Foundations of Deep Learning: Project Paper

Yang Hu @ SEAS, Harvard

5/3/2023

Topic: Learn to Cooperate in Iterative Social Dilemmas

1 Introduction

In the past few years, sequential decision-making in multi-agent systems has become an increasingly popular field of research. Enabled by the recent exciting advancements of deep neural networks, Multi-Agent Reinforcement Learning (MARL) algorithms have shown their strengths in achieving human-level performance in various tasks ranging from games like StarCraft II [1] and Google Football [2] to industrial applications including smart grid [3] and inventory control [4]. In short, the goal of MARL algorithms is to let autonomous agents who are incentivized by individual rewards interact with a common environment, and learn policies to complete certain tasks and/or optimize over certain performance metrics.

Learning to cooperate has been a crucial yet hard problem to solve in multi-agent systems (and particularly, for MARL algorithms). Although it is always possible to promote cooperative behavior via reward-sharing or additional training signals [5, 6], it is still unknown and interesting enough to study whether there exist algorithms to encourage spontaneous cooperative behavior among autonomous without extra design. It is worth pointing out that cooperation is not a pattern that naturally emerges in multi-agent systems, since in the standard setting MARL agents are designed to maximize their own rewards without knowledge of others' rewards, and popular value-based RL algorithms tend to converge to (pure-strategy) Nash equilibria in a wide range of games [7, 8].

The difficulty of discovering cooperative policies largely depends on the underlying game structure. In identical-interest games, cooperation could emerge very naturally since individual rewards are identical; in zero-sum games, cooperation does not make sense at all since an agent's extra reward is exactly another agent's loss. Therefore, an ideal benchmark should lie between the extremes. Iterative social dilemmas are a family of games that are suitable for showing the difficulty of cooperation by nature. They expose the ubiquitous conflict between collective and individual rationality, where cooperation leads to better outcomes for all, but parasitic strategies like free-riding jeopardizes the stability of alliance. Iterative Prisoner's Dilemma, the repeated version of the best-known game-theoretic example, is among these social dilemmas. A lot of efforts have been made to solve the problem [9–13], yet the solutions are far from satisfactory.

The reason why the problem of learning to cooperate is important is two-fold. On the one hand, cooperative behavior is actually expected in many real-world multi-agent systems, and thus developing algorithms that promote spontaneous cooperation can assist better decision-making in practice. On the other hand, a detailed study into the problem (probably in combination with

theoretical analysis) also helps to reveal the intrinsic difficulty of cooperation and the fundamental limit of MARL algorithms, which is still an unknown area to the best of our knowledge.

Related work. There is a wide range of empirical work in the field of deep MARL that tries to promote cooperation among agents in the identical-interest setting [14], the challenges of which mainly lie in the complexity of the tasks and a lack of training signals (most often due to sparse rewards or difficulty in decomposing rewards). For example, a line of work [15, 16] focuses on appropriately decomposing the global reward to assign credit to each agent corresponding to their actual contribution to the social welfare so as to encourage cooperation. Another line of work focuses on the role of communication in cooperation, trying to design sparse communication protocols [17, 18] or even enable agents to learn such protocols [19–21] so as to let them effectively and efficiently communicate. There are also a few works studying how to introduce additional training signals to improve the efficiency and robustness of learning [5, 6], or focusing on independent learning agents without centralized coordination [22, 23]. However, despite the wide spectrum of topics they have covered and the level of practical success they have achieved, the majority of these works are specifically designed for a narrow scope of tasks which could be hard to generalize to generic Markov games, nor are they compatible with the game-theoretic setting or analysis.

From a theoretical perspective, our understanding of MARL is unexpectedly poor. People have been aware of the unusual difficulty for multiple agents to display cooperative behavior for a long time [24, 25], but up till now there are only restricted or over-simplified attempts to break the curse of cooperation [26, 27] or to explain the intrinsic difficulties of cooperative MARL.

It is also worth mentioning that, as the birth-giver and the most active user of game theory, people from the economics community are also interested in analyzing multi-agent cooperation in a game-theoretic sense, though they tend to take quite different approaches. For example, economists may analyze social dilemmas in order to explain (rather than predict) human behavior, and there is a full line of work focusing on the modeling of collective behavior patterns and the identification of parameters therein using a few representative social dilemmas [28–31]. It is not until quite recently that they start to touch the computational aspect of social dilemmas, among which [32] claims to enable learning agents to spontaneously follow stable cooperative policies in finite time using the simple ε-greedy Q-learning algorithm, which seems just too good to be true.

The aim of this project. In this project, we would like to make a few first attempts in designing an efficient algorithm that efficiently finds a "cooperative policy" of a repeated game, and empirically evaluates the algorithms in a few standard social dilemma settings.

2 Problem Formulation

Instead of the generic Markov Game setting that MARL algorithms usually work with, we want to simplify the setting as much as possible. Hence we will focus on such repeated games where each stage game is a normal-form game [33].

Definition 2.1. A *normal-form game* can be described as a 3-tuple $(n, \{\mathcal{A}_i\}_{i=1}^n, \{r_i\}_{i=1}^n)$, where:

- n is the number of *players* ("players" and "agents" will be used interchangeably);
- $\mathcal{A}_i$ is the *action space* of player $i \in [n]$, and the *joint action space* is $\mathcal{A} := \prod_{i=1}^n \mathcal{A}_i$;

- $r_i : \mathcal{A} \to \mathbb{R}$ is the *reward* (*utility*) *function* of player $i \in [n]$ over ant possible joint action.

Definition 2.2 (repeated game)**.** A *repeated game* with normal-form stage games can be described as a 3-tuple (G, T, γ), where:

- $G = (n, \{\mathcal{A}_i\}_{i=1}^n, \{r_i\}_{i=1}^n)$ is a normal-form game, referred to as the *stage game*;
- $T \in \mathbb{R} \cup \{\infty\}$ is the *horizon* of the game;
- $\gamma \in (0, 1]$ is the *reward discount factor*.

The *cumulative discounted reward* of player $i \in [n]$ is defined as $R_i(a^{1:T}) := \sum_{t=1}^T \gamma^{t-1} r_i(a^t)$.

We point out that the terminologies from the RL literature are preferred here over those from game theory literature, though the repeated game defined above is stateless by itself and thus RL algorithms cannot be directly applied.

Definition 2.3 (policy)**.** A *policy* of player $i \in [n]$ is a collection $\{\pi_i^t \mid t \in [T]\}$, where $\pi_i^t : \mathcal{O}_{t-1} \to \Delta(\mathcal{A}_i)$ is a mapping from the history of the game to a (potentially mixed) strategy of the next stage game. The *joint policy* is defined as $\pi^t := (\pi_1^t, \ldots, \pi_n^t)$, and $\pi := (\pi^1, \ldots, \pi^T)$. For repeated games with normal-form stage games, we shall regard $\mathcal{O}_{t-1} = \mathcal{A}^{t-1}$ (i.e., all historical joint actions).

We shall further overload the notation of rewards a bit to include expected rewards of mixed strategies. Specifically, player i's reward with respect to a joint policy π^t and history $a^{1:t-1}$ is

$$r_i(\pi^t \mid a^{1:t-1}) := \sum_{a \in \mathcal{A}} \pi^t(a \mid a^{1:t-1}) r_i(a), \tag{1}$$

and the cumulative discounted reward of player $i \in [n]$ with respect to π is

$$R_i(\pi) := \sum_{t=1}^T \gamma^{t-1} r_i(\pi^t \mid a^{1:t-1}). \tag{2}$$

Note that the reward of any player i also depends on the policies of the other players.

2.1 Two Representative Social Dilemmas

In this section, we formally define two social-dilemma-style stage games that we will be mainly working with. These games are especially well-known and well-studied in economics literature.

Iterated Prisoner's Dilemma (IPD). The stage game of IPD, which is known as the *Prisoner's Dilemma* (*PD*), is perhaps the most prominent game-theoretic setting. In literature, PD may refer to a family of two-player matrix games, where actions are called *cooperate* (*C*) and *deviate* (*D*), and the payoff matrix should satisfy certain constraints to guarantee a unique Nash Equilibrium (D, D). In this project, we will use the following payoff matrix

	C	D
C	$(3, 3)$	$(0, 5)$
D	$(5, 0)$	$(1, 1)$

$$\begin{array}{c} \\ 0 \\ 1 \end{array} \begin{array}{c} \begin{array}{cc} 0 & 1 \end{array} \\ \begin{bmatrix} * & * \\ * & * \end{bmatrix} \end{array}$$

Note that PD is not a zero-sum game. For simplicity, label C as action 0, and D as action 1. Further, as we will see below, we shall use the joint action of the last time step as the *state* of the

game, and we shall only consider *Markovian policies* that is a function of the state. Under these conventions, any deterministic policy for PD can be written as a 2-by-2 matrix, as shown above.

Iterated Bertrand competition. In the economics literature, Bertrand competition is a classic model for modeling monopoly and duopoly behavior in the market. Specifically, there are n firms and an external source (labelled as firm 0) providing homogeneous goods. Each firm is allowed to price its own product at p_i. For a fixed price profile p, the market share of firm i is

$$q_i(p) = \frac{\exp\left(\frac{a_i - p_i}{\mu}\right)}{\sum_{j=1}^{n} \exp\left(\frac{a_i - p_i}{\mu}\right) + \exp\left(\frac{a_0}{\mu}\right)}, \quad i = 1, 2, \ldots, n, \tag{3}$$

where a_i is the *quality index* (a.k.a. "vertical differentiation") of the product, and μ is a scaling factor ("hoizontal differentiation"). Note that here action is defined as the price of the good. The reward of firm i is then calculated by $r_i(p) = (p_i - c_i)q_i$, where c_i is the marginal cost of production. It is straight-forward to see that the Nash equilibrium is symmetric $(p_N, \ldots, p_N)$ for some *Nash price* p_N, and is thus given by solving

$$\left.\frac{\partial r_i(p)}{\partial p_i}\right|_{p=(p_N,\ldots,p_N)} = 0; \tag{4}$$

on the other hand, the optimal *monopoly price* p_M (i.e., the optimal price to set if one can decide a uniform price for all firms) is given by solving

$$\frac{\mathrm{d} r_i(p_M, \ldots, p_M)}{\mathrm{d} p_M} = 0. \tag{5}$$

It can be shown that, in the non-degenerative case we have $p_M > p_N > 0$.

For consistency, in this project we will focus on the 2-player case and use the following parameters: $a_0 = 1$, $\mu = 0.5$, $a_1 = a_2 = 2$, $c_1 = c_2 = 1$, in which p_M and p_N are

$$p_N \approx 1.61338 \implies r_i(p_N, p_N) \approx 0.11338, \tag{6a}$$

$$p_M \approx 1.73153 \implies r_i(p_M, p_M) \approx 0.11576. \tag{6b}$$

Further, the action space is discretized as the 15 equipartition points of the interval $[p_N - \xi(p_M - p_N), p_M + \xi(p_M - p_N)]$, where $\xi = 0.1$. In the discretized action space, p_N is approximately the 2^{nd} action, while p_M is approximately the 14^{th} action. Note that the setting here is exactly the same as the one used in [32] since we will try to reproduce the results therein.

2.2 Preliminaries on RL

We will rely on RL algorithms to enable agents to learn. To model the repeated game as an Markov game $(\mathcal{S}, \mathcal{A}, \mathbb{P}, \{r_i\})$, we need to define the state space $\mathcal{S}$ and the transition kernel $\mathbb{P}$. We have pointed out that the repeated game is *stateless* itself, yet the (finite) state space can be viewed as the memory of the agent that stores information from the history. Under this intuition, we will use the joint action space as the state space (i.e., $\mathcal{S} := \mathcal{A} \times \mathcal{A} \cup \{\perp\}$), where $\perp$ denotes the initial state, and the transition kernel simply assigns the next state as the last joint action (i.e.,

$\mathbb{P}(s' \mid s, a) = \mathbb{1}\{s' = a\}$. The initial state is only introduced for completeness and will not be used.

The algorithms designed in this project are all *value-based*. For a fixed joint policy π, the *Q-function* of agent i with respect to π is defined as

$$Q_i^\pi(s, a) = r_i(a) + \gamma \mathbb{E}_{s' \sim \mathbb{P}(\cdot|s,a)} \left[\max_{a' \in \mathcal{A}} Q_i^\pi(s', a') \right], \tag{7}$$

and is also called the *Q-table* in the tabular case. For a fixed π, the following *value iteration*

$$\hat{Q}_i(s, a) \leftarrow \hat{Q}_i(s, a) + \alpha \left[r_i + \gamma \max_{a' \in A} \hat{Q}_i(s', a') - \hat{Q}_i(s, a) \right] \tag{8}$$

converges to the actual Q-function Q_i^π.

3 LSTM-Based Agent

The key challenge for autonomous agents to learn to cooperate partly lies in the difficulty to predict whether their opponents are willing to cooperate based on historical data. Intuitively, the agents shall extract some "latent state" from the history to represent such willingness.

In this section, we will present a novel design of learning agents that incorporates an LSTM-based "actor" module to utilize the history moves of the opponent so as to assist the prediction of future values. Experimental results in the IPD setting will be shown to evaluate the design.

3.1 Agent Design

The structure of the LSTM-based agent is shown below in Figure 1. It is similar to the classical actor-critic structure in which we have an actor module that generates action from state and an critic module that evaluates the value of a state. However, since the games we are dealing with are discrete (or at least can be practically discretized) by nature, the critic module is not necessary, and we shall simplify the design by replacing it with a Q-table. To avoid extra large state space, we still use the joint action of the last step as the state.

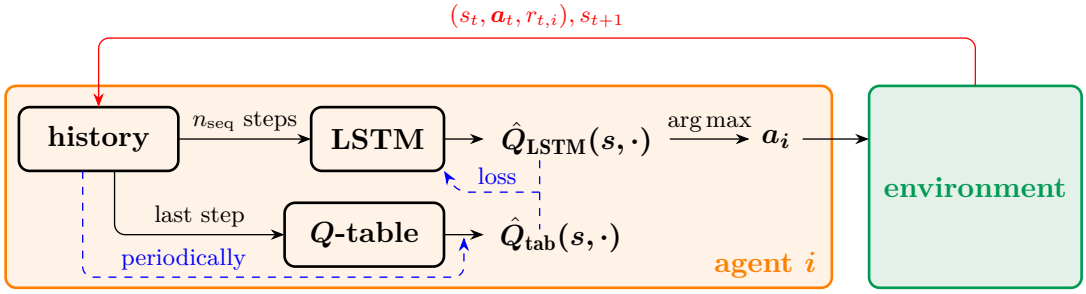

Figure 1 Structure of the LSTM-based agent.

The Long Short-Term Memory (LSTM) network is incorporated into the agent to extract information from history. Specifically, it takes the last n_{seq} steps and predict the Q-value for the current state. Note that the notation $\hat{Q}_{\text{LSTM}}(s, \cdot)$ is symbolic, in that the network does not actually

compute the whole $\hat{Q}_{\text{LSTM}}$ table in execution. Then the agent determines the next action by

$$a_{i,t} \leftarrow \arg\max_{a_i \in A_i} \hat{Q}_{\text{LSTM}}(s_t, a_i). \tag{9}$$

The *lookback window* n_{seq} of the LSTM network, i.e. the number of past steps the LSTM network observes (whenever available), is a tunable hyper-parameter.

The LSTM network is trained against a surrogate for the ground-truth Q value, which is the "critic" module that performs Q-learning independently from the "actor" module. Of course, it is not completely decoupled from the "actor" module, since the latter generates the trajectory. The loss of the LSTM network is then defined as

$$\mathcal{L}(\boldsymbol{\theta}) := \sum_{\tau=t-B}^{t} \left\| \hat{Q}_{\text{LSTM}}(s_\tau, \cdot; \boldsymbol{\theta}) - \hat{Q}_{\text{tab}}(s_\tau, \cdot) \right\|_2^2, \tag{10}$$

where B is the *batch size*, and $\hat{Q}_{\text{tab}}$ is the Q-table maintained by the "critic" module. Note that here we may train in batches to avoid thrashing due to constantly changing policies.

3.2 Experimental Results

In the experiment, the LSTM agent is trained either against each other or against a player with a fixed strategic policy. Since the training is time-consuming as the network scales up, experiments are only conducted in the IPD setting. Most of the hyper-parameters are fixed across all experiments (which can be found in the appendix), while the only hyper-parameters of interest here are the number of layers $L \in \{1, 2\}$, the horizon $n_{\text{seq}} \in \{1, 2, 4, 8, 16, 32\}$, and the batch size $B \in \{1, 32\}$.

Table 1 Terminal policies of LSTM-agents (self-play) with different hyper-parameters.

Setting	n_{seq}	1	2	4	8	16
1-layer $B = 1$	**Agent 1**	$\begin{bmatrix}1&1\\0&1\end{bmatrix}$	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$
	Agent 2	$\begin{bmatrix}1&1\\0&1\end{bmatrix}$	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$
2-layer $B = 1$	**Agent 1**	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$	$\begin{bmatrix}0&0\\0&1\end{bmatrix}$	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$	$\begin{bmatrix}0&0\\1&1\end{bmatrix}$	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$
	Agent 2	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$	$\begin{bmatrix}0&0\\0&1\end{bmatrix}$	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$	$\begin{bmatrix}0&0\\0&1\end{bmatrix}$	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$
2-layer $B = 32$	**Agent 1**	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$
	Agent 2	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$	$\begin{bmatrix}0&1\\1&1\end{bmatrix}$

LSTM-agent vs. LSTM-agent. We first evaluate the agents trained in a self-play manner. The learned policies are shown as matrices (as stated before) in the above Table 1. Detailed training curves are deferred to the appendix due to limited space.

It can be observed that, in most executions, the agents simply converge to the Nash equilibrium, where the Q-tables are very likely to converge to the case where they will only cooperate if both of them have cooperated in the last step. A detailed examination of the trajectories shows that a substantial fraction among these non-cooperative agents actually display some transient cooperative behavior along the trajectory, yet their Q-tables gradually drift away towards deviation.

Only two of the executions lead to somewhat cooperative policies upon termination. However, repeated experiments with the same hyper-parameters do not reproduce the results in most of the cases. Therefore, it seems that stochasticity is dominant in the emergence of different behavior patterns, while the tuning of hyper-parameters only have minor effects. Overall, the emergence of cooperative behavior is scarce and volatile.

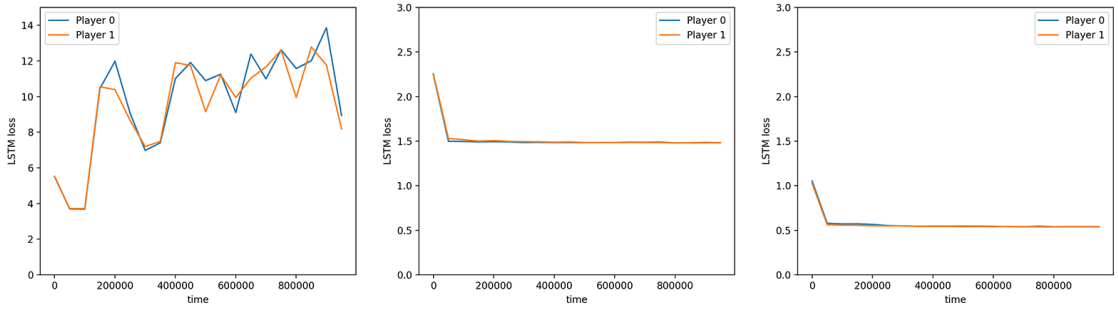

Figure 2 Training losses of three LSTM networks along trajectory.

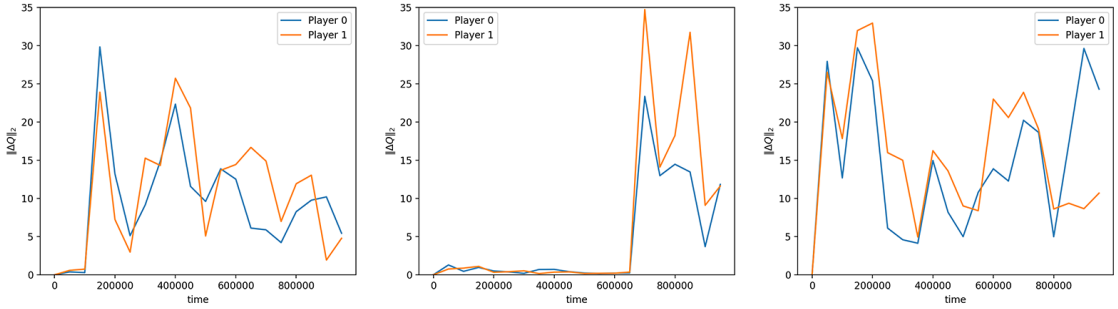

Figure 3 Q-table differences of three agents along trajectory.

Further, we shall examine the convergence of the LSTM-based agents in terms of LSTM training loss (see Figure 2) and the difference of consecutive Q-tables (see Figure 3; Q-tables are logged every 5×10^4 steps). It can be seen from the figures that the training loss of the LSTM network might keep oscillating around or stop decreasing after a very short length of time, while the update in Q-functions might be drastic throughout the training procedure. These signs show that the current training scheme do not lead to satisfactory convergence performance. This might be attributed to the nonstaionarity of the multi-agent system and insufficient exploration.

LSTM-agent vs. Periodic Tit-for-tat. The results in the previous section are really frustrating, which makes us doubt whether the LSTM network is actually capable of learning patterns to assist cooperation. To verify this, we try to train our LSTM agents against a player with a

well-known strategic fixed policy known as *P-periodic tit-for-tat*, which is an automata policy that punishes each deviation of the opponent by P deviations in a row.

The learned polices are shown as matrices in Table 2 below.

Table 2 Terminal policies of LSTM-agents (vs. tit-for-tat) with different hyper-parameters.

Period P	**1**	**2**	**4**	**8**	**16**	**32**
$n_{\text{seq}} = 8$	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$
$n_{\text{seq}} = 32$	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

It can be observed that, when the period P of the tit-for-tat opponent is small, the LSTM-agent almost surely learns how to cooperate with them (and actually in a very timely way, as shown in the learning curves in appendix). However, when the period P becomes a bit larger, the LSTM-agent immediately fails, and in most of the cases it even fails to learn to keep cooperating at state $(0, 0)$. It is also clear that the lookback window n_{seq} plays only a minor part in the ability of finding cooperative policies.

The above results further confirms the conjecture that the poor performance is likely to be owe to a lack of exploration. In fact, when playing with tit-for-tat agent with longer period, the LSTM-agent with a long enough "memory" (i.e., lookback window) should be able to discover the pattern given sufficient exploration. However, when P is large, it takes much longer to visit all possible combinations of P states along the trajectory — suppose most of the state combinations different from the converged behavior is visited via ε-exploration, since the probability is only ε^P, an average time of $O(\varepsilon^{-P})$ is expected, and there are $2^{O(P)}$ of them! This definitely poses a fundamental challenge for agents to learn about the long-term behavioral patterns of their opponents.

4 Q-Learning Agent

It is proposed in [32] that we shall use the naive ε-greedy Q-learning agent to achieve cooperative policies in the IBC setting. In addition to the standard learning dynamics in Algorithm 1, [32] also proposes a decaying exploration rate $\varepsilon_t = \exp(-\beta t)$ and a specific initialization scheme of the Q-tables $Q_i^\star(s, a_i) = \frac{\sum_{a_{-i}} r_i(a_i, a_{-i})}{(1-\gamma)|\mathcal{A}_{-i}|}$ (i.e., the Q-table for the uniformly random joint policy).

Algorithm 1 ε-greedy Q-learning algorithm

1: Initialize Q-table $Q(\cdot, \cdot) \leftarrow 0$, $s \leftarrow s_{\text{init}}$.
2: **for** time $t \leftarrow 1, 2, \ldots, T$ **do**
3: Take action $a_t \leftarrow \begin{cases} \arg\max_{a \in A} Q(s, a) & \text{with probability } (1 - \varepsilon_t) \\ \text{Unif}(A) & \text{with probability } \varepsilon_t \end{cases}$.
4: Observe reward r and next state s'.
5: Perform update $Q(s, a) \leftarrow Q(s, a) + \alpha\big(r + \gamma \max_{a' \in A} Q(s', a') - Q(s, a)\big)$.
6: $s \leftarrow s'$.

We repeat the experiment with exactly the same hyper-parameters as in [32] (see the appendix

for details), and surprisingly the result is actually reproducible in a very stable way. In most of the executions, the joint terminal policy of the agents will keep the trajectory at a fixed point around $(9, 9)$, roughly the same as reported in the paper (a price below monopoly price p_M, yet way above the duopoly Nash price p_N). However, when we examine the Q-table carefully (see Figure 4 below for a sample), we see no trace of any threat-based policies. Further, the learning curves (see the appendix for details) becomes flat (marking the de facto end of learning) very soon. These signs suggest that the convergence is fake and the overall performance of the algorithm is unsatisfactory.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	3	1	3	0	9	5	5	3	0	6	12	2	1	4	10
1	0	3	3	3	6	4	3	4	1	1	0	9	1	9	4
2	3	3	4	1	2	10	2	1	9	1	2	1	10	5	11
3	0	0	0	5	6	3	3	1	6	5	5	0	1	4	3
4	7	3	0	2	5	1	6	1	6	10	11	9	2	6	8
5	1	0	0	0	1	8	6	9	1	1	1	3	6	4	5
6	0	0	3	2	1	5	4	1	1	3	10	5	1	1	2
7	0	0	3	0	5	7	5	2	1	5	4	8	10	0	8
8	1	1	2	2	10	0	6	2	11	1	5	0	7	6	1
9	1	1	0	8	4	5	2	1	0	**9**	9	6	8	3	0
10	0	3	5	2	2	0	11	1	1	1	7	7	3	3	0
11	5	0	2	0	6	6	9	5	0	2	1	8	3	5	9
12	9	4	7	0	9	1	6	9	5	7	6	2	1	6	6
13	1	3	7	2	4	5	1	0	3	1	0	0	5	6	6
14	3	11	4	5	5	4	9	7	4	7	7	3	4	1	8

Figure 4 A sample Q-table of the Q-learing agent upon termination.

4.1 Ablation Studies

To figure out why the algorithm shows a misleading sign of convergence, we perform some ablation studies with regard to the algorithm.

The exploration rate ε_t. After some calculation, it is obvious that the exploration rate ε_t is decaying too fast. In fact, the exploration rate drops to e^{-5} after 10^6 steps, which basically means no exploration. The total number of explorations and the change in the Q-tables of both agents are shown below in Table 3.

Table 3 Number of explorations and change in $\boldsymbol{Q}$-tables along a trajectory.

time ($\times 10^5$)	$[0, 5]$	$[5, 10]$	$[10, 15]$	$[15, 20]$
# explorations	74908	5	0	0
$\|\boldsymbol{\Delta Q_1}\|_2$	0.1327	4.28×10^{-10}	0	0
$\|\boldsymbol{\Delta Q_2}\|_2$	0.0767	2.93×10^{-11}	0	0

To correct this erroneous behavior, we shall set a lower bound on ε_t, i.e., decaying the exploration rate as $\varepsilon_t = \min\{\exp(-\beta t), \varepsilon_{\min}\}$. After correction, the algorithm performs in a significantly different way. As can be seen in the appendix, the Q-table basically fails to converge and the best move keeps drifting around from time to time. Therefore, it is reasonable to conclue that the reason for the abnormal cooperative behavior in [32] is that the agents are not exploring enough.

Initialization scheme. Another suspicious part in the proposed algorithm is the specific initialization scheme for the Q-table. Initializing to the exact Q-function of uniformly random joint policy

might seem reasonable at first, but when we examine the initialized values, it turns out that the terminal price is very close to the maximizer of the initial $Q^\star$-values, which makes it significantly easier for agents to explore that state in the beginning. Then after a short time that specific state becomes dominant, while the Q-values of all other actions are still far from the true value and it takes much more time for the Q-function to really converge through occasional explorations.

To verify the influence of Q-table initialization schemes, we conduct the following experiment (here $Q^\star$ is defined above, and Q_{rand} refers to the Q-function with Unif($[0,1]$)-entries). The terminal policies of the Q-learning agents are shown below in Table 4.

Table 4 Terminal policies of Q-learning agents under different initialization schemes.

initialization	$(Q^\star, Q^\star)$	$(Q^\star, 0)$	$(0,0)$	$(2.5Q_{\text{rand}}, 2.5Q_{\text{rand}})$	$(10Q_{\text{rand}}, 10Q_{\text{rand}})$
unbounded ε_t	$(9,9)$	$(7,4)$	$(5,5)$	$(9,8)$	$(11,12)$
bounded ε_t	—	$(6,6)$	—	$(6,8)$	$(7,8)$

It can be concluded that, with larger initial values in the Q-tables, agents become more likely to explore the higher prices first, and the frequently visited states soon become dominant afte a few updates. Then it will take a long time before sufficiently many exploration steps happen to correct the learned Q-values, and this is impossible if the exploration rate decays too fast without lower bounds.

4.2 ε-Greedy Works for IPD

Still, the proposed Algorithm 1 does work for simpler games, e.g. the IPD setting. With exploration rate ε_t lower bounded by 0.01, in almost all executions the agents discover the following joint policy: cooperate when the last joint actions are identical, and deviate if they are different. This is, of course, a kind of threat-based policy we would like to see in a broader range of settings.

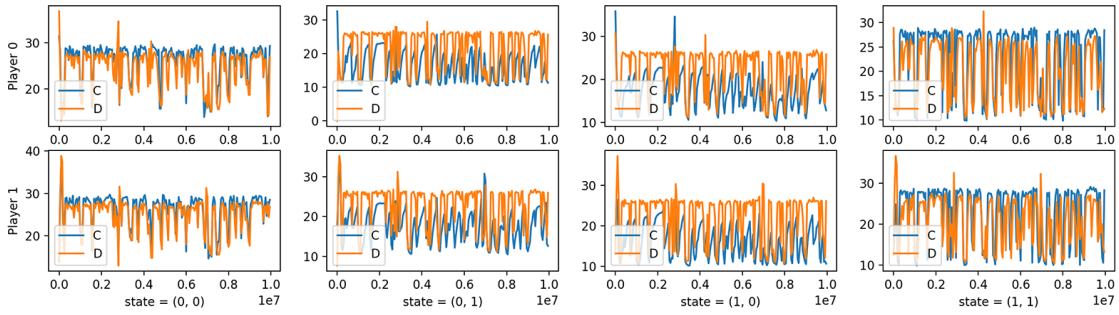

Figure 5 Q-functions along a typical trajectory in the IPD setting.

5 Conclusion

It is a little frustrating that our initial attempts to enable agents to learn to cooperate almost completely failed during the course of this project. Still, it reflects the intrinsic difficulty of this problem. The key challenge, as it seems now, still roots from insufficient exploration in a highly nonstationary environment. Therefore, a possible way out is to think about how to encourage efficient and effective exploration without centralized coordination.

References

[1] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

[2] Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zając, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, et al. Google research football: A novel reinforcement learning environment. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34(04), pages 4501–4510, 2020.

[3] Martin Roesch, Christian Linder, Roland Zimmermann, Andreas Rudolf, Andrea Hohmann, and Gunther Reinhart. Smart grid for industry using multi-agent reinforcement learning. *Applied Sciences*, 10(19):6900, 2020.

[4] Afshin Oroojlooyjadid, MohammadReza Nazari, Lawrence V Snyder, and Martin Takáč. A deep Q-network for the beer game: Deep reinforcement learning for inventory optimization. *Manufacturing & Service Operations Management*, 24(1):285–304, 2022.

[5] Tonghan Wang, Tarun Gupta, Anuj Mahajan, Bei Peng, Shimon Whiteson, and Chongjie Zhang. Rode: Learning roles to decompose multi-agent tasks. *arXiv preprint arXiv: 2010.01523*, 2020.

[6] Weixun Wang, Jianye Hao, Yixi Wang, and Matthew Taylor. Towards cooperation in sequential prisoner's dilemmas: a deep multiagent reinforcement learning approach. *arXiv preprint arXiv:1803.00162*, 2018.

[7] Junling Hu and Michael P Wellman. Nash Q-learning for general-sum stochastic games. *Journal of machine learning research*, 4(Nov):1039–1069, 2003.

[8] Jianqing Fan, Zhaoran Wang, Yuchen Xie, and Zhuoran Yang. A theoretical analysis of deep Q-learning. In *Learning for Dynamics and Control*, pages 486–489. PMLR, 2020.

[9] Valerio Capraro, Matteo Venanzi, Maria Polukarov, and Nicholas R Jennings. Cooperative equilibria in iterated social dilemmas. In *Algorithmic Game Theory: 6th International Symposium, SAGT 2013, Aachen, Germany, October 21-23, 2013. Proceedings 6*, pages 146–158. Springer, 2013.

[10] Nan Rong and Joseph Y. Halpern. Cooperative equilibrium: A solution predicting cooperative play. *arXiv preprint arXiv:1412.6722*, 2014.

[11] Christian Hilbe, Bin Wu, Arne Traulsen, and Martin A Nowak. Cooperation and control in multiplayer social dilemmas. *Proceedings of the National Academy of Sciences*, 111(46):16425–16430, 2014.

[12] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. *arXiv preprint arXiv:1702.03037*, 2017.

[13] Tom Eccles, Edward Hughes, János Kramár, Steven Wheelwright, and Joel Z Leibo. Learning reciprocity in complex sequential social dilemmas. *arXiv preprint arXiv:1903.08082*, 2019.

[14] Afshin Oroojlooy and Davood Hajinezhad. A review of cooperative multi-agent deep reinforcement learning. *Applied Intelligence*, pages 1–46, 2022.

[15] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[16] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *The Journal of Machine Learning Research*, 21(1):7234–7284, 2020.

[17] Sergio Valcarcel Macua, Aleksi Tukiainen, Daniel García-Ocaña Hernández, David Baldazo, Enrique Munoz de Cote, and Santiago Zazo. Diff-DAC: Distributed actor-critic for multitask deep reinforcement learning. *arXiv preprint arXiv:1710.10363*, 2017.

[18] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[19] Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 29, 2016.

[20] Emilio Jorge, Mikael Kågebäck, Fredrik D Johansson, and Emil Gustavsson. Learning to play guess who? and inventing a grounded language as a consequence. *arXiv preprint arXiv:1611.03218*, 2016.

[21] Jiechuan Jiang and Zongqing Lu. Learning attentional communication for multi-agent cooperation. *Advances in neural information processing systems*, 31, 2018.

[22] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. In *International conference on machine learning*, pages 1146–1155. PMLR, 2017.

[23] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

[24] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

[25] Ping Xuan, Victor Lesser, and Shlomo Zilberstein. Communication decisions in multi-agent cooperation: Model and experiments. In *Proceedings of the fifth international conference on Autonomous agents*, pages 616–623, 2001.

[26] Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Basar. Fully decentralized multi-agent reinforcement learning with networked agents. In *International Conference on Machine Learning*, pages 5872–5881. PMLR, 2018.

[27] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control*, pages 321–384, 2021.

[28] Graham Kendall, Xin Yao, and Siang Yew Chong. *The iterated prisoners' dilemma: 20 years on*, volume 4. World Scientific, 2007.

[29] Martin Beckenkamp. A game-theoretic taxonomy of social dilemmas. *Central European Journal of Operations Research*, 14(3):337–353, 2006.

[30] Valerio Capraro. A model of human cooperation in social dilemmas. *PloS one*, 8(8):e72427, 2013.

[31] Christian Hilbe, Arne Traulsen, and Karl Sigmund. Partners or rivals? strategies for the iterated prisoner's dilemma. *Games and economic behavior*, 92:41–52, 2015.

[32] Emilio Calvano, Giacomo Calzolari, Vincenzo Denicolo, and Sergio Pastorello. Artificial intelligence, algorithmic pricing, and collusion. *American Economic Review*, 110(10):3267–3297, 2020.

[33] Martin J Osborne and Ariel Rubinstein. *A course in game theory*. MIT press, 1994.

Appendix

A Experiment Set-ups

IBC game setting. The following parameters are used: $a_0 = 1$, $\mu = 0.5$, $a_1 = a_2 = 2$, $c_1 = c_2 = 1$. In this case, $p_N \approx 1.61338$, $r_i(p_N, p_N) \approx 0.11338$; $p_M \approx 1.73153$, $r_i(p_M, p_M) \approx 0.11576$. The action space is discretized as the 15 equipartition points of the interval $[p_N - \xi(p_M - p_N), p_M + \xi(p_M - p_N)]$, where $\xi = 0.1$.

Q-learning. The following hyper-parameters are used:

- the learning rate of Q-learning is $\alpha = 0.1$,
- the exploration rate decay factor is $\beta = 2 \times 10^{-5}$,
- the reward discount factor is $\gamma = 0.95$,
- the state space is set to be $\mathcal{S} = \mathcal{A} \times \mathcal{A}$.

LSTM network. The following hyper-parameters are used:

- the number of layers $L \in \{1, 2\}$,
- the dimension of hidden layer $d_{\text{hid}} = 32$,
- the lookback window $n_{\text{seq}} \in \{1, 2, 4, 8, 16, 32\}$,
- the batch size $B \in \{1, 32\}$,
- the learning rate $\eta_t = 0.01 \cdot 0.9995^t$,
- the output of LSTM passes through a fully connected network for a valid Q-vector prediction.

B Learning Curves for the IPD Setting

Figure 6 Learning curve of the LSTM agent in the IPC setting (1-layer, $B = 1$).

Figure 7 Learning curve of the LSTM agent in the IPC setting (2-layer, $B = 1$).

Figure 8 Learning curve of the LSTM agent in the IPC setting (1-layer, $B = 32$).

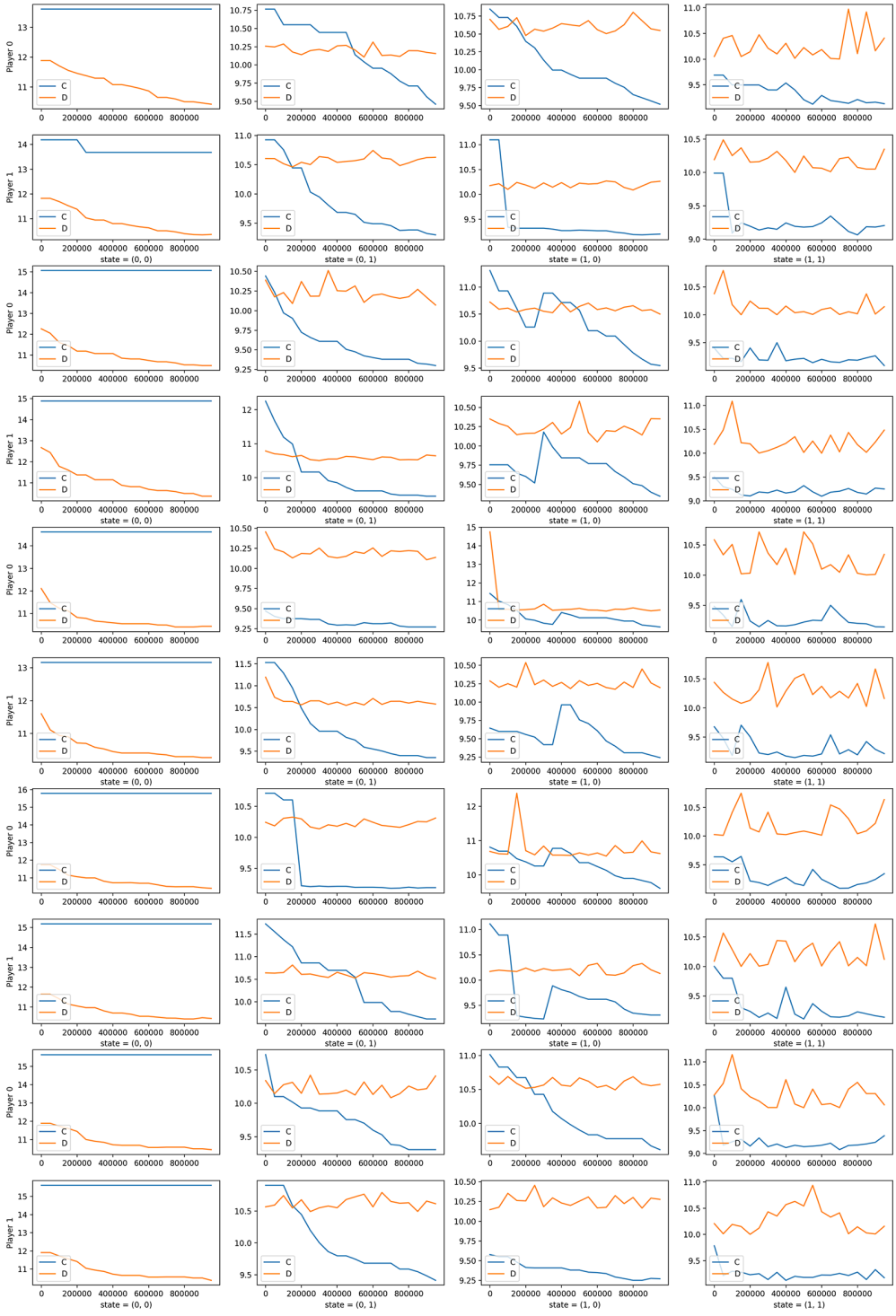

Figure 9 Learning curve of the LSTM agent in the IPC setting (2-layer, $B = 32$).

C Learning Curves for the IBC Setting

Figure 10 Learning curve of the Q-learning agent in the IBC setting (unbounded ε_t).

Figure 11 Learning curve of the Q-learning agent in the IBC setting (bounded ε_t).

Figure 12 Learning curve of the Q-learning agent in the IBC setting (bounded ε_t, batched).