The Application of Q-Learning in the Prisoner's Dilemma: Achieving Nash Equilibrium in Multi-Agent Systems

Yuanchang Hao

Computer Science, Ohio State University, Columbus, USA
ychao@iastate.edu

Abstract. The Prisoner's Dilemma is a classic example of a game in game theory where two players must decide independently whether to cooperate or betray the other, with the outcome of their choices affecting both their rewards or punishments. This study utilizes the Q-Learning algorithm to solve the problem of achieving Nash equilibrium in the Prisoner's Dilemma using the Parallel-Env environment from the PettingZoo library. Q-Learning allows for multiple random selection of action strategies, providing corresponding rewards and updating Q-values through the Bellman equation. The ε-greedy strategy is used to balance exploration and exploitation, ensuring that the agents sufficiently explore various actions while gradually converging to the optimal strategy. This approach is particularly well-suited for problems like the Prisoner's Dilemma, where limited and definite actions lead to quantifiable rewards. By designing the environment using PettingZoo's Parallel-Env class, all agents are allowed to make decisions simultaneously at each step. In conclusion, this project demonstrates the practicality and efficiency of Q-Learning in solving multi-agent dilemmas and reinforces its applicability in broader multi-agent systems.

Keywords: Q-learning, prisoners dilemma, multi-Agent reinforcement learning

1 Introduction

Multi Agent Reinforcement Learning (MARL) is multiple agents making decisions in the same environment. Each agent may cooperate or compete with each other, or both [1]. As for the reinforcement learning, the main principle is trial and error, that is, the agent interacts with the environment and receives feedback from it. The agent then iteratively optimizes based on the feedback information, which is commonly referred to as a Markov Decision Process [2-4].

For MARL, there are multiple agents in the same environment, each following the process of reinforcement learning. This means that for each agent, their decisions are not only influenced by the action of other agents, but also affect the behaviors of other agents. The main application of multi-agent reinforcement learning is to solve an unpredictable or unpredictable environment. In the face of this kind of problem, there

Y. Wang (ed.), *Proceedings of the 2024 International Conference on Artificial Intelligence and Communication (ICAIC 2024)*, Advances in Intelligent Systems Research 185,
https://doi.org/10.2991/978-94-6463-512-6_77

are usually three solutions: 1) Design effective agent rules to avoid conflicts. 2) Use means of communication so that agents can exchange information with each other. 3) The learning mechanism is added, and the behavior strategy of the agent can be optimized and improved through iteration

In previous studies and related information, there are many that introduce the types of multi-agent reinforcement learning, reinforcement learning coding environments, and reinforcement learning methods. In the paper, Zhang et al [1] introduce about three types of relationship between the agent including: fully competitive, fully collaborate, and mix. Also, python libraries such as Gym, Pettingzoo and Magent, introduce a convenient environment on simulating the agent problems. For Gym, it is design for single Agent while Magent is design for an environment that contains a huge number of quantities of agent, about over a thousand, pettingzoo, it's better to use for multiple agents but under a thousand. However, there are few papers that analyze how to integrate these three aspects, namely, which methods and environments should be used for different types of multi-agent reinforcement learning and how to implement them in code. Also, a few papers discuss the types of decisions made by agents. To be more detailed, for agents, the decisions they make are either limited choices or non-selective.

This paper will focus on the range of agents numbering approximately under and explores how to use the PettingZoo library, based on Python, to apply Q-learning and genetic algorithms to solve decision-making problems faced by two types of agents: those with choice-based decisions and those with non-choice-based decisions. In addition, the paper will use the Nash equilibrium in the prisoner's dilemma, and the gradual evolution of evasion strategies by predators to illustrate choice-based decisions and non-choice-based decisions. Furthermore, it will provide a methodology and implementation process using Q-learning and genetic learning. These approaches offer agents a reference for implementing two different behavioral strategy choices.

2 Methods

2.1 The Introduction of Prisoner's Dilemma

The Prisoner's Dilemma is a fundamental scenario in game theory as shown in Fig. 1: two criminals are imprisoned in separate cells and cannot communicate with each other. Each person has the option either to remain silent or confess. If both prisoners choose to cooperate, they will each be sentenced to one year in prison. If both choose to defect, they will each be sentenced to two years in prison. However, if one opts to cooperate while the other defects, the cooperative prisoner will receive a three-year sentence, whereas the defecting one will be released without charges [5-7].

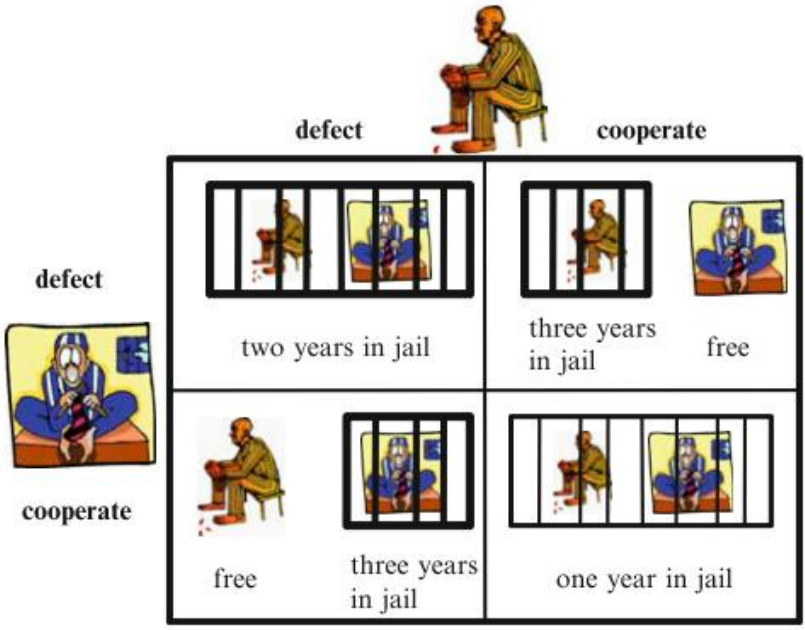

Fig. 1. The schematic of the prisoner's dilemma [8].

For both prisoners, there are only two choices: to remain silent or to confess. If considering all possible strategy combinations for the two prisoners and the corresponding payoffs for each combination, then for a prisoner, when he chooses to cooperate (remain silent), his payoff function is $u1(-1,-3)$. When he chooses to defect (confess), his payoff function is $u2(0,-2)$. When comparing the two payoff functions, it can be found that choosing to defect, represented by $u2$, yields the highest benefit for the prisoner. Assuming both prisoners are completely rational and self-interested, they will both choose to defect, reaching an equilibrium state. In this scenario, if both choose to defect, they will each receive a payoff of -2, resulting in both being imprisoned for two years. This equilibrium state is known as the Nash Equilibrium.

2.2 Q Learning

Q-learning is a model-free reinforcement learning algorithm used to learn a policy by interacting with the environment to maximize cumulative rewards. It determines the expected cumulative reward of taking a specific action in a given state by learning a value function known as the Q-function [9, 10].

Some basic and important definition as well as rules about Q learning are provided as follows:

1) Q value function: The Q-function represents the expected total reward obtained by taking action a in state s. It is denoted as $Q(s,a)$

2) Q-table: In discrete state and action spaces, Q-values can be stored in a table.

3) Update Rule (Bellman Equation): The Q-learning algorithm updates the Q-values using the Bellman equation

About the Bellman Equation:

$Q(s,a) \leftarrow Q(s,a)+\alpha[r+\gamma \max a' \quad Q(s',a')-Q(s,a)]$

←: This symbol means "gets assigned to" or "is updated to". It's indicating that the value of $Q(s,a)$ will be updated based on the expression on the right-hand side of the equation.

• Q(s,a): The Q-value of the current state s and action a, representing the long-term value of taking a specific action in the current state.

• a: The learning rate, which controls the extent to which new information influences the current Q-value. It's typically a small decimal between 0 and 1.

• r: The immediate reward, indicating the immediate feedback received after taking action aa.

• γ: The discount factor, which balances the influence of future rewards on current decisions. It's usually a value between 0 and 1, where values closer to 1 prioritize future rewards more.

• max $a'Q(s',a')$: The maximum Q-value in the next state s′. This part represents the maximum long-term value achievable by selecting the best action in the next state.

2.3 Introduction About Pettingzoo and Parallel-env

PettingZoo is a Python-based library primarily used for simulating and handling multi-agent reinforcement learning problems. Also, it includes many preset environments and problem simulations.

Parallel-Env is an abstract class in the PettingZoo library used to define multi-agent parallel environments. Unlike PettingZoo's Agent-Environment Cycle Environment (AECEnv) class, Parallel-Env allows all agents to take actions simultaneously within the same timestep. This parallel mode is particularly suitable for environments where all agents need to make decisions at the same time. In the coding part about the Prisoner's Dilemma, both prisoners make their decisions at the same time, and there is no communication between them, therefore, it is suitable to use the environment provided by Parallel-Env.

2.4 Training Process and Q-learning Implementation

During the training process, the Q-learning algorithm is used to train and optimize the agents. The training process is as follows: first, the environment and the actions of the agents are initialized. At each timestep, the agents select actions using the ε-greedy strategy, which balances exploration (randomly choosing actions) and exploitation (choosing the current best action). Then, the agents interact with the environment based on the chosen actions and receive feedback in the form of rewards and new states. The Q-table is updated using the Bellman equation to gradually optimize the expected reward for each state-action pair. Throughout the training process, the exploration rate ε is gradually decayed, allowing the agents to explore more in the early stages and focus on exploiting the best strategies in the later stages. After multiple training episodes, the agents eventually reach the Nash Equilibrium in the Prisoner's Dilemma.

3 Results and Discussion

By printing the results of the prisoners' choices before and after training shown in Table 1 and Table 2, it can be observed that the final outcome reaches a Nash Equilibrium. In the two tables, the first column shows the prisoners' choices: 1 represents confessing, and 0 represents remaining silent. The second column displays the prisoners' payoffs. As previously described, if both prisoners choose to confess, their payoff is 1 each. If prisoner A confesses while prisoner B remains silent, A's payoff is 5 and B's is 0. If both prisoners choose to remain silent, their payoff is 3 each.

Table 1. Prisoners' choices without training.

Agent Actions	Rewards
player_0': 1, 'player_1': 0	player_0': 5, 'player_1': 0
player_0': 0, 'player_1': 0	player_0': 3, 'player_1': 3
player_0': 1, 'player_1': 1	player_0': 1, 'player_1': 1
player_0': 0, 'player_1': 1	player_0': 0, 'player_1': 5
player_0': 0, 'player_1': 1	player_0': 0, 'player_1': 5

Table 2. Prisoners' choices after the training.

Agent Actions	Rewards
player_0': 1, 'player_1': 1	player_0': 1, 'player_1': 1
player_0': 1, 'player_1': 1	player_0': 1, 'player_1': 1
player_0': 1, 'player_1': 1	player_0': 1, 'player_1': 1
player_0': 1, 'player_1': 1	player_0': 1, 'player_1': 1
player_0': 1, 'player_1': 1	player_0': 1, 'player_1': 1

In Table 1, without Q-learning training, the prisoners' choices are based purely on randomness, and it could be observed that their choices are disordered. That is, the two prisoners might choose either 0 or 1. In the second table, it could be observed that both prisoners choose 1, which means they both defect and confess, resulting in a payoff of 1 for each of them. This paper randomly selects actions using the Q-values table and updates the Q-values using the Bellman equation. Through multiple updates, the Q-values stabilize.

When dealing with problems like the Prisoner's Dilemma, the Q-learning method demonstrates significant advantages. Firstly, in the Prisoner's Dilemma, each agent needs to choose from a set of possible actions. Q-learning effectively balances exploration and exploitation through the ε-greedy strategy, helping the agent widely explore various options in the early stages of training and gradually focus on the optimal strategy. Secondly, the payoff function in the Prisoner's Dilemma is clear and easy to quantify, allowing Q-learning to directly use these numerical rewards for Q-value updates.

Specifically, through continuous updates of the Q-table, the agent learns to choose actions that maximize the expected reward in each state, gradually mastering the optimal strategy combination. Given sufficient training time and an appropriate learning rate, the agent's strategy will converge to the Nash Equilibrium, where no agent can gain a higher payoff by unilaterally changing their strategy. The convenience of this method and its ability to handle quantifiable rewards make Q-learning an effective tool for solving game-theoretic problems like the Prisoner's Dilemma. Although Q-Learning is effective in handling small-scale problems like the Prisoner's Dilemma, it faces scalability issues when applied to larger state-action spaces. The increase in the number of states and actions results in exponential growth in memory and computational requirements, making it impractical for more complex environments.

4 Conclusion

This paper provides an overview of multi-agent reinforcement learning, using a fundamental problem—the Prisoner's Dilemma—to explore how such problems can be solved using the Q-learning algorithm. It discusses the implementation and benefits of the algorithm in this context and offers insights into potential future research directions. However, when dealing with problems involving more agents, this leads to further computational demands. The scalability issue becomes a challenge that needs to be addressed in future work.

References

1. Zhang, K., Yang, Z., Başar, T.: Multi-agent reinforcement learning: A selective overview of theories and algorithms. Handbook of reinforcement learning and control, 321-384 (2021).
2. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Journal of Artificial Intelligence Research 4, 237-285 (1996).
3. Li, Y.: Deep reinforcement learning: An overview. arXiv preprint arXiv:1701.07274. (2017)
4. Wiering, M.A., Van Otterlo, M.: Reinforcement learning. Adaptation, learning, and optimization 12(3), 729 (2012).
5. Axelrod, R.: Effective choice in the prisoner's dilemma. Journal of Conflict Resolution 24(1), 3-25 (1980).
6. Nowak, M.A., Sigmund, K.: The alternating prisoner's dilemma. Journal of Theoretical Biology 168(2), 219-226 (1994).
7. Rapoport, A., Chammah, A.M.: Prisoner's dilemma: A study in conflict and cooperation. University of Michigan Press (1965).
8. Prisoner's dilemma https://kr.pinterest.com/pin/315533517622875895/, (2024).
9. Watkins, C.J., Dayan, P.: Q-learning. Machine Learning 8, 279-292 (1992).
10. Clifton, J., Laber, E.: Q-learning: Theory and applications. Annual Review of Statistics and Its Application 7, 279-301 (2020).