Towards cooperation in learning games

Jesse Clifton [1] [2] Maxime Riché [1]

[1] Center on Long-Term Risk, [2] Polaris Research Institute

Abstract

Suppose that several actors are going to deploy learning agents to act on their behalf. What principles should guide these actors in designing their agents, given that they may have competing goals? An appealing solution concept in this setting is *welfare-optimal learning equilibrium*. This means that the learning agents should constitute a Nash equilibrium whose payoff profile is optimal according to some measure of total welfare (welfare function). In this work, we construct a class of learning algorithms in this spirit called *learning tit-for-tat (L-TFT)*. L-TFT algorithms maximize a welfare function according to a specified optimization schedule, and punish their counterpart when they detect that they are deviating from this plan. Because the policies of other agents are not in general fully observed, agents must infer whether their counterpart is following a cooperative learning algorithm. This requires us to develop new techniques for making inferences about counterpart learning algorithms. In two sequential social dilemmas, our L-TFT algorithms successfully cooperate in self-play while effectively avoiding exploitation by and punishing defecting learning algorithms.

1 Introduction

As the capabilities of machine learning systems improve, it is increasingly important to understand how to achieve cooperation between systems deployed by actors with competing goals. Moreover, because intelligent systems may continue to learn after they are deployed, designing agents who can cooperate requires accounting for ways their policies may change post-deployment. In this paper, we look at the setting where multiple learning agents are deployed to act on behalf of their respective principals.

There are three main goals of this work. The first is to articulate an appealing criterion for the design of learning agents, which is *welfare-optimal learning equilibrium*. The second is to present a general approach to designing learning agents who approximately fulfill this criterion, which we call *learning tit-for-tat (L-TFT)*. This framework is generic to agents who update their policies over time, rather than depending on a particular class of reinforcement learning algorithms. The third is to illustrate some of the strategic considerations which need to be addressed in the design of L-TFT agents. We do this by constructing an L-TFT agent using model-free reinforcement learning algorithms, and examining its performance in two sequential social dilemmas. This agent converges to welfare-optimal payoffs in self-play, while successfully avoiding exploitation and punishing defecting learning algorithms.

A learning equilibrium is simply a Nash equilibrium of a *learning game*, in which strategies are learning algorithms, and payoffs are the cumulative rewards associated with the profile of learning algorithms chosen by each principal [Brafman and Tennenholtz, 2003]. Principals will want to choose the learning agents they deploy strategically. This means that it is not enough for a learner to perform well against a specified set of other players (e.g., Shoham and Leyton-Brown 2008). It is also not enough that a learner converges to Nash equilibrium of the game which is being learned (e.g. Bowling and Veloso 2001, Conitzer and Sandholm 2007, Balduzzi et al. 2018, Letcher et al. 2018), as this does not guarantee that a player cannot benefit by deviating from that profile (i.e., deploying a different learning algorithm).

Like other sequential games, learning games will generally have many learning equilibria. Independent equilibrium selection on part of the principals may lead to poor performance. Thus, to guarantee socially optimal payoffs, principals need to coordinate when designing their agents. L-TFT addresses this problem via a *welfare function*. A welfare function measures the social value of a payoff profile; for instance, a simple welfare function is the sum of the principals' respective payoffs. An L-TFT agent learns a policy profile which optimizes this welfare function, and punishes its counterpart if it detects that they are not doing the same. Thus agents are incentivized to follow a learning algorithm which optimizes the welfare function.

Coordination can thus be achieved via coordination on a welfare function and on a class of learning algorithms for optimizing it.

In our experiments, learners observe only the actions taken by their counterpart, and not their counterpart's (stochastic) policy. This requires us to construct L-TFT agents who model their counterpart's learning schedule based on their actions and conduct a hypothesis test as to whether it matches a cooperative learning schedule. This in turn introduces incentives to exploit this testing procedure. We examine the performance of L-TFT against an algorithm designed specifically to exploit the way in which it detects defections, as well as its performance in self-play and against a naive selfish learner. Lastly, there is a strategic element in choosing the sensitivity with which L-TFT's hypothesis test detects defections. In Section 5.3, we examine this element in a learning game between an L-TFT agent and a potential exploiter choosing how cautious to be in deviating from a cooperative learning algorithm.

2 Related work

Learning equilibrium was first discussed (under that name) by Brafman and Tennenholtz [2003]. Their construction of learning equilibria involves evaluating each of a finite set of policies in a finite stochastic game, whereas the algorithms we focus on use incremental stochastic updates towards the welfare-optimal policy profile.

Our online learning setting complements the setting in which agents are trained fully offline and then deployed. This is the setting typically considered in applications of reinforcement learning to cooperation in social dilemmas. For instance, Peysakhovich and Lerer [2017], Lerer and Peysakhovich [2017], Wang et al. [2018] each develop methods in which punishment and cooperative policies are trained offline, and cooperation at deployment time is sustained by the threat of punishment.

However, as many agents may learn post-deployment, the strategic aspects of *learning* itself — rather than just policy selection — must be addressed. The manipulation of the learning of counterpart agents has been explored in the literature on opponent-shaping [Foerster et al., 2018, Letcher et al., 2018]. Perhaps the most similar to our approach is the variant of Foerster et al. [2018]'s learning with opponent learning awareness which uses opponent modeling (LOLA-OM). LOLA-OM uses supervised learning to estimate the policy followed by the other agent, and updates its own policy in a way that is intended to manipulate the direction in which the other player updates their policy. On the other hand, our L-TFT uses opponent modeling to infer whether the other agent is following a cooperative learning algorithm.

3 Preliminaries: Learning games

Consider a situation in which multiple agents interact with their environment over time, incrementally updating the policies which they use to choose actions. These updates are controlled by their learning algorithms, which are fixed by their principals before the agents begin interacting with the environment. It is helpful to distinguish between two different games that are being played here. There is the **base game**, which is defined by the dynamics of the environment, the reward functions of the agents, and the policies available to each agent. In this paper, the "base game" will be a stochastic game (described below). There is also the **learning game**, in which the principals simultaneously submit learning algorithms to be deployed in the base game environment, and attain payoffs equal to some measure of cumulative reward.

We work with stochastic games [Littman, 1994], a generalization of Markov decision processes to the multi-agent setting. We will assume only two players, $i = 1, 2$. We'll refer to player i's counterpart with the index j. In a stochastic game, players simultaneously take actions at each time $t \in \mathbb{N}$; observe a reward depending on the current state; and the state evolves according to a Markovian transition function. Write the state space as $\mathcal{S}$ and the action spaces as $\mathcal{A}_i$. Let the reward functions be mappings $r_i : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \to \mathbb{R}$, with the reward to player i at time t given by $R_i^t = r_i(S^t, A_1^t, A_2^t)$. Let H^t be the history of observations until time t. We will assume that both agents fully observe the state and each others' actions and rewards, such that $H^t = \{(S^v, A_1^v, A_2^v, R_1^v, R_2^v)\}_{v=0}^t$.

A *policy* π_i gives a probability distribution over actions to be taken by player i at each state. Players wish to learn the parameters θ_i of a policy which (in some sense) leads to large cumulative reward. Then we define a learning algorithm.

Definition 3.1 (Learning algorithm)**.** A learning algorithm for player i is a mapping σ_i from histories H^t to parameters, i.e., $\sigma_i : \mathcal{H} \to \Theta_i$, where $\mathcal{H}$ is the set of histories and Θ_i is the space of policy parameters for player i.

A learning game is a game whose strategies are learning algorithms. We define payoffs via average rewards. Define the value to player i of learning algorithm profile (σ_1, σ_2) starting from history h^t as $V_i(h^t, \sigma) = \liminf_{T\to\infty} T^{-1}\mathbb{E}_{\sigma_1,\sigma_2}\left(\sum_{v=t}^{t+T} R_i^v \mid H^t\right)$, where $\mathbb{E}_{\sigma_1,\sigma_2}$ is the expectation taken with respect to trajectories in which agents follow learning algorithms σ_1, σ_2 at each step. Let the initial state s^0 be fixed so that $h^0 = \{s^0\}$. Then a learning game is a game whose strategy spaces are spaces of learning algorithms Σ_1, Σ_2, and whose payoffs are given by $V_i(\sigma_1, \sigma_2) \equiv V_i(h^0, \sigma_1, \sigma_2)$. A learning equilibrium is a Nash equilibrium of a learning game:

Definition 3.2 (Learning equilibrium)**.** Learning algorithm profile (σ_1, σ_2) is a *learning equilibrium* of the learning game with learning algorithm spaces Σ_1, Σ_2 if

$$\sup_{\sigma_1' \in \Sigma_1} V_1(\sigma_1', \sigma_2) \leq V_1(\sigma_1, \sigma_2) \text{ and } \sup_{\sigma_2' \in \Sigma_2} V_2(\sigma_1, \sigma_2') \leq V_2(\sigma_1, \sigma_2).$$

Finally, let a *welfare function* w be a function measuring the social value of different learning algorithm profiles. A straightforward welfare function is the *utilitarian welfare*, $w(\sigma_1, \sigma_2) = V_1(\sigma_1, \sigma_2) + V_2(\sigma_1, \sigma_2)$. Others commonly discussed are the Nash welfare [Nash, 1950] and the egalitarian welfare (e.g., Kalai 1977). We say that a profile of learning algorithms (σ_1, σ_2) is a *welfare-optimal learning equilibrium* if it is a learning equilibrium and if it maximizes a welfare function w. We will construct learning algorithms which 1) optimize a welfare function in self-play and 2) are minimally exploitable, in the spirit of welfare-optimal learning equilibrium.

4 Learning tit-for-tat (L-TFT)

Tit-for-tat (TFT) famously enforces cooperation in the iterated Prisoner's Dilemma by cooperating when its counterpart cooperates, and defecting when they defect. Similarly, the "folk theorems" say that equilibria in repeated games are constructed by specifying a target payoff profile, a profile of target strategies which achieve this payoff profile, and a punishment strategy that is used to punish agents who deviate from the target strategy (see Mailath et al. [2006] for a review).

Learning tit-for-tat (L-TFT) is a class of learning algorithms that are analogous to tit-for-tat for learning games. For L-TFT, defection consists in playing a policy other than the current estimate of the cooperative policy. This depends on a notion of cooperation and a class of algorithms for learning a "cooperative" policy. These are encoded in a welfare function and a class of learning algorithms which learn an optimal policy with respect to that welfare function. Thus an L-TFT agent punishes their counterpart when it detects that they are not learning a welfare-optimal policy using a particular set of algorithms, and otherwise plays according to their current estimate of the welfare-optimal policy.

Because policies are in general stochastic, and because we do *not* assume that policy parameters are mutually visible, we also need a method for inferring whether the other player is following this cooperative algorithm. In particular, we need to construct a test for the hypotheses:

$\mathbf{H}_0$: My counterpart is following an algorithm for learning a welfare-optimal policy.

$\mathbf{H}_1$: My counterpart is not following an algorithm for learning a welfare-optimal policy.

In order to be able to cooperate and punish, as well as detect defections by their counterpart, an agent i playing L-TFT maintains the following policy estimates:

- A cooperative policy $\theta_i^{C,t}$ trained to optimize the welfare function. This network is updated according to a learning algorithm $\mathcal{O}^C$. In our experiments, $\mathcal{O}^C$ corresponds to double deep Q-networks (DDQN; Van Hasselt et al. 2015) trained on the reward signal $R_1^t + R_2^t$;
- A punishment policy $\theta_i^{P,t}$, updated according to a learning algorithm $\mathcal{O}^P$. In our experiments, this corresponds to DDQN trained on the reward signal $-R_j^t$;

- An estimate $\widehat{\theta}_j^{C,t}$ of the other player's current policy which corresponds to the hypothesis that they are playing according to a cooperative learning algorithm. This policy is estimated by specifying a class of cooperative learning algorithms Σ_j^C; finding the learning algorithm $\widehat{\sigma}_j^{C,t}$ in this class which best fits the other player's actions; and taking $\widehat{\theta}_j^{C,t} = \widehat{\sigma}_j^{C,t}(H^t)$. In our experiments, the L-TFT agent assumes that the learning algorithms Σ_j^C apply DDQN to $R_1^t + R_2^t$ and follow a Boltzmann exploration policy with an unknown initial temperature parameter. In order to obtain $\widehat{\theta}_j^{C,t}$ we must therefore estimate this temperature parameter, which we do using an approximate maximum likelihood algorithm;
- An estimate $\widehat{\theta}_j^{D,t}$ of the other player's current policy which corresponds to the hypothesis that they are not playing according to a cooperative learning algorithm. This policy is estimated by specifying a class of learning algorithms Σ_j^D, finding the learning algorithm $\widehat{\sigma}_j^{D,t}$ in this class which best fits the other player's actions; and taking $\widehat{\theta}_j^{D,t} = \widehat{\sigma}_j^{D,t}(H^t)$. In our experiments, we approximate this procedure by obtaining $\widehat{\theta}_j^{D,t}$ via supervised learning on player j's actions given the state.

Algorithm 1 summarizes L-TFT. Note that, while our experiments use a standard model-free learning algorithm for the base algorithms $\mathcal{O}^C, \mathcal{O}^P$, this framework is highly general. For instance, the players might be model-based planners who choose actions by optimizing against an environment model with parameter θ_i, which is updated over time. Moreover, with more sophisticated methods for inferring the other player's learning algorithm, L-TFT algorithms may also be constructed in games with partial state observability, or uncertainty about the other players' rewards.

Algorithm 1: Learning tit-for-tat (L-TFT)

Input: Observation history H^t, Other player punishment indicator `being_punished`, defection at previous timestep indicator `defection_detected`, cooperative learning algorithm $\mathcal{O}^C$, punishment learning algorithm $\mathcal{O}^P$, defection hypothesis test `HypothesisTest`

`punishing` $\leftarrow$ `is_punishing`(H^t);
$\theta_i^{P,t} \leftarrow \mathcal{O}^P(H^t)$;
if not `being_punished` *or* `defection_detected` **then**
  $\theta_i^{C,t} \leftarrow \mathcal{O}^C(H^t)$;
if `defection_detected` *and* not `being_punished` **then**
  $A_i^t \sim \pi_{\theta_i^{P,t}}(\cdot \mid S^t)$;
  Broadcast punishing;
else
  $A_i^t \sim \pi_{\theta_i^{C,t}}(\cdot \mid S^t)$;
`defection_detected` $\leftarrow$ `HypothesisTest`(H^t);
return A_i^t, `defection_detected`

5 Experiments

5.1 L-TFT with DDQN + Boltzmann exploration

For the cooperative and punishment learning algorithms, we use DDQN applied to reward signals $R_1^t + R_2^t$ and $-R_j^t$, respectively. Agents follow Boltzmann exploration with respect to the estimated Q-function, i.e., $\pi_{\theta_i}(a_i \mid s; \tau_i) \propto \exp\left\{\sum_{a_j} Q_{\theta_i}(s, a_i, a_j)/\tau_i\right\}$, where Q_{θ_i} is a neural network with parameter θ_i. The initial temperature τ_i is decayed according to a schedule $g(\cdot, \tau_i)$, such that the temperature at each time step t is $\tau_i^t = g(t, \tau_i)$. (In our experiments g linearly decreases to a minimum value, and is constant thereafter.) Our L-TFT algorithm punishes for one episode after it has detected a defection. We assume that players may broadcast punishment to their counterpart. L-TFT agents do not update their cooperative and defecting policy estimates if their counterpart broadcasts punishment, and do not punish if they are currently being punished; for simplicity we suppress the dependence of agents' policies and of the state on these broadcasts.

Testing for defection: $\texttt{L-TFT}_q$

To decide whether to punish, our agent conducts a goodness-of-fit test for the hypothesis that the other player is cooperating. A goodness-of-fit test (e.g., Lemeshow and Hosmer Jr 1982) compares the fit of a simple model with the fit of a flexible model. If the fit of the simple model is sufficiently close to that of the flexible model, the simple model is deemed adequate. Otherwise, the simple model is said to fit poorly. Here, the simple model we assess is the hypothesis that the other player is using cooperative DDQN with Boltzmann exploration. The flexible model we would like to compare to is that the other player is following an arbitrary learning algorithm (which need not be cooperative). One approach to estimating this model could be to solve the penalized maximum likelihood problem

$$\widehat{\theta}_j^{D,1}, \ldots, \widehat{\theta}_j^{D,t} = \underset{\theta_j^{D,1},\ldots,\theta_j^{D,t}}{\arg\max} \sum_{v=1}^{t} \log \pi^D_{\theta_j^{D,v}}(A^v \mid S^v) + t^{-1}\lambda \sum_{v=1}^{t} \|\theta_j^{D,v}\|^2, \tag{1}$$

where $\pi^D_{\theta_j}$ is a class of neural networks that output a distribution over actions given a state. But solving 1 is computationally infeasible. As an approximation, we maintain a sequence $\{\widehat{\theta}_j^{D,v}\}_{v=1}^t$ by taking gradient steps on the supervised learning loss at each time step. We maintain a buffer of log likelihoods under the hypothesis of defection, $\boldsymbol{\mathcal{L}}_j^{D,t} = \{\log \pi^D_{\widehat{\theta}_j^{D,v}}(A_j^v \mid S^v)\}_{v=t-t'}^t$. (The estimate $\widehat{\theta}_j^{D,t}$ and log likelihood buffer $\boldsymbol{\mathcal{L}}_j^{D,t}$ are not updated while the other player broadcasts punishment.)

On the other hand, to obtain likelihoods under the hypothesis of cooperation, player j is cooperating, we need to estimate the initial temperature τ_j under the assumption that player j is following a cooperative learning algorithm for some τ_j. We would thus like to solve

$$\widehat{\tau}_j^t = \underset{\tau_j}{\arg\max} \sum_{v=1}^{t} \log \pi_{\theta_j^{C,v}} \left\{A^v \mid S^v; g(v, \tau_j)\right\}. \tag{2}$$

As a tractable approximation to the solution of problem 2, we maintain an estimated cooperative Q-network $\widehat{\theta}_j^{C,t}$ via DDQN updates on reward signal $R_1^t + R_2^t$ (only when not being punished). We track the likelihood of player j's actions under $\widehat{\theta}_j^{C,t}$ over a set of candidate values for τ_j'; and take $\widehat{\tau}_j^t$ to be the highest-likelihood such value at time t. For each τ_j', we also maintain a buffer of log likelihoods[1] $\boldsymbol{\mathcal{L}}_j^{C,t}(\tau_j') = \{\log \pi_j^{C,v}\left\{A_j^v \mid S^v; g(v, \tau_j')\right\}\}_{v=t-t'}^t$. (Estimate $\widehat{\theta}_j^{C,t}$ and buffers $\boldsymbol{\mathcal{L}}_j^{C,t}(\tau_j')$ are again not udpated while the opponent broadcasts punishment.)

Finally, the hypothesis test is conducted at the end of each episode by bootstrapping [Efron, 1992] the difference in mean log likelihoods over buffers $\boldsymbol{\mathcal{L}}_j^{C,t}(\widehat{\tau}_j^t)$ and $\boldsymbol{\mathcal{L}}_j^{D,t}$. The hypothesis of cooperation is rejected when the q^{th} quantile of this distribution is less than 0. This hypothesis test is summarized in Algorithm 2. We refer to the variant of L-TFT which uses this test as $\texttt{L-TFT}_{\boldsymbol{q}}$.

Exploiting $\texttt{L-TFT}_q$

We also construct *exploiter* agents to play against $\texttt{L-TFT}_q$. These agents assume their counterpart is $\texttt{L-TFT}_q$ for a particular value of q, simulate the hypothesis test used by $\texttt{L-TFT}_q$, and act (myopically) depending on the result of the test. That is, if an exploiter predicts that a defection would be detected by a test with cutoff q, it acts according to the cooperative policy; otherwise, it takes a selfish action (according to a DDQN network trained on its reward signal only). Call this agent $\texttt{Exploiter}_q$. The networks maintained by $\texttt{Exploiter}_q$ are: 1) a cooperative network trained on $R_1^t + R_2^t$, 2) a selfish network trained on R_i^t, 3) an estimate of *its own* policy under the hypothesis that it is cooperating $\widetilde{\theta}_j^{C,l}$, and 4) an estimate of its own policy under the hypothesis that it is defecting $\widetilde{\theta}_j^{D,l}$.

[1] When computing these likelihoods, we set the Q-value of an action equal to the mean of all Q-values which are within some distance of one another (in our experiments, 0.25). This is important in Coin Game, where there are multiple paths of equal length to a coin. Thus, although estimated Q-values along different paths may be very similar, these differences are amplified by the exponentiation used in Boltzmann exploration when the temperature is small, possibly leading to false detections of defection.

Algorithm 2: Hypothesis test for L-TFT$_q$

Input: Defection log likelihood buffer $\boldsymbol{\mathcal{L}}_j^D$, cooperative log likelihood buffer $\boldsymbol{\mathcal{L}}_j^C(\widehat{\tau}_j^t)$, cutoff quantile q, test statistics buffer $\{\Delta_j^{(q),v}\}_{v=t-u}^{t-1}$

/* t indexes episodes */

$\boldsymbol{\Delta}_j \leftarrow \emptyset$

for $b = 1, \ldots, B$ **do**

  $\boldsymbol{\mathcal{L}}_j^{C,b} \leftarrow \texttt{Bootstrap}\left\{\boldsymbol{\mathcal{L}}_j^{C,t}(\widehat{\tau}_j^t)\right\}$

  $\boldsymbol{\mathcal{L}}_j^{D,b} \leftarrow \texttt{Bootstrap}(\boldsymbol{\mathcal{L}}_j^{D,t})$

  $\mathcal{L}_j^{C,b} \leftarrow B^{-1}\sum_{\ell_j^C \in \boldsymbol{\mathcal{L}}_j^{C,b}} \ell_j^C$

  $\mathcal{L}_j^{D,b} \leftarrow B^{-1}\sum_{\ell_j^D \in \boldsymbol{\mathcal{L}}_j^{D,b}} \ell_j^D$

  $\boldsymbol{\Delta}_j \leftarrow \boldsymbol{\Delta}_j \cup \left\{\mathcal{L}_j^{C,b} - \mathcal{L}_j^{D,b}\right\}$

$\Delta_j^{(q),t} \leftarrow \texttt{Percentile}(\boldsymbol{\Delta}_j, q)$

$\bar{\Delta}_j^{(q),t} \leftarrow u^{-1}\sum_{v=t-u}^{t} \Delta_j^{(q),v}$ // Average previous test statistics for stability

if $\bar{\Delta}_j^{(q),t} < 0$ **then**

  defection_detected $\leftarrow$ True

else

  defection_detected $\leftarrow$ False

return defection_detected

5.2 Environments

The environments we use are the iterated Prisoner's Dilemma (IPD) and the Coin Game environment [Lerer and Peysakhovich, 2017][2]. In both cases, episodes are 20 timesteps long. Our version of the IPD involves repeated play of the matrix game with expected payoffs as in Table 1. Policy π_{θ_i} gives the probability of each action given player j's action at the previous timestep. In our implementation, this means that the state passed to the estimated Q-function is the profile of actions taken at the last step.

		Player 2	
		C	D
Player 1	C	$-1, -1$	$-3, 0$
	D	$0, -3$	$-2, -2$

Table 1: Prisoner's Dilemma expected payoffs.

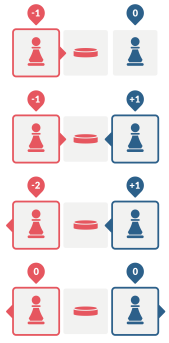

Figure 1: Coin Game Payoffs.

Write the expected reward function, corresponding to the payoffs in Table 1, as $r_i(a_i, a_j)$. We introduce randomness by generating rewards as $R_i^t \sim N\left\{r_i(A_i^t, A_j^t), 0.1\right\}$.

In the Coin Game environment [Lerer and Peysakhovich, 2017] depicted in Figure 1, a Red and Blue player navigate a grid and pick up randomly-generated coins. Each player gets a reward of 1 for picking up a coin of any color. But, a player gets a reward of -2 if the other player picks up their coin. This creates a social dilemma in which the socially optimal behavior is to only get one's own coin, but there is incentive to defect and try to get the other player's coin as well.

[2]These experiments were implemented using extensions of OpenAI Gym [Brockman et al., 2016] and the SLM Lab framework for reinforcement learning [Keng and Graesser, 2017].

The state space consists of encodings of each player's location and the location of the coin. We use networks with two convolutional layers and two fully connected feedforward layers.

5.3 Results

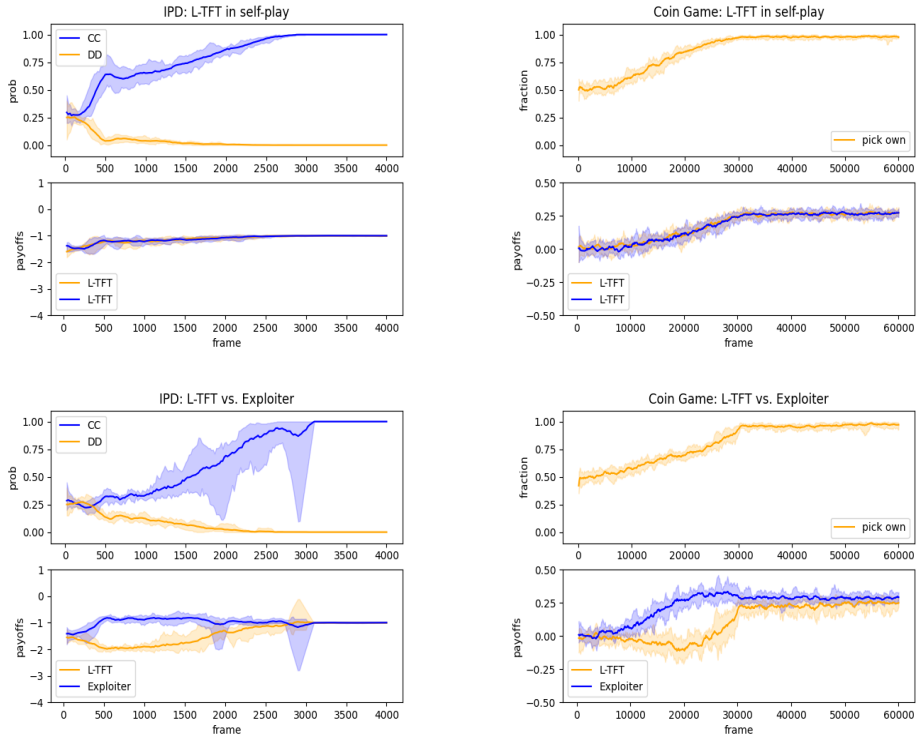

Figure 2: Performance of $\texttt{L-TFT}_{0.95}$ in the IPD (left) and Coin Game (right). Counterparts are $\texttt{L-TFT}_{0.95}$ (top), $\texttt{Exploiter}_{0.95}$ (bottom). For IPD, the `prob` axis is the probability of mutual cooperation (`cc`) and mutual defection (`dd`), For Coin Game, the `pick_own` axis is the proportion of times the coin is picked by the agent of its own color (i.e., the agents cooperate). The `payoffs` axis shows a moving average of player rewards after each episode. These values are averaged over 10 replicates and bands are (min, max) values. Note that temperature values are constant in the IPD after 3000 episodes and in the Coin Game after 30000 episodes.

Figure 2 shows the performance of $\texttt{L-TFT}_q$ in self-play, against $\texttt{Exploiter}_q$ (i.e., an exploiter with a correctly-specified value of q). $\texttt{L-TFT}_q$ is able to converge to welfare-optimal payoffs in self-play and against the $\texttt{Exploiter}_q$ (suggesting that the exploiter eventually anticipates being punished if it defects, and therefore chooses to cooperate). This indicates that our hypothesis test effectively detects defection. $\texttt{L-TFT}_q$ also effectively punishes a naive selfish learner (i.e., one trained with DDQN on R_j^t); see Figure 3 in the Supplementary Materials. Punishing the naive learner comes as the expense of low payoffs for L-TFT. However, this is the expected behavior, just as the punishments which enforce equilibria in classical iterated games can lead to low payoffs for each player.

Strategic choice of q

While our previous experiments with $\texttt{Exploiter}_q$ assumed that the exploiter knew the cutoff quantile used in L-TFT's hypothesis test, this seems like too strong an assumption for real-world agents deployed by different principals. Here we consider a game played between 1) one principal deciding which value of q_1 to specify for its $\texttt{L-TFT}_{q_1}$ agent, and 2) a second principal deciding among $\texttt{L-TFT}_{q_2}$ and $\texttt{Exploiter}_{q_2}$ for different values of q_2. Table 2 shows the average payoffs to each player over a time horizon of 4000 steps.

	q in $\texttt{L-TFT}_q$			q in $\texttt{Exploiter}_q$		
	0.55	0.75	0.95	0.55	0.75	0.95
$\texttt{L-TFT}_{0.55}$	$-1.28, -1.2$	$-0.97, -1.48$	$-0.98, -1.38$	$-1.03, -1.41$	$-1.06, -1.43$	$-0.91, -1.81$
$\texttt{L-TFT}_{0.75}$	$-1.44, -0.97$	$-1.13, -1.1$	$-1.09, -1.13$	$-1.19, -1.11$	$-1.25, -1.11$	$-1.21, -1.35$
$\texttt{L-TFT}_{0.95}$	$-1.21, -1.06$	$-1.12, -1.1$	$-1.11, -1.11$	$-1.2, -1.06$	$-1.27, -1.03$	$-1.41, -0.98$

Table 2: Empirical payoff matrix for a learning game in which the base game is an iterated Prisoner's Dilemma played for 4000 timesteps. The row player chooses $\texttt{L-TFT}_q$ for $q \in \{0.55, 0.75, 0.95\}$. The column player chooses either $\texttt{L-TFT}_q$ or $\texttt{Exploiter}_q$ for $q \in \{0.55, 0.75, 0.95\}$. Cells give (row player payoffs, column player payoffs), where payoffs are the mean of rewards over the span of the game, averaged over 10 replicates. Standard errors are all less than or equal to 0.1.

It turns out that the unique Nash equilibrium of this empirical learning game[3] is one in which row player chooses $\texttt{L-TFT}_{0.95}$ with probability 0.88 and $\texttt{L-TFT}_{0.55}$ with probability 0.12, while the column player chooses $\texttt{L-TFT}_{0.55}$ with probability 0.88 and $\texttt{Exploiter}_{0.95}$ with probability 0.12. The expected payoffs in equilibrium are $-1.23, -1.08$ for row and column player, respectively. This suggests that $\texttt{L-TFT}_q$ is vulnerable to some amount of exploitation on short time horizons, although still does better than the mutual-defection payoff of -2. On the other hand, $\texttt{L-TFT}_{q_1}$ converges to mutual cooperation[4] against both $\texttt{L-TFT}_{q_2}$ and $\texttt{Exploiter}_{q_2}$ for each value of q_1, q_2. Thus L-TFT avoids exploitation over longer time horizons in this learning game.

6 Discussion

Powerful learning agents should be designed to avoid conflict when they are eventually deployed. One way to avoid such outcomes is to construct profiles of agents which avoid conflict and which rational principals prefer to deploy. One appealing criterion for joint rationality on part of the principals is welfare-optimal learning equilibrium (rather than mere convergence to Nash equilibrium of the base game, for instance). We have presented a class of learning algorithms, learning tit-for-tat (L-TFT), which approximately implement welfare-optimal learning equilibrium in the sense of converging to welfare-optimal payoffs in self-play and discouraging exploitation.

Many open questions remain. First, while we have relied on a welfare function, we have said little about which welfare function should be used. The ideal scenario would be for the principals of the reinforcement learning systems in question to coordinate on a welfare function before deploying these systems. Moreover, it may be necessary to develop novel reinforcement learning methods tailored for the optimization of different welfare functions. For instance, our use of DDQN with the utilitarian welfare is unproblematic, as the sum of the players' reward signals still admits a Bellman equation for their cumulative reward. But the welfare functions other than the utiliarian welfare (such as the Nash welfare [Nash, 1950]) do not admit a Bellman equation (see the discussion of nonlinear scalarization functions in Roijers et al. [2013]'s review of multi-objective reinforcement learning).

Many generalizations of our setting can be studied, including partial observability and more than two agents. Perhaps the most restrictive of our assumptions is that the agents' reward functions are known to each other. A complete framework should address the problem of incentives to misrepresent one's reward function to improve one's payoffs in the welfare-optimal policy profile. One direction would be to allow for uncertainty about the other players' reward functions, analogously to Bayesian games [Harsanyi, 1967] but in a way that is tractable in complex environments. This may in turn require the development of novel techniques for inference about other players' preferences in complex environments; see Song et al. [2018], Yu et al. [2019] for recent steps in this direction. Another approach to reward uncertainty would be to use mechanism design techniques [Nisan et al., 2007, Ch. 9] to incentivize the truthful reporting of the principals' utility functions.

[3]Computed using the Nashpy Python library [Knight and Campbell, 2018].

[4]Payoffs averaged over the final 10 episodes equal to -1 ± 0.03.

Acknowledgements

Many thanks to Tobias Baumann, Anthony DiGiovanni, Flo Dorner, Daniel Kokotajlo, Alex Lyzhov, and Johannes Treutlein for helpful comments on drafts of this document.

References

David Balduzzi, Sebastien Racaniere, James Martens, Jakob Foerster, Karl Tuyls, and Thore Graepel. The mechanics of n-player differentiable games. *arXiv preprint arXiv:1802.05642*, 2018.

Michael Bowling and Manuela Veloso. Rational and convergent learning in stochastic games. In *International joint conference on artificial intelligence*, volume 17, pages 1021–1026. Lawrence Erlbaum Associates Ltd, 2001.

Ronen I Brafman and Moshe Tennenholtz. Efficient learning equilibrium. In *Advances in Neural Information Processing Systems*, pages 1635–1642, 2003.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Vincent Conitzer and Tuomas Sandholm. Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Machine Learning*, 67(1-2):23–43, 2007.

Bradley Efron. Bootstrap methods: another look at the jackknife. In *Breakthroughs in statistics*, pages 569–593. Springer, 1992.

Jakob Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 122–130. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

John C Harsanyi. Games with incomplete information played by "bayesian" players, i–iii part i. the basic model. *Management science*, 14(3):159–182, 1967.

Ehud Kalai. Proportional solutions to bargaining situations: interpersonal utility comparisons. *Econometrica: Journal of the Econometric Society*, pages 1623–1630, 1977.

Wah Loon Keng and Laura Graesser. Slm lab. `https://github.com/kengz/SLM-Lab`, 2017.

Vincent Knight and James Campbell. Nashpy: A python library for the computation of nash equilibria. *Journal of Open Source Software*, 3(30):904, 2018.

Stanley Lemeshow and David W Hosmer Jr. A review of goodness of fit statistics for use in the development of logistic regression models. *American journal of epidemiology*, 115(1):92–106, 1982.

Adam Lerer and Alexander Peysakhovich. Maintaining cooperation in complex social dilemmas using deep reinforcement learning. *arXiv preprint arXiv:1707.01068*, 2017.

Alistair Letcher, Jakob Foerster, David Balduzzi, Tim Rocktäschel, and Shimon Whiteson. Stable opponent shaping in differentiable games. *arXiv preprint arXiv:1811.08469*, 2018.

Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

George J Mailath, J George, Larry Samuelson, et al. *Repeated games and reputations: long-run relationships*. Oxford university press, 2006.

John F Nash. The bargaining problem. *Econometrica: Journal of the Econometric Society*, pages 155–162, 1950.

Noam Nisan et al. Introduction to mechanism design (for computer scientists). *Algorithmic game theory*, 9: 209–242, 2007.

Alexander Peysakhovich and Adam Lerer. Consequentialist conditional cooperation in social dilemmas with imperfect information. *arXiv preprint arXiv:1710.06975*, 2017.

Diederik M Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *Journal of Artificial Intelligence Research*, 48:67–113, 2013.

Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press, 2008.

Jiaming Song, Hongyu Ren, Dorsa Sadigh, and Stefano Ermon. Multi-agent generative adversarial imitation learning. In *Advances in neural information processing systems*, pages 7461–7472, 2018.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*, 2015.

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.

Lantao Yu, Jiaming Song, and Stefano Ermon. Multi-agent adversarial inverse reinforcement learning. *arXiv preprint arXiv:1907.13220*, 2019.

Supplementary Materials

L-TFT vs. naive DDQN

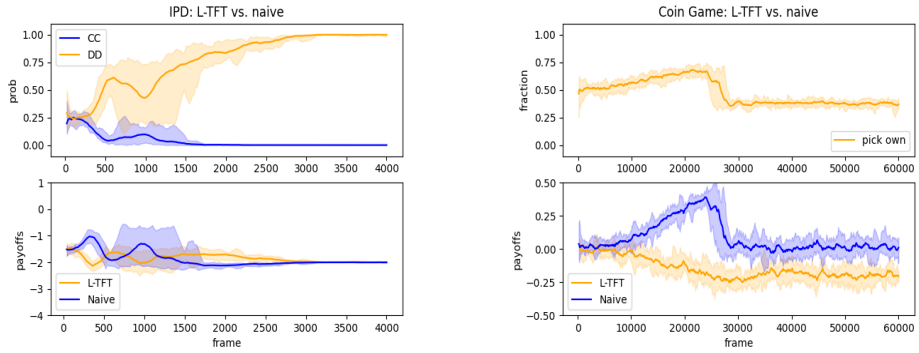

Figure 3: Performance of $\texttt{L-TFT}_{0.95}$ in the IPD (left) and Coin Game (right) against a DDQN learner trained on R_j^t (`naive`). For IPD, the `prob` axis is the probability of mutual cooperation (`cc`) and mutual defection (`dd`), For Coin Game, the `pick_own` axis is the proportion of times the coin is picked by the agent of its own color (i.e., the agents cooperate). The `payoffs` axis shows a moving average of player rewards after each episode. These values are averaged over 10 replicates and bands are (min, max) values. Note that temperature values are constant in the IPD after 3000 episodes and in the Coin Game after 30000 episodes.