Final Report of Question 1

NING Wentao

1 Literature Review

Reinforcement learning (RL) aims to train the agents to learn how to act based on the environmental state to maximize the expected long-term accumulated reward. By interacting with the environment, the agents can learn the optimal policy by trial and error [6]. Deep Q-Learning (DQN) [3] is the most famous RL algorithm which learns a value function to estimate the future rewards based on the current environmental state and the action to be adopted. It uses the experience replay [4] method by storing the historical transitions into a buffer and randomly choosing samples from it for training. Double DQN [7] introduces double Q-learning to reduce observed overestimations of Q values, which employs two networks for optimal action selection and value estimation, respectively. All of these algorithms rely on finding the optimal value function. Therefore, they are also called value-based RL methods.

Besides value-based methods, policy-based methods are another category of RL methods, which optimizes the parameterized policy directly. Actor-critic [6] learns a policy and a value function simultaneously, where the value function is used to evaluate the policy. Asynchronous advantage actor-critic (A3C) [2] collects data asynchronously, where the agent interacts with the environment simultaneously in multiple threads, thus speeding up the sample collection process. Proximal policy optimization (PPO) [5] utilizes a trust-region constraint to make sure the old and new policies would not diverge too much after each parameter update. Deep deterministic policy gradient (DDPG) [1] learns deterministic policies and expands them into continuous action space through the actor-critic architecture.

2 Methodology

In this project, I plan to choose DQN as the solution algorithm. DQN is the most famous RL method that has been applied in many games [3]. Since it is widely used and easy to implement, I chose it as the potential method. Next, I will elaborate in detail on the designed solutions of Part 1 and Part 2 of Question 1.

2.1 Part 1

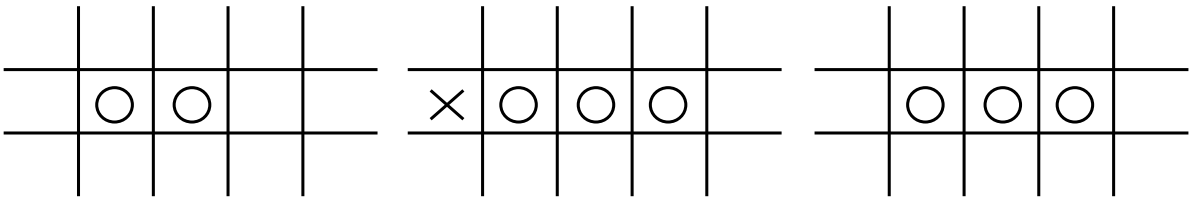
Recall that RL follows a Markov Decision Process (MDP) [6] formulation, which contains a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a decision policy $\mathcal{P}$, and a reward function $\mathcal{R}$. Here, I design the key components and formalize them with the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$.

State ($\mathcal{S}$): The state is a $9 \times 9 \times 5$ matrix, which is stacked by five 9×9 binary matrices.

1. The first matrix is the opponent's play history, 0 means no stone, and 1 means there is a piece placed by the opponent player.

2. The second matrix is the current player's history, 0 means no stone, and 1 means there is a piece placed by the current player.

3. The third matrix is the position of the opponent's latest move; only one entry is 1 and the others are 0. If the board is empty now, all entries are 0.



(a) Active two (b) Non-active three (c) Active three

Figure 1: Three chess types.

4. The fourth matrix is whether the current player is the first hand. an array full of 1 means yes, and an array full of 0 means no.

5. The fifth matrix represents the occupied positions' adjacent positions (within a distance of two grids). An entry of 1 means this position is empty and there is at least 1 piece within two grids of this position, and 0 means the opposite.

Action ($\mathcal{A}$): The action space is all the grids in the game board. Since there are $9 * 9 = 81$ grids in the game board, an action can be any integer that falls within [0, 80] range. It represents a grid in the board at index (action // 9, action % 9).

Policy ($\mathcal{P}$): Here, I implement the policy network (Q network) by stacking several convolutional neural network (CNN) layers. This is because the input state is a three-dimensional matrix, and CNNs are very good at capturing spatial information.

Reward ($\mathcal{R}$): To better illustrate how to design the reward function, I first define the three chess types, which are shown in Figure 1.

- **Active two**: Means *open-at-two-ends 2-in-a-row* shown in Figure 1(a).
- **Non-active three**: Means *open-at-one-end 3-in-a-row* shown in Figure 1(b).
- **Active three**: Means *open-at-two-ends 3-in-a-row* shown in Figure 1(c).

Table 1 shows the reward that each move will get. For example, when the player wins the game, he will get a +30 reward and his opponent will get a -30 reward. When a player gets an active two/non-active three/active three, he will get a +1/+3/+9 reward and his opponent will get a -1/-3/-9 reward.

Besides, there are two more reward functions designed for better training.

- **Alive (-0.1)**: If the player is still alive (i.e., the game is not finished yet), the player will get a -0.1 reward. This is designed to let the agent learn to make a quick win.
- **No pieces around (-0.3)**: If the player chooses a place where there is no piece within *two* grids, the player will get a -0.3 reward. This is because the place with the pieces around (within two grids) is usually the right place to go, we want the agent to learn it.

Self-play training. To get enough data to train the agent, I will use a self-play strategy to let the agent play with itself. By making the agent play the game for several episodes, the experience data are stored in the replay buffer and sampled to train the agent.

Table 1: Reward design for each chess type.

Type	Reward
Active two	± 1
Non-active three	± 3
Active three	± 9
Win/Lose	± 30

Table 2: The win rate of the trained RL agent

	Play first	Play second
Part 1	99.8%	100.0%
Part 2	99.7%	100.0%

Some strict rules. It should be noted that although the trained RL agent is generally capable of selecting the correct place to go, there are instances where it may make mistakes. As a result, I propose some strict rules to enhance the agent's capabilities.

1. If there is a place where the current player can win directly, place the stone right down there.
2. Else if there is a place where the opponent player can win, place the stone right down there.
3. Else, let the trained RL policy decide where to go.

2.2 Part 2

In Part 2, we also utilize the DQN algorithm to train the RL agent. However, unlike in Part 1, players are required to determine the amount of energy they will utilize to ensure the successful execution of their moves. Consequently, modifications to the design of the action space are necessary to ensure compatibility with the DQN method.

Action ($\mathcal{A}$): For the energy points that each round can use, let the agent choose from $\{0, 0.2, 0.4, 0.6, 0.8, 1.0\}$ (6 values in total). Since the agent will also choose the next position (81 possible positions in the game board in total), the agent will have $6 * 81 = 486$ possible actions in total. Therefore, an action can be any integer within the [0, 485] range in this situation. The energy points to use is 0.2 * (action // 81) and the next position is at index (action % 81 // 9, action % 81 % 9).

Some strict rules. The rules in Part 2 are also slightly different from Part 1.

1. If there is a place where the current player can win directly, *use as many energy points as possible* and place the stone right down there.
2. Else if there is a place where the opponent player can win, *use as many energy points as possible* and place the stone right down there.
3. Else, let the trained RL policy decide where to go.

The state, policy network, reward design, and training processes remain the same as in Part 1.

Table 3: The win rate of the trained RL agent without the use of strict rules

	Play first	Play second
Part 1	99.1%	98.3%
Part 2	96.8%	97.2%

3 Experiments

I compared the trained RL agent with the random policy. Specifically, let the trained agent play 1000 rounds against the random policy as the first and second players, separately. The results are shown in Table 2. The win rates are **close or equal to 100% in all the cases**, showing the effectiveness of our proposed methods.

Besides, I also conduct an ablation study to show the effectiveness of the proposed strict rules. We conduct the same experiments without using rules in the trained RL agent. The win rates are shown in Table 3. The results show that the proposed rules can effectively improve the capabilities of the trained RL agents. Moreover, the win rates of the trained RL agents *remain consistently high even without strict rules* (all over 95%). This indicates the effectiveness of our proposed training and reward designs.

All codes can be found and run in Github[1]. There are also some videos[2] that record the agent-self-play games and the games between the trained RL models and the random policy.

[1] https://github.com/Stevenn9981/tic_tac_toe

[2] https://github.com/Stevenn9981/tic_tac_toe/blob/master/videos/

References

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

[2] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1928–1937. JMLR.org, 2016.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[4] M. Pieters and M. A. Wiering. Q-learning with experience replay in a dynamic environment. In *SSCI*, pages 1–8. IEEE, 2016.

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[6] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. *IEEE Trans. Neural Networks*, 9(5):1054–1054, 1998.

[7] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100. AAAI Press, 2016.