Final Projct Report

學號: t11902210　姓名：張一凡 （交換生）

Reinforcement learning is a branch of machine learning, which emphasizes taking action based on the environment to realize long-term benefits. Reinforcement learning eradicates the state of the environment, executes actions, and guides the output of better actions based on the rewards provided by environmental feedback. The game can quickly generate a large number of state action data with natural annotations, which is very suitable for training with Reinforcement learning methods to generate agents that can play games and obtain high scores. In this question, Texas poker can also affect participants' strategies from different perspectives, even if they receive poor cards, they can effectively avoid risks through their strategies. There are action states and reward modes, so Reinforcement learning can adapt to the scene well. In the report, I introduced my use of the model, comparison and selection process, parameter settings, and analysis and conclusions of the final results. At the end, I introduced the **file directory structure** in my implementation (which is very **important**)

1 Method

Monte Carlo

The Monte Carlo agent computes the winning rate through the simulated win-loss hand multiple times and chooses different strategies based on different winning rates. It chooses actions based on the current game state and hand. First, it uses the `estimate_hole_card_win_rate` function to estimate the winning rate of the opponent's card If you can, calculate the number of notes needed. Select action according to the winning rate: if the winning rate is high, select raise (raise); If the win rate is low, choose discard (fold), otherwise choose follow up (call).

D3QN

The Dueling Double Deep Q Network (D3QN) algorithm is based on the Dueling DQN algorithm and incorporates the idea of the Double DQN algorithm. The difference between Dueling DQN and DQN is that the middle hidden layer of Dueling DQN outputs value and advantage function, respectively. Combining the two to output the final Q value, Double DQN uses two DQN networks to determine actions using evaluation networks and action value using target networks, thereby eliminating the problem of overestimation of Q values. In the D3QN algorithm, the calculation method for the target value is

$$y_t = r_{t1}\gamma q(s_{t1}, argmax_a q(s_{t1}, a; w_e); w_t)$$

where use the evaluation network to obtain the action corresponding to the optimal action value in state s_{t1}, and then using the target network to calculate the action value of that action, thereby obtaining the target value. The interaction between the two networks effectively avoids the problem of overestimation in the algorithm. Where w_e and w_t represent the parameters of the evaluation network and the target network, respectively.

2 Experiment & Comparison

Agents mainly attempt to use D3QN agents and Monte Carlo agents. The agent mainly attempts to use D3QN agent and Monte Carlo agent. D3QN agent uses D3QN algorithm for training, while Monte Carlo agent calculates the winning rate by simulating the winning or losing situation of the hand multiple times, and selects different strategies based on different winning rates, such as raising chips when the winning rate exceeds 0.85. However, Monte Carlo agent is difficult to make appropriate judgment according to the field conditions, so we use it more as a training competitor to assist D3QN agent training. Table 1 shows some hyperparameters of the model.

Table 1 Agent training hyperparameters

Hyperparameters	Meaning	Value
batch_size	batch size	256
update_freq	update model frequency	50
y	discount	0.99
start_E	starting chance of random action	1
end_E	final chance of random action	0.2
num_episodes	total games of poker	2000
pre_train_steps	steps of random action before training begin	300
h_size	network intermediate parameters	128
tau	rate to update two network	0.01
is_dueling	whether or not to use dueling architecture	True
betas	optimizer parameters	(0.9, 0.999)
lr	optimizer parameters	0.0001

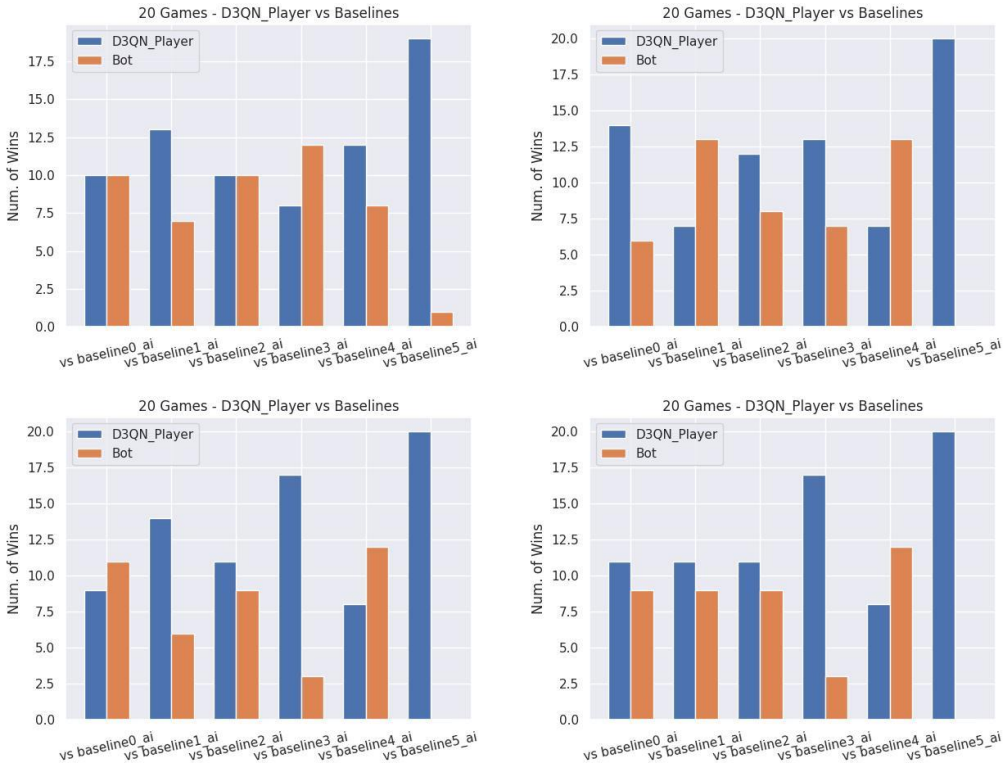

Figure 1 D3QN agents using different agents for competitive training

Figure 1 shows the game competition training results using different models, and compares the number of wins between the model and baseline through 20 games. The first two sub graphs compete with random agent and baseline 4 agent, and the last two sub graphs compete with Monte Carlo agent.

`model_random.pth` represents the agent trained with the random agent. `model_base4.pth` represents the agent trained with the baseline 4 agent. `model_MC.pth`, `model_ MC_2.pth` represent the agent trained with the Monte Carlo agent with different `num_episodes`. `hole_card_estimation.pkl` estimates the strength of hole cards. `MC_player.py` saves the Monte Carlo agent specific code. `emulator.py` emulates the running of poker games, this file is from `pypokerengine`. `train.py` defines the training process and `util.py` gives some commonly used functions.

We mainly try to use D3QN agent to conduct one-to-one training through different agents to improve its decision-making effect. It can be seen that competing against random agents can only beat three of these baseline agents, while competing against baseline 4 can beat four of them, and after debugging, competing against Monte Carlo agents can beat five of baseline agents. However, when dealing with multi-agent competition, the agent effect is poor, which may be due to only conducting one-on-one competition during training.

3 Discussion & Conclusion

We try to build a D3QN model to conduct reinforcement learning training on

Texas Hold'em Poker to build a high-level agent on the basis of meeting Texas Hold'em Poker rules. The results indicate that the results obtained through different training methods in this model can ensure competitiveness on baselines. In the end, we chose the D3QN agent that competed with the Monte Carlo agent training. We believe that the agent competing with the Monte Carlo agent can better evaluate the quality of the hand and make better decisions.

4 File Location Description

This section is mainly about my deployment instructions to understand the location of the files I am placing. Because I use the model generated by Reinforcement learning confrontation to read and solve the Texas poker problem of the D3QN method (see the introduction of the report model for specific methods), I also need to add five model files (hole_card_estimation. pkl), (model_base4. pth), etc. When I asked the teaching assistant, he said he would place src in the same position as start_ Game. py on the same layer and at start_ Use from src. agent import setup in game.py_ AI is calling the agent submitted by our classmates, so we should need to include both util. py and the. pth file used during execution in the src. Therefore, I placed them in the src folder (located side by side with the start_game.py file) and set the path to (./src/modelname) when reading the model. The following is the directory structure I tested, with the root directory on the left and the directory file where agent.py is located on the right (in the src folder of the job). In addition, I did not include the files that generated the images in the report in the submitted homework to avoid using the drawing tool library to affect the final correction.