Software ENgineering

Learning from induced changes in opponent (re)actions in multi-agent games

P.J. 't Hoen, S.M. Bohte, J.A. La Poutré

REPORT SEN-E0513 NOVEMBER 2005

ISSN 1386-369X

Learning from induced changes in opponent (re)actions in multi-agent games

ABSTRACT
Multi-agent learning is a growing area of research. An important topic is to formulate how an agent can learn a good policy in the face of adaptive, competitive opponents. Most research has focused on extensions of single agent learning techniques originally designed for agents in more static environments. These techniques however fail to incorporate a notion of the effect of own previous actions on the development of the policy of the other agents in the system. We argue that incorporation of this property is beneficial in competitive settings. In this paper, we present a novel algorithm to capture this notion, and present experimental results to validate our claims.

2000 Mathematics Subject Classification: 91A20, 91A26
Keywords and Phrases: Multi-Agent Learning

Learning from Induced Changes in Opponent (Re)Actions in Multi-Agent Games

P.J. 't Hoen, S.M. Bohte, and J.A. La Poutré
hoen@cwi.nl,sbohte@cwi.nl,hlp@cwi.nl
CWI, The Netherlands Centre for Mathematics and Computer Science
Kruislaan 413, NL-1098 SJ Amsterdam, The Netherlands

Abstract

Multi-agent learning is a growing area of research. An important topic is to formulate how an agent can learn a good policy in the face of adaptive, competitive opponents. Most research has focused on extensions of single agent learning techniques originally designed for agents in more static environments. These techniques however fail to incorporate a notion of the effect of own previous actions on the development of the policy of the other agents in the system. We argue that incorporation of this property is beneficial in competitive settings. In this paper, we present a novel algorithm to capture this notion, and present experimental results to validate our claims.

1 Introduction

Acting intelligently in a dynamic environment shared with other competing agents is a hard task, as daily life as well as a host of work on learning in multi-agent settings demonstrates. Without the presence of a teacher, and with only the observed actions and rewards to learn from, the framework of Reinforcement Learning (RL) (Sutton & Barto, 1998) is an obvious choice. Conceptually, the focus moves from the realm of Markov Decision Problems to that of Game Theory (GT) and stochastic games (Shoham, Powers, & Grenager, 2004).

Recent work has proposed a number of extensions from the single agent setting to the multi-agent domain. State-of-the-art Multi-Agent Reinforcement Learning (MARL) algorithms improve on single-agent RL approaches by incorporating models of the *current* opponent agent's behavior. This allows an agent to model the current "environment" including opponents, against which it has to optimize its own behavior by playing a best-response. In a multi-agent setting, the policy that maximizes payoff will depend on the *changing* actions of adversaries. Importantly, the adaptations of adversaries are likely to be reactions to *ones own* actions. This point is largely ignored in current (MARL) algorithms. Such strategic reasoning can be paramount when an agent is repeatedly interacting with the same opponents, and earlier actions influence the future behavior of these learning adversaries.

The Prisoner's Dilemma (PD) is illustrative in that in repeated play the optimal policy differs from blindly playing the myopic best response, and that it can be profitable to have a good estimation of the opponents reactions to ones own play. In one single game, two competing agents can each choose from actions cooperate or defect ($\{C, D\}$), and the maximum *joint* payoff is achieved when both agents choose to Cooperate. However, when the other agents plays Cooperate, an agent obtains an even larger payoff from playing Defect. From a game theoretical perspective then, for the single shot game $\{D, D\}$ is the dominant strategy and a Nash-Equilibrium (see Section 2).

In iterated play of the PD (iPD) game, there is no longer a clear dominant strategy, as both agents can achieve a higher aggregated *and* individual reward by cooperating and playing the joint action $\{C, C\}$, *provided* there is some strategic incentive to rarely unilaterally defect. That is: defecting as a strategy is discouraged by the likely *negative future* reaction of the opponent (see also the famous Folk Theorem discussed in Section 2).

It is important to notice that just modeling the *current* behavior of the opponent in iPD can lead to $\{D, D\}$ outcome in repeated play when the reaction of the opponent to ones own play of defection is not taken into account. An agent that does not take into account the changing behavior of the opponent may estimate that a cooperating opponent may be exploited, and ignores the fact that defection will result in an opponent that also defects. When repeatedly playing a game against an adversary, the question is which moves one should play to maximize the individual reward given the *dependent* adaptive behavior of the adversary.

We argue that current state-of-the-art MARL algorithms (M. H. Bowling & Veloso, 2002; M. Bowling, 2004; Tesauro, 2003; Conitzer & Sandholm, 2003; Greenwald & Hall, 2003; Banerjee & Peng, 2004; Weinberg & Rosenschein, 2004) (see also Shoham et al. (2004) for a more in depth discussion) do not incorporate a sufficient notion of the longer term impact of their actions on the dynamics of the environment. This environment includes other adaptive agents that react to moves of their opponents as well. For example, current MARL algorithms that play the iPD will universally converge to the worst possible outcome of $\{D, D\}$ in self-play[1](Crandall & Goodrich, 2005). The one exception to the full defection outcome in repeated play of the iPD we are aware of is a recent paper by Crandall and Goodrich (2005), which we will discuss further below.

In this paper, we present a novel MARL framework called Strategic Opponent Policy Modeling (StrOPM) that goes beyond current Myopic Opponent Policy Modeling (MOPM) MARL approaches. StrOPM, as most other MARL's, takes into account the current (estimated) policy of the opponents and the rewards for actions. Importantly, our algorithm also estimates how the policies of the opponents change over time due to actions played by StrOPM. The policy

[1]An agent playing against an identical opponent.

of the agent using the StrOPM algorithm is adjusted to increase the expected future reward by taking into consideration 1) the immediate and future rewards of actions 2) the estimated policy of the opponent, and 3), most importantly, the impact of the chosen actions on the future policy of the opponent. The last point is the novel contribution of our work.

Our algorithm is a policy-gradient like method with roots in Q-learning. Q-like values for states are calculated based on estimated values of actions and estimated opponent policies. Policies are updated along the gradient of expected increased rewards. To more accurately model the (future) changes in the environment due to an agent's own actions, the agent computes a continuously updated estimate of the *change* in the choice of opponents actions due to its own actions. StrOPM then adjusts the reward gradient by this estimate to account for the opponent's likely *re*action.

We present our results for the well studied domain of iterated-play two-player two-action matrix games. We show that our algorithm quickly arrives at the desired
Nash-Equilibrium for games that are unproblematic. These games are relatively straightforward in the sense that optimal play in a single-shot play of the game is also a good solution to iterated play of the game.

Importantly, we show that the proposed StrOPM algorithm achieves cooperation ($\{C, C\}$) in repeated play of the PD. After an initial exploration period, StrOPM learns that defection leads to reciprocal defection, and an overall lower value of future play than a high level of cooperation. The agents learn to optimize their own interest as a function of the played game and the reciprocal behavior of the evolving opponent. The (possible) cooperation strategy is not a precoded option in the algorithm, but is achieved by optimizing beyond the immediate best response to the current opponent play. The StrOPM agents in self-play exhibit a learned Tit-For-Tat strategy (Axelrod, 1984). The Tit-For-Tat algorithm plays C until the opponent defects upon which a D is played followed by again C until the next defection to encourage cooperation. The StrOPM agents learn this strategy and cooperates to reach a good equilibrium, and yet also evolves a threat state where defection is returned in order to guard against exploitation by the opponent.

We also compare our approach to the recent algorithm M-Qubed of Crandall and Goodrich (2005) for the studied matrix games. M-Qubed achieves the (C, C) equilibrium in the iPD by a meta step. The M-Qubed algorithm is designed to balance play of a best-response strategy with the precoded strategy of playing one action consistently. In particular, the algorithm during play continuously considers the value of switching to the pure, precoded strategy of playing C. Using a precoded strategy may lead to being exploited, or be unsuitable for more complex settings, as we will show.

We present results for StrOPM versus M-Qubed and show that the two type of agents playing against each other can together achieve cooperation, but from entirely different principles. The StrOPM algorithm in self-play finds good strategies significantly faster than M-Qubed for the iPD game. When playing against each other, this high speed of learning leads to an initial period where StrOPM exploits M-Qubed. Furthermore, StrOPM is shown to exploit the M-Qubed algorithm in a simple matrix game where the pre-coded strategy of playing one action predictably can result in inferior strategies in repeated play. Using precoded strategies has limited application. For example, precoded strategies cannot be used when the number of possible precoded strategies must be large, the precoded strategies cannot be determined, or do not generalize to novel, or more complex settings. Fundamentally, a MARL should remain flexible in novel settings.

In summary, we have presented three important contributions. First of all, we have signaled the need for Multi-Agent learning algorithms to take into account the impact of their actions on the learned policy of the opponents. This concept is as yet lacking in the MARL community. Secondly, we presented a novel framework that allows agents to apply the above observation to achieve desired outcomes for representative iterated matrix games, and, more importantly, also for the iPD. The latter is known to be difficult for MARL approaches; the presented StrOPM framework solves the iPD game by learning and recognizing "threat states", where it will punish an opponent's uncooperative behavior with a Tit-for-Tat type strategy (and also learns the reciprocal reaction). Lastly, we demonstrate the danger of incorporating pre-coded fixed strategies in learning algorithms as these are vulnerable to exploitation or may not be suited to novel or more complex settings.

Structure The remainder of this document is structured as follows. In Section 2, we present iterated matrix games, and the iterated Prisoners Dilemma. In Section 3, we present our general framework. We present a Multi-Agent RL framework, StrOPM, that in the limit plays best-response against stationary opponents. Importantly, StrOPM can reason strategically in games where it is relevant in how an opponent will react to the history of play. In Section 4, we present experimental results. In Section 5, we discuss and conclude.

2 Model and problem setting

In this section, we give some introductory definitions and notation from Game Theory, Reinforcement Learning, and the iterated matrix games we study as problem domain. We discuss the iterated Prisoners Dilemma as a game of special interest.

2.1 Agents and Matrix games

We consider multi-agent Reinforcement Learning for iterated play of games, more specifically **matrix games**, and introduce some well-known concepts from Game Theory (GT).

In general, let S denote the set of states in the game and let A_i denote the set of actions that agent/player i may select in each state $s \in S$. Let $a = (a_1, a_2, \ldots, a_n)$, where $a_i \in A_i$ be a joint action for n agents, and let

$A = A_1 \times \cdots \times A_n$ be the set of possible joint actions. **Zero-sum games** are games where the rewards of the agents for each joint action sum to zero. **General sum games** allow for any sum of values for the reward of a joint action.

A **strategy (or policy)** for agent i is a probability distribution $\pi(\cdot)$ over its actions set A_i. Let $\pi(S)$ denote a strategy over all states $s \in S$ and let $\pi(s)$ (or π_i) denote a strategy in a single state s. A strategy may be a **pure strategy** (an agent selects an action deterministically) or according to a **mixed strategy** (a strategy that plays a random action, according a probability distribution). A **joint strategy** played by n agents is denoted by $\pi = (\pi_i, \ldots, \pi_n)$. Let a_{-i} and π_{-i} refer to the joint action and strategy of all agents except agent i.

We focus on the more restricted **matrix game**, defined by a set of matrices $R = \{R_1, \ldots, R_n\}$. Let $R(\pi) = (R_1(\pi), \ldots, R(\pi_n))$ be a vector of expected payoffs when the joint strategy π is played. Also, let $R_i(\pi_i, \pi_{-i})$ be the expected payoff to agent i when it plays strategy π_i and the other agents play π_{-i}. A strategy is **dominant** if, regardless of what any other players do, the strategy earns a player a larger payoff than any other strategy. Also, let $R_i(\begin{bmatrix} a_i \\ a_{-i} \end{bmatrix})$ be the payoff for agent i playing action a_i while the other agents play action a_{-i}.

A **stage game** is a single iteration of a matrix game, and a **repeated game** is the indefinite repetition of the stage game between the same agents. While matrix games do not have state, agents can encode the previous w joint actions taken by the agents as state information, as for example illustrated by Sandholm and Crites (1995).

Each individual matrix game has certain classic game theoretic values. The **minimax** value for player i is $m_i = \max_{\pi_i} \min_{a_{-i}} R_i(\pi_i, a_{-i})$, i.e. the least reward that can be achieved if the game is known and the game is only played once. A **Best-Response** (BR) to the opponents strategy π_{-i} is defined by $BR = \pi^* = \max_{\pi} R_i(\pi, \pi_{-i})$. This corresponds to the (expected) most reward that can be gained from playing, under the assumption that the game is known; the game is only played once; and the opponent strategy is known.

A **Nash Equilibrium** (NE) is a joint strategy such that no agent may unilaterally change its strategy without lowering its expected payoff in the one shot play of the game. Nash (1951) showed that every n-player matrix game has at least one such NE. A **Pareto optimal** solution of the game is a joint strategy such that no agent may unilaterally increase its expected payoff without making another agent worse off. A (joint) strategy π_1 is said to **Pareto dominate** a strategy π_2 if the expected payoff for π_1 is at least as high as for π_2 and higher for at least one of the agents. A joint strategy is **Pareto deficient** if it is not Pareto optimal.

In the studied matrix games, we assume that an agent can observe its own payoffs as well as the actions taken by all agents in each stage game, but only after the fact. All agents concurrently choose their actions. A possible adaptation of the agents' policy, i.e. learning as a result of observed opponent behavior, only takes effect in the next stage game. Repeated games are modeled as a series of stage games with the same opponent(s). Each agent then aims to maximize

R=0.35 R=0.35	T=0.5 S=0
S= 0 T=0.5	P=0.1 P=0.1

Table 1: Example payoffs for the (symmetrical) PD

its reward from iterated play of the same matrix game.

We restrict our investigation to two-player, two-action games as these are well classified (Rapoport, Guyer, & Gordon, 1976). The arguments and formalism presented in the rest of the paper are however applicable to more general settings. The next section discusses the Prisoners Dilemma (PD), a special two-player two-action game.

2.2 The iterated Prisoner's Dilemma

We present the iterated **Prisoners Dilemma** (iPD) as a special matrix game. In this game, each player has a choice of two operations: either **cooperate** (C) with the other player or **defect** (D). The payoff matrix for joint actions is shown in Table 1. If both players cooperate, they both receive a given payoff R. However, if one player plays Cooperate, and the other plays Defect, the defector receives a payoff $T > R$, and the cooperator receives a much lower payoff $S < R$. If both players play Defect, they receive the low payoff P. For each individual, the incentive is thus present to defect, hoping that the other player plays "cooperate". Myopic play by even one of the players already quickly leads both players to arrive at the suboptimal outcome of both players receiving a low reward P.

The PD game is of particular interest as it has specific properties that make it unique among the matrix games. The PD is the only game of the taxonomy of two-player, two-action games listed by Rapoport et al. (1976) for which the natural outcome is stable (a NE), but Pareto-deficient: there are outcomes that Pareto-dominate the NE. This unique property also makes it non-trivial for competitive learning algorithms in repeated play; how can learners consider the possible future gains that are non-existent in the single shot game?

Most multi-agent learning algorithms to date have focused on an individual agent learning a (myopic) Best Response to the *current* strategies of the other agents. Play between such agents using this approach often converge, and have as goal to converge, to a one-shot NE. However, a famous result from game theory (the **folk theorem**) suggests that the goal of reaching a one-shot NE may be inappropriate in repeated games.

The folk theorem implies that, in many games, there exists NEs for repeated games, repeated Nash-Equilibria (rNEs), that yield higher individual payoffs to all agents than do one-shot NEs, i.e. the rNE Pareto dominates the NE. Hence, in repeated games, a successful set of agents should learn to play profitable rNEs.

However, since many repeated games have an infinite number of rNEs, the folk theorem does little to indicate which one the agents should play. Littman and Stone (2003) present an algorithm for computing rNEs that satisfies a set of desiderata, but how to learn these strategy online is unknown. Additionally, an agent may have preferences between rNEs and play one above the other, if allowed by its opponents.

It is however not given that the mainstream, or state of the art RL algorithms, will (approximately) learn these rNE equilibriums in self play or against various classes of opponents. As discussed in Section 1, and by Crandall and Goodrich (2005), the claim is that current MARL algorithms will not converge to good rNE for iPD-like games.

3 The StrOPM Framework

We give a formal definition of the StrOPM framework. We first introduce the concepts from the single state perspective for inital exposition. With this in place, we then present the multi state formalism framework. Lastly, we show (very successful) experimental results with an instantiation of the framework for matrix games in Section 4.

We can formulate the goal of an agent i as wanting to maximize the total reward it obtains from playing T times against an adversary:

$$R_i^{tot} = \sum_{t=0}^{t=T} val_i(\pi_{i,t}, \pi_{-i,t}). \tag{1}$$

where π_t is the policy at time t and val_i the expected reward for a agent i using policy $\pi_{i,t}$ and opponent policies $\pi_{-i,t}$. Thus, we want to specify or learn the time-varying policy $\pi_{i,t}$ that maximizes the reward obtained.

We make several observations: first, we have to start playing from some policy $\pi_{i,t=0}$, and we may not know the payoff of (joint) actions yet. We may also not know the policy of the adversary yet, nor to what extent it is adaptive.

We assume that we can define a state representation for an agent i that includes sufficient history to make the evolution of both its own policy and the opponent's policy Markovian: $\pi_{i,t+1} = f(s(t), \pi_{i,t})$, where $f(s(t), \pi_{i,t})$ is the state dependent function that adapts the agent's policy based on the agent's current policy $\pi_{i,t}$ and the current state $s(t)$. Likewise, the agent assumes that the opponent behaves similarly Markovian: $\pi_{-i,t+1} = g(s(t), \pi_{-i,t})$, where $g(s(t), \pi_{-i,t})$ is an unknown function that adapts the adversarie's policy. We assume that part of the state information in the $g(s(t))$ term depends on the actions taken by the agent i.

At its core, the StrOPM algorithm attempts to estimate this unknown adaptive function $g(s(t), \pi_{-i,t})$, and then computes forward all possible future action sequences resulting from the thus evolving policies $\pi_{i,t}$ and $\pi_{-i,t}$. It then updates the policy $\pi_{i,(t+1)}$ in the direction of those future action sequences that promise the most reward.

3.1 StrOPM: Single State

The StrOPM algorithm learns the reward of joint actions, estimates the policy of the opponents, and updates its own policy to increase its expected reward. The policy is updated along the gradient of increased reward. StrOPM however moderates its changes in policy by taking into account how the opponent will react to its own chosen actions.

Learning the values of joint actions We introduce $V_i : A_i \times A_{-i} \rightarrow \Re$ that estimates the value of a single joint action to agent i. Through V_i, agent i learns its part of the rewards of the game, dependent upon the actions of the opponent. The function V_i is updated using the exponential moving average (EMA):

$$V_{i,t+1} = (1 - \alpha_{EMA1})V_{i,t} + \alpha_{EMA1} \times r_t, \tag{2}$$

where r_t is the reward received by agent i in epoch t, and $0 < \alpha_{EMA1} \leq 1$ is the learning rate for the update.

Estimation of the opponent policy We define π_{-i} : as the **estimate** of the policy of agent $-i$ by agent i. The opponent policy is estimated online again using Exponential Moving Average (EMA). The estimate of π_{-i} after observing action a_{-i} is adjusted according:

$$\pi_{-i,t+1}(a_{-i}) = (1 - \alpha_{EMA2})\pi_{-i,t}(a_{-i}) + \alpha_{EMA2}. \tag{3}$$

After this update, the policy $\pi_{-i,t+1}$ is normalised to retain π_{-i} as a probability distribution.

The (estimated) value of two policy pairs π_i of agent i and estimated policy π_{-i} of agent(s) $-i$ is defined as

$$val(\pi_i, \pi_{-i}) = \sum_{a_i} \sum_{a_{-i}} V_i \begin{bmatrix} a_i \\ a_{-i} \end{bmatrix} \times \pi_i(a_i) \times \pi_{-i}(a_{-i}). \tag{4}$$

Based on this Equation 4, an agent can deliberate upon the value of joint policies from its own perspective, i.e. what is the value of any policy π_i given an estimate of the opponent policies π_{-i}. An agent i can play best response as defined in Section 2, or can deliberate upon more advanced strategies.

Estimation of impact actions We introduce $\xi(\pi_{-i}(s), a_i)$ to estimate the impact of playing of an action a_i on the development of the policy of the opponent. This function[2] predicts the change in policy of the opponent upon playing action a_i; i.e. $\pi_{-i,t+1} = \xi(\pi_{-i,t}, a_i)$. We postpone the definition of ξ to Section 3.2, as we then have a better perspective to introduce this aspect.

[2]More complex functions like $\xi(\pi_i, \pi_{-i}, a)$ can be considered if required for good rNE.

Updating of the policy We introduce **policy update actions**. A policy update action pua_i dictates whether the likelihood of an action a_i should be increased. Additionally, we introduce the *null* policy update action pua_{null} to indicate that the policy should not be changed. Let $\pi_i^{pua_i}$ be the policy achieved by applying the policy update action pua_i to π_i. The policy π_i of agent i for a policy update action pua_i not equal to the null action is updated according to:

$$\pi_i^{pua_i} = (1 - \alpha_{LEARN})\pi_{i,t}(a_i) + \alpha_{LEARN}, \tag{5}$$

where α_{LEARN} is the learning rate. The probabilities $\pi_i(\cdot)$ for actions $a_j \neq a_i$ are normalized to retain π_i as a probability distribution.

Let $G(pua_i)$ be the estimated value of the game for agent i after applying policy update action pua_i and playing action a_i. The estimated value G of the game becomes:

$$G(pua_i) = \sum_{a_j \in A_i} \big(\pi_i^{pua_i}(a_j) \times val(\pi_i^{pua_i}, \xi(\pi_{-i}, a_j))\big). \tag{6}$$

The change in value of the game is the difference between the value of the original policy pair (π, π_{-i}) and the value of the newly updated policy $\pi_i^{pua_i}$ with respect to the policy of the opponent influenced by playing actions a_j, i.e. $\xi(\pi_{-i}, a_j))$.

The value of the null policy update action is calculated using the weighted reaction to the current policy π_i of agent i; just because agent i does not adapt its policy does not signify that its actions will not have impact.

$$G(pua_{null}) = \sum_{a_i \in A_i} \big(\pi_i(a_i) \times val(\pi_i, \xi(\pi_{-i}, a_i))\big). \tag{7}$$

In each epoch, the highest[3] valued policy update action pua^* is calculated according to

$$pua^* = \max_{pua_j} G(pua_j). \tag{8}$$

Note however that we keep the learning rate for policy updating **constant** over time, in contrast to most existing approaches. A decreasing learning rate over time is a classic assumption (Watkins & Dayan, 1992) for convergence of single agent learning algorithms and many multi-agent learning algorithms. A decreasing learning rate however trivially opens an agent to exploitation by continuously adaptive agents: an opponent can easily exploit a converged competitor after the opponent becomes near stationary due to its minimal learning rate. The StrOPM algorithm converges by choosing pua_{null} as its policy update action to maintain a stationary policy if this is deemed best. Should the environment however change, the StrOPM player is able to react due to non-zero learning rates.

[3]One is picked at random in case of tie.

One Epoch The learning algorithm for an agent i in each epoch sequentially:

1. Updates policy π_i using the highest valued policy update pua^* action as defined in Equation 8.
2. Plays action $a_i \in A_i$ based on the choosen policy update action pua^*. StrOPM plays action a_i for choosen policy update action $pua_i = pua^*$ if pua^* is not the null policy update action. Otherwise choose the action according to π_i.
3. Receives reward $R_i(\begin{bmatrix} a_i \\ a_{-i} \end{bmatrix})$ for the joint action determined by agents $-i$.
4. Updates the estimate of the reward for the joint action $V(\begin{bmatrix} a_i \\ a_{-i} \end{bmatrix})$ using Equation 2 and the opponent policy π_{-i} using Equation 3.

The StrOPM algorithm updates its policy along the gradient of expected reward. Actions are chosen to maximise reward based on the estimation of the opponent policy. Importantly, StrOPM does not pursue a pure Best-Response in the sense that the chosen actions are also based on changes expected in the opponent behavior due to choice of StrOPM's own actions.

3.2 StrOPM: Multi-State

Above, we introduced the StrOPM algorithm from the single state perspective. We now extend the StrOPM framework for multiple states now that the basic concepts have been introduced. In this Section, we now define the ξ function as we introduce state representations that encode recent history of play.

States Let S denote the set of states of agent i. We introduce a **transition function** $T : S \times A_i \times A_{-i} \rightarrow S$ to return the next state of agent i upon playing a (joint) action from the current state. We stretch the previously used notations to include state where relevant; i.e. $\pi_i(s, a)$ is the probability of agent i playing action a in state s.

Estimating the impact of actions We postponed the definition of ξ in the single state StrOPM framework to the definition of StrOPM in a multi-state setting. There are various possibilities to build ξ, and we present a first implementation below; more sophisticated versions of ξ are possible, for example by taking into account the history of last played joint actions over multiple steps. Here, we restrict ourselves to a simple linear extrapolation that we show in works very well already (see also Section 4). It also allows for intuitive interpretations within the domain of matrix games.

We estimate the impact of an action a_i by agent i on the policy $\pi_{-i}(s)$ of the opponent. As a first implementation of this function, we take the approach that the changes in the opponent policy are, at least in part, caused by an agents own actions. We have as state representation an encoding of the last played joint action; the current state is determined in the transition function by the last played joint action. Furthermore, we have a state-differentiated policy for the agent itself, and a state-differentiated estimation of the opponent policy. With the above information, StrOPM can track the chances in estimated policy $\pi_{-i}(s)$ over time and estimate the impact of its actions for each state.

More generally, as states (also) encode the estimated policy for the opponent as a function of the last played joint action, this information can be applied to estimate the shift in opponent policy as a function of ones own play. StrOPM estimates the change in policy as a continuation of the change in policy from estimation of the opponent policy N epochs in the past, i.e. how was the opponent policy at time $t-N$?. For state s' reached after playing action joint action $\begin{bmatrix} a_i \\ a_{-i} \end{bmatrix}$ from state s, i.e. $s' = T(s, \begin{bmatrix} a_i \\ a_{-i} \end{bmatrix})$, the change in policy for this new state, as a consequence of the action played, is estimated as a linear extrapolation of past reaction to ones own actions:

$$\xi_{i,t}(s' = T(s, \begin{bmatrix} a_i \\ a_{-i} \end{bmatrix})) = \frac{\pi_{-i,t}(s') - \pi_{-i,t-N}(s')}{N} + \pi_{i,t}(s'), \tag{9}$$

where we limit ξ to $[0, 1]$.

Calculating the best policy update action In Section 3.1, we define the value of a policy update action in Equations 6 and 7 for an agent with single state. We generalize for agents with multiple states games by considering how the value of the game will change due to an adaption in the agents own policy for the current state.

To approximate the above change in the value G of the game, we calculate for the presented results of Section 4 the value of the extensive form tree representation of the next n epochs of the game being played. The estimated value of the possible future moves of "depth" n is calculated by using Equation 4 to estimate the value of the current state. To this value, the expected reward in the reachable states in the following $n-1$ epochs is added. $G(s, 1) = val(\pi_i(s), \pi_{-i}(s))$, and

$$\begin{gathered} G(s, n+1) = val(\pi_i(s), \pi_{-i}(s)) + \\ \sum_{a_i} \sum_{a_{-i}} \pi_i(s)(a_i) \pi_{-i}(s)(a_{-i}) G(T(s, \begin{bmatrix} a_i \\ a_{-i} \end{bmatrix}), n) \end{gathered} \tag{10}$$

The StrOPM algorithm reasons about the impact of its actions by replacing $\pi_{-i}(s)$ by $\xi(s)$ in the above equations for $G(s, n)$. Call this new recursive equation G'. The marginal value of a policy update action pua_i that increases the probability of playing a_i from state s is then

$$\Delta G(pua_i) = G'(s, n)[\pi_i(s) \leftarrow \pi_i^{pua_i}] - G(s, n), \tag{11}$$

where $[\pi_i(s) \leftarrow \pi_i^{pua_i}]$ indicates that $\pi_i^{pua_i}$, the changed policy, should be used instead of $\pi_i(s)$ in calculation G'.

The above approximation of the games was sufficient for the StrOPM algorithm to estimate the impact of changes in its policy for $n = 4$. For large state spaces or novel settings, G will have to be estimated in other ways, using for example discounting of future states in the leaves of the extensive form game tree. An extensive form of the future game cannot be constructed indefinitly for increasing n. Methods like Monte Carlo sampling may be a possible direction to take; this however is ouside the scope of this paper.

3.3 Multi-State Matrix Games

For the experiments in Section 4, we use use one separate state to encode the last joint play of action from $A_i \times A_{-i}$. For the two player matrix games, the StrOPM agents hence have four states. In particular, for the iPD, the current state encodes whether the last joint action played was either $\begin{bmatrix} C \\ C \end{bmatrix}$, $\begin{bmatrix} C \\ D \end{bmatrix}$, $\begin{bmatrix} D \\ C \end{bmatrix}$, or $\begin{bmatrix} D \\ D \end{bmatrix}$ for the iPD.

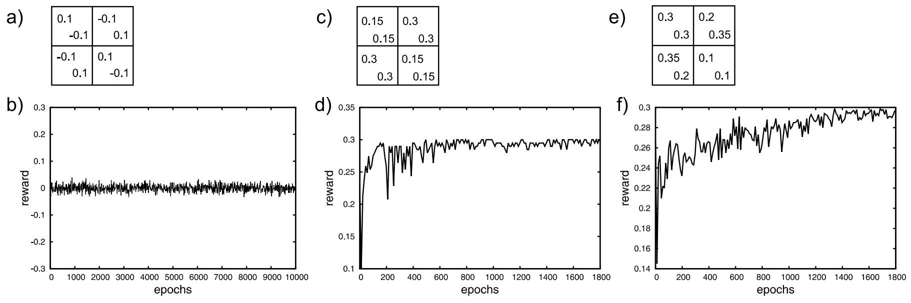


Figure 1: a) payoff matrix for "Matching Pennies", reward representation as in Table 1. b)two StrOPM players playing Matching Pennies. c,d) payoff matrix and two StrOPM players playing a Coordination Game. e,f) payoff matrix and two StrOPM players playing a game of Chicken.

4 Experiments

We first investigate the StrOPM algorithm for various well-known two-player, two-action games.

In Figures 1b,d,f, we show the results for StrOPM in a variety of games. Figure 1b shows the results for self-play for the game of "matching pennies", Figure 1d for a variant of a coordination game[4], and Figure 1f for the game of Chicken (see also Rapoport et al. (1976)). The corresponding payoff matrices

[4]In the simple coordination game, the two agents must learn to either play $\begin{bmatrix} C \\ D \end{bmatrix}$ or $\begin{bmatrix} D \\ C \end{bmatrix}$ to achieve the highest reward.

are plotted above the respective graphs in Figures 1a,c,e. Experiments are averaged over 50 runs. A learning rate of 0.01 was used for all the EMA equations of Section 3. The StrOPM algorithm looks back $N = 10$ epochs in Equation 9. Additionally, we added an $\epsilon = 0.01$ probability of the StrOPM agent taking an exploration move in each epoch to ensure all states are sufficiently sampled in play. Note that we reuse the payoff labels C and D for ease of exposition.

In the games shown in Figure 1, the StrOPM algorithm finds the optimal Nash-Equilibria rapidly, as was to be expected since these games are unproblematic in the sense that the Nash-Equilibria of the games is also a good solution for iterated play of the game. The results show that the StrOPM algorithm finds good solutions for these diverse games quickly, as it should since it is not designed with a bias for any particular matrix game.

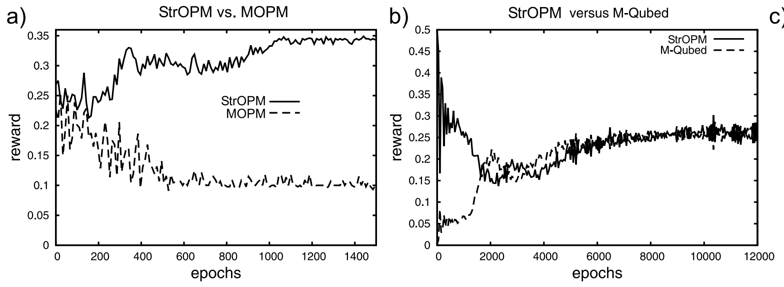
StrOPM in the iPD Of greater interest than the previous games is the iterated Prisoner's Dilemma, as it is a much harder, open problem for MARLs and is also the motivating example for the work of Crandall and Goodrich (2005).

In Figure 2a, we present results of the StrOPM algorithm for self-play in the iPD game of Figure 2a (using the payoff matrix of Table 1). To highlight the difference between a forward-thinking algorithm like StrOPM, and more reactive MOPM algorithms, we compare in Figure 1 the results for the StrOPM algorithm in self-play with that of self-play for an MOPM player in the form of a reduced variant of StrOPM that does not anticipate reactions to its actions. With the latter, we mean that the used ξ functions do not predict a change in policy of the opponent, i.e $\xi(\pi_{-i}, a_i) = \pi_{-i}$. As such, the StrOPM implementation reverts to an opponent modeler type player that simply moves its policy to a best-response to the recently observed play of the opponent.

The StrOPM learner in self-play (Figure 2a, solid line) converges to playing the (optimal) Cooperate-Cooperate ($\{C, C\}$) strategy with reward 0.35. The full cooperation equilibrium is reached as the StrOPM learner estimates that leaving the full cooperation state leads to states where more and more defection is expected. Additionally, the full cooperation state is expected to lead to more cooperation. In contrast, the MOPM variant quickly learns cooperation is risky and moves to full defection(Figure 2a, dashed line).

Furthermore, a closer study of the state-based policy of the StrOPM players reveals that the agents in self-play exhibit a learned Tit-For-Tat strategy (Axelrod, 1984). The Tit-For-Tat algorithm plays C until the opponent defects upon which a D is played followed by again C until the next defection to encourage cooperation. For a four state agent, with each state encoding the last joint action of play, the policy is to play C in state $\begin{bmatrix} C \\ C \end{bmatrix}$, play D from state $\begin{bmatrix} C \\ D \end{bmatrix}$, and C from $\begin{bmatrix} D \\ D \end{bmatrix}$. The StrOPM agents learn this strategy and play C to cooperate and reach a good equilibrium, except for occasional exploratory moves. At the same time the StrOPM players also evolve a "threat" state where defection is retaliated in order to guard against exploitation by the opponent. The need for learning a threat state as in the Tit-For-Tat play of the iPD is an interesting venue of research for iterated play of games.

StrOPM versus M-Qubed: In Figure 2b we show results for an agent using the StrOPM algorithm (using the same settings as above) playing against an agent using the M-Qubed algorithm by Crandall and Goodrich (2005). We choose M-Qubed an a state-of-the-art MARL algorithm to compare StrOPM to, as Crandall & Goodrich claim it performs on par with other state-of-the-art MARLs on simple matrix games, and is additonally capable of successfully solving the iPD game.



c)

0.35 0.35	0.5 0.0
0.0 0.5	0.1 0.1

Figure 2: a) StrOPM players playing the iPD (solid line) and the MOPM-variant (dashed line). b) A StrOPM player playing vs M-Qubed for the iPD. c) iPD payoff matrix

M-Qubed as introduced in Crandall and Goodrich (2005) operates as a mix of a Q-learning type algorithm and a pure strategy player. The algorithm learns Q-like values that encode the received reward and the discounted future reward. The algorithm investigates the policy of playing according to the Q-like values various exploration strategies. At the same time, the algorithm considers whether to play any of its actions consistently to promote possibly good interactions with the opponent. A parameter $\beta \in [0, 1]$ is used in M-Qubed and adapted in play to learn the best overall strategy; M-Qubed plays a game using the Q-values with probability $(1 - \beta)$ or it plays the highest valued action as a pure strategy with probability β. We used for M-Qubed the same settings as Crandall & Goodrich note in their work[5].

When a StrOPM player plays against a player using M-Qubed, the full average reward of 0.35 is not reached as alternating Cooperate/Defect followed by Defect/Cooperate patterns often emerged, giving an average reward of 0.25 per epoch. This is not the optimal C, C outcome, but still much better than the typical D, D outcome other MARLs arrive at. This result suggests that the switching policy in M-Qubed is repeately "tripped". The non-linear switching policy threshold is hard to approximate in StrOPM's linear opponent change

[5]These settings allowed us to successfully replicate the results in Crandall and Goodrich (2005).

model. Including more history of non-linear update terms in future extensions may solve this issue.

The trouble with precoded strategies The M-Qubed algorithm has a clear potential weakness since it has to decide between the pure (or at least precoded) strategy of playing full cooperation, and a myopic Q-Learning type strategy. We would expect it to perform badly when the optimal strategy involves a more complex strategy that is not precoded. Our StrOPM algorithm on the other hand is not biased to playing precoded strategies.

We tested this hypothesis in the game with the payoff matrix of Figure 3a, dubbed "Switching Bait", which we specifically designed so that the highest (average) joint reward is achieved when agents alternate Cooperate/Defect and Defect/Cooperate (or at least play these two joint actions equally often). Playing the Cooperate/Defect, Defect/Cooperate mixed strategy yields an average reward of 0.45 per epoch for both agents.

In experiments playing the Switching Bait game, M-Qubed was frequently not able to find the optimal mixed strategy in selfplay, as it predominantly converged to playing C or D and achieves a reward of only 0.4 per epoch. Higher rewards of $\frac{0.8+0.1}{2}$ where achived by one of the M-Qubed learners by playing C and D alternatingly, while the other M-Qubed player plays C or D continouously for an average reward of $\frac{0.1+0.4}{2}$. For the Switching Bait new game, M-Qubed often decides to play the safer strategy of full cooperation or defection.

The StrOPM algorithm, unlike M-Qubed, is not biased to playing pure strategies. For the Switching Bait game, it often finds the alternating Cooperate/Defect pattern and matches it to the opponent's Defect/Cooperate pattern in self play. Importantly, a StrOPM player playing againts an M-Qubed player was found to either achieve this mutually beneficial equilibrium or was repeatedly able to exploit M-Qubed if the opponent decided to play a (precoded) pure strategy. An example of this latter behavior is shown in Figure 3b, where the cumulative payoff achieved during play is plotted. The StrOPM algorithm achieves a higher payoff as M-Quebed opts for the safer, pure strategy. This clearly demonstrates the weakness of using precoded fixed strategies (as in M-Qubed).

5 Discussion and Conclusions

We have presented a number of important contributions in this paper. First, we argued the need for Multi-Agent learning algorithms to take into account the impact of their actions on the adaptive policy of the opponents. We then presented a novel framework that allows agents to apply the above observation to achieve desired outcomes for representative iterated matrix games, and, importantly, also for the iterated Prisoner's Dilemma. The latter is notoriously difficult to solve for current state-or-the-art MARL approaches. Our MARL framework allows an agent to learn profitable strategies for long term behavior. Finally, we

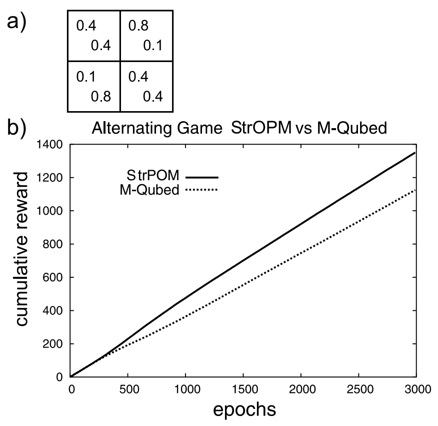


Figure 3: a) payoff matrix for "Switching Bait" game b) respective cumulative reward obtained by a StrOPM player playing against an M-Qubed player.

demonstrated that incorporating fixed precoded strategies in MARL algorithms makes agents vulnerable to exploitation, especially when mixed strategies can be optimal (of which there is a continuous spectrum). Additionally, fixed precoded strategies may not be suited to novel settings.

The important novel component of this work is the concept of learning the changes in opponents policy due ones own action. This is an idea that can in fact be incorporated into the quickly growing literature on MARL. One can foresee more and more complex nested opponent models (Hu & Wellman, 1998) to extract every bit of reward from complex games like iPD. Although perfectly learning about an opponent while at the same time perfectly learning to adjust oneself is problematic (Nachbar & Zame, 1996), there is still much scope to be "smarter" than your opponents.

Our future challenge is to apply our new concept to large, complex state spaces. One interesting direction may be to integrate our framework with new ideas about state space representations like Predictive State Representations (Wolfe, James, & Singh, 2005).

References

Axelrod, R. (1984). *The evolution of cooperation.* Basic Books, New York, NY.

Banerjee, B., & Peng, J. (2004). The role of reactivity in multiagent learning. In *Proc. 3rd AAMAS* (p. 538-545).

Bowling, M. (2004). Convergence and no-regret in multiagent learning. In *Proc. NIPS-17* (pp. 209–216).

Bowling, M. H., & Veloso, M. M. (2002). Multiagent learning using a variable learning rate. *Artificial Intelligence*, *136*(2), 215-250.

Conitzer, V., & Sandholm, T. (2003). AWESOME: A General Multiagent

Learning Algorithm that Converges in Self-Play and Learns a Best Response Against Stationary Opponents. In *Proc. 20th ICML* (p. 83-90).

Crandall, J. W., & Goodrich, M. A. (2005). Learning to compete, compromise, and cooperate in repeated general-sum games. In *Proc. 22nd ICML.*

Greenwald, A. R., & Hall, K. (2003). Correlated Q-learning. In *Proc. 20th ICML* (p. 242-249).

Hu, J., & Wellman, M. (1998). Online learning about other agents in a dynamic multiagent system. In *Proc ACM Conf. on Autonomous Agents* (p. 239-246).

Littman, M. L., & Stone, P. (2003). A polynomial-time nash equilibrium algorithm for repeated games. In *Proc. 4th ACM Conf. on Electronic Commerce* (p. 48-54).

Nachbar, J. H., & Zame, W. R. (1996). Non-computable strategies and discounted repeated games. *Economic Theory*, *8*, 103– 122.

Nash, J. (1951). Non-cooperative games. *Annals of Mathematics*, *54*, 286–295.

Rapoport, A., Guyer, M., & Gordon, D. (1976). *The 2x2 game.* MI: University of Michigan Press.

Sandholm, T., & Crites, R. (1995). Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems*, *37*, 147-166.

Shoham, Y., Powers, R., & Grenager, T. (2004). Multi-agent reinforcement learning: a critical survey. In *AAAI Fall Symposium on Artificial Multi-Agent Learning.*

Sutton, R., & Barto, A. (1998). *Reinforcement learning: An introduction.* Cambridge, MA: MIT Press.

Tesauro, G. (2003). Extending Q-learning to general adaptive multi-agent systems. In *Nips-16* (pp. 871–878).

Watkins, C. J. C. H., & Dayan, P. (1992). Technical note Q-learning. *Machine Learning*, *8*, 279-292.

Weinberg, M., & Rosenschein, J. S. (2004). Best-response multiagent learning in non-stationary environments. In *Proc. 3rd AAMAS* (p. 506-513). New York.

Wolfe, B., James, M. R., & Singh, S. (2005). Learning predictive state representations in dynamical systems without reset. In *Proc. 22nd ICML.*