Project Report

Saiyun Dong

December 16, 2024

1 Introduction

Game playing has long been a fascinating domain for artificial intelligence (AI) research, serving as a benchmark for testing and advancing novel algorithms. From early breakthroughs like Deep Blue defeating a world chess champion to AlphaGo mastering the complex game of Go, the evolution of AI in game playing has showcased the potential of machine learning techniques. Among these, deep reinforcement learning (DRL) has emerged as a powerful approach, enabling agents to learn optimal strategies through interaction with their environments.

Deep reinforcement learning combines the representational power of deep neural networks with the decision-making capabilities of reinforcement learning. This synergy has led to remarkable achievements in diverse gaming contexts, from classic board games to modern video games and real-time strategy simulations. Unlike traditional data-based AI, DRL agents learn by trial and error, refining their policies to maximize long-term rewards without requiring explicit programming of strategies.

The application of DRL in game playing is not only a testament to its algorithmic strength but also a source of valuable insights into decision-making, adaptability, and strategic planning. These qualities have implications far beyond gaming, influencing fields such as robotics, finance, and autonomous systems. As we delve into the intersection of deep reinforcement learning and game playing, it becomes evident that this area of research continues to push the boundaries of what AI can achieve, offering a dynamic platform for innovation and exploration.

In this project, I applied deep Q-learning(DQN) and proximal policy optimization(PPO) methods to the large-scale ARPG game "Black Myth: Wukong" released in August 2024, and compared the advantages and disadvantages of these two methods.

2 Related Work

Previous works have mostly focused on accessing video game APIs to read in-game environmental and action information. For instance, the framework proposed by Wang et al.[1] has been successfully applied in the game Minecraft. Agents can achieve autonomous mining, building, and attacking enemies in the game. However, this approach does not align with how humans play games, and most games do not offer open APIs.

Recently, the emergence of vision language models (VLMs) has further enhanced the visual understanding capabilities of agents. For example, the Cradle framework[2] has been implemented in Red Dead Redemption 2 (RDR2). It directly uses game screenshots from RDR2 as input, rather than using an API to read game memory information. However, Cradle relies heavily on text-based guiding information in the game screenshots to create new skills. For tasks or games with weak textual guidance, such as some action role-playing games(ARPG), Cradle is unable to leverage the effective performance of VLMs[3].

For ARPGs, reinforcement learning methods are widely used, where penalties and rewards are predefined for specific tasks. RL are combined them with computer vision to understand the graphic information of games. In Atari games such as Space Invaders, Brick Strike, etc., DQN[4] has demonstrated performance beyond human players. Some people also apply DQN in ARPG[5], but its performance in complex environments is limited. OpenAI used evolution strategies(ES) to train the five member AI team "OpenAI Five" in Dota 2[6]. Schulman et al.[7] proposed the PPO algorithm, which simplifies policy optimization by introducing

clipping functions and is widely used in reinforcement learning tasks. OpenAI demonstrated the application of PPO in multi-agent competition and sparked the emergence of complex behaviors[8]. AlphaStar partially utilized imitation learning in Starcraft 2[9], extracting strategies from human players' operational data.

3 DQN

Deep Q-learning is an algorithm that combines reinforcement learning and deep learning to solve decision problems in complex environments, especially in high-dimensional state spaces such as video games.

Q-learning is a value function method used to solve Markov decision processes (MDPs). Our goal is to find the optimal strategy $\pi * (s) = argmax_a Q(s, a)$ by learning the state action value function $Q(s, a)$. The Q function is defined as:

$$Q(s, a) = \mathbb{E}[R_t + \gamma \max_{a'} Q(s', a')|s, a]$$

where R_t is current reward, γ is discount factor(used to balance short-term and long-term rewards), s' is the next state, a' is the next action.

However, for high-dimensional state spaces (such as game graphics), it is not possible to directly use tables to store $Q(s, a)$. DQN solves this problem by introducing deep neural networks $Q(s, a; \theta)$ as Q-function approximators, where θ is the network parameter.

DQN use an independent target network $Q(s, a; \theta^-)$ to calculate the target value y, which parameters θ^- are copied from the main network parameter θ every certain number of steps. The goal of DQN is to minimize the time difference (TD) error, that is:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim D}[(y - Q(s, a; \theta))^2]$$

where $y = r + \max_{a'} Q(s', a'|\theta^-)$. DQN also introduce a replay buffer to store past experiences (s, a, r, s') and randomly select samples for training, breaking the temporal correlation of samples and improving training stability.

4 PPO

Proximal Policy Optimization(PPO) is a policy gradient method in deep reinforcement learning. The strategy gradient method directly optimizes the parameter θ of the strategy $\pi_\theta(a|s)$ to maximize the cumulative reward.

In reinforcement learning, we aim to maximize the cumulative expected reward:

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

where r_t means the reward for time step t, γ means discount factor that used to balance short-term and long-term rewards. By using the strategy gradient method, we can take the derivative of $J(\theta)$ and optimize the parameter θ:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta log \pi_\theta(a|s) A(s, a)]$$

where $\nabla_\theta log \pi_\theta(a|s)$ is the strategy gradient representing the rate of change of strategy probability on parameters, and $A(s, a)$ is the advantage function measuring the performance of the current action a relative to the strategy π_θ. To calculate the dominance function, PPO typically uses generalized advantage estimation (GAE):

$$A_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+1}$$

where:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

δ_t is time difference error. $V(s)$ is the value function represents the long-term expected return of state s, which is predicted by a neural network. λ is the hyperparameter that controls the bias and variance.

In reinforcement learning, sampling environment interaction data (states, actions, rewards, etc.) is expensive, especially when the environment is complex or the simulation speed is slow. If every policy update requires the use of the latest policy π_θ to interact with the environment(called on-policy), we will need a large amount of new data, which will greatly reduce data utilization. PPO introduces data from the old strategy $\pi_{\theta_{old}}$ and corrects it through importance sampling, allowing the old data to also be used to optimize the new strategy(called off-policy). This significantly improves data utilization while reducing the frequency of interaction with the environment. PPO use probability ratio $r_t(\theta)$ to measure the difference between the current strategy and the old strategy:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

After adjusting the distribution through importance sampling, the expected value can be calculated under the distribution of the old strategy:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_{\theta_{old}}}[r_t(\theta) \cdot \nabla_\theta log \pi_\theta(a|s) A(s,a)]$$

Due to the fact that the $\pi_{\theta_{old}}$ used for estimation cannot differ significantly from π_θ, PPO further limits the probability ratio to control the magnitude of policy updates. The objective function is defined as two forms:

$$L^{KL}(\theta) = \mathbb{E}_t[r_t(\theta)A_t + \beta D_{KL}(\pi_\theta|\pi_{\theta_{old}})]$$

and

$$L^{CLIP}(\theta) = \mathbb{E}_t[\min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)]$$

The second one is more commonly used. PPO simultaneously optimizes the value function $V(s)$, using mean square error as the loss function:

$$L^{VF}(\theta) = \mathbb{E}_t\left[(V_\theta(s_t) - R_t)^2\right]$$

where $R_t = \sum_{l=0}^{\infty} \gamma^l r_{t+l}$ is the accumulative return. By optimizing the value function, we can estimate the advantage function A_t more accurately. The total loss function of PPO consists of the following three parts:

$$L(\theta) = L^{CLIP}(\theta) - c_1 L^{VF}(\theta) + c_2 H(\pi_\theta)$$

where $H(\pi_\theta)$ is the entropy of strategy is used to encourage exploration.

5 Experiments

5.1 Architecture Description

My DQN framework is provided by [5](Originally used in "SekiroShadows Die Twice"). My PPO framework is provided by [10](Originally used in "Elden Ring"). This framework is based on Stable Baselines3 (SB3) implementation, which is an open-source library based on Python specifically designed for the implementation of RL algorithms. We apply both frameworks in "Black Myth: Wukong".

In the DQN framework, we define our game environment and rewards in the file "env_wukong.py". The network is defined in the file "dqn.py". You can start the project by "train.py".We only use visual features as inputs to the network. Use ResNet-50 as the baseline for extracting visual features. You can replace the baseline with any type of neural network, as long as the real-time performance of the entire network is guaranteed.

In the PPO framework, we define our game environment in "WukongEnv.py" and rewards in "WukongReward.py". The network is defined in the file "train.py". You can start the project by "main.py".We use multi-input policy in PPO, which means inputting visual features as well as the agent state(a vector of Health, Mana, Stamina, ...) as inputs to the network. Use ResNet-50 as the baseline for extracting visual features.

5.2 Results

We ran 50,000 epochs on DQN. Figure 1 shows the trend of the reward function of DQN.

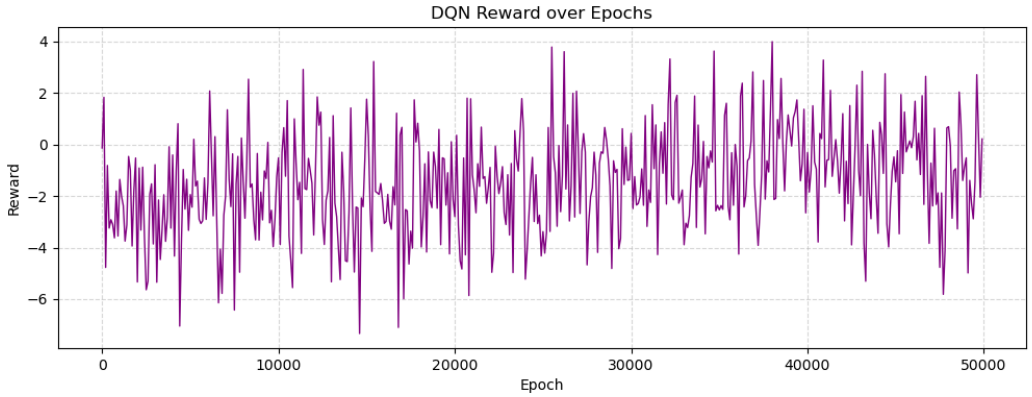

Figure 1: The reward over epochs of DQN.

We ran 50,000 epochs on PPO. Figure 1 shows the trend of the reward function of PPO.

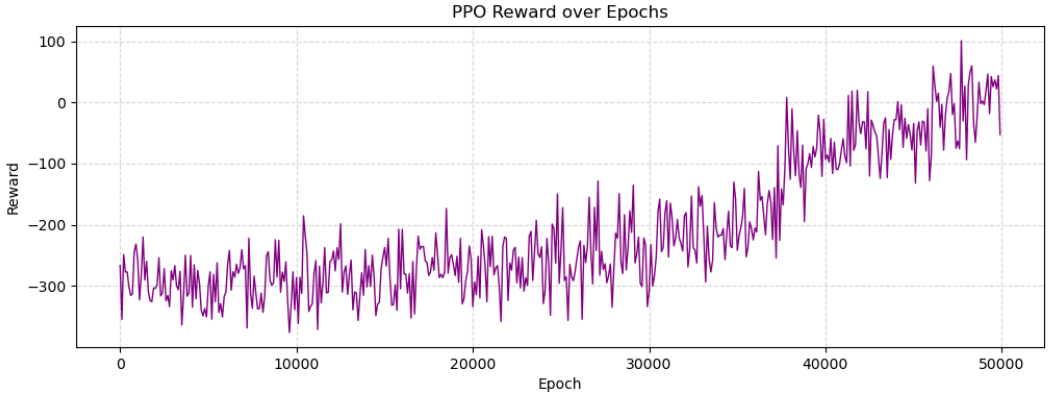

Figure 2: The reward over epochs of PPO.

Based on the comparison between Figure 1 and Figure 2, we can see that the rewards obtained by DQN did not show a significant improvement, but the rewards obtained by PPO overall continued to increase. And PPO has a leap improvement after a certain step. I think this is because the net has learned some key operations of the agent.

As of the number of steps I have trained so far, DQN did not defeat the enemy 'Tiger Vanguard', while PPO emerged victorious.

6 Conclusion and Outlook

6.1 Result Analysis

According to our experimental results, PPO performs better than DQN in large-scale 3D games.

The reason I analyzed is that DQN is a greedy algorithm, while PPO directly learns a probability distribution. We will definitely choose the action with the highest Q value. Therefore, in a complex environments, it is easy to focus only on local optimal solutions. PPO is a strategy gradient method suitable for both continuous and discrete action spaces. Therefore, there is more randomness and exploration. And due to its constraint on the entire trajectory, its prediction of actions is more holistic. It can learn to give up short-term reward and gain more long-term reward.

6.2 Difficulties

During my experiment, I encountered the following difficulties:

1. Due to the various delays, pre-inputs, and animation effects that large games often have, it is difficult to accurately capture the moment when the action ends. For example, after pressing a button, the game requires a certain amount of time to input commands. Other times the game need to play an animation to show the state change. So the information you captured on the screen is likely not the next state. Therefore, you receive a incorrect reward.
2. The game and network must run on the same computer(because of real-timely). Unable to use remote server. Therefore, it will cause a shortage of computing resource.
3. Highly dependent on the setting of reward. Sometimes it is difficult to make reasonable conditional judgments.
4. The replay time is too long, and training the network is very time-consuming and hard-to-debugging. You can't do anything else on your computer during training.

6.3 Future Works

In order to enhance the effectiveness of RL in large-scale 3D games, we will improve these aspects of our work in the future:

1. Use the sound of game as input to the network. (Sound is an important basis for intelligent agents to respond).
2. Use each frame refreshed by the game as the action input time. This can make the particles of the action space finer. Maybe the agents can learn more complex techniques, such as combination skills, cancel skills to dodge, etc.
3. Use a better visual backbone(3D pose estimation, behavior recognition, e.g. OpenPose), while maintaining real-time network operation.

References

[1] Wang G, Xie Y, Jiang Y, et al. Voyager: An open-ended embodied agent with large language models[J]. arXiv preprint arXiv:2305.16291, 2023.

[2] Tan W, Ding Z, Zhang W, et al. Towards general computer control: A multimodal agent for red dead redemption ii as a case study[J]. arXiv preprint arXiv:2403.03186, 2024.

[3] Chen P, Bu P, Song J, et al. Can VLMs Play Action Role-Playing Games? Take Black Myth Wukong as a Study Case[J]. arXiv preprint arXiv:2409.12889, 2024.

[4] Mnih V. Playing atari with deep reinforcement learning[J]. arXiv preprint arXiv:1312.5602, 2013.

[5] https://github.com/analoganddigital/DQN_play_sekiro

[6] Salimans T, Ho J, Chen X, et al. Evolution strategies as a scalable alternative to reinforcement learning[J]. arXiv preprint arXiv:1703.03864, 2017.

[7] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms[J]. arXiv preprint arXiv:1707.06347, 2017.

[8] Bansal T, Pachocki J, Sidor S, et al. Emergent complexity via multi-agent competition[J]. arXiv preprint arXiv:1710.03748, 2017.

[9] Vinyals O, Babuschkin I, Czarnecki W M, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning[J]. nature, 2019, 575(7782): 350-354.

[10] https://github.com/ocram444/EldenRL