Achieving Cooperation Through Deep Multiagent Reinforcement Learning in Sequential Prisoner's Dilemmas

Weixun Wang, Jianye Hao *, Yixi Wang
School of Computer Software, Tianjin University
Tianjin, China
{wxwang,jianye.hao}@tju.edu.cn,yixiwang2017@outlook.com

Matthew E. Taylor
Washington State University
Pullman, Washington, USA
taylorm@eecs.wsu.edu

ABSTRACT

The Iterated Prisoner's Dilemma has guided research on social dilemmas for decades. However, it distinguishes between only two atomic actions: cooperate and defect. In real world prisoner's dilemmas, these choices are temporally extended and different strategies may correspond to sequences of actions, reflecting grades of cooperation. We introduce a Sequential Prisoner's Dilemma (SPD) game to better capture the aforementioned characteristics. In this work, we propose a deep multiagent reinforcement learning approach that investigates the evolution of mutual cooperation in SPD games. Our approach consists of two phases. The first phase is offline: it synthesizes policies with different cooperation degrees and then trains a cooperation degree detection network. The second phase is online: an agent adaptively selects its policy based on the detected degree of opponent cooperation. The effectiveness of our approach is demonstrated in two representative SPD 2D games: the Apple-Pear game and the Fruit Gathering game. Experimental results show that our strategy can avoid being exploited by exploitative opponents and achieve cooperation with cooperative opponents.

KEYWORDS

Multiagent learning; Deep learning; Learning agent-to-agent interactions (negotiation, trust, coordination)

1 INTRODUCTION

Learning is the key to achieving coordination with others in multiagent environments [25]. Over the last couple of decades, a large body of multiagent learning techniques have been proposed that aim to coordinate on various solutions (e.g., Nash equilibrium) in different settings, e.g., minimax Q-learning [19], Nash Q-learning [16], and Conditional-JAL [2], to name just a few.

One commonly investigated class of games is the Prisoner's Dilemma (PD), in which a Nash Equilibrium solution is not a desirable learning target. Until now, a large body of work [1, 2, 4, 6, 11, 21, 23] has been devoted to incentivize rational agents towards mutual cooperation in repeated matrix PD games. However, all the above works focus on the classic repeated PD games, which ignores several key aspects of real world prisoner's dilemma scenarios. In repeated PD games, the moves are atomic actions and can be easily labeled as cooperative or uncooperative or learned from the payoffs [3]. In contrast, in real-world PD scenarios, cooperation/defection behaviors are temporally extended and the payoff signals are usually delayed (available after a number of steps of interactions).

Crandall [5] proposes the Pepper framework for repeated stochastic PD games (e.g., two-player gate entering problem), which can extend strategies originally proposed for classic repeated matrix games. Later some techniques extend Pepper in different scenarios, e.g., playing against opponents with changing strategy [7, 13]. However, these approaches rely on hand-crafted state inputs and tabular Q-learning techniques to learn optimal policies. Thus, they cannot be directly applied to more realistic environments whose states are too large and complex to be analyzed beforehand.

Leibo et al. [17] introduce a 2D Fruit Gathering game to better capture the real world social dilemma characteristics, while also maintaining the characteristics of classical iterated PD games. In this game, at each time step, an agent selects its action based on its image observation and cannot directly observe the actions of the opponent. Different policies represent different levels of cooperativeness, which is a graded quantity. They investigate the cooperation/defection emergence problem by leveraging the power of deep reinforcement learning (DRL) [22] from the descriptive point of view: how do multiple selfish independent agents' behaviors evolve when agents update their policy using deep Q-learning? In contrast, this paper takes a prescriptive and non-cooperative perspective and considers the following question: *how should an agent learn effectively in real-world social dilemma environments when it is faced with different opponents?*

To this end, in the paper, we formally introduce the general notion of sequential prisoner's dilemma (SPD) to model real world social dilemma problems. We then propose a multiagent deep reinforcement learning approach for mutual cooperation in SPD games. Our approach consists of two phases: offline and online phases. The offline phase generates policies with varying cooperation degrees and trains a cooperation degree detection network. To generate policies, we propose using the weighted target reward and two schemes, independent Actor-Critic (IAC) and joint Actor-Critic (JAC), to train the baseline policies with varying cooperation degrees. Then we propose using the policy generation approach to synthesize the full range of policies from these baseline policies. Lastly, we propose a cooperation degree detection network implemented as an LSTM-based structure with an encoder-decoder module. The online phase adaptively selects a policy with the proper cooperation degree from a continuous range of candidates based on the detected cooperation degree of an opponent. Intuitively, our algorithm is cooperation-oriented and also robust against selfish exploitation. We evaluate the performance of our approach using two 2D SPD games (the Fruit Gathering and Apple-Pear games). Experimental results show that our agent can efficiently achieve mutual cooperation under self-play and also perform well against opponents with switching stationary policies.

Table 1: Prisoner's Dilemma

	C	D
C	R, R	S, T
D	T, S	P, P

2 BACKGROUND

2.1 Matrix Games and the Prisoner's Dilemma

A matrix game can be represented as a tuple $\langle N, \{A_i\}_{i\in N}, \{R_i\}_{i\in N}\rangle$, where N is the set of agents, A_i is the set of actions available to agent i with $\mathcal{A}$ being the joint action space $A_1 \times \ldots \times A_n$, and R_i is the reward function for agent i. One representative class of matrix games is the prisoner's dilemma game, as shown in Table 1. In this game, each agent has two actions: cooperate (C) and defect (D), and is faced with four possible rewards: R, P, S, and T. The four payoffs satisfy the following four inequalities under a prisoner's dilemma game:

- $R > P$: mutual cooperation is preferred to mutual defection.
- $R > S$: mutual cooperation is preferred to being exploited by a defector.
- $2R > S + T$: mutual cooperation is preferred to an equal probability of unilateral cooperation and defection.
- $T > R$: exploiting a cooperator is preferred over mutual cooperation.
- $P > S$: mutual defection is preferred over being exploited.

2.2 Markov Game (Stochastic Game)

A Markov game $\mathcal{M}$ is defined by a tuple $\langle N, S, \{A_i\}_{i\in N}, \{R_i\}_{i\in N}, \mathcal{T}\rangle$, where S is the set of states and N is the number of agents, $\{A_i\}_{i\in N}$ is the collection of action sets, with A_i being the action set of agent i, and $\{R_i\}_{i\in N}$ is the set of reward functions, $R_i : S\times A_1\times \ldots \times A_n \to \mathcal{R}$ is the reward function for agent i. $\mathcal{T}$ is the state transition function: $S \times A_1 \times \ldots \times A_n \to \Delta(S)$, where $\Delta(S)$ denotes the set of discrete probability distributions over S. Matrix games are the special case of Markov games when $|S| = 1$. Next we formally introduce SPD by extending the classic iterated PD game to multiple states.

2.3 Definition of Sequential Prisoner's Dilemma

SPD is a specific form of SSD [17]. SSD models the social dilemma including prisoner's dilemma game, stag hunt game and chicken game, and requires agents to learn policies for implementing the intention. Following this work [17], specifically, we focus on the sequential prisoner's dilemma (SPD) in social dilemma. A two-player SPD is a tuple $\langle \mathcal{M}, \Pi\rangle$, where $\mathcal{M}$ is a 2-player Markov game with state space S. Π is the set of policies with varying cooperation degrees. The empirical payoff matrix $(R(s), P(s), S(s), T(s))$ can be induced by policies $(\pi^C, \pi^D \in \Pi)$, where π^C is more cooperative than π^D. Given two policies, π^C and π^D, the corresponding empirical payoffs (R, P, S, T) under any starting state s with respect to the payoff matrix in Section 2.1 can be defined as $(R(s), P(s), S(s), T(s))$ through their long-term expected payoff, where

$$R(s) := V_1^{\pi^C,\pi^C}(s) \quad = V_2^{\pi^C,\pi^C}(s), \tag{1}$$

$$P(s) := V_1^{\pi^D,\pi^D}(s) \quad = V_2^{\pi^D,\pi^D}(s), \tag{2}$$

$$S(s) := V_1^{\pi^C,\pi^D}(s) \quad = V_2^{\pi^D,\pi^C}(s), \tag{3}$$

$$T(s) := V_1^{\pi^D,\pi^C}(s) \quad = V_2^{\pi^C,\pi^D}(s), \tag{4}$$

We can define the long-term payoff $V_i^{\vec{\pi}}(s)$ for agent i when the joint policy $\vec{\pi} = (\pi_1, \pi_2)$ is followed starting from s.

$$V_i^{\vec{\pi}}(s) = E_{\vec{a}_t\sim\vec{\pi}(s_t), s_{t+1}\sim\mathcal{T}(s_t,\vec{a}_t)}[\sum_{t=0}^{\infty}\gamma^t r_i(s_t, \vec{a}_t)] \tag{5}$$

A Markov game is an SPD when there exists a state $s \in S$ for which the induced empirical payoff matrix satisfies the five inequalities in Section 2.1. Since SPD is more complex than iterated PD, the existing learning strategies for iterated PD games cannot be directly applied in SPD. Moreover, in SSD, also exist other types of dilemma games including Stag hunt game and Chicken games. Stag hunt game has two Nash equilibria (cooperation, cooperation) and (defection, defection), and is essentially a coordination game. Similarly, for chicken game, it also has two Nash equilibria (cooperation, defection) and (defection, cooperation), and is essentially an anti-coordination game. For both games, the only problem is how agents can coordinate on the same Nash equilibrium. In contrast, for SPD games, only mutual defection is a Nash equilibrium. However, we are interested in agents exhibiting different non-Nash equilibrium cooperative behaviors against cooperative opponents and defective behaviors (converging to Nash equilibrium) faced with purely selfish ones.

2.4 Deep Reinforcement Learning

Q-Learning and Deep Q-Networks: Q-learning and Deep Q-Networks (DQN) [22] are value-based reinforcement learning approaches to learn optimal policies in Markov environments. Q-learning makes use of an action-value function for policy π as $Q^\pi(s, a) = E_{s'}[r(s, a)+ \gamma E_{a'\sim\pi}[Q^\pi(s', a')]]$. DQN uses a deep convolutional neural network to estimate Q-values and the optimal Q-values are learned by minimizing the following loss function:

$$y = r + \gamma max_{a'}\bar{Q}(s', a'), \tag{6}$$

$$L(\theta) = E_{s,a,r,s'}[(Q(s, a|\theta) - y)^2] \tag{7}$$

where $\bar{Q}$ is a target Q network whose parameters are periodically updated with the most recent θ.

Policy Gradient and Actor-Critic Algorithms: Policy Gradient methods are for a variety of RL tasks [27, 29]. Their objective is to maximize $J(\theta) = E_{s\sim p^\pi, a\sim\pi_\theta}[R]$ by taking steps in the direction of $\nabla_\theta J(\theta)$, where

$$\nabla_\theta J(\theta) = E_{s\sim p^\pi, a\sim\pi_\theta}[\nabla_\theta log\pi_\theta(a|s)Q^\pi(s, a)] \tag{8}$$

where p^π is the state transition distribution. Practically, the value of Q^π can be estimated in different ways. For example, $Q^\pi(s, a)$ serves as a critic to guide the updating direction of π_θ, which leads to a class of actor-critic algorithms [28].

3 DEEP RL: TOWARDS MUTUAL COOPERATION

Algorithm 1 describes our deep multiagent reinforcement learning approach, which consists of two phases. In the offline phase, we first seek to generate policies with varying cooperation degrees. Since the number of policies with different cooperation degrees is infinite, it is computationally infeasible to train all the policies from scratch. To address this issue, we first train representative policies using Actor-Critic until convergence (i.e., cooperation and defection

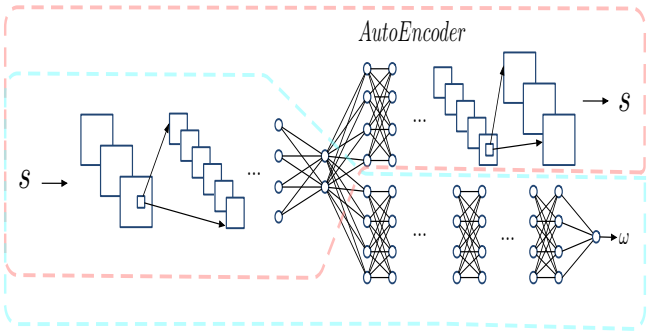


Figure 1: Cooperation Degree Detection Network

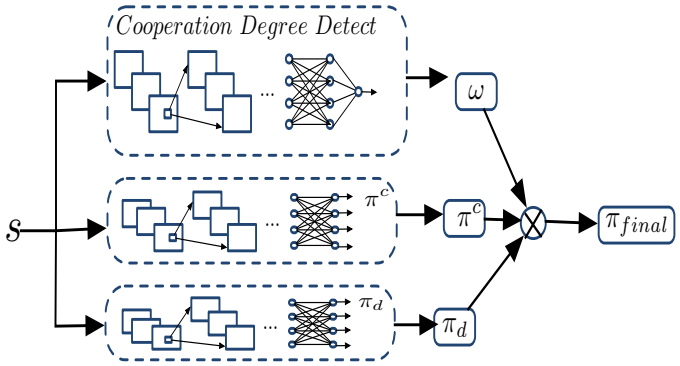


Figure 2: The Structure of Deep Reinforcement Learning Approach Towards Mutual Cooperation

baseline policies) (Lines 3-5) detailed in Section 3.1; second, we synthesize the full range of policies (Lines 6-7) from the above baseline policies, which will be detailed in Section 3.2. Another task is to how to effectively detect the cooperation degree of the opponent. We divide this task into two steps: we first train a cooperation degree detection network offline (Lines 8-10), which will be then used for real-time detection during the online phase, detailed in Section 3.3. In the online phase, our agent plays against any opponent by reciprocating with a policy of a slightly higher cooperation degree than that of the opponent we detect (Lines 12-17), detailed in Section 3.4. Intuitively, on one hand, our algorithm is cooperation-oriented and seeks for mutual cooperation whenever possible; on the other hand, our algorithm is also robust against selfish exploitation and resorts to defection strategy to avoid being exploited whenever necessary.

3.1 Train Baseline Policies with Different Cooperation Degrees

One way of generating policies with different cooperation degrees is changing the key parameters of the environments. For example, Leibo et al. [17] investigate the influence of the resource abundance degree on the policy's cooperation tendency in sequential social dilemma games where agents compete for limited resources. It is found that when both agents employ deep Q-learning algorithms, more cooperative behaviors can be learned when resources are plentiful and vice versa. We may leverage similar ideas of modifying game settings to generate policies with different cooperation degrees. However, this type of approach requires perfectly understanding the environment, and also may not be feasible when one cannot modify the underlying game engine.

Another more generalized way of generating policies with different cooperation degrees is to modify agents' reward signals during learning. Intuitively, agents with the sum of all agents' immediate rewards would learn towards cooperation policies to maximize the expected accumulated social welfare, and agents maximizing only their individual reward would learn selfish (defecting) policies. Formally, for a two-player environment (agent i and j), agent i computes a weighted target reward r_i' as follows:

$$r_i' = r_i + att_{ij} \times r_j \tag{9}$$

where agent i's attitude att_{ij} reflects the relative importance of agent j in its perceived reward. By setting att_{ij} and att_{ji} to 0 or 1, agents would maximize their own return or overall return respectively. The

Algorithm 1 The Approach of Deep Multiagent Reinforcement Learning Towards Mutual Cooperation

1: *//offline phase*
2: initialize the size N_t, N_g of training and generation policy set, the number N_d, N_e of training data set and episode, the step number N_r of each episode
3: **for** training policy set index t = 1 to N_t **do**
4: set agents' attitudes
5: train agents' policy set P_t using weighted target reward
6: **for** generation policy set index g = 1 to N_g **do**
7: use policy set $\{P_t\}_{t\in N_t}$ to generate policy set P_g
8: **for** training data set index d = 1 to N_d **do**
9: generate training data set D_d as $\{P_t\}_{t\in N_t} \cup \{P_g\}_{g\in N_g}$
10: train cooperation degree detection network
11: *//online phase*
12: initialize $agent_1's$ cooperation degree
13: **for** episode index e = 1 to N_e **do**
14: **for** step index r = 1 to N_r **do**
15: $agent_1$ and $agent_2$ take actions and get rewards
16: detect $agent_2's$ cooperation degree cd_2^r using n-state trajectory and update cd_1^r based on cd_2^r
17: generate a policy with cd_1^r using policy generation

greater agent one's attitude towards agent two is, the higher the cooperation degree of agent one's learned policy would be.

Given the modified reward signal, the next question is how agents learn to effectively converge to the expected behaviors. One natural way is to resort to IAC or some independent DRL, i.e., equipping agent i with an individual DQN with reward r_i'. However, the key element to the success of DQN, experience replay memory, may prohibit effective learning in multiagent environments [9, 26]. Due to the nonstationarity problem, data in the replay memory may no longer reflect the current dynamics during learning, IACs get confused by obsolete experience and impede the learning process. A number of methods have been proposed to remedy this issue [8, 9, 20] and we omit the details which are out of the scope of this paper.

Since the baseline policy training step is performed offline, another way of improving training is to use JAC by treating both

agents as a single learner. Thus both agents share the same underlying network that learns the optimal policy over the joint action space. Note that we use JAC only for the purpose of training baseline policies offline, and we do not require controlling the policies that an opponent may use online. In this way, the aforementioned nonstationarity problem can be avoided. Besides, compared with IAC, the training efficiency can be improved significantly since the network parameters are shared across agents in JAC.

In JAC, the weighted target reward is defined as follows:

$$r_{total} = \sum_{i \in N} att_i \times r_i \tag{10}$$

where att_i represents the relative importance of agent i on the overall reward. The smaller the value of att_i, the higher the cooperation degree of agent i's learned policy and vice versa. Given the learned joint policy $\pi^{joint}(a_1, a_2|s, att_1, att_2)$, agent i can easily obtain its individual policy π_i as $\pi_i = \sum_{a_j,\ j \neq i} \pi^{joint}(a_1, a_2|s, att_1, att_2)$.

It is computationally prohibitive to train a large number of policies with different cooperation degrees due to the high training cost of IAC and JAC, and the infinite policy space. To alleviate this issue, we propose that only two baseline policies, cooperation policy π_c and defection policy π_d, need to be trained. Other policies with cooperation degree between them can be synthesized efficiently, detailed in Section 3.2.

3.2 Policy Generation

Given the baseline policies π_i^c and π_i^d, we synthesize multiple policies. Each continuous weighting factor $w_c \in [0, 1]$ corresponds to a new policy $\pi_i^{w_c}$ defined as follows:

$$\pi_i^{w_c} = w_c \times \pi_i^c + (1 - w_c) \times \pi_i^d \tag{11}$$

The weighting factor w_c is defined as policy $\pi_i^{w_c}$'s cooperation degree. The linear combination of two policies has two advantages — it 1) generates policies with varying cooperation degrees and 2) ensures low computational cost. Any synthesized policy $\pi_i^{w_c}$ should be more cooperative than π_i^d and more defecting than π_i^c. The higher the value of w_c is, the more cooperative the corresponding policy $\pi_i^{w_c}$ is. It is important to mention that the cooperation degrees of synthesized policies are ordinal, i.e., the cooperation degree of policies only reflect their relative cooperation ranking. For example, considering two synthesized policies $\pi_i^{0.6}$ and $\pi_i^{0.3}$, we cannot say that $\pi_i^{0.6}$ is twice as cooperative as $\pi_i^{0.3}$. Our way of synthesizing new policies can be understood as synthesizing policies over expert policies [12]. The previous work applies a similar idea to generate new policies to better respond with different opponents in competitive environments, however, our goal here is to synthesize policies with varying cooperation degrees in sequential Prisoner's dilemmas.

3.3 Opponent Cooperation Degree Detection

In classical iterated PD games, a number of techniques have been proposed to estimate the opponent's cooperation degree, e.g., counting the cooperation frequency when action can be observed, using a particle filter or Bayesian approach otherwise [6, 14, 15, 17]. However, in SPD, the actions are temporally extended, and the opponent's information (actions and rewards) cannot be observed directly.

Figure 3: The Fruit Gathering game (left) and the Apple-Pear game.

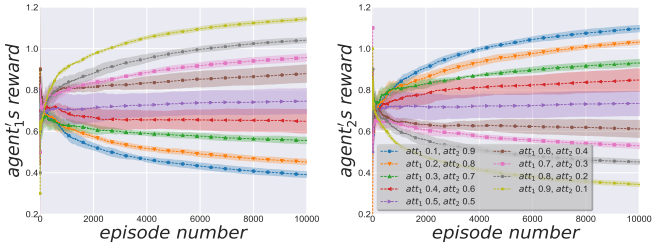


Figure 4: Agents' average rewards under policies trained with different cooperation attitudes.

Thus previous works cannot be directly applied. We need a way of accurately predicting the opponents' cooperation degree from the observed sequence of moves in a qualitative manner. In SPD, given the sequential observations (time-series data), we propose an LSTM-based cooperation degree detection network. We transform it into a supervised learning problem: given a sequence of moves of an opponent, our task is to detect the cooperation degree (label) of this opponent. We propose a recurrent neural network, which combines an autoencoder and a recurrent classifier, as shown in Figure 1. Combining an autoencoder with a recurrent classifier brings two major benefits here. It ensures the effective feature extraction of the observed moves to accelerate the training speed of the classifier and reduce fluctuation.

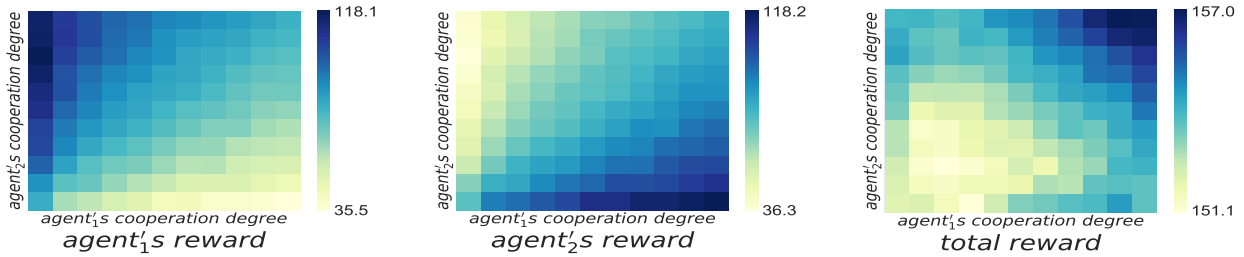
The network is trained on experiences collected by agents. Both agents i and j interact with the environment starting with initialized policies $\pi_i \in \{\pi_i^c, \pi_i^d\}$ and $\pi_j \in \Pi_j$, where π_i^c and π_i^d are baseline policies for agent i, Π_j is the learned policy set of its opponent j. Then we can get a set of trajectories under the joint policy (π_i, π_j). For each trajectory $\langle s_1, s_2, \ldots, s_d \rangle$, its label is the cooperation degree of agent i's policy. If π_i is π_i^c, the label is set 1 and 0 otherwise. The network is trained to minimize the weighted cross entropy.

3.4 Play Against Different Opponents

Once we have detected the cooperation degree of the opponent, the final question arises as to how an agent should select proper policies to play against that opponent. Unlike the self-interested approach, we seek a solution that can allow agents to achieve cooperation while avoiding being exploited.

At each time step t, agent i uses its previous n-step sequence of observations as the input of the detection network, and obtain the detected cooperation degree cd_j^t. However, the one-shot detection of the opponent's cooperation degree might be misleading due to either the detection error of our classifier or the stochastic behaviors of the opponent. Thus this may lead to high variance of our detection outcome and our response policy thereafter. To reduce the variance, agent i uses the exponential smoothing to update its current

(a) The Apple-Pear game



(b) The Fruit Gathering game

Figure 5: Average and total rewards under different cooperation degrees. The cooperation degrees of $agent_1$ and $agent_2$ increase from left to right and from bottom to top respectively. Each cell corresponds the rewards of different policy pairs.

cooperation degree estimation of its opponent j as follows:

$$cd_i^t = (1-\alpha) \times cd_i^{t-1} + \alpha \times cd_j^t \tag{12}$$

where cd_i^{t-1} is agent i's cooperation degree in the last time step, and α is the cooperation degree changing factor.

Finally, agent i sets its own cooperation degree equal to cd_i^t plus its reciprocation level, and then synthesizes a new policy with the updated cooperation degree following Equation (11) as its next-step strategy. Note that the reciprocation level can be quite low and still avoid significant loss while, produce cooperation when the opponent reciprocates in the same way. Also, our overall approach is general and any existing multiagent strategy selection approaches can be applied here. The idea here, adjusting policy by detecting the change of the opponent's cooperation degree, is similar with the work of Foerster et al. [10]. They propose that the agent considers the policy change of its opponent as the second order term of its policy gradient, and can learn towards Tit-for-Tat policy in coin game under self-play. However, they assume that both policies of the agent and its opponent have minor changes, and only focus on the process of learning together with opponents. Thus, agents cannot face opponents with fast-changing policies. In contrast, we use policy generation rather than policy gradient, and can allow agents to quickly adapt to fast-changing policies to ensure self-interests and social welfare of agents.

4 SIMULATION AND RESULTS

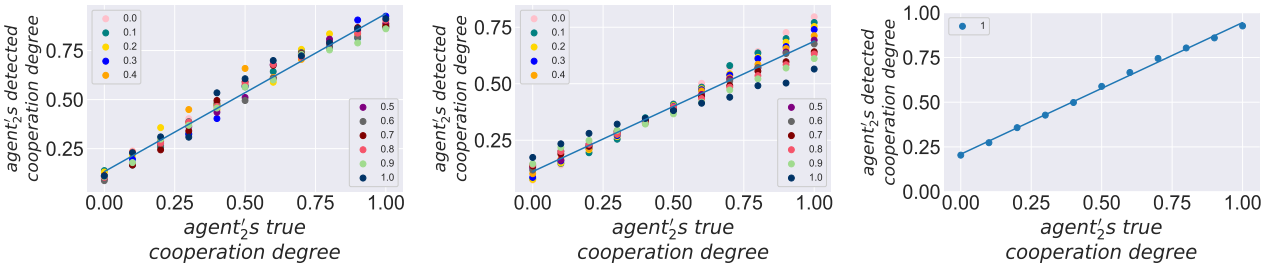
4.1 SPD Game Descriptions

In this section, we adopt the Fruit Gathering game [17] to evaluate the effectiveness of our approach and propose another game Apple-Pear, which also satisfies real world social dilemma conditions we mentioned before. Each game involves two agents (in blue and red) who receive the raw RGB screenshots as observations. The task of an agent in the Fruit Gathering game is to collect as many apples, represented by green pixels, as possible (Figure 3 (left)). An agent's action set is: step forward, step backward, step left, step right, rotate left, rotate right, use beam, and stand still. The agent obtains the corresponding fruit when it steps on the same square as the fruit is located. When an agent collects an apple, it will receive a reward of 1. The apple will be removed from the environment and respawn after 40 frames. Each agent can also emit a beam in a straight line along its current orientation. An agent is removed from the map for 20 frames if it is hit by the beam twice. Intuitively, a more defecting policy is one which frequently tags the rival agent.

For the Apple-Pear game, there is a red apple and a green pear (Figure 3 (right)). The blue agent prefers apple while the red agent prefers pear. Each agent has four actions: step right, step left, step backward, step forward, and each step of moving incurs a cost of 0.01. The fruit is collected when the agent enters its square. When the blue (red) agent collects an apple (pear) individually, it receives a higher reward of 1. When agents collect their less favored fruit, they receive a lower reward of 0.5. One exception is that they both receive a half of their corresponding rewards when they share a pear or an apple. In this game, a more defecting policy tends to collect fruits regardless of the type. In Section 4.4, we observe that both games satisfy the definition of SPD games in Section 2.3 by using policies with different cooperation degrees to play with each other.

4.2 Network Architecture and Parameter Settings

In both games, our network architectures for training the baseline policies follow standard AC networks, except that we allow both



(a) Apple-Pear (b) Fruit Gathering (c) Fruit Gathering ($cd_1 = 1$)

Figure 6: The detection results for $agent_2$ under different cooperation degrees of $agent_1$: (a) Apple-Pear game; (b) Fruit Gathering game; (c)$agent_1$'s cooperation degree as 1 in Fruit Gathering game

Figure 7: Self-play in Apple-Pear with agents using our strategy.

actor and critic to share the same underlying network to reduce the parameter space. For the underlying network, the first hidden layer convolves 32 filters of 8×8 with stride 4 with the input image and applies a rectifier nonlinearity. The second hidden layer convolves 64 filters of 4×4 with stride 2, again followed by a rectifier nonlinearity. This is followed by a third convolutional layer that convolves 64 filters of 3×3 with stride 1 followed by a rectifier. For the actor, on the basis of sharing network, the next layer includes 128 units with rectifier nonlinearity, and the final softmax layer has as many units as the number of actions. The critic is similar to the actor, but with only one scalar output.

The recurrent cooperation degree detection network is shown in Figure 1. The autoencoder and the detection network share the same underlying network. The first hidden layer convolves 10 filters of 3×3 with stride 2 with the input image and applies a rectifier nonlinearity. The second hidden layer is the same as the first one and the third hidden layer convolves 10 filters of 3×3 with stride 3 and applies a rectifier nonlinearity. The autoencoder is followed by a fourth hidden layer that deconvolves 10 filters of 3×3 with stride 3 and applies a sigmoid and its output shape is $21 \times 21 \times 10$. The next layer deconvolves 10 filters of 3×3 with stride 2 and applies a sigmoid and the output shape is $42 \times 42 \times 10$. The final layer deconvolves 10 filters of 3×3 with stride 2 and applies a sigmoid and the output shape is $84 \times 84 \times 3$. Cooperation degree detection network is followed by two LSTM layers of 256 units. The final layer is an output node.

For the Apple-Pear game, each episode has at most 100 steps. The exploration rate is annealed linearly from 1 to 0.1 over the first 20000 steps. The weight parameters are updated by soft target updates [18] every 4 steps to avoid the update fluctuation as follows:

$$\theta' \leftarrow 0.05\theta + 0.95\theta' \tag{13}$$

where θ is the parameter of the policy network and θ' is the parameter of the target network. The learning rate is set to 0.0001 and memory is set to 25000. The batch size is 128. For the loss function in the cooperation degree detection network, we set w_1 and w_2 as 1 and 2. When agents play with different opponents online, we assign the number of states visited to the length n of the state sequence which is the input of cooperation detection network. The cooperation degree changing factor α is set to 1.

The Fruit Gathering game uses the same detection network architecture. The actor-critic uses independent policy networks. Each episode has at most 100 steps during training. The exploration rate and the memory are the same as the Apple-Pear game. The weight parameters are updates the same as the Apple-Pear game:

$$\theta' \leftarrow 0.001\theta + 0.999\theta' \tag{14}$$

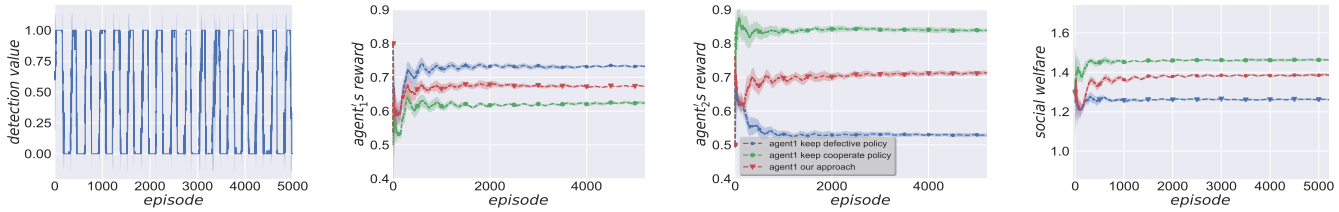
When agents play with different opponents online, we set state sequence length n as 50 and the changing factor α is set to 0.02.

4.3 Effect of Baseline Policy Generation

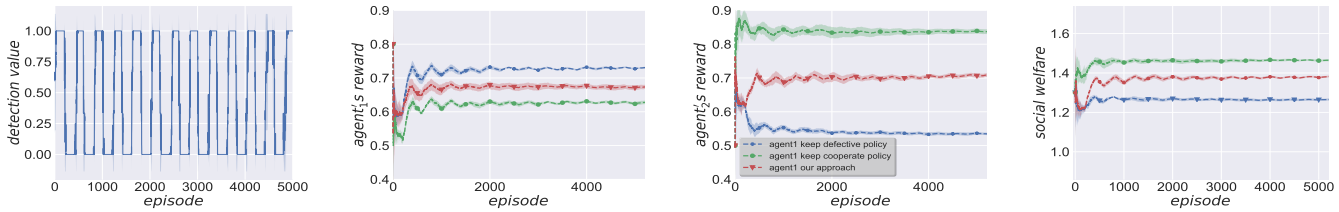
For the Apple-Pear game, the baseline policies are trained using the JAC scheme. We set the attitude in Equation (10) between 0.1 and 0.9 to train baseline policies. The reason that attitudes are not set 0 and 1 is that these settings will cause an agent with meaningless policy, e.g, the agent whose attitude equals 0 makes no influence to total reward in Equation (10) and hence will always avoid collecting fruit. This would, in turn, affect the learned policy quality of the other agent (i.e., lazy agent problem [24]). Figure 4 shows the average rewards of agents under policies trained with different weighted target rewards. The individual rewards of each agent increase gradually as its attitude increases and the other agent's attitude decreases, as seen in the results. We also evaluate the IAC scheme and do experiments in the Fruit Gathering game. Results are similar and are omitted for space.

4.4 Effect of Policy Generation

For the Apple-Pear game, the baseline cooperation policies π_1^c and π_2^c are trained under the setting of $\{att_1 = att_2 = 0.5\}$. The baseline defection policies π_1^d and π_2^d has $\{att_1 = 0.75, att_2 = 0.25\}$ and $\{att_1 = 0.25, att_2 = 0.75\}$ respectively. Then we generate the policy set of $agent_1$ $\Pi_1 = \{\pi = w_1 \times \pi_1^c + (1 - w_1) \times \pi_1^d | w_1 = 0.0, 0.1, \ldots, 1.0\}$ and the policy set of $agent_2$ Π_2 in the same way. After that, two policies π_1 and π_2 are sampled from Π_1 and Π_2 and matched against each other in the games for 200,000 episodes. The

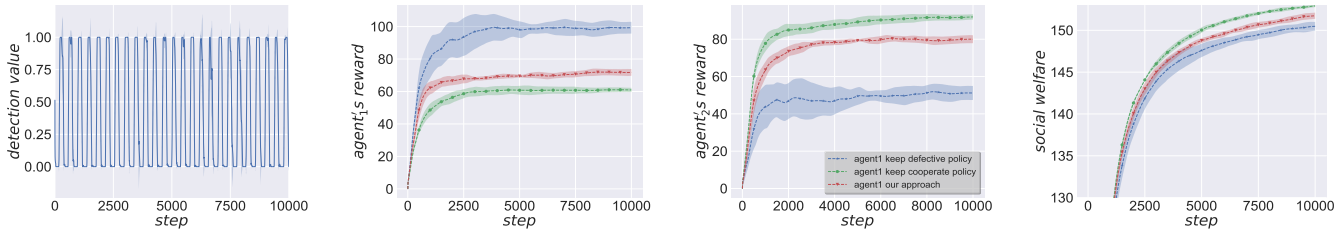


(a) policy varies between π^c and π^d every 150 episodes

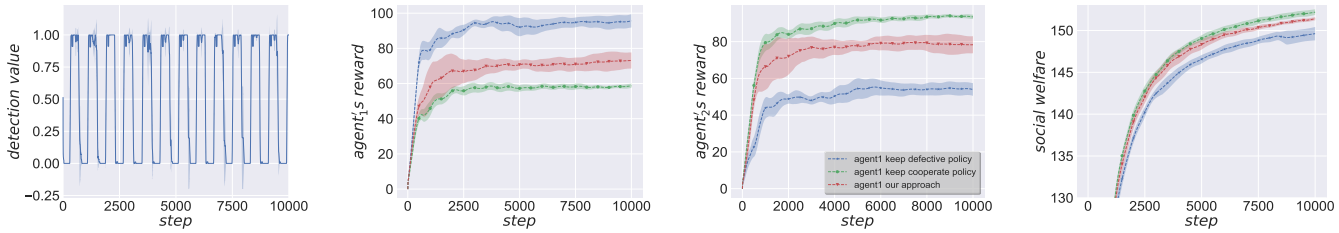


(b) policy varies between π^c and π^d every 200 episodes

Figure 8: Apple-Pear game: $agent_2$'s policy varies between π^c and π^d every 150 and 200 episodes. The average rewards of the $agent_1$ are higher when using our approach than using π^c, which means our approach can avoid being exploited by defective opponents. The social welfare is higher than using π^d, indicating that our approach can seek for cooperation against cooperative ones.



(a) $agent_2$'s policy varies between π^c and π^d every 200 steps



(b) $agent_2$'s policy varies between π^c and π^d every 350 steps

Figure 9: Fruit Gathering game: $agent_2$'s policy varies between π^c and π^d every 200 and 350 steps. Similar phenomenon can be observed as in Figure 8.

average rewards are assigned to individual cells of different attitude pairs, in which π_1 and π_2 correspond to policies with varying cooperation degrees w_1 and w_2 for agent 1 and 2 respectively (Figure 5 (a)). We can observe that when both of their cooperation degrees increase, which means they are more cooperative, their total rewards increase, and vice versa. Besides, given a fixed cooperation degree of $agent_1$, $agent_2$'s reward is increased as its cooperation degree decreases, and vice versa. Similar patterns can be observed for agent 1 as well. Therefore, it indicates that we can successfully synthesize policies with a continuous range of cooperation degrees. It also confirms that the Apple-Pear game can be seen as an SPD following the definition in Section 2.3. Moreover, similar results for the Fruit Gathering game can be observed from Figure 5 (b).

4.5 Effect of Cooperation Degree Detection

This section evaluates the detection power of the cooperation degree detection network. For the Fruit Gathering game , we collect 4000 data for each state sequence lengths $\{40, 50, 60, 70\}$, which includes 2000 data labeled as 1 and 0, respectively. The network is trained to minimize the weighted cross entropy and the cooperation degree detection accuracy is high (Figure 6 (a)). We can observe that the detection results are almost linear with the true values. Figure 6 (c) shows the detection results of $agent_2$ when $agent_1$'s policy is fixed with cooperation degree 1. We can see that the true cooperation degree of $agent_2$ can be easily obtained by fitting a linear curve between the predicted and true values. Thus for each policy of $agent_1$ in Π_1, we can fit a linear function to evaluate the true cooperation degrees of $agent_2$. During online learning, when $agent_1$ uses policies of varying cooperation degrees, it firstly chooses the function whose corresponding policy is closest to the used policy, and then computes the cooperation degree of $agent_2$.

Note that the opponent may take polices with cooperation degree between 0 and 1 whereas its cooperation degree policy is fixed as 0 or 1 during training which will be introduced in Section 4.6. This shows that our detection network can generalize to detect policies with varying cooperation degrees. Even for previously unseen policies which are not generated by pi_c and pi_d, they can be mapped to proper cooperation degrees. For example, a defection policy is always to emit a beam whereas the emitting frequency varies for different cooperation degrees. Our detection network can learn aforementioned key patterns of opponents' behaviors and thus accurately predict the different levels of cooperative degrees.

4.6 Performance under Self-Play

Next, we evaluate the learning performance of our approach under self-play. Each agent starts with cooperation or defection policy. Agents converge to full cooperation for all initial cases. Due to space limitation, we only present the results in the case when both agents start with a defection policy, which is the most challenging one (Figure 7) . It is more likely for them to model each other as defecting and thus both play defect thereafter. Our approach enables agents to detect the cooperation tendency of their opponents from sequential actions and eventually converge to cooperation.

4.7 Playing with Opponents with Changing Strategies

Now, we evaluate the performance against opponents, switching between cooperation and defection. In the Fruit Gathering game we vary it in the range of $[100, 500]$ at the interval of 25. Due to space limitation , only one set of results for the game are shown in Figure 8 and Figure 9. Similar results can be observed for other values of N_{change}. We can observe that the average rewards of $agent_1$ are higher than its rewards using π^c. And the social welfare (the total rewards) is higher than that using π^d. This indicates that we can prevent agents from being exploited by defecting opponents and also seek mutual cooperation. By comparing the results for different values of N_{change}, we find that the detection accuracy decreases when the opponent changes its policy quickly. Since the agent requires observing several episodes to detect the cooperation degree of the opponent, when the agent realizes its opponent changes the policy and adjusts its policy, the opponent may change the policy again. This problem becomes less severe when T is comparatively large.

5 CONCLUSIONS

In this paper, we investigate multiagent learning problem in large-scale PD games by leveraging the recent advance of DRL. A deep multiagent RL approach is proposed towards mutual cooperation in SPD games to support adaptive end-to-end learning. As a step towards solving multiagent learning problem in large-scale environments, we believe there are many interesting questions remaining for future work. One worthwhile direction is how to generalize our approach to other classes of large-scale multiagent games. Generalized policy detection and reuse techniques should be proposed, e.g., by extending existing approaches in traditional reinforcement learning contexts [13–15].

ACKNOWLEDGMENTS

The work is supported by the National Natural Science Foundation of China under Grant No.: 61702362, and Special Program of Artificial Intelligence of Tianjin Municipal Science and Technology Commission (No.:17ZXRGGX00150), and Special Program of Talents Development for High Level Innovation and Entrepreneurship Team in Tianjin.

REFERENCES

[1] R. Axelrod. The evolution of cooperation, 1984.
[2] D. Banerjee and S. Sen. Reaching pareto-optimality in prisoner's dilemma using conditional joint action learning. *Autonomous Agents and Multi-Agent Systems*, 15(1):91–108, 2007.
[3] L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, And Cybernetics-Part C: Applications and Reviews, 38 (2), 2008*, 2008.
[4] J. W. Crandall and M. A. Goodrich. Learning to teach and follow in repeated games. In *AAAI workshop on Multiagent Learning*, 2005.
[5] J. W. Crandall. Just add pepper: extending learning algorithms for repeated matrix games to repeated markov games. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 399–406, 2012.
[6] S. Damer and M. L. Gini. Achieving cooperation in a minimally constrained environment. In *AAAI*, pages 57–62, 2008.
[7] M. Elidrisi, N. Johnson, M. Gini, and J. Crandall. Fast adaptive learning in repeated stochastic games by game abstraction. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1141–1148, 2014.
[8] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual multi-agent policy gradients. *arXiv preprint arXiv:1705.08926*, 2017.
[9] J. Foerster, N. Nardelli, G. Farquhar, P. Torr, P. Kohli, S. Whiteson, et al. Stabilising experience replay for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1702.08887*, 2017.
[10] Jakob N Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. *arXiv preprint arXiv:1709.04326*, 2017.
[11] J. Hao and H. Leung. Introducing decision entrustment mechanism into repeated bilateral agent interactions to achieve social optimality. *Autonomous Agents and Multi-Agent Systems*, 29(4):658–682, 2015.
[12] H. He, J. Boyd-Graber, K. Kwok, and H. Daumé III. Opponent modeling in deep reinforcement learning. In *International Conference on Machine Learning*, pages 1804–1813, 2016.
[13] P. Hernandez-Leal and M. Kaisers. Towards a fast detection of opponents in repeated stochastic games. In *The Workshop on Transfer in Reinforcement Learning*, 2017.
[14] P. Hernandez-Leal, B. Rosman, M. E. Taylor, L. E. Sucar, and E. M. D. Cote. A bayesian approach for learning and tracking switching, non-stationary opponents. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 1315–1316, 2016.
[15] P. Hernandez-Leal, M. E. Taylor, B. Rosman, L. E. Sucar, and E. M. D. Cote. Identifying and tracking switching, non-stationary opponents: a bayesian approach. In *Multiagent Interaction without Prior Coordination Workshop at AAAI*, 2016.

[16] J. Hu and M. P. Wellman. Nash q-learning for general-sum stochastic games. *Journal of machine learning research*, 4(Nov):1039–1069, 2003.

[17] J. Z. Leibo, V. Zambaldi, M. Lanctot, J. Marecki, and T. Graepel. Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pages 464–473, 2017.

[18] 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. *arXiv preprint arXiv:1509.02971*, 2015.

[19] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the eleventh international conference on machine learning*, volume 157, pages 157–163, 1994.

[20] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*, 2017.

[21] P. Mathieu and J. Delahaye. New winning strategies for the iterated prisoner's dilemma. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1665–1666, 2015.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[23] M. Nowak and K. Sigmund. A strategy of win-stay, lose-shift that outperforms tit-for-tat in the prisoner's dilemma game. *Nature*, 364(6432):56–58, 1993.

[24] J. Perolat, J. Z. Leibo, V. Zambaldi, C. Beattie, K. Tuyls, and T. Graepel. A multi-agent reinforcement learning model of common-pool resource appropriation. *arXiv preprint arXiv:1707.06600*, 2017.

[25] P. Stone and M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[26] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[27] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[28] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

[29] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.