To be a fast adaptive learner: using game history to defeat opponents

Guangzhao Cheng, Siliang Tang
Zhejiang University
cgz@zju.edu.cn, siliang@zju.edu.cn

Abstract

In many real-world games, such as trader repeatedly bargaining with customers, it is very hard for a single AI trader to make good deals with various customers in a few turns, since customers may adopt different strategies even the strategies they choose are quite simple. In this paper, we model this problem as fast adaptive learning in the finitely repeated games. We believe that past game history plays a vital role in such a learning procedure, and therefore we propose a novel framework (named, F3) to fuse the past and current game history with an Opponent Action Estimator (OAE) module that uses past game history to estimate the opponent’s future behaviors. The experiments show that the agent trained by F3 can quickly defeat opponents who adopt unknown new strategies. The F3 trained agent obtains more rewards in a fixed number of turns than the agents that are trained by deep reinforcement learning. Further studies show that the OAE module in F3 contains meta-knowledge that can even be transferred across different games.

1 Introduction

Game Theory can be defined as the study of mathematical models of conflict and cooperation between intelligent rational decision-makers. [1] It can be applied in different ambit of Artificial Intelligence, such as AI Pluribus in multiplayer poker [2,3], Multi-agent AI systems [4,5], Imitation and Reinforcement Learning [6,7] and so on.

Interactions in the real world, like negotiation, are challenging tasks. The opponents are usually various and the turns of interactions are usually multiple but limited. However previous interactions, historical information, are rarely used. This kind of interaction can be modeled as a finitely repeated game. In the face of various opponents or strategies, how to make the agent adapt quickly is an important issue. Many new strategies are integration or variants of the old so we can use historical information to make the agent have fast adaptability.

The repeated game consists of a number of repetitions of some base game (also called a stage game) [8] which may be broadly divided into two classes, finite and infinite, depending on how long the game is being played for. Finite games are those in which both players know that the game is being played for a specific and finite number of rounds. When the game ends, all the players will receive end terminal [9] and final rewards.

In repeated games, game history is the main basis for making decisions and defeating opponents. Memory is a special form of knowledge [9]. Judging the type of opponent, inferring incomplete information, and predicting the behavior of the opponent all depend on it. In the absence of other additional information from the opponent, historical information can be said to be the sole basis.

But historical information has different types and different history have different meanings. The set of histories which have terminals [9] can be defined as Past game history and the set of actions that have been taken in the current event can be defined as Current game history.
history consists of histories after which the chance player and the decision maker move [9]. As shown in Figure 1, in finitely repeated games, the Past game history is the set of events that have an end terminals and have received the final rewards. The Current game history is the set of actions that have already made in the ongoing game.

Current studies about the use of historical information in repeated games are mainly limited to the Current game history (e.g. [9, 10, 11]). The studies about modeling opponents (e.g. [12, 13]) treat all their opponents as one type. But in the real world, the type of opponents may be fickle. In a new game, we may meet opponents that we have never encountered before. Different from one-shot interactions, players would consider more about the long-term payoffs.

In finite repeated games, the opponents’ many new strategies are integration or variants of the old. For example, in Iterated Prisoners Dilemma (IPD), Spiteful Tit For Tat (Prison1998), Hard Tit For Two Tats [14] and Suspicious Tit For Tat [15] are all proposed based on the Tit For Tat [16]. The Past game history information library stores the strategies that have been encountered and the best response under various circumstances. When agents encounter a new strategy, they can infer or learn new strategies quickly to defeat their opponents from the similar strategies in the Past game history.

In order to make the agent have the fast adaptability, we propose a novel method of modeling Past game history and a novel framework (named, F3) that fuses current and past game history. As shown in Figure 2, the framework consists mainly of six components: (1) Current Game History Memory, (2) Past Game History Memory, (3) Opponent Action Estimator (OAE), (4) Hierarchical History Encoder (HE), (5) Action Decoder (AD) and (6) History Updater (HU).

Given the Current game history, the Opponent Action Estimator (OAE) module with a structure akin to memory networks [20] predicts an embedding vector which represents an estimation of the opponent’s future behavior based on Past game history. The Hierarchical History Encoder (HE) module will extract information from different dimensions of the game history and fuse the estimated vector and the current game history representation.

We experiment on several classic repeated games whose number is fixed. In each game, the opponent’s strategy is different. Because in the real world, the opponents are usually various and the turns of interactions are multiple. The experimental results show that good results can be achieved only by using past history. The highest score difference can be achieved by fusing past and current history. The Opponent Action Estimator (OAE) module can estimate the opponent’s next action. It can be reused in new games which means it has the cross-game adaptability.

In summary, our main contributions are as follows:

• In this paper, we proposed a framework to model the Current game history and the Past game history in the meantime.
Figure 2: The diagram of our framework that consists of six main components: (1) Current Game History, (2) Past History Memory, (3) Opponent Action Estimator (OAE), (4) Hierarchical History Encoder (HE), (5) Action Decoder (AD) and (6) History Updater (HU).

- The model and framework we proposed can make learners to be fast adaptive agents. They can quickly defeat opponents on new strategies.
- The OAE model can be reused in new games. So this framework is easy to implement quickly even in the face of new games, which makes agents have the cross-game adaptability.

2 Method

2.1 Basic Notation

To simplify exposition, we start with some notations. We use \( p_l \) to represent the learner (our trained agent). The set of opponents players is denoted by \( P = \{ p_1, p_2, ..., p_n \} \). \( P \) is consistent with the Opponent Strategy Library. Each opponent \( p_i \) represents a strategy. The action set denoted by \( A = \{ a_1, a_2, ..., a_n \} \). The set action of the learner is the same as the opponent’s. A complete repeated game \( G \) consists of \( n \) turns stage games \( g_i \). Thus, \( G = \{ g_1, g_2, ..., g_n \} \).

Before each repeated game \( G_i \) starts, the agent \( p_l \) randomly selects an opponent \( p_o \) from the opponents set \( P \), then they \( (p_l, p_o) \) play \( n \) turns stage games. Note that all players cannot determine the upper turns limit \( n \), which means they do not know when the \( G_i \) will end. In every stage game \( g_i \), the players choose actions from \( A \). A stage game action profile \( a^l = (a^l_1, a^l_2) \) consists of the action of player \( p_l \) and the action of the opponent player \( p_o \), then they will separately get a stage reward \( r^l, r^o \) from the payoff function \( g : A \Rightarrow \mathbb{R} \). When a complete repeated game \( G_i \) is over, the final score of each player is the average of all stage rewards:

\[
R^l = \frac{1}{n} \sum_{i=1}^{n} r^l_i, \quad R^o = \frac{1}{n} \sum_{i=1}^{n} r^o_i
\]

When a complete repeated game \( G_i \) ends, History Updater decides whether it will be added to the Past Game History Memory \( P = \{ G_1, G_2, ..., G_m \} \). If a repeated game \( G_i \) proceeds to step \( k \), the set of \( k - 1 \) step actions before is defined as Current Game History Memory \( C = \{ a^l_1, a^l_2, ..., a^l_k, a^o_1, a^o_2, ..., a^o_k \} \), \( a^l_i = (a^l_i, a^o_i) \). Thus the entire history \( H = P + C \).
2.2 Framework Overview

The framework is shown in Figure 2. The Current Game History Memory is the actions that have been made in the current game which includes yours and your opponent’s. (We assume that actions by both parties can be observed.) The Past Game History Memory stores some complete history which have terminals and final scores. The Opponent Action Estimator inputs the current history and find K-similar past history items and outputs a predict vector about the opponent’s future behavior through a network. It contains one step opponent action estimation and multiple steps opponent action estimation. The Hierarchical History Encoder fuses current history and the predict vector. The Action Decoder converts vectors into the final action. If the current game is over and gets the final scores, the History Updater will update the Past History Memory.

2.3 Opponent Action Estimator (OAE)

Many new strategies are integrations or variants of the old. The past history contains more knowledge to defeat new opponents. But facing new challenges, how to find entries similar to the current problem in Past game history and extract useful information from them is a huge challenge. We propose Opponent Action Estimator (OAE) to solve this problem.

Given the Current game history $C$ of $m$ turns, the OAE module predicts an embedding vector which represents an estimate of the opponent’s future behavior. This estimation is based on Past game history.

First, the $m$ turns $C = \{(a'_1, a''_1), (a'_2, a''_2), ..., (a'_m, a''_m)\}$ is used to find the K-similar items from the past game history. As we defined in Section 3.1, the Past History Memory $P = \{G_1, G_2, G_3, ..., G_t\}$. In $P$, each game $G_i = \{(a'_1, a''_1), (a'_2, a''_2), ..., (a'_m, a''_m), (a'_{m+1}, a''_{m+1}), ..., (a'_n, a''_n)\}$. Each $G_i$ will be divided into two subsets: $G_i^m = \{(a'_1, a''_1), (a'_2, a''_2), (a'_3, a''_3), ..., (a'_m, a''_m)\}$ and $G_i^{n-m} = \{(a'_{m+1}, a''_{m+1}), ..., (a'_n, a''_n)\}$. We calculate the similarity between $C$ and each $G_i^m$ to find the K-similar items $G_{sim} = \{G_1, G_2, ..., G_k\}$.

Second, the estimate network, a deep neural network with a structure akin to memory networks [20], inputs the $m$ turns Current game history $C$ and the K-similar items $G_{sim}$ and outputs an embedding vector $E_y$. The Current game history $C$ is embedded into a d-dimensional representation $c$ using LSTM. At each layer $l$ of the model, the K-similar items $G_{sim}$ was also embedded into $A_i$ and
$C_i$ using two LSTMs. The weight tying of the estimate network, we take **Adjacent**: the output embedding for one layer is the input embedding for the one above, i.e. $A_{k+1} = C_k$. The output of each layer is calculated by adding the weighted sum of another embedding of the similar items. To include the previous item representation.

Finally, from the K-similar items, based its $G_i^{n-m}$, we extract the opponent’s action set $O_i$ after the step $m$. $O_i = \{a_{o_{m+1}}, a_{o_{m+2}}, ... , a_{o_n}\}$. The opponent’s action set of the K-similar items $O = \{O_1, O_2, ... , O_k\}$ is also fed into a network, FusionNet, to generate a vector $E_y$. The structure of the FusionNet is LSTM [18], followed by a Multi-Layer Perceptrons (MLP).

Our goal is to make $E_x$ and $E_y$ equal. So we learn a estimate network $\phi_1$ parametrized by $\alpha_1$ and a fusion network $\phi_2$ parametrized by $\alpha_2$ by minimizing the following loss:

$$L_{OAE} = |\phi_1(C, G^{\text{sim}}; \alpha_1) - \phi_2(O; \alpha_2)| = |E_x - E_y| \quad (2)$$

**One-Step Opponent Action Estimator (O-OAE)** From each item $G_i^{\text{sim}}$, if we only extract the opponent’s action of step $m + 1$, then $O_i = \{a_{o_{m+1}}\}$. This method is called One-Step Opponent Action Estimator (see Figure 3 for details).

**Multi-Steps Opponent Action Estimator (M-OAE)** From each item $G_i^{\text{sim}}$, if we extract all the opponent’s action from $G_i^{n-m}$, then $O_i = \{a_{o_{m+1}}, ..., a_{o_n}\}$. This method is called Multi-Step Opponent Action Estimator.

### 2.4 Hierarchical History Encoder (HE)

Game is an interactive process. Sometimes different strategies can only be manifested according to the behavior of the opponent. So we use **Hierarchical History Encoder (HE)** to extract information from different dimensions of the game history.

In repeated games, given the current game history $C = \{(a^1_1, a^2_1), (a^1_2, a^2_2), ..., (a^k_n)\}$, we use **Action-Level Encoder** and **History-Level Encoder** to encode it. If the length of the game history $C$ is $n$, the **Action-Level Encoder** is a LSTM $f_{\text{action}}: \mathbb{L}^{n \times 1} \Rightarrow \mathbb{L}^d$ mapping the agent’s action $a_{self} = (a^1_1, a^2_1, ..., a^k_n)$ or the opponent’s action $a_{oppo} = (a^1_1, a^2_1, ..., a^k_n)$ into a $d$-dimensional vector $h_{\text{action}}$ or $h_{\text{oppo}}$ as the action-level representation of the current history; and the **History-Level Encoder** is a LSTM $f_{\text{history}}: \mathbb{L}^{n \times 2} \Rightarrow \mathbb{L}^d$ mapping the union action $C = \{(a^1_1, a^2_1), (a^1_2, a^2_2), ..., (a^k_n)\}$ into a $d$-dimensional vector $h_{\text{history}}$ as the history-level representation. So there are two **Action-Level Encoders** and one **History-Level Encoder** in our games. If the number of players is $t$, you can use the $t$ **Action-Level Encoders** to encode each player’s action and then the **History-Level Encoder** will be a LSTM $f_{\text{history}}: \mathbb{L}^{n \times t} \Rightarrow \mathbb{L}^d$.

Through OAE, we will get an embedding vector $E_x$ which represents an estimate of the opponent’s future behavior. From HE, we can get $h_{\text{self}}$, $h_{\text{oppo}}$ and $h_{\text{history}}$ which represents the feature of the current history. Then, we use **CombineNet**, a MLP, to fuse these vectors and get a $d$-dimensional vector $h_c$.

### 2.5 Action Decoder (AD)

Through the above module, we can get $h_c$. The last step is to turn this vector into the next action for the learner. We use **Action Decoder (AD)** to get the final action. The size of the action set $A = \{a_1, a_2, ..., a_s\}$ is $s$. $h_c$ is a $d$-dimensional vector. We can also use a MLP $f_{\text{decoder}}: \mathbb{M}^d \Rightarrow \mathbb{M}^s$ to map the $h_c$ to a $s$-dimensional one-hot vector $a$.

If you only want to encode Past game history, you can also get the final action directly from OAE. The OAE outputs a $s$-dimensional embedding vector $E_x$ which represents an estimate of the opponent’s future behavior, then $a = \text{MLP}(E_x)$ (see Figure 3 for details).
| Payoff |  |  |
|---|---|---|---|---|
| | Cooperate(C) | Defect(D) | Swerve(S) | Going(G) |
| Cooperate(C) | 3, 3 | 0, 5 | 2, 2 | 1, 5 |
| Defect(D) | 5, 0 | 1, 1 | 5, 1 | 0, 0 |

2.6 Past History Updater (HU)

When the players receive the end signal, the current game $G_i$ is over. Then the agent will get the final reward $R^l$ and the opponent will get the final reward $R^o$. As shown in formula (1), the final score of each player is the average of all stage rewards from the payoff function $g$. The score difference $\Delta R = R^l - R^o$. The action set $G = \{(a_1^1, a_1^o), (a_2^1, a_2^o), ..., (a_n^1, a_n^o)\}$, the final reward $R^l$ and $R^o$ compose a complete history item $h = \{G, R^l, R^o\}$.

When the framework starts, if the Past History Memory is empty or the maximum capacity is not reached, history item $h$ will be added to the memory. If the memory reaches its maximum capacity, when the current game is over, history item $h$ will be added to the memory and meanwhile the item with the smallest difference will be deleted.

Note that once the OAE and Past History Updater are trained, their parameters will be fixed. When the current game is over, we use the final score difference $\Delta R = R^l - R^o$ as a guide and use REINFORCE algorithm \[19\] to update the networks parameters which contain Hierarchical History Encoder and Action Decoder.

3 Experiments

3.1 Game Setting

We experiment on two classic games: Prisoner’s Dilemma and Chicken. They have different reward functions. Their payoff functions are shown in Table 1 and Table 2. Axelrod\[1\] is a python library for the Iterated Prisoner’s Dilemma research as easily as possible and create the de-facto tool for future Iterated Prisoner’s Dilemma research. It has access to over 200 strategies, including original and classics like Tit For Tat and Win Stay Lose Shift. New strategies will be encouraged to contribute and new games are easy to defined. Note that all players cannot determine the upper turns limit $n$ in finitely repeated games, which means they do not know when the $G_i$ will end. In our experiments, the game turns is fixed. Each game will play 50 turns.

3.2 Evaluation Metric

When the current game is over, the agent will get the final reward $R^l$ and the opponent will get the final reward $R^o$. In multiple tests, we calculate their average scores $\overline{R^l}$ and $\overline{R^o}$ (see Equation 1). In all games, the number of turns is fixed. Therefore, a player with a higher score means mastering the opponent ’s strategy more quickly, which means it has faster adaptability. The difference $\Delta R = \overline{R^l} - \overline{R^o}$ is the evaluation metric.

3.3 Baselines

Q-Learning Q-learning \[21\] is the easiest way to use Current game history. In this paper, experiments are performed on Second-order Markov process.

Deep Q Network Deep Q learning (DQN) \[22\] leverages advances in deep learning to learn policies from high dimensional sensory input. Different from Atari 2600 games using convolutional networks, we use Long Short Term Memory (LSTM) to convert the game history to vectors. The Current game history $C = \{a_1^1, a_2^1, ..., a_i^1, ..., a_k^1\}$, so $h$ =LSTM($C$). A two-layer Multi-Layer Perceptrons (MLP) is then used to predict the next action.

\[http://axelrod.readthedocs.org/\]
Table 3: The average scores difference $\Delta R = R_l - R_o$. The Old Strategies is the training strategies set and New Strategies is the new strategies set. After each 500 times training epoch, we will test one time.

| Model               | Old Strategies | New Strategies | Old Strategies | New Strategies |
|---------------------|----------------|----------------|----------------|----------------|
| Q-learning          | -1.022         | -1.055         | -0.872         | -0.840         |
| HE+AD(DQN)          | 0.522          | 0.301          | 0.434          | 0.461          |
| HE+AD(PG)           | 1.199          | 1.091          | 1.105          | 0.943          |
| O-OAE+AD            | 1.148          | 0.958          | 0.842          | 0.764          |
| M-OAE+AD            | 1.079          | 0.894          | 0.851          | 0.793          |
| O-OAE+HE+AD         | 1.288          | 1.116          | 1.123          | 1.009          |
| M-OAE+HE+AD         | 1.236          | 1.118          | 1.131          | 1.001          |

Table 4: The average scores difference $\Delta R$ of Reused OAE.

| Model               | New Trained OAE | Reused OAE |
|---------------------|-----------------|------------|
| O-OAE+AD            | 0.840           | 0.747      | 0.842       | 0.764 |
| M-OAE+AD            | 0.854           | 0.780      | 0.851       | 0.793 |
| O-OAE+HE+AD         | 1.134           | 1.010      | 1.123       | 1.009 |
| M-OAE+HE+AD         | 1.139           | 1.031      | 1.131       | 1.001 |

**Policy Gradient**  The network structure is the same as DQN which uses LSTM to encode the current history and uses MLP to map the action set $A$. When the current history ends, we train the networks using Policy Gradient [19]. The network will only be updated when the game is over.

3.4 Use Current History Only

Although after each stage game, all players will receive a stage reward, but in the finite repeated game, the final reward is the basis for determining the outcome when the game is over. Q-Learning only relies on the history of the previous $n$ turns to make the next decision, instead of making decisions from the whole of the finite repeated game. DQN relies on all current history, but it only considers the next decision, which is obviously greedy. Because sometimes, in order to maximize the overall benefit, players may make compromises in some stage games. While, the Policy Gradient methods target at modeling and optimizing the policy directly. In a finite repeated game, the opponent’s strategy usually constant. So the Policy Gradient can model this strategy and defeat it.

From the Table 3 we can see that the Q-Learning score is negative, which means that the method can not beat the opponent at all. The DQN method beats its opponent by about 0.5 points on the Old Strategies and about 0.3 points on the New Strategies. While the Policy Gradient score difference exceeds 1 whether it is a new strategy or an old strategy. Experiments show that the Policy Gradient is suitable for finite repeated game. Note that the first three rows of experiments in Table 3 only use the Current game history. Since Policy Gradient is the best method for finite repeated games, all next experiments use this method for optimization.

3.5 The Performance of OAE

The Opponent Action Estimator(OAE) module use current history to retrieve the K-similar items form the past history and generates an estimated vector. To test the performance of OAE, after getting the estimated vector about the opponent’s next step, we use Action Decoder (AD) directly to decode it into the action. We use the Policy Gradient [19] to train the AD module. As shown in Figure 3 although the scores do not reach the level of using only the current history, but also achieved a good score.
3.6 Comparative Experiments

The current history is the most important basis for decision-making, we use Hierarchical Encoder to encode it. The optimal response of strategies is stored in the past game history, it can estimate the opponent’s next action and provide more information for the best response. As shown in Figure 3, the highest score difference can be achieved by fusing Past game history and Current game history.

The One-Step Opponent Action Estimator (O-OAE) only relies on the opponent’s one-step decision in past history while the Multi-Steps Opponent Action Estimator (M-OAE) use opponent’s multi-steps decision. But as shown in Figure 3 whether it is directly using OAE or fusing with the past history, we find that they have no obvious difference.

3.7 Reuse of OAE

Given the Current game history, the OAE module akin to memory networks [20] predicts a embedding vector which represents an estimate of the opponent’s future behavior. Essentially, it is a retrieval model. As long as the current history and past history belong to the same game, the OAE can be reused. We trained the O-OAE and M-OAE on Prisoner’s Dilemma, but we reuse them on Chicken Game. We experiment two kinds of OAE on Chicken Game: (a) the New Trained OAE is the Opponent Action Estimator module trained on the Chicken game and tested on the Chicken game and (b) the Reused OAE is the module trained on the Prisoner’s Dilemma game and reused on the Chicken game.

4 Related Work

When a static non-cooperative strategic game is repeated over time, players iteratively interact by playing a similar stage game. This type of game is called as a repeated game. It consists of a number of repetitions of some base game (also called a stage game) [8]. Repeated games may be broadly divided into two classes, finite and infinite, depending on how long the game is being played for. Infinite games are those in which the game is being played an infinite number of times. Finite games are those in which both players know that the game is being played a specific and finite number of rounds. When the game ends, all the players will receive end terminal [9] and final rewards.

In the fields of Economics, related research has focused on Modeling Bounded Rationality. [9] proposed the set of histories $H$ can be partitioned into three subsets: (1) $Z$: the set of terminal histories; (2) $C$: the set of histories after which the chance player moves and (3) $D = H - Z - C$: the set of histories after which the decision maker moves. Inspired by this, we define the set of terminal histories $Z$ the Past history and explore the fusion of Past game history and Current game history.

Current studies about the use of historical information and limited memory in repeated games are mainly limited to the Current game history. In Multi-Agent Reinforcement Learning (MARL), [17] proposed a new algorithm which demonstrates strong performance in empirical tests against a variety of opponents in a wide range of environments. But the bounded memory here is limited to Past history. [18] talked about the Memory-one strategies. This manuscript investigates best responses to a collection of memory-one strategies as a multidimensional optimisation problem. But in this article, we not only discuss the current history, but also try to use the Past history.

In the research of Game Theory, the new strategies on the prisoner’s dilemma and other games are constantly being proposed: Tit For Tat [16], Spiteful Tit For Tat (Prison1998), Hard Tit For Two Tats [14], Suspicious Tit For Tat [13] and so on. Our proposed method can defeat the new strategy based on learning the old strategy, which makes agents easily cope with the new strategies that are constantly proposed.

The method of modeling teammates in MARL also can quickly defeat the opponents. [12] proposed Self Other-Modeling (SOM), in which an agent uses its own policy to predict the other agent’s actions and update its belief of their hidden state in an online manner. [13] presented Learning with Opponent-Learning Awareness (LOLA), a method that reasons about the anticipated learning of the other agents. They treat all their opponents as one type. But in the real world, the type of opponents may be variable. In a new game, we may meet opponents that we have never encountered before.


5 Conclusion

In this paper, we propose a model to make use of past history and a framework to fuse past history and current history. The model and framework we proposed can make learners to be fast adaptive agents. They can quickly defeat opponents on new strategies. The experimental results show that good results can be achieved only by using past history. The highest score difference can be achieved by fusing past and current history. The OAE model can be reused in new games which means it has the cross-game adaptability.

Broader Impact

Whether it is a dialogue or a game, in the process of human interaction, it needs an agent with fast adaptability. The model and framework proposed in this paper can make better use of past and current game history, so that the agent can adapt quickly.

The agent has the ability to adapt quickly, and can bring a better experience to humans in human-computer interaction. AI can be used to develop more agents, which makes AI serve humans better.

References

[1] Myerson R B. Game theory [M]. Harvard university press, 2013.
[2] Brown N, Sandholm T. Superhuman AI for multiplayer poker [J]. Science, 2019, 365(6456): 885-890.
[3] Brown N, Sandholm T. Safe and nested subgame solving for imperfect-information games [C] // Advances in neural information processing systems. 2017: 689-699.
[4] Parsons S, Wooldridge M. Game theory and decision theory in multi-agent systems [J]. Autonomous Agents and Multi-Agent Systems, 2002, 5(3): 243-254.
[5] Semsar-Kazerooni E, Khorasani K. Multi-agent team cooperation: A game theory approach [J]. Automatica, 2009, 45(10): 2205-2213.
[6] Alos-Ferrer C, Schlag K H. Imitation and learning [M] // The handbook of rational and social choice. Oxford University Press, 2009.
[7] Tuyls K, Nowé A. Evolutionary game theory and multi-agent reinforcement learning [J]. The Knowledge Engineering Review, 2005, 20(1): 63-90.
[8] Game Theory: Breakthroughs in Research and Practice [M]. IGI Global, 2018: 247.
[9] Rubinstein A. Modeling bounded rationality [M]. MIT press, 1998.
[10] Powers R, Shoham Y. Learning against opponents with bounded memory [C] // IJCAI. 2005, 5: 817-822.
[11] Glymantsi N E, Knight V A. Stability of defection, optimisation of strategies and the limits of memory in the Prisoner’s Dilemma [J]. arXiv preprint arXiv:1911.12112 2019
[12] Raileanu R, Denton E, Szlam A, et al. Modeling others using oneself in multi-agent reinforcement learning [J]. arXiv preprint arXiv:1802.09640 2018.
[13] Foerster J, Chen R Y, Al-Shedivat M, et al. Learning with opponent-learning awareness [C] // Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems. International Foundation for Autonomous Agents and Multiagent Systems, 2018: 122-130.
[14] Stewart A J, Plotkin J B. Extortion and cooperation in the Prisoner’s Dilemma [J]. Proceedings of the National Academy of Sciences, 2012, 109(26): 10134-10135.
[15] Hilbe C, Nowak M A, Traulsen A. Adaptive dynamics of extortion and compliance [J]. PloS one, 2013, 8(11).
[16] Axelrod R, Hamilton W D. The evolution of cooperation [J]. science, 1981, 211(4489): 1390-1396.
[17] Glymantsi N E, Knight V A. A systematic literature review of the Prisoner’s Dilemma; collaboration and influence [J]. arXiv preprint arXiv:1911.06128 2019.
[18] Hochreiter S, Schmidhuber J. Long short-term memory [J]. Neural computation, 1997, 9(8): 1735-1780.
[19] Williams R J. Simple statistical gradient-following algorithms for connectionist reinforcement learning [J]. Machine learning, 1992, 8(3-4): 229-256.
[20] Sukhbaatar S, Weston J, Fergus R. End-to-end memory networks [C] // Advances in neural information processing systems. 2015: 2440-2448.
[21] Watkins C J C H, Dayan P. Q-learning[J]. Machine learning, 1992, 8(3-4): 279-292.

[22] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning[J]. arXiv preprint arXiv:1312.5602, 2013.

[23] Sukhbaatar S, Weston J, Fergus R. End-to-end memory networks[C]//Advances in neural information processing systems. 2015: 2440-2448.

[24] Luce R D, Raiffa H. Games and decisions: Introduction and critical survey[M]. Courier Corporation, 1989.

[25] Nowak M, Sigmund K. A strategy of win-stay, lose-shift that outperforms tit-for-tat in the Prisoner's Dilemma game[J]. Nature, 1993, 364(6432): 56-58.