# Reinforcement Learning in Atari Game, Pacman

**Shangquan SUN, Jie GONG,**

SJTU-Michigan Joint Institute, Shanghai Jiao Tong University

527112517@sjtu.edu.cn, kevinjiegong@gmail.com

## Abstract

In the project we apply several powerful reinforcement learning algorithms to train an agent of ability to play an Atari game, Pacman. We try PPO (Proximal Policy Optimization) and DQN (Deep Q-network) and investigate the effectiveness of the methods. Specifically, we modify the open-source codes of the algorithms from *OpenAI Baselines* [DHK+17] and run them on the environment of Atari game, "MsPacman-v4" from *OpenAI Gym* [BCP+16]. Our goal is to apply the reinforcement learning algorithms and compare the efficiency of the algorithms.

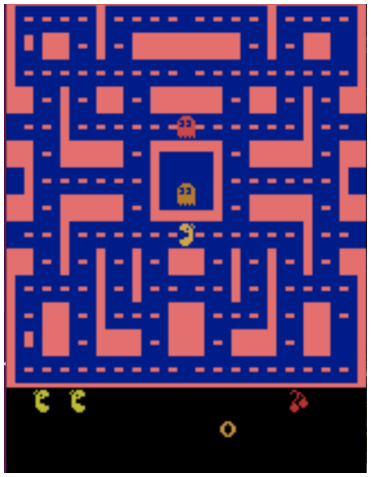

Figure 1: The map of Pacman.

## 1 Introduction

The rule of game is to eat as many points as possible while avoiding being caught by four moving ghosts. In addition, the player can also scare ghosts during a short time after eating a special point. The more points are eaten, the higher score is got. Once all the points in the screen are eaten, the player wins. Conversely, once the player is caught by a ghost, he or she loses. The map is shown in Figure 1.

A player must always keep in mind a concern of avoiding ghosts when achieving a high score, which is not a simple problem that can be solved by supervised or unsupervised learning any more. Therefore, we shall use reinforcement learning to train an agent.

The motivation of the project is that training a human-level control agent by reinforcement learning is very fascinating. The inputs of the algorithms are captured from MsPacman-v4 [BCP+16] with state, action and reward. We denote them $s_t$, $a_t$ and $R_t$. And images including the positions and directions of the player, points and ghosts from Pacman are considered as input of states.

## 2 Related Work

A group of researchers from Stanford University have experimented with Q-learning, approximate Q-learning and Deep Q-learning to train an agent of Pacman. [GFA17] They have concluded that Q-learning works poorly in Pacman, while approximate Q-learning performs better. DQN also works well in this case. Based on their preliminary conclusion, we will directly use DQN to train our agent.

Regarding to the topic of applying DQN to playing Atari games, the most prevalent and relevant work is done by Mnih et al. ([MKS+13], [MKS+15]). We will also refer to the two papers to some extent.

In a work of *OpenAI*, the algorithm of PPO (Proximal Pol-

icy Optimization) [SWD+17] is raised. They also implement the codes of PPO and make them open source [DHK+17]. We will generally do reinforcement learning based on their work.

## 3 Data set and Features

Since we are using reinforcement learning, data sets are directly obtained from the environment, which is MsPacman-v4 from *OpenAI Gym* [BCP+16], during training.

The action space has nine elements, including doing nothing, moving towards left, right, up, down, downright, downleft, upright and upleft. Therefore, the reward returned by the environment will be a $8 \times 1$ vector.

The state space is very large. It is from the image of screen and has 4 dimensions including the information of the locations of ghosts and player.

## 4 Methods

### 4.1 Brief Recap of Reinforcement Learning

In the scope of reinforcement learning, we no longer have a data set with $\{x^{(1)}, \cdots, x^{(n)}\}$ for unsupervised learning or $\{(x^{(1)}, y^{(1)}), \cdots, (x^{(n)}, y^{(n)})\}$ for supervised learning. Instead, we directly train an agent by value update or policy update. In a state $s_{t-1}$ at $t-1$ time, if we take an action $a_t$, then the environment will give a feedback of a new state $s_t$ and reward $R_t$ as Figure 2.

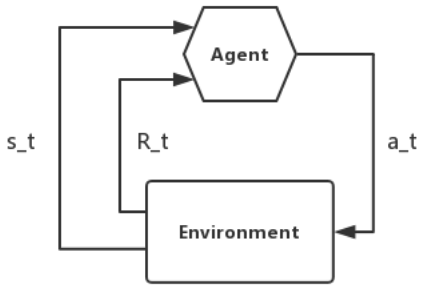


Figure 2: Flowchart of reinforcement learning.

### 4.2 DQN (Deep Q-network)

Firstly, we consider Q-learning, also known as SARSA, whose $Q$-value update is shown in Eq. (1). Suppose $s_t$ and $a_t$ is the state and action at t timestep and $Q(s_t, a_t)$ is the value function with respect to a tuple of state and action

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[ R_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right], \tag{1}$$

where $\alpha$ is a reasonably tuned learning rate and $\gamma$ is a discounting rate. $R_{t+1}$ is the reward function at t+1 timestep.

Then coming back to Deep Q-learning, we are going to estimate the value function $Q(s, a)$ by applying a CNN (Convolutional Neural Network) which can process features implicitly and produce an estimated $Q(s, a)$ for different actions. Also, an experience replay buffer is included. In the implementation of *OpenAI*, the CNN is completely connected for all hidden layers. The structure of the algorithm can be illustrated as Figure 3.

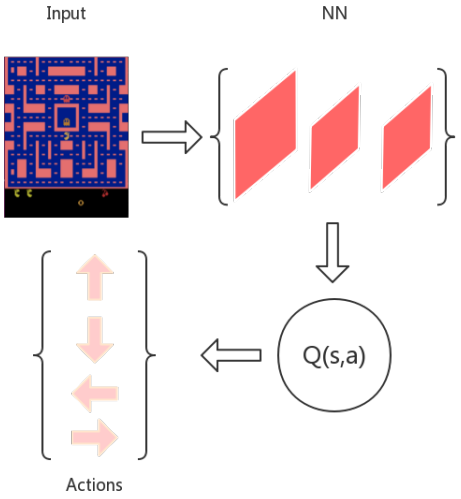


Figure 3: Architecture of DQN with a CNN.

In addition, there are two advanced DQN algorithms, DDQN (Double Deep Q-network) and Dueling-DDQN (Dueling Double Deep Q-network) [WdFL15]. They respectively include two sets of weights and dueling network. According to the experiment, involving a dueling network will enhance the accuracy of estimated value function. Therefore, we directly use Dueling-DDQN in our project.

### 4.3 PPO (Proximal Policy Optimization)

The algorithm of Proximal Policy Optimization is raised by *OpenAI* in 2017 [SWD+17]. Further modified based on TRPO

(Trust Region Policy Optimization) as Eq. (2), the surrogate objective function of PPO is as Eq. (3) as

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[ \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[ r_t(\theta) \hat{A}_t \right], \quad (2)$$

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[ \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right], \quad (3)$$

where $\hat{A}_t$ is an estimator of advantage function at t timestep as shown in Eq. (4), and

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

is the policy probability ratio.

$$\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T) \quad (4)$$

Inside the min, there are two terms. The first term is just the objective of TRPO which is an unclipped term, while the second one is a clipped term. By this way, we could use multiple epochs of small batch SGD (Stochastic Gradient Descent) to optimize the objective. The method of PPO has been proved to be better than TRPO in continuous control and thus in our case, Atari game. Besides, the implementation of PPO is simpler than TRPO and it is more general.

## 5 Results

### 5.1 DQN

We utilize the open source codes of *OpenAI* [DHK+17] and then manage to modify the codes so that they can train on Atari environment, MsPacman-v4, as well as save the model trained after a fixed interval of episodes. Then we have to spend a considerable time and efforts to tune the hyperparameters of DQN. To do the experiments, we make use of a computer engine of *Google Cloud Platform* to train agents.

There are generally five hyperparameters that we can tune in DQN, the replay buffer size, the structure of neural network, discounting rate $\gamma$, learning rate and $\epsilon$-greedy. In our case, since we want the agent to keep learn replay experiences at very beginning, thus we just keep the replay buffer size larger. Then the default structure of neural network is complicated enough and we does not change it significantly. What we have focused on is $\gamma$, learning rate and $\epsilon$. Regarding to the discounting rate, we just focus on the reward in a short term instead of a long term, since the agent should get a high score in just tens or hundreds of timesteps. Therefore, we try 0.99, 0.95 and 0.9 for the value $\gamma$. Finally, we find when it equals to 0.99, the mean reward increases most efficiently. For $\epsilon$, it is the probability of exploration. In the experiments, we find if it is too small, the agent will get stuck in a corner and never turn for lack of explorations. Conversely, if it is too large, the agent will never keep moving ahead and always go back and forth due to a huge exploration rate. In addition, this probability will decease over timesteps. Hence, for number of maximum timesteps equals to $2 \times 10^7$, $\epsilon = 1$ will be the best among 0.5, 1, 2, 3, 4 and 5. For learning rate, it will not affect the convergence significantly. Therefore, we choose $10^{-4}$.

In Table 1, we show the mean reward of DQN with different parameters, $\gamma$ and $\epsilon$, at 100 episodes. And thus we have $\gamma = 0.99$ and $\varepsilon = 1$ for best choice. Furthermore, we can find for same $\varepsilon$, $\gamma = 0.99$ is basically the best, and $\varepsilon$ larger than 1 causes better performance.

| $\varepsilon$ \ $\gamma$ | 0.99 | 0.95 | 0.9 |
|---|---|---|---|
| 0.5 | 437 | 363 | 413 |
| 1 | 543 | 522 | 348 |
| 2 | 374 | 294 | 310 |
| 3 | 440 | 257 | 438 |
| 4 | 446 | 437 | 522 |
| 5 | 425 | 404 | 524 |

Table 1: The mean reward of DQN with different parameters, $\gamma$ and $\epsilon$, at 100 episodes.

The result in Figure 4 is the default settings of DQN, which is not satisfying because the agent is learning in a very low speed.

And for the result of optimal hyperparameters in Figure 5, the mean rewards are high but unstable. The mean rewards go up and down frequently. It can achieve the maximum reward, 802, at around 850 episodes. But finally after 1000 episodes, the mean reward is only around 630 with a high variance. The demo results shows a poor performance.

After searching the previous work online, we find that the performance of DQN on the Atari game, MsPacman, is relatively bad and lower than human level significantly [Mni17]. Therefore, we try another method, PPO, which is supposed to

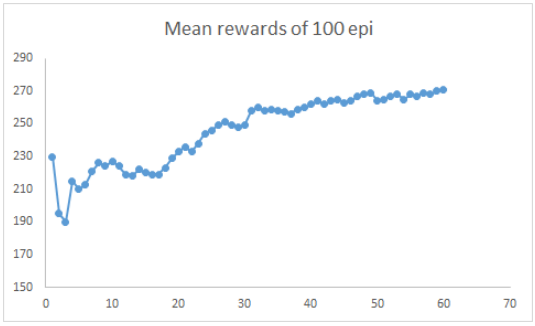


Figure 4: The average rewards versus the number of episodes for DQN.

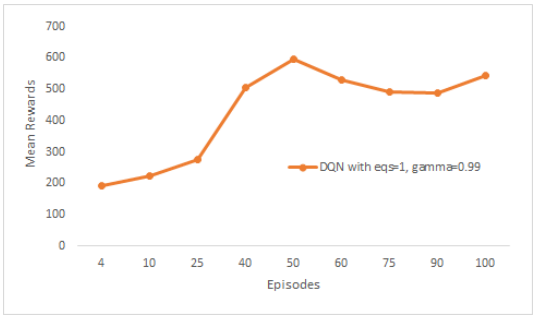


Figure 5: The average rewards versus the number of episodes for DQN with optimal hyperparameters.

work well in a continuous control problem [Sch17]. This can preliminarily shown in Figure 6, which will be showed more in Section 5.3.

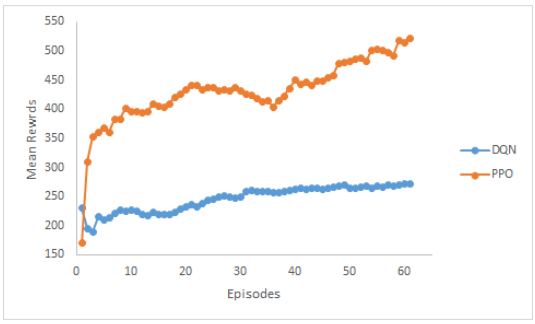


Figure 6: The mean rewards versus the number of episodes for DQN and PPO with default settings.

## 5.2 PPO

We firstly modify the codes of *OpenAI* so that it can work on MsPacman-v4 [BCP+16] successfully. We find that the 8 possible rewards outputted by the environment will only be 0 when the player does not eat a point and 1 when the player eats a point, which is very simple and may not generate a good model. Hence, we should change the implementation of this function. Besides, the method of estimating surrogate objective function is optional including CNN (Convolutional Neural Network), MLP (Multi-Layer Perceptron) and LSTM (Long Short Term Memory). Therefore, we can also compare these three neural networks.

**Comparison of different Implementations of Reward Function**

The default implementation of rewards are just set to be one if the action $a_t$ taken by the agent at $t$ timestep can lead to eating one point, and zero if not. However, this may be too simple and thus we modify this slightly. Instead of one or zero, we set, at $t$ timestep, the reward of eating one point as 4, eating no point as -1 and losing one life as -10, -100 or -1000. We run this three cases and the default one for 1,500 episodes and the result of the mean rewards is shown in Figure 7.

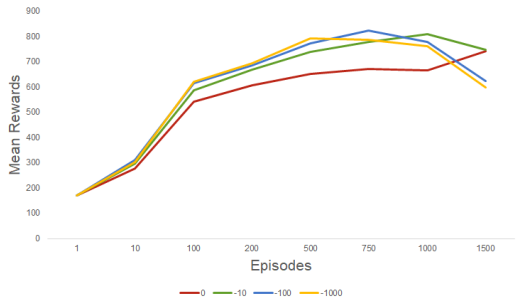


Figure 7: The mean rewards versus the number of episodes for PPO with different reward functions.

We can find that the default implementation of reward function is much worse than the modified version of reward implementations. However, after running for 1,000 episodes, the mean reward of the other three ones start to decrease and on the other hand, that of the default one is still increasing and is almost the same as the best among the three modified ones. Comparing the three modified ones, we find that the curve that the reward of losing one life is -1000 rises the most significantly before 500 episodes and however, decreases the earliest and most significantly. On the other hand, the curve of the reward set to be -10 starts to decrease after 1,000 episodes which is the most lately. Therefore, though the number of our experiments may be insufficient, we can draw a preliminary conclusion that the default setting of the reward function is

not good in short term and nevertheless, is probably better in long term.

Furthermore, we have obtained several models after training them for about ten thousand episodes, one is using the default reward setting and the other one is from the modified one, and the performance of the models can support the conclusion. The videos have been uploaded onto iCloud[1].

After running the model for multiple episodes, we can find the result which is attached in Table 2 is that in long term, *e.g.* for 10,740 episodes, the performance of the model with a default setting of reward function is much better than that of a modified reward function, which can be an evidence of the conclusion drawn before.

| | PPO with reward -10 | PPO with default reward |
|---|---|---|
| 1 | 1100 | 1380 |
| 2 | 1130 | 1880 |
| 3 | 660 | 2670 |
| 4 | 700 | 1390 |
| 5 | 990 | 2050 |
| 6 | 690 | 1940 |
| 7 | 800 | 1170 |
| 8 | 840 | 1370 |
| 9 | 1060 | 1390 |
| 10 | 720 | 1330 |
| Mean Scores | 869 | 1657 |

Table 2: The scores performed of the agents trained by PPO with the default reward function and the modified reward function (the reward of losing one life is -10).

**Comparison of different types of NN**

We also compare the performance of different types of neural network. In the project, we use CNN (Convolutional Neural Network), MLP (Multi-Layer Perceptron) and LSTM (Long Short-Term Memory). We do the experiment for 100 episodes and the result can be shown in Figure 8.

We can find that the default one, CNN, is much better than the rest two neural networks within 100 episodes. However, we do not have a further investigation quantitatively in long term. Instead, we have trained a model of MLP and another one of CNN for about 10,740 episodes. The videos of the agents playing MsPacman have been uploaded onto iCloud[2].

[1]The details are in Appendix.
[2]The details are in Appendix.

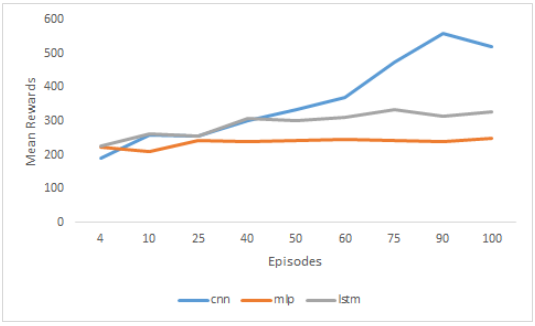


Figure 8: The mean rewards versus the number of episodes for PPO with different neural networks.

We run the agents for multiple times and then look into their performances as shown in Table 3.

| | PPO with MLP | PPO with CNN |
|---|---|---|
| 1 | 1260 | 1100 |
| 2 | 1600 | 1130 |
| 3 | 790 | 660 |
| 4 | 1270 | 700 |
| 5 | 1140 | 990 |
| 6 | 1000 | 690 |
| 7 | 580 | 800 |
| 8 | 1880 | 840 |
| 9 | 1660 | 1060 |
| 10 | 1260 | 720 |
| Mean Scores | 1244 | 869 |

Table 3: The scores performed of the agents trained by PPO with CNN and MLP.

Though the sample number is not large enough, We can find that in long term, the performance of PPO with MLP is much better that of PPO with CNN. However, we previously mention that in short term it is the opposite.

## 5.3 Comparison between DQN and PPO

As shown in Figure 6, the mean rewards for DQN and PPO with default settings has huge difference within 60 episodes. Now we will show the mean rewards within 100 episodes for both DQN and PPO with the optimal settings as Figure 9.

Although we can find the mean rewards for DQN is higher than that for PPO, DQN has a larger variance. For further training, because the time is limited, we cannot collect data in all 1000 episodes. However, we have trained several agents for 10,740 episodes and after running them 10 times, we collect the mean scores of the models which is shown in Table

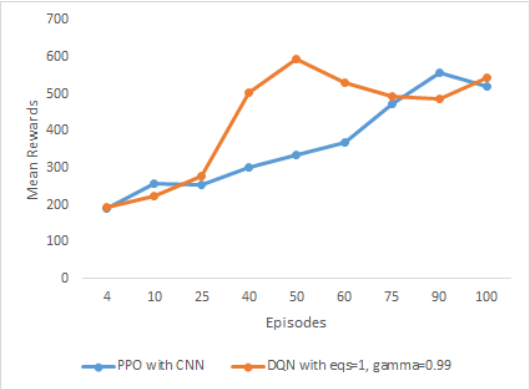


Figure 9: The mean rewards versus the number of episodes for DQN and PPO with the optimal settings.

4. Though the sample number may be insufficient, we can still acquire some information from the comparison. In addition, some videos of the agents' performance can be found on iCloud[3].

| | DQN with opti param | PPO |
|---|---|---|
| 1 | 390 | 1380 |
| 2 | 260 | 1880 |
| 3 | 320 | 2670 |
| 4 | 210 | 1390 |
| 5 | 440 | 2050 |
| 6 | 470 | 1940 |
| 7 | 520 | 1170 |
| 8 | 640 | 1370 |
| 9 | 1160 | 1390 |
| 10 | 220 | 1330 |
| Mean Scores | 463 | 1657 |

Table 4: The scores performed of the agents trained by PPO with CNN and DQN with the optimal parameters.

We can find that the mean score of PPO is much better than that of DQN. The former one is almost four times as the latter one. Therefore, we can safely draw the conclusion that the performance of PPO is much better than DQN.

## 6 Conclusion

Even the Deep Q-Network with the optimal parameters, $\gamma = 0.99$ and $\epsilon = 1$ cannot perform well in the Atari game, MsPacman, no matter how we tune its parameters. We can find after training for ten thousand episodes, the mean reward is still 463, which is much lower than human level. The potentials reason may be raised as below.

1. DQN is optimizing the Q value. However, the reward in MsPacman cannot be summarized into one or two events. All of eating normal points, eating special points and scaring ghosts will generate higher scores. Therefore, dealing with value instead of policy may be a worse choice.
2. DQN performs badly in continuous control of actions.
3. DQN cannot solve a problem of long-time memory [Mni17]. The agent of DQN averagely die after 2,000 timesteps in one game.
4. The CNN to estimate the value function may not converge since we have not tuned the parameters of the CNN.

Regarding the much better performance of PPO comparing with DQN, we can also give several potential reasons.

1. The PPO algorithm is a policy optimization method which may be more suitable in our case, MsPacman.
2. PPO can perform well in continuous control [Sch17].
3. The neural network here may or may not converge. Or the CNN and MLP in PPO have a better convergence than DQN.
4. The PPO finds a new policy after one iteration so that the loss function is minimized and keeps the difference from the previous policy small.

Talking about different kinds of reward functions, the conclusion is that the default one which is set as 1 for eating one point and 0 for not eating one point, is performing significantly well in long term, however, worse in short term, comparing with the other three modified versions which is set as +4 for eating one point, -10 for losing one life and -1 for not eating one point.

Finally considering different types of neural networks in PPO, we can conclude that the PPO with MLP is bad in short term, however, better in long term comparing with the PPO with CNN.

Although PPO can perform well, it is still far from human level.

[3]The details are in Appendix.

## 7 Acknowledgements

Thanks to *Google Cloud Platform* for a powerful computer engine to train our agents.

## References

[BCP+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[DHK+17] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Openai baselines. `https://github.com/openai/baselines`, 2017.

[GFA17] Abeynaya Gnanasekaran, Jordi Feliu Faba, and Jing An. Reinforcement learning in pacman. `http://cs229.stanford.edu/proj2017/final-reports/5241109.pdf`, 2017.

[MKS+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[Mni17] Volodymyr Mnih. Deep q-networks. `https://drive.google.com/file/d/0BxXI_RttTZAhVUhpbDhiSUFFNjg/view`, 2017.

[Sch17] John Schulman. Advanced policy gradient methods: Natural gradient, trpo, and more. `https://drive.google.com/file/d/0BxXI_RttTZAhMVhsNk5VSXU0U3c/view`, 2017.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[WdFL15] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015.

## Appendix: The Demo Videos Performed by Trained Models

(1) DQN: https://www.icloud.com/iclouddrive/0OjmWf3py05Hw66IdNFh_jM5Q#DQN.mp4

(2) PPO with default settings: https://www.icloud.com/iclouddrive/0giE4VosBBnEUMgnBJ0mU_aXw#PPO%5Fdefault.mp4

(3) PPO with CNN and revised reward function: https://www.icloud.com/iclouddrive/0g2QrdURcXdHj167qmW9xGBog#PPO%5FCNN.mp4

(4) PPO with MLP and revised reward function: https://www.icloud.com/iclouddrive/0ks3SHZPdX3ePaMxsWEXrqqtA#PPO%5FMLP.mp4