacting with each other in a shared environment. MADDPG has been applied to a variety of multi-agent tasks, including cooperative and competitive environments [4]. Its ability to handle complex, multi-agent interactions make it a valuable tool for studying emergent behavior in multi-agent systems. SAC is a model-based algorithm that Inherits the advantages of value - based and policy - based algorithms. SAC uses a soft Q-function and an actor function to learn the optimal policy for a given environment. The soft Q-function estimates the expected return of taking a particular action in an existing state, while the actor function determines the optimal action to take based on the current state [5]. SAC has been shown to outperform other RL algorithms on many kinds of tasks, including control of robotic systems, game playing, and recommendation systems. Its ability to learn robust policies even in the presence of uncertainty makes it a valuable tool for real-world applications.

This paper mainly discusses the effect of PPO, MADDPG and SAC three commonly used reinforcement Learning algorithms on the Machine Learning Agents' tennis environment. At the same time, by adjusting the parameters of different algorithms, the training results of different algorithms are compared, and the situation affected by the parameters is finally drawn. The outcomes can

contribute to people who are ready to choose better algorithms or better parameters to achieve better performance and results when encountering other similar reinforcement learning environments. It also makes people more familiar with the advantages and disadvantages of each of the three algorithms.

2. Method

This part is the about the introduction of the chosen algorithms and the result each algorithm leads to. Training in the same model, the researchers set a goal of 2.5 total scores for each algorithm to reach. And due to the features of different algorithms, they performed variously.

2.1 Proximal Policy Optimization(PPO) Algorithm

PPO algorithm as a new kind of off-policy gradient algorithm, is based on Actor Critic architecture. By clip the importance sampling ratio, which can be limited it to a range [6].

The traditional policy gradient algorithm uses the following formula as the loss of the policy network:

$$\hat{g} = \hat{E}_t\left[\nabla_\theta \log \pi_\theta\left(a_t \mid s_t\right)\widehat{A_t}\right] \tag{1}$$

However, this method is easy to lead to large differences between $\pi_\theta(a_t \mid s_t)$ and $\pi_{\theta_{new}}(a_t \mid s_t)$, which is not conducive to convergence. For this reason, PPO uses clip [7]. The policy network loss value of PPO before the clip is as follows:

$$L^{CPI}(\theta) = \hat{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\widehat{A_t}\right] = \hat{E}_t\left[r_t(\theta)\widehat{A_t}\right] \tag{2}$$

Then it does the following for the above equation:

$$L^{CLIP}(\theta) = \hat{E}_t\left[\min\left(r_t(\theta)\widehat{A_t}, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\widehat{A_{t_t}}\right)\right] \tag{3}$$

That's the core idea of PPO and can prevents excessive model updating, making the algorithm effective and easy to implement and tune parameters. As the default reinforcement learning algorithm for OpenAI to test new environment, this paper used this algorithm for training.

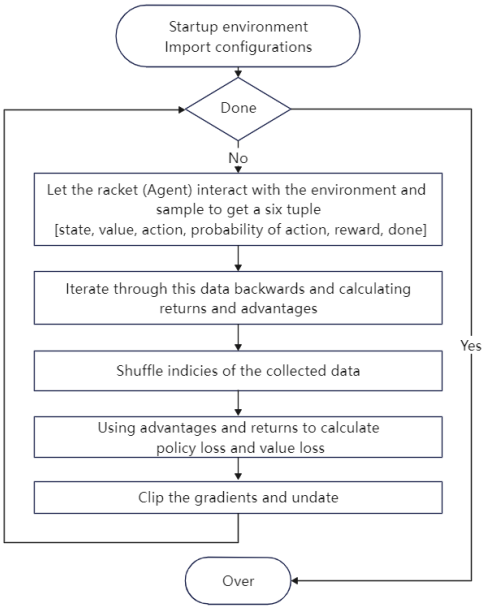

Fig 1. Flow Chart of PPO Algorithm to play Tennis

Because of for each racket (Agent), its observation of the environment and choice of action both are the same, so we can combine the two Agents' networks, just need two networks to train the Agent. One network input the current state and outputs the probability $\pi(a_t \mid s_t)$ of each action, which is called the policy network. A value network inputs the current state and outputs the value of the current state, which can be used to evaluate the value of an action, which is called critic network. In this article, the racket moves up, down, left and right, so use a two-dimensional vector to represent, because of this, that's a multidimensional action problem. That's the reason why we use log standard deviation in our code to output multiple mean and variance to generate multiple distributions to sample. The definition of state has been mentioned in the environmental description, so a high-dimensional vector is enough to represent it. Training with PPO algorithm to play Tennis game can be summarized by the figure 1.

First, load the Tennis environment, read the data and the algorithm configuration. Then let the two rackets (Agent) interact with the environment, get the probability of all actions through the actor network. Meanwhile, sample an action by probability, which will be input into the environment to get s_1 and r_1. After thar, get current status value $v(s_0)$through critic network. When all of these already done, $(s_0, a_0, r_1, v(s_0), \log P(a_0 \mid s_0), done\ or\ not)$ can be collected, that's the data of racket interact with the environment. After iterate through this data backwards we can calculate returns and advantages:

$$A^{\theta}(s_t, a_t) = \sum_{t'>t} \gamma^{t'-t} r_{t'} - V_{\phi}(s_t) \tag{4}$$

The fourth step is to Shuffle indices of the collected data and use advantages and returns to calculate policy loss and value loss. So, we can calculate the loss just like this:

$$L_t^{CLIP+VF+S}(\theta) = \widehat{E}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_{\theta}](s_t)] \tag{5}$$

At the end, we should make sure to Clip the gradients before update the network. Through all these steps, one episode is already done and we need to steadily train the model till the convergence.

2.2 Multi-Agent Deep Deterministic Policy Gradients(MADDPG) Algorithm

People use Q- network to make state-action affect values. It may determine the best action which has the largest estimated state-action value by feeding it current state [7]. To carry it out, it's required to do a series of random actions and collect information. After it has got tons of information, it starts updating the model. But for DQN, it is designed to work in spaces with continuous-action. This is caused by the feature that Q-Learning computes all possible state-actions values and then chooses the highest one. It's impossible to carry out the method of Q iteration in continuous action spaces since continuous policy-making is needed. That's why people adopt Actor-Critic methods. Besides using an actor-critic method, the Deep Deterministic Policy Gradient (DDPG) algorithm additionally uses a replay buffer and target networks that updates Actor and Critic periodically to adopt Deep Q-networks in off-policy learning [8]. Such modifications made the algorithm more stable. Multi-Agent Deep Deterministic Policy Gradients (MADDPG) is based on the theory of DDPG. It's adopted in multi-agent topics. And it has several features. It extends DDPG into a multi-agent policy gradient algorithm. And it centralized the training while decentralizing execution at the same time. It only uses local information to update the policy. what makes it special is that it balances between competition and cooperation of multi-agents.

This algorithm was used to train the tennis model and aim to see its result in training two agents. The action in the tennis model chosen for the research is continuous, and this made MADDPG a reasonable choice among all algorithms developed for reinforcement learning. To use MADDPG algorithm in the tennis model, we update the training in the following method:

Before training process starts, initial state x. And each episode after training begins, for each agent i, select an action with current policy and exploration:

$$a_i = \mu_{\theta_i}(o_i) + N_t \tag{6}$$

Execute $a = (a_1, a_2, ...a_N)$, get the reward r and new state x'.After the execution, store (x, a, r, x') in buffer D, and then overlap x with x'.

Then for agent i = 1 to N, Sample a random minibatch of S samples (x^j, a^j, r^j, x'^j) from D and set y^i:

$$y^i = r_i^j + \gamma Q_i^{\mu'}(x'^j, a'_1, ..., a'_N) \mid a'_k = \mu'_k(o^j_k) \tag{7}$$

With y^i, update critic by minimizing the loss and update actor using sample policy gradient. In this step, calculate loss and sampled policy gradient. This process will form actions of each agent in the next step.

After the updating of Actor&Critic, it's still required to update target network parameters for each agent i:

$$\theta'_i = \tau\theta_i + (1-\tau)\theta'_i \tag{8}$$

In this research, the number of agents is just 2, and this makes the adoption less complex than other MADDPG trainings.

2.3 Soft Actor-Critic(SAC) Algorithm

The problem that SAC algorithm solves is the reinforcement learning problem of discrete action space and continuous action space, namely the reinforcement learning algorithm of off-policy [9]. So, the Machine Learning Agents' tennis environment is very consistent with the characteristics of SAC, and it is feasible to use SAC algorithm training on tennis (Table 1).

Table 1. Soft Actor-Critic Algorithm

Algorithm SAC
Initialize parameter $\boldsymbol{\psi}, \bar{\boldsymbol{\psi}}, \boldsymbol{\theta}, \boldsymbol{\phi}$. **for each iteration do**

for each iteration do
$\mathbf{a_t} \sim \boldsymbol{\pi_\phi}(\mathbf{a_t}|\mathbf{s_t})$
$\mathbf{s_{t+1}} \sim \mathbf{p}(\mathbf{s_{t+1}}|\mathbf{s_t}, \mathbf{a_t})$
$\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s_{t+1}}, \mathbf{a_t}, \mathbf{r}(\mathbf{s_t}, \mathbf{a_t}), \mathbf{s_{t+1}})\}$
end for
for each step do
$\boldsymbol{\psi} \leftarrow \boldsymbol{\psi} - \lambda_V \hat{\nabla}_{\psi} J_V(\boldsymbol{\psi})$
$\boldsymbol{\theta_i} \leftarrow \boldsymbol{\theta_i} - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\boldsymbol{\theta_i})$ **for i** $\in \{1, 2\}$
$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \lambda_V \hat{\nabla}_{\phi} J_\pi(\boldsymbol{\phi})$
$\bar{\boldsymbol{\psi}} \leftarrow \boldsymbol{\tau}\bar{\boldsymbol{\psi}} + (\mathbf{1} - \boldsymbol{\tau})\bar{\boldsymbol{\psi}}$
end for
end for

The structure of SAC includes an actor network and four critic networks, which are respectively state value estimation v and Target v network.

$$\psi \leftarrow \psi - \lambda_V \hat{\nabla}_{\psi} J_V(\psi) \tag{9}$$

$$\bar{\boldsymbol{\psi}} \leftarrow \tau\bar{\psi} + (1-\tau)\bar{\boldsymbol{\psi}} \tag{10}$$

Action-state value estimates for Q0 and Q1 networks.

$$\theta_i \leftarrow \theta_i - \lambda_Q \widehat{\nabla}_{\theta_i} J_Q(\theta_i) \text{ for i} \in \{1,2\} \tag{11}$$

In order to encourage exploration, the concept of entropy is increased in the SAC algorithm, so the training objectives of its actor and critic network are different from those of conventional algorithms without entropy [10].

In the SAC algorithm, the more actions output by the actor network can make a composite index larger, the better. If the action value Q of Q critic network output is more accurate (according to Behrman equation, whether q is accurate depends on whether v is accurate), the better it will be.

If the more accurate the state value V of the V critic network output is, then the better. But it's important to note that because SAC has added the concept of entropy, the state value v is not v (s) as we usually understand it, it has added the entropy term.

3. Result

Through many experiments, this section gives the running results of three algorithms in general, and compares the results of PPO algorithm and SAC algorithm under different parameter settings. At last, through comparative analysis, different phenomena under different parameter settings are summarized and analyzed.

The abscissa represents the episode of training, and the ordinate represents the training result which can be quantified into the scores the agents get in the training. With the parameters, the training can reach the default goal of 2.5 in this research. This guarantees the reliability of the algorithm on this topic. To demonstrate the cooperation between two agents, the figure posts the scores of both agents. In analyzing the feature of this algorithm, we extract two indexes: the total scores of two agents and average scores the agents gained by the increase of episodes.

The figure 3-5 shows the results obtained by using three different algorithms, namely DDPG, PPO, and SAC, to solve this environmental problem. The horizontal axis represents the number of iterations required by the algorithm from the beginning to the convergence, and the vertical axis represents the magnitude of the reward value obtained.

Fig 2. Performance comparison of three algorithms

By observing training time and data efficiency, it can be seen that PPO algorithm is better than DDPG algorithm in terms of training time. This is because PPO as an on-policy algorithm, it can realize offline update strategy through the idea of importance sampling, that is, the data obtained from the behavior strategy can be reused to update the target strategy, which can significantly increase the data utilization and improve the training speed. Therefore, compared with the DDPG algorithm, its effect is undoubtedly better. However, compared with the effect obtained by SAC algorithm, it is slightly inferior. To some extent, this is because PPO algorithm does not play its advantages in supporting parallel training in this paper, and also because each iteration only uses the data of the last episode, which leads to the model cannot fit the space explored more long ago, so the sample efficiency is still relatively low. However, if parallel training is adopted in this environment, although the data efficiency of the algorithm may not change much but it is believed that the training time of PPO algorithm can be greatly reduced.

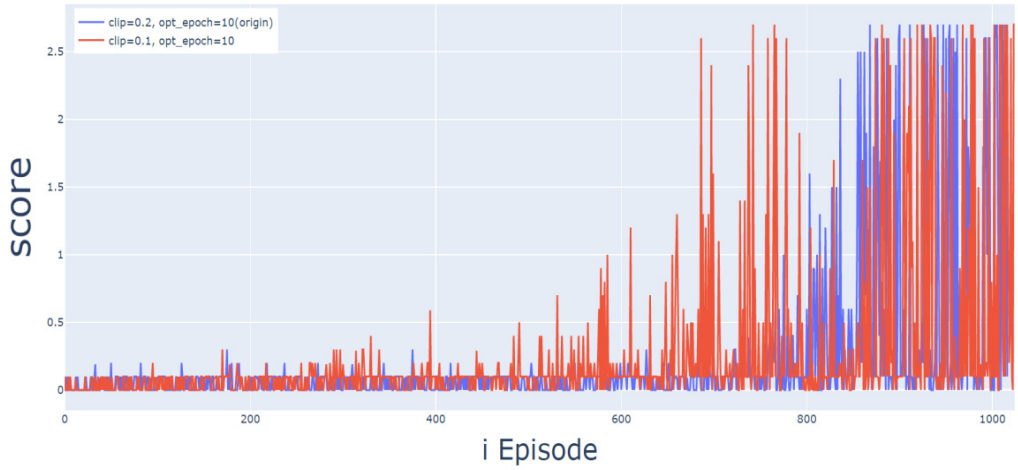

Fig 3. Performance comparison under different clip

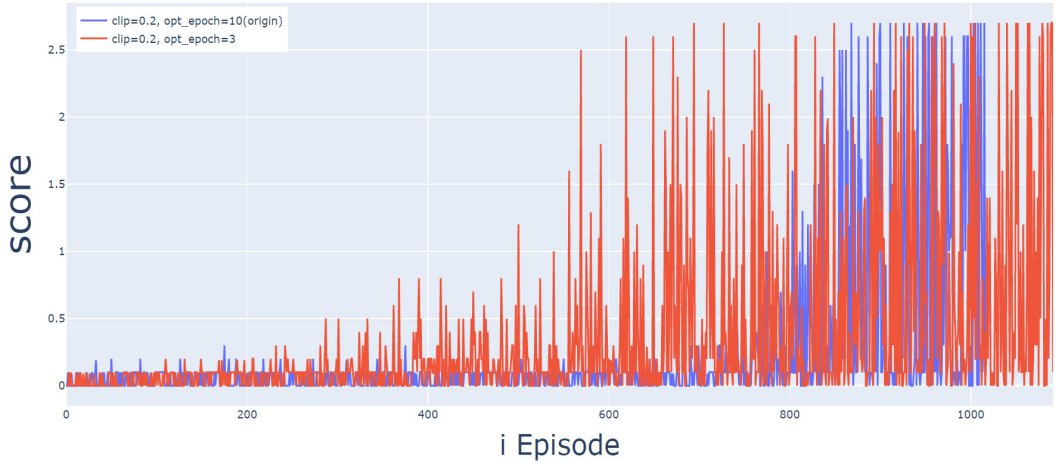

Fig 4. Performance comparison under different opt_epoch

Through the above diagram, with clip = 0.1 and optimization epochs=10, this task is basically completed in the same number of iterations with 1025 iterations. With clip = 0.2 and optimization epochs=3 the result is 1092. It can be seen that by modifying several core hyperparameters of the PPO algorithm it still achieved the same effect in an approximate number of iterations, which shows that the algorithm is robust to hyperparameters, and it also explains why it is selected as the basis algorithm of OpenAI. The core reason for this phenomenon is that the algorithm uses the idea of CLIP to limit the range of updates, which limits the non-convergence of training caused by excessive

updates, so it can obtain more stable algorithm performance. So, when encounter a new environment, PPO is a good choice to have a first try.

4. Conclusion

The research is to adopt reinforcement-learning algorithms in Dual-Agent environment and make a comparison between them. To reach the goal, the researchers chose the tennis model in Unity3D. PPO, SAC and MADDPG were used to train. With all reaching the score of 2.5, comparison on the algorithms can be made. Among the three algorithms, SAC takes the shortest training time of less than 900 episodes as it has higher sample efficiency. PPO takes a little more training time compared with SAC. But as on-policy algorithm, it has high data utilization. Considering the fact that SAC is too dependent on the initial state distributions, it's reasonable to think PPO as a more robust algorithm. In the training, MADDPG takes more iterations to reach 2.5. It focuses more on the asymmetric upgrade of multi-agents. This benefited from its good features in balancing between competition and cooperation. PPO is a good choice for the tennis environment since it is both robust and effective, and it's also easier to switch its parameters and control the training result compared with SAC and MADDPG. But in a more stable environment, SAC may lead to better training result. The research chose 3 algorithms commonly used in reinforcement-learning topics and trained an environment with 2 agents. Further study may take more algorithms into consideration like A3C, C51 and TRPO. Also, to study more features of the 3 algorithms, it's feasible to adopt them in environments of more than 3 agents and see the training result. The increase in agents' amount may change the performance of algorithms, and raise the complexity of research, too.

References

[1] Li, Y. Deep reinforcement learning: An overview. 2017, arXiv preprint arXiv:1701.07274.

[2] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Wierstra, D. Continuous control with deep reinforcement learning. 2015 arXiv preprint arXiv:1509.02971.

[3] Gu, Y., Cheng, Y., Chen, C. P., & Wang, X. Proximal Policy Optimization with Policy Feedback. IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2020, 29:434-447.

[4] Schulman, J., Wolski, F., Dhariwal, P., Radford, A Klimov, O. Proximal policy optimization algorithms. 2017, arXiv preprint arXiv:1707.06347.

[5] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y, Wierstra, D. Continuous control with deep reinforcement learning. 2015, arXiv preprint arXiv:1509.02971.

[6] Wang, Z., Wan, R., Gui, X., Zhou, G. Deep reinforcement learning of cooperative control with four robotic agents by MADDPG. In 2020 International Conference on Computer Engineering and Intelligent Control 2020: 287-290.

[7] Haarnoja, T., Zhou, A., Abbeel, P., Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In International conference on machine learning 2018: 1861-1870.

[8] de Jesus, J. C., Kich, V. A., Kolling, A. H., Grando, R. B., Cuadros, M. A. D. S. L., & Gamarra, D. F. T. Soft actor-critic for navigation of mobile robots. Journal of Intelligent & Robotic Systems, 2021, 102(2), 1-11.

[9] Juliani, A., Berges, V. P., Teng, E., Cohen, A., Harper, J., Elion, C., Lange, D. Unity: A general platform for intelligent agents. 2018, arXiv preprint arXiv:1809.02627.

[10] Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Levine, S. Soft actor-critic algorithms, and applications. 2018 arXiv preprint arXiv:1812.05905.