Proximal Learning With Opponent-Learning Awareness

Stephen Zhao
University of Toronto and Vector Institute
stephen.zhao@mail.utoronto.ca

Chris Lu
FLAIR, University of Oxford
christopher.lu@exeter.ox.ac.uk

Roger Grosse
University of Toronto and Vector Institute
rgrosse@cs.toronto.edu

Jakob Foerster
FLAIR, University of Oxford
jakob.foerster@eng.ox.ac.uk

Abstract

Learning With Opponent-Learning Awareness (LOLA) (Foerster et al. [2018a]) is a multi-agent reinforcement learning algorithm that typically learns reciprocity-based cooperation in partially competitive environments. However, LOLA often fails to learn such behaviour on more complex policy spaces parameterized by neural networks, partly because the update rule is sensitive to the policy parameterization. This problem is especially pronounced in the opponent modeling setting, where the opponent's policy is unknown and must be inferred from observations; in such settings, LOLA is ill-specified because behaviourally equivalent opponent policies can result in non-equivalent updates. To address this shortcoming, we reinterpret LOLA as approximating a proximal operator, and then derive a new algorithm, proximal LOLA (POLA), which uses the proximal formulation directly. Unlike LOLA, the POLA updates are *parameterization invariant*, in the sense that when the proximal objective has a unique optimum, behaviourally equivalent policies result in behaviourally equivalent updates. We then present practical approximations to the ideal POLA update, which we evaluate in several partially competitive environments with function approximation and opponent modeling. This empirically demonstrates that POLA achieves reciprocity-based cooperation more reliably than LOLA.

1 Introduction

As autonomous learning agents become more integrated into society, it is increasingly important to ensure these agents' interactions produce socially beneficial outcomes, i.e. those with high total reward. One step in this direction is ensuring agents are able to navigate social dilemmas [Dawes, 1980]. Foerster et al. [2018a] showed that simple applications of independent reinforcement learning (RL) to social dilemmas usually result in suboptimal outcomes from a social welfare perspective. To address this, they introduce *Learning With Opponent-Learning Awareness* (LOLA), which actively shapes the learning step of other agents. LOLA with tabular policies learns *tit-for-tat* (TFT), i.e. reciprocity-based cooperation, in the iterated prisoner's dilemma (IPD), a 2-agent social dilemma.

However, for the same IPD setting, we show that LOLA with policies parameterized by neural networks often converges to the Pareto suboptimal equilibria of unconditional defection. This shows that the learning outcome for LOLA is highly dependent on *policy parameterization*, one factor that makes LOLA difficult to scale to higher dimensional settings. This problem is especially pronounced in the *opponent modeling* setting, where the opponent's policy is *unknown* and must be inferred

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

from observations; in such settings, LOLA is ill-specified because behaviourally equivalent opponent policies can result in non-equivalent updates and hence learning outcomes.

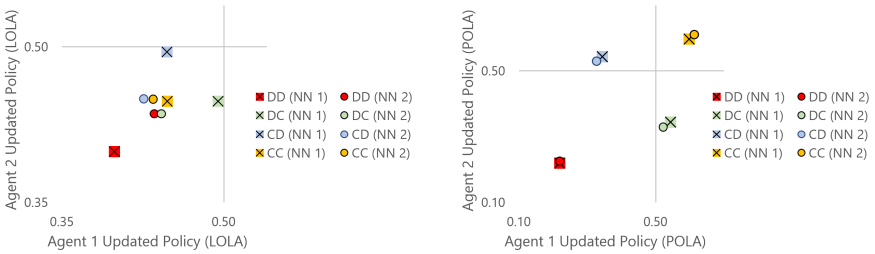

Figure 1: In the IPD (Section 4.1), we initialize two neural networks using the same architecture with different weights that produce the same policy, and plot the probability of cooperation in 4 states (DD, DC, CD, CC) after a single update of LOLA and *outer POLA* (Section 3.3). Left: using LOLA; comparing circles to crosses, the different policy parameterizations result in very different updated policies despite the same starting policies and hyperparameters. Right: using *outer POLA*; the two updated policies are much more similar.

To address these issues, we build on ideas from *proximal algorithms* [Parikh and Boyd, 2014], reinterpreting LOLA as a 1-step approximation of such a proximal operator, and then devise a new algorithm, *proximal LOLA* (POLA), which uses this proximal formulation directly. POLA replaces the gradient updates in LOLA with proximal operators based on a *proximity penalty* in policy space, which makes POLA invariant to policy parameterization. When the proximal objective has a unique optimum, behaviourally equivalent policies result in behaviourally equivalent updates.

Solving for the exact POLA update is usually intractable, so we also provide practical algorithms for settings with and without environment rollouts that rely on approximate versions of the POLA update. We show empirically that our algorithms learn reciprocity-based cooperation with significantly higher probability than LOLA in several settings with *function approximation*, even when combined with opponent modeling.

Figure 1 illustrates our core idea: LOLA is sensitive to policy parameterization, whereas POLA is not. We compare the same policy parameterized by two different sets of weights in the same neural network architecture in the IPD, showing that our approximation to the POLA update is largely parameterization independent, while the LOLA update varies depending on parameterization.

For reproducibility, our code is available at: `https://github.com/Silent-Zebra/POLA`.

1.1 Summary of Motivation, Contributions and Outline of Paper Structure

Motivation: Reciprocity-based cooperation is a desired learning outcome in partially competitive environments (Section 2.2). LOLA typically learns such behaviour in simple policy spaces, but not on more complex policy spaces parameterized by neural networks. Our goal is to more reliably learn reciprocity-based cooperation in such policy spaces.

Contributions:

- We identify and demonstrate a previously unknown problem with LOLA that helps explain the aforementioned issue: LOLA is sensitive to policy parameterization (Section 3.1).
- To address this sensitivity, we conceptually reinterpret LOLA as approximating a proximal operator, and derive a new algorithm, *ideal POLA*, which uses the proximal formulation directly (Section 3.2).
- *Ideal POLA* updates are invariant to policy parameterization, but usually intractable in practice. So, we develop approximations to the *ideal POLA* update for use in practice (Sections 3.3, 3.4).
- We demonstrate that our approximations more reliably learn reciprocity-based cooperation than LOLA in several partially competitive environments (Section 4.1), including with function approximation and opponent modeling (Sections 4.2, 4.3).

2 Background

2.1 Reinforcement Learning

We consider the standard fully-observable reinforcement learning and Markov Game formulation:

An N-agent fully observable Markov Game $\mathcal{M}$ is defined as the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A} = \{A_1, ..., \mathcal{A}_N\}$ is a set of action spaces, where $\mathcal{A}_i$ for $i \in \{1, ..., N\}$ denotes the action space for agent i, $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times ... \times \mathcal{A}_N \to \mathcal{P}(\mathcal{S})$ is a transition function mapping from states and actions to a probability distribution over next states, $\mathcal{R} = \{\mathcal{R}_1, ..., \mathcal{R}_N\}$ is a set of reward functions, where $\mathcal{R}_i : \mathcal{S} \times \mathcal{A}_1 \times ... \times \mathcal{A}_N \to \mathbb{R}$ denotes the reward function for agent i, and $\gamma \in \mathbb{R}$ is a discount factor. Each agent acts according to a policy $\pi_{\theta^i} : \mathcal{S} \to \mathcal{P}(\mathcal{A}_i)$, with parameters θ^i.

Let $\boldsymbol{\theta}$ be the concatenation of each of the individual θ^i (so for $N = 2, \boldsymbol{\theta} = \{\theta^1, \theta^2\}$) and $\pi_{\boldsymbol{\theta}}$ be the concatenation of each of the individual π_{θ^i}. Each agent's objective is to maximize its discounted expected return: $J^i(\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{s \sim P(\mathcal{S}), \mathbf{a} \sim \pi_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{T} \gamma^t r_t^i \right]$ where $\mathbf{a} \sim \pi_{\boldsymbol{\theta}}$ is a joint action $\mathbf{a} = \{a_1, ..., a_N\} \in \mathcal{A}_1 \times ... \times \mathcal{A}_n$ drawn from the policies $\pi_{\boldsymbol{\theta}}$ given the current state $s \in \mathcal{S}$, $P(\mathcal{S})$ is the probability distribution over states given the current policies and transition function, r_t^i denotes the reward achieved by agent i at time step t as defined by the reward function $\mathcal{R}_i$ and $T \in \mathbb{N}$ is the total time horizon or episode length. Throughout this paper, we assume all states are visited with non-zero probability. This is true in practice with stochastic policies, which we use throughout our experiments.

Let $L^i(\pi_{\boldsymbol{\theta}}) = -J^i(\pi_{\boldsymbol{\theta}})$. Each agent's objective is equivalently formulated as minimizing $L^i(\pi_{\boldsymbol{\theta}})$. We use this as it better aligns with standard optimization frameworks. For ease of exposition throughout this paper we consider the $N = 2$ case and consider updates from the perspective of agent 1. Updates for agent 2 follow the same structure, but with agents 1 and 2 swapped.

2.2 Reciprocity-Based Cooperation and the IPD

Reciprocity-based cooperation refers to cooperation *iff* others cooperate. Unlike *unconditional cooperation*, this encourages other self-interested learning agents to cooperate. One well-known example is the *tit-for-tat* (TFT) strategy in the IPD [Axelrod and Hamilton, 1981]. At each time step in the IPD, agents cooperate (C) or defect (D). Defecting always provides a higher individual reward compared to cooperating for the given time step, but both agents are better off when both cooperate than when both defect. The game is played repeatedly with a low probability of termination at each time step, which is modeled by the discount factor γ in RL. TFT agents begin by cooperating, then cooperate *iff* the opponent cooperated at the previous time step. Against a TFT agent, the best strategy for a self-interested agent is to cooperate at every time step. Other examples of reciprocity-based cooperation include contributing to a common good when others contribute (to avoid punishment), or reciprocating by collecting only resources that do not harm others (such as in Section 4.3). Crucially, *unconditional cooperation*, for example in the single step prisoner's dilemma or against defecting opponents in the IPD, is usually a *dominated strategy* and thus not a desired learning outcome. Thus, we do not compare against works that modify the learning objective and can in principle learn *unconditional cooperation* such as Hughes et al. [2018], Wang et al. [2018], Jaques et al. [2019], McKee et al. [2020].

2.3 Learning with Opponent-Learning Awareness (LOLA)

LOLA [Foerster et al., 2018a] introduces *opponent shaping* via a gradient based approach. Instead of optimizing for $L^1(\pi_{\theta^1}, \pi_{\theta^2})$, i.e. the loss for agent 1 given the policy parameters (θ^1, θ^2), agent 1 optimizes for $L^1(\pi_{\theta^1}, \pi_{\theta^2 - \Delta\theta^2(\theta^1)})$, the loss for agent 1 after agent 2 updates its policy with one naive learning (NL) gradient step $\Delta\theta^2(\theta^1) = \eta \nabla_{\theta^2} L^2(\pi_{\theta^1}, \pi_{\theta^2})$, where η is the opponent's learning rate. Importantly, $\Delta\theta^2$ is treated as a function of θ^1, and LOLA differentiates through the update $\Delta\theta^2(\theta^1)$ when agent 1 optimizes for $L^1(\pi_{\theta^1}, \pi_{\theta^2})$. Appendix A.1 provides more details, including pseudo-code for LOLA. Foerster et al. [2018a] also provide a policy gradient based formulation for use with environment rollouts when the loss cannot be calculated analytically, and use opponent modeling to avoid needing access to the opponent's policy parameters.

In the IPD with tabular policies, LOLA agents learn a TFT strategy, with high probability cooperating when the other agent cooperates and defecting when the other defects [Foerster et al., 2018a]. Thus, LOLA showed that *with the appropriate learning rule* self-interested agents can learn policies that lead to socially optimal outcomes in the IPD.

2.4 Proximal Point Method

Following Parikh and Boyd [2014], define the proximal operator $\text{prox}_{\lambda f} : \mathbb{R}^n \to \mathbb{R}^n$ of a closed proper convex function f as:

$$\text{prox}_{\lambda f}(v) = \arg\min_x \left(f(x) + \frac{1}{2\lambda}||x - v||_2^2 \right)$$

where $||\cdot||_2$ is the Euclidean (L2) norm and λ is a hyperparameter. If f is differentiable, its first-order approximation near v is:

$$\hat{f}_v^{(1)}(x) = f(v) + \nabla f(v)^T (x - v)$$

The proximal operator of the first-order approximation is:

$$\text{prox}_{\hat{f}_v^{(1)}}(v) = \arg\min_x \left(f(v) + \nabla f(v)^T (x - v) + \frac{1}{2\lambda}||x - v||_2^2 \right) = v - \lambda \nabla f(v)$$

which is a standard gradient step on the original function $f(v)$ with step size λ.

The proximal point method starts with an iterate x_0, then for each time step $t \in \{1, 2, ...\}$, calculates a new iterate $x_t = \text{prox}_{\lambda f}(x_{t-1})$. Gradient descent is equivalent to using the proximal point method with a first-order approximation of f.

3 Proximal LOLA (POLA)

In this section, we first explore how LOLA is sensitive to different types of changes in policy parameterization (Section 3.1). Next, we introduce *ideal POLA* (Section 3.2), a method invariant to all such changes. Lastly, we present approximations to POLA and resulting practical algorithms (Sections 3.3, 3.4).

3.1 Sensitivity to Policy Parameterization

LOLA is sensitive to policy parameterization, as it is defined in terms of (Euclidean) gradients, which are not invariant to smooth transformations of the parameters. Specifically, parameterization affects not only the convergence rate, but also which equilibrium is reached, as we illustrate with a simple toy example. Consider the case of tabular policies with one time step of memory for the IPD. There are five possible states: DD (both agents last defected), DC (agent 1 defected while agent 2 cooperated), CD (agent 1 cooperated, agent 2 defected), CC (both agents last cooperated), and the starting state. For each agent $i \in \{1, 2\}$, $\theta^i \in \mathbb{R}^5$ parameterizes a tabular policy over these 5 states, so that $\pi_{\theta^i}(s) = \text{sigmoid}(\theta^i)[s]$, where $v[s]$ denotes the element of vector v corresponding to state s. To illustrate the dependence on parameterization, we apply the invertible transformation $\mathbf{Q}^i\theta^i$ where:

$$\mathbf{Q}^1 = \begin{pmatrix} 1 & 0 & -2 & 0 & 0 \\ 0 & 1 & -2 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -2 & 1 & 0 \\ 0 & 0 & -2 & 0 & 1 \end{pmatrix}, \mathbf{Q}^2 = \begin{pmatrix} 1 & -2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & -2 & 1 & 0 & 0 \\ 0 & -2 & 0 & 1 & 0 \\ 0 & -2 & 0 & 0 & 1 \end{pmatrix}$$

Thus, agent i's policy is now $\pi_{\mathbf{Q}^i\theta^i}(s) = \text{sigmoid}(\mathbf{Q}^i\theta^i)[s]$.

Definition 3.1. For policies $\pi_{\theta^{ia}}, \pi_{\theta^{ib}}$, we say $\pi_{\theta^{ia}} = \pi_{\theta^{ib}}$ when for all $s \in \mathcal{S}$, $\pi_{\theta^{ia}}(s) = \pi_{\theta^{ib}}(s)$.

The transformation matrices $\mathbf{Q}^1, \mathbf{Q}^2$ are non-singular and represent changes of basis. Thus, for any policy π_{θ^i}, there exists some $\theta^{i\prime}$ such that $\pi_{\mathbf{Q}^i\theta^{i\prime}} = \pi_{\theta^i}$. Despite this, LOLA fails to learn TFT in this transformed policy space (Figure 3). We also observe this issue when comparing tabular LOLA to LOLA with policies parameterized by neural networks in Figure 3; changes in policy representation materially affect LOLA. Relatedly, LOLA is sensitive to different policy parameterizations with the same policy representation, as Figure 1 (left) shows. In the next subsection, we propose an algorithm that addresses these issues.

3.2 Ideal POLA

In this section, we first formalize the desired invariance property. Next, we introduce an idealized (but impractical) version of POLA, and show that it achieves the desired invariance property.

Definition 3.2. An update rule $u : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ that updates policy parameters $\theta^1 \in \mathbb{R}^n$ using auxiliary information $\theta^2 \in \mathbb{R}^m$ is invariant to policy parameterization when for any $\theta^{1a} \in \mathbb{R}^n, \theta^{1b} \in \mathbb{R}^n, \theta^{2a} \in \mathbb{R}^m, \theta^{2b} \in \mathbb{R}^m$ such that $\pi_{\theta^{1a}} = \pi_{\theta^{1b}}$ and $\pi_{\theta^{2a}} = \pi_{\theta^{2b}}$, $\pi_{u(\theta^{1a},\theta^{2a})} = \pi_{u(\theta^{1b},\theta^{2b})}$. In short, if the original policies were the same, so are the new policies (but not necessarily the new policy parameters).

To achieve invariance to policy parameterization, we introduce our new idealized version of proximal LOLA (*ideal POLA*). Similarly to PPO [Schulman et al., 2017], each player adjusts their policy to increase the probability of highly rewarded states while penalizing the distance moved in policy space. Crucially, each player also assumes the other player updates their parameters through such a proximal update. More formally, at each policy update, agent 1 solves for the following $\theta^{1\prime}$:

$$\theta^{1\prime}(\theta^1, \theta^2) = \arg\min_{\theta^{1\prime\prime}} \left(L^1(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime}(\theta^{1\prime\prime},\theta^2)}) + \beta_{\text{out}} D(\pi_{\theta^1} || \pi_{\theta^{1\prime\prime}}) \right) \tag{1}$$

where $D(\pi_{\theta^i} || \pi_{\theta^{i\prime\prime}})$ is shorthand for a general divergence defined on policies; specific choices are given in subsequent sections. For $\theta^{2\prime}$ in Equation 1, we choose the following proximal update:

$$\theta^{2\prime}(\theta^{1\prime\prime}, \theta^2) = \arg\min_{\theta^{2\prime\prime}} \left(L^2(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime\prime}}) + \beta_{\text{in}} D(\pi_{\theta^2} || \pi_{\theta^{2\prime\prime}}) \right) \tag{2}$$

For the above equations to be well defined, the $\arg\min$ must be unique; we assume this in all our theoretical analysis. If different parameters can produce the same policy, or multiple different policies have the same total divergence and loss, then the $\arg\min$ is non-unique. However, non-unique solutions is not an issue in practice, as our algorithms approximate $\arg\min$ operations with multiple gradient updates.

Theorem 3.3. *The POLA update rule $u(\theta^1, \theta^2) = \theta^{1\prime}(\theta^1, \theta^2)$, where $\theta^{1\prime}$ is defined based on Equations 1 and 2, is invariant to policy parameterization.*

The proof is in Appendix A.2. In short, this follows from the fact that all terms in the $\arg\min$ in Equations 1 and 2 are functions of policies, and not directly dependent on policy parameters.

This invariance also is a major advantage in opponent modeling settings, where agents cannot directly access the policy parameters of other agents, and must infer policies from observations. LOLA is ill-specified in such settings; the LOLA update varies depending on the assumed parameterization of opponents' policies. Conversely, POLA updates are invariant to policy parameterization, so any parameterization can be chosen for the opponent model.

To clarify the relation between LOLA and POLA, LOLA is a version of POLA that uses first-order approximations to all objectives and an L2 penalty on policy parameters rather than a divergence over policies. This follows from gradient descent being equivalent to the proximal point method with first order approximations (Section 2.4); we provide a full proof in Appendix A.3.

Exactly solving Equations 1 and 2 is usually intractable, so in the following Sections 3.3 and 3.4, we formulate practical algorithms that approximate the POLA update.

3.3 Outer POLA

To approximate *ideal POLA*, we use a first order approximation to agent 2's objective, which is equivalent to agent 2 taking a gradient step (see Theorem A.3 for details). That is, instead of finding $\theta^{2\prime}$ via Equation 2, we use $\theta^{2\prime} = \theta^2 - \Delta\theta^2$. Agent 1 thus solves for:

$$\theta^{1\prime}(\theta^1, \theta^2) = \arg\min_{\theta^{1\prime\prime}} \left(L^1(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^2 - \Delta\theta^2(\theta^{1\prime\prime})}) + \beta_{\text{out}} D(\pi_{\theta^1} || \pi_{\theta^{1\prime\prime}}) \right) \tag{3}$$

Solving the $\arg\min$ above exactly is usually intractable. For a practical algorithm, we differentiate through agent 2's gradient steps with unrolling, like in LOLA, and repeatedly iterate with gradient steps on agent 1's proximal objective until a fixed point is found; Algorithm 1 shows pseudocode. We choose $D(\pi_{\theta^1} || \pi_{\theta^{1\prime\prime}}) = \mathbb{E}_{s \sim U(\mathcal{S})}[D_{KL}(\pi_{\theta^1}(s) || \pi_{\theta^{1\prime\prime}}(s))]$, where $U(\mathcal{S})$ denotes a uniform distribution over states, as this is most analogous to an L2 distance on tabular policies, and we only test *outer POLA* on settings where tabular policies can be used.

Algorithm 1 Outer POLA 2-agent formulation: update for agent 1

Input: Policy parameters θ^1, θ^2, proximal step size α_1, learning rate η, penalty strength β_{out}
Make copy: $\theta^{1\prime\prime} \leftarrow \theta^1$
repeat
  $\theta^{2\prime\prime} \leftarrow \theta^2 - \eta \nabla_{\theta^2} L^2(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^2})$
  $\theta^{1\prime\prime} \leftarrow \theta^{1\prime\prime} - \alpha_1 \nabla_{\theta^{1\prime\prime}}(L^1(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime\prime}}) + \beta_{\text{out}}(\mathbb{E}_{s\sim U(\mathcal{S})}[D_{KL}(\pi_{\theta^1}(s)||\pi_{\theta^{1\prime\prime}}(s))]))$
until $\theta^{1\prime\prime}$ has converged to a fixed point
Output: $\theta^{1\prime\prime}$

3.4 POLA-DiCE

Outer POLA assumes we can calculate the exact loss, but often we need to estimate the loss with samples from environment rollouts. For these cases, we introduce a policy gradient version of POLA adapted to work with DiCE [Foerster et al., 2018b]. DiCE is a sample-based estimator that makes it easy to estimate higher order derivatives using backprop. More details about DiCE and its combination with LOLA are in the Appendix (A.4, A.5) and in Foerster et al. [2018b].

POLA-DiCE approximates the $\arg\min$ in *ideal POLA* (Equations 1 and 2) with a fixed number of gradient steps on the proximal objectives, where steps on the outer objective (1) differentiate through the unrolled steps on the inner objective (2). We choose $D(\pi_{\theta^i}||\pi_{\theta^{i\prime\prime}}) = \mathbb{E}_{s\sim\mathcal{S}}[D_{KL}(\pi_{\theta^i}(s)||\pi_{\theta^{i\prime\prime}}(s))]$, and approximate the expectation under the true state visitation with an average over states visited during rollouts. $s_{\leq T}$ denotes the states from a set of rollouts with T time steps: $s_{\leq T} \triangleq \{s_t : t \in \{1, ..., T\}\}$, where s_t is the state at time step t. $D(\pi_{\theta^i}, \pi_{\theta^{i\prime\prime}}|s_{\leq T}) \triangleq \frac{1}{|s_{\leq T}|} \sum_{s\in s_{\leq T}} [D_{KL}(\pi_{\theta^i}(s)||\pi_{\theta^{i\prime\prime}}(s))]$ denotes our sample based approximation to $\mathbb{E}_{s\sim\mathcal{S}}[D_{KL}(\pi_{\theta^i}(s)||\pi_{\theta^{i\prime\prime}}(s))]$. As in LOLA-DiCE, rollouts are done in a simulator, and used in the DiCE loss $\mathcal{L}^i_{\boxdot(\pi_{\theta^1}, \pi_{\theta^2})}$. For our experiments, we assume full knowledge of transition dynamics, but in principle an environment model could be used instead. Algorithm 2 provides pseudo-code for POLA-DiCE.

Algorithm 2 POLA-DiCE 2-agent formulation: update for agent 1

Input: Policy parameters θ^1, θ^2, learning rates α_1, α_2, penalty hyperparameters $\beta_{\text{in}}, \beta_{\text{out}}$, number of outer steps M and inner steps K
Initialize: $\theta^{1\prime\prime} \leftarrow \theta^1$
for m in $1...M$ **do**
  Initialize: $\theta^{2\prime\prime} \leftarrow \theta^2$
  for k in $1...K$ **do**
    Rollout trajectories with states $s^{\text{in}}_{\leq T}$ using policies $(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime\prime}})$
    $\theta^{2\prime\prime} \leftarrow \theta^{2\prime\prime} - \alpha_2 \nabla_{\theta^{2\prime\prime}}(\mathcal{L}^2_{\boxdot(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime\prime}})} + \beta_{\text{in}} D(\pi_{\theta^2}, \pi_{\theta^{2\prime\prime}}|s^{\text{in}}_{\leq T}))$
  end for
  Rollout trajectories with states $s^{\text{out}}_{\leq T}$ using policies $(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime\prime}})$
  $\theta^{1\prime\prime} \leftarrow \theta^{1\prime\prime} - \alpha_1 \nabla_{\theta^{1\prime\prime}}(\mathcal{L}^1_{\boxdot(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime\prime}})} + \beta_{\text{out}} D(\pi_{\theta^1}, \pi_{\theta^{1\prime\prime}}|s^{\text{out}}_{\leq T}))$
end for
Output: $\theta^{1\prime\prime}$

For sufficiently large inner steps K and outer steps M, and sufficiently small learning rates, POLA-DiCE iterates until a fixed point is found. Unfortunately, iterating to convergence often requires a very large amount of rollouts and memory. When $M = 1$ and $\beta_{\text{in}}, \beta_{\text{out}} = 0$, POLA-DiCE is equivalent to LOLA-DiCE [Foerster et al., 2018b].

Without opponent modeling, LOLA-DiCE and POLA-DiCE directly access the other agent's policy parameters, using those in simulator rollouts for policy updates. In the opponent modeling (OM) setting, agents can only access actions taken by the other agent, from real environment rollouts. For policy updates, agents must use learned policy models in simulator rollouts. We learn policy models by treating observed actions as targets in a supervised classification setting, as in behaviour cloning

[Ross et al., 2011]. Figure 2 illustrates the process at each time step; we keep policy models from previous iterations and update incrementally on newly observed actions.

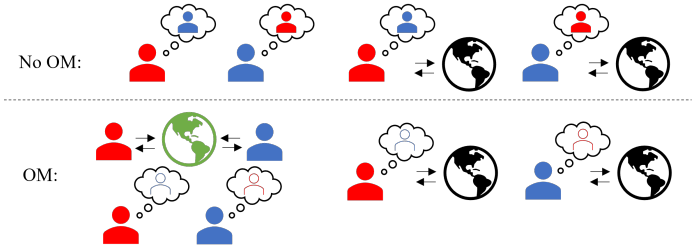

Figure 2: Illustration of the training process at each time step for LOLA-DiCE and POLA-DiCE. Without opponent modeling, each agent directly uses the other agent's policy parameters in simulator rollouts for policy updates. With opponent modeling, each agent first learns a policy model of the opponent based on observed actions from real environment rollouts. Agents then use the learned models in simulator rollouts for policy updates. We assume known dynamics, so no environment model needs to be learned for simulator rollouts.

4 Experiments

To investigate how useful our approximations to *ideal POLA* are, we run experiments to answer the following questions: 1) Does *outer POLA* learn reciprocity-based cooperation more consistently than LOLA, across a variety of policy parameterizations, in the IPD with one-step memory (Section 4.1)? 2) Does POLA-DiCE learn reciprocity-based cooperation more consistently than LOLA-DiCE in settings that require function approximation and rollouts (IPD with full history (Section 4.2) and coin game (Section 4.3))? If so, do these results still hold when using opponent modeling?

In all settings, during training, agents interact with each other only.

4.1 One-Step Memory IPD

With one-step memory, the observations are the actions by both agents at the previous time step. In this setting, we can use the exact loss in gradient updates (see Appendix B.1.1 for details).

To investigate behaviour across settings that vary the difficulty of learning reciprocity-based cooperation, we introduce a *cooperation factor* $f \in \mathbb{R}$, which determines the reward for cooperating relative to defecting. At each time step, let c be the number of agents who cooperated. Each agent's reward is $c * f/2 - \mathbb{1}[\text{agent contributed}]$. This is a specific instance of the *contribution game* from Barbosa et al. [2020], with two agents. $1 < f < 2$ satisfies the dilemma characteristics of the IPD outlined in Section 2.2. For more details on the problem setup, see Appendices B.1.2 and B.1.3.

Figure 3 compares LOLA and *outer POLA* (Section 3.3) using various f and policy parameterizations. To provide a succinct graphical representation of how well agents learn reciprocity-based cooperation, we test how often agents *find TFT*. We consider agents cooperating with each other with high probability, achieving average reward $> 80\%$ of the socially optimal, but both cooperating with probability < 0.65 if the opponent last defected, to have *found TFT*. These thresholds are somewhat arbitrary; Appendix B.1.5 provides detailed policy probabilities that support our conclusions without such thresholds.

We reproduce Foerster et al. [2018a]'s result that naive learning converges to unconditional defection, while LOLA using tabular policies finds TFT. However, changes in policy parameterization greatly hinder LOLA's ability to find TFT. In contrast, *outer POLA* finds TFT even with function approximation, and finds TFT significantly more often than LOLA in the pre-conditioned setting described in Section 3.1. Appendix B.1.4 further discusses hyperparameter settings.

Qualitatively, Figure 1 shows that in the IPD with one-step memory, $f = 1.33$, and neural network parameterized policies, *outer POLA* closely approximates the invariance provided by *ideal POLA*.

To demonstrate that POLA allows for any choice of opponent model, Appendix B.1.6 provides further experiments in the IPD with opponent modeling, using a version of POLA similar to POLA-DiCE.

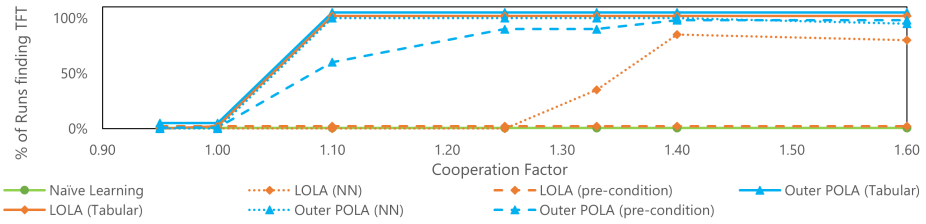

Figure 3: Comparison of LOLA and *outer POLA* in the one-step memory IPD with various f, plotting the percentage of 20 runs that *find TFT*. "NN" denotes policies parameterized by neural networks. "Pre-condition" denotes the setting in Section 3.1. Given hyperparameter tuning, tabular LOLA always finds TFT (for $f > 1$), whereas LOLA with function approximation fails on lower f. In the pre-conditioned setting, LOLA completely fails to find TFT. In contrast, *outer POLA* finds TFT regardless of whether the policy is tabular or a neural network, and retains most of its performance in the pre-conditioned setting. As sanity checks, naive learning ($\eta = 0$) always fails to find TFT, and all algorithms always defect for $f < 1$.

4.2 Full History IPD

Next, we relax the assumption that agents are limited to one-step memory and instead consider policies that condition on the entire history, which makes using function approximation and rollouts necessary. We parameterize policies using GRUs [Cho et al., 2014] and test whether POLA-DiCE still learns reciprocity-based cooperation within this much larger policy space. We use $f = 1.33$ for the reward structure. For more details on the problem setting, policy parameterization, and hyperparameters, see Appendix B.2.

Figure 4 shows results in this setting. LOLA agents sometimes learn reciprocity-based cooperation but often learn to always defect, achieving low scores against each other on average. POLA agents learn reciprocity-based cooperation much more consistently, almost always cooperating with each other but defecting with high probability if the opponent always defects. Opponent modeling works well with POLA; POLA agents behave similarly using opponent modeling instead of accessing policy parameters directly.

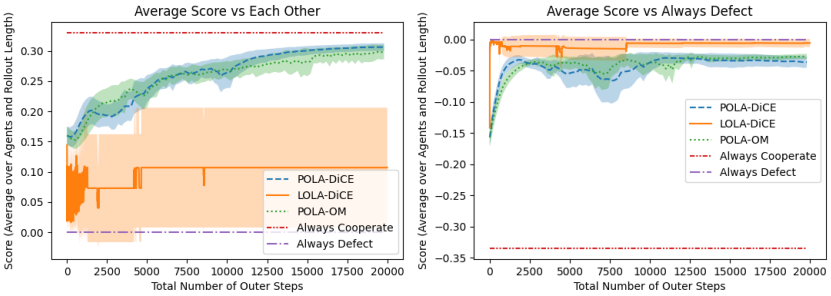

Figure 4: Comparison of LOLA-DiCE and POLA-DiCE on the IPD with GRU parameterized policies that condition on the full action history. Left: POLA-DiCE agents learn to cooperate with each other with high probability, achieving close to the socially optimal reward (always cooperate), whereas LOLA agents cooperate with each other much less consistently, often learning to always defect. Right: we test the learned policies against a hard-coded rule that always defects. POLA-DiCE agents defect against such an opponent with high probability, achieving close to the optimal reward of 0, showing POLA agents cooperate only when the other agent reciprocates. POLA-OM (POLA-DiCE with opponent modeling) agents show similar behaviour to POLA-DiCE agents. All results are averaged over 10 random seeds with 95% confidence intervals shown.

4.3 Coin Game

To investigate the scalability of POLA-DiCE under higher dimensional observations and function approximation, we test LOLA-DiCE and POLA-DiCE in the coin game setting from Lerer and Peysakhovich [2017]. The coin game consists of a 3x3 grid in which two agents, red and blue, move around and collect coins. There is always one coin, coloured either red or blue, which spawns with

the other colour after being collected. Collecting any coin grants a reward of 1; collecting a coin with colour different from the collecting agent gives the other agent -2 reward. The coin game embeds a temporally extended social dilemma; if agents defect by trying to pick up all coins, both agents get 0 total reward in expectation, whereas if agents cooperate by only picking up coins of their own colour, they achieve positive expected average reward (maximum: $\frac{1}{3}$ per time step). Appendix B.3 provides more detail. We again parameterize the agents' policies with GRUs [Cho et al., 2014].

Figure 5 shows that POLA-DiCE agents learn to cooperate with higher probability than LOLA-DiCE agents,[1] picking up a larger proportion of their own coins, and again do not naively cooperate. POLA-DiCE agents with opponent modeling also learn reciprocity-based cooperation, but slightly less so than with direct access to policy parameters, likely due to the noise from opponent modeling.

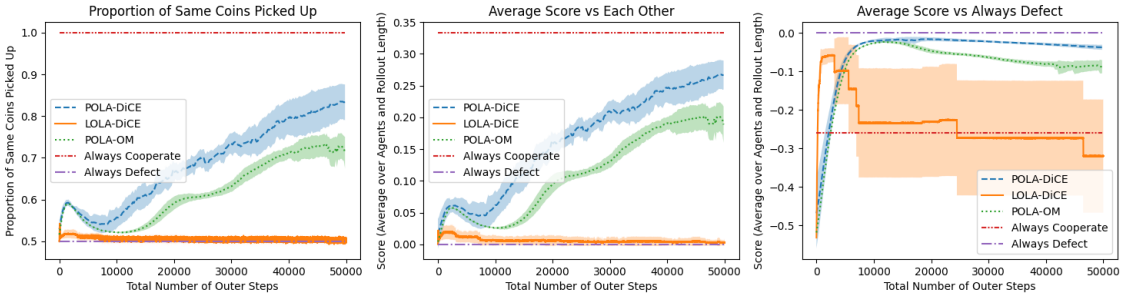

Figure 5: Comparison of LOLA-DiCE and POLA-DiCE on the coin game. Left: number of coins collected where the coin's colour matched the agent's colour, divided by the total number of coins collected. POLA agents cooperate more than LOLA agents, collecting a larger proportion of their own coloured coins. Middle: average score for the agents against each other. POLA-DiCE agents achieve close to the maximum cooperative reward. Right: we test the learned policies against a hard-coded rule that always *defects* by taking the shortest path to the coin regardless of colour. POLA agents learn to defect, achieving scores close to the optimal of 0 against such an opponent, again showing POLA agents cooperate based on reciprocity. LOLA agents quickly learn to always defect; subsequent training is highly unstable.

5 Related Work

POLA-DiCE (Section 3.4) can alternatively be motivated as replacing the policy gradient update inherent in LOLA-DiCE with a proximal-style update like that of TRPO [Schulman et al., 2015a] or the adaptive KL-penalty version of PPO [Schulman et al., 2017].

There are other extensions to LOLA that address problems orthogonal to policy parameterization sensitivity. In concurrent work, Willi et al. [2022] highlight that LOLA is inconsistent in self-play; the update each player assumes the opponent makes is different from their actual update. They propose COLA, learning update rules $\Delta\theta^1$ and $\Delta\theta^2$ that are consistent with each other. However, their notion of consistency is gradient-based and thus not invariant to policy parameterization, whereas POLA is not consistent. SOS [Letcher et al., 2018], which combines the advantages of LOLA and LookAhead [Zhang and Lesser, 2010], is also not invariant to policy parameterization.

Al-Shedivat et al. [2018] formulate a meta-RL framework with a policy gradient theorem, and apply PPO as their optimization algorithm; this is one instance of combining proximal style operators with reinforcement learning style policy gradients for optimization with higher order gradients. However, they focus on adaptation rather than explicitly modeling and shaping opponent behaviour.

MATRL [Wen et al., 2021] sets up a metagame between the original policies and new policies calculated with independent TRPO updates for each agent, solving for a Nash equilibrium in the metagame at each step. MATRL finds a best response without actively shaping its opponents' learning, so it is similar to LookAhead [Zhang and Lesser, 2010] (which results in unconditional defection) rather than LOLA.

M-FOS [Lu et al., 2022] sets up a metagame where each meta-step is a full environment rollout, and the meta-reward at each step is the cumulative discounted return of the rollout. They use model-free

[1] See Appendix B.3 for why the LOLA results cannot be directly compared with Foerster et al. [2018a]

optimization methods to learn meta-policies that shape the opponent's behaviour over meta-steps. We do not compare against it as it is concurrent work without published code at the time of writing.

Badjatiya et al. [2021] introduce *status-quo loss*, in which agents imagine the current situation being repeated for several steps; combined with the usual RL objective, this leads to cooperation in social dilemmas. However, their method learns to defect in the one-step memory IPD in the states DC and CD, and thus does not learn reciprocity-based cooperation.

Zhao et al. [2021] provide a detailed numerical analysis of the function approximation setting for two-player zero-sum Markov Games. Fiez et al. [2021] considers a proximal method for minimax optimization. In contrast to both, we consider the general-sum setting.

6 Limitations

Outer POLA and POLA-DiCE approximate *ideal POLA*, so are not completely invariant to policy parameterization; Figures 1 and 3 show how close the approximation is in the one-step memory IPD. POLA introduces additional hyperparameters compared to LOLA (e.g. $\beta_{\text{in}}, \beta_{\text{out}}$ and number of outer steps for POLA-DiCE). POLA usually requires many rollouts for each update; future work could explore ways to mitigate this. For example, in Appendix A.8 we formulate a version of POLA-DiCE closer to PPO [Schulman et al., 2017], that repeatedly trains on samples for improved sample efficiency.

7 Conclusion and Future Work

Motivated by making LOLA invariant to policy parameterization, we introduced *ideal POLA*. Empirically, we demonstrated that practical approximations to *ideal POLA* learn reciprocity-based cooperation significantly more often than LOLA in several social dilemma settings.

Future work could explore and test: policy parameterization invariant versions of LOLA extensions like COLA [Willi et al., 2022] and SOS [Letcher et al., 2018]; POLA formulations with improved sample efficiency (e.g. Appendix A.8); the N-agent version of POLA (Appendices A.9, A.10) in N-agent versions of the IPD [Barbosa et al., 2020] and larger environments such as Harvest and Cleanup [Hughes et al., 2018]; POLA formulations with adaptive penalty parameters $\beta_{\text{in}}, \beta_{\text{out}}$; connections to Mirror Learning [Kuba et al., 2022]; adapting approximations to proximal point methods such as the extra-gradient method [Mokhtari et al., 2020] to work with POLA.

We are excited and hopeful that this work opens the door to scaling LOLA and related algorithms to practical settings that require function approximation and opponent modeling.

Acknowledgements

[SZ: Please add to this section anything missing]

Resources used in this research were provided, in part, by the Province of Ontario, the Government of Canada, and companies sponsoring the Vector Institute. The Foerster Lab for AI Research (FLAIR) is grateful to the Cooperative AI Foundation for their support. We thank the anonymous NeurIPS reviewers for helpful comments on earlier versions of this paper.

References

Maruan Al-Shedivat, Trapit Bansal, Yura Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. Continuous adaptation via meta-learning in nonstationary and competitive environments. In *International Conference on Learning Representations*, 2018.

Robert Axelrod and William Donald Hamilton. The evolution of cooperation. *science*, 211(4489): 1390–1396, 1981.

Pinkesh Badjatiya, Mausoom Sarkar, Nikaash Puri, Jayakumar Subramanian, Abhishek Sinha, Siddharth Singh, and Balaji Krishnamurthy. Status-quo policy gradient in multi-agent reinforcement learning. *arXiv preprint arXiv:2111.11692*, 2021.

João Vitor Barbosa, Anna H Reali Costa, Francisco S Melo, Jaime S Sichman, and Francisco C Santos. Emergence of cooperation in n-person dilemmas through actor-critic reinforcement learning. In *Adaptive Learning Workshop (ALA), AAMAS*, 2020.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Robyn M Dawes. Social dilemmas. *Annual review of psychology*, 31(1):169–193, 1980.

Gregory Farquhar, Shimon Whiteson, and Jakob Foerster. Loaded dice: trading off bias and variance in any-order score function estimators for reinforcement learning. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 8151–8162, 2019.

Tanner Fiez, Chi Jin, Praneeth Netrapalli, and Lillian J Ratliff. Minimax optimization with smooth algorithmic adversaries. *arXiv preprint arXiv:2106.01488*, 2021.

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, 2018a.

Jakob Foerster, Gregory Farquhar, Maruan Al-Shedivat, Tim Rocktäschel, Eric Xing, and Shimon Whiteson. Dice: The infinitely differentiable monte carlo estimator. In *International Conference on Machine Learning*, pages 1529–1538. PMLR, 2018b.

Edward Hughes, Joel Z Leibo, Matthew Phillips, Karl Tuyls, Edgar Dueñez-Guzman, Antonio García Castañeda, Iain Dunning, Tina Zhu, Kevin McKee, Raphael Koster, et al. Inequity aversion improves cooperation in intertemporal social dilemmas. In *Advances in neural information processing systems*, pages 3326–3336, 2018.

Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro Ortega, DJ Strouse, Joel Z Leibo, and Nando De Freitas. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In *International conference on machine learning*, pages 3040–3049. PMLR, 2019.

Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *In Proc. 19th International Conference on Machine Learning*. Citeseer, 2002.

Jakub Grudzien Kuba, Christian Schroeder de Witt, and Jakob Foerster. Mirror learning: A unifying framework of policy optimisation. *arXiv preprint arXiv:2201.02373*, 2022.

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.

Chris Lu, Timon Willi, Christian Schroeder de Witt, and Jakob Foerster. Model-free opponent shaping. *arXiv preprint arXiv:2205.01447*, 2022.

Kevin R McKee, Ian Gemp, Brian McWilliams, Edgar A Duéñez-Guzmán, Edward Hughes, and Joel Z Leibo. Social diversity and social preferences in mixed-motive reinforcement learning. *arXiv preprint arXiv:2002.02325*, 2020.

Aryan Mokhtari, Asuman Ozdaglar, and Sarath Pattathil. A unified analysis of extra-gradient and optimistic gradient methods for saddle point problems: Proximal point approach. In *International Conference on Artificial Intelligence and Statistics*, pages 1497–1507. PMLR, 2020.

Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends in optimization*, 1(3): 127–239, 2014.

Stéphane Ross, Geoffrey J Gordon, and J Andrew Bagnell. No-regret reductions for imitation learning and structured prediction. In *In AISTATS*. Citeseer, 2011.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015a.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Jane X Wang, Edward Hughes, Chrisantha Fernando, Wojciech M Czarnecki, Edgar A Duéñez-Guzmán, and Joel Z Leibo. Evolving intrinsic motivations for altruistic behavior. *arXiv preprint arXiv:1811.05931*, 2018.

Ying Wen, Hui Chen, Yaodong Yang, Zheng Tian, Minne Li, Xu Chen, and Jun Wang. A game-theoretic approach to multi-agent trust region optimization. *arXiv preprint arXiv:2106.06828*, 2021.

Timon Willi, Johannes Treutlein, Alistair Letcher, and Jakob Foerster. Cola: Consistent learning with opponent-learning awareness. *arXiv preprint arXiv:2203.04098*, 2022.

Chongjie Zhang and Victor Lesser. Multi-agent learning with policy prediction. In *Twenty-fourth AAAI conference on artificial intelligence*, 2010.

Yulai Zhao, Yuandong Tian, Jason D Lee, and Simon S Du. Provably efficient policy gradient methods for two-player zero-sum markov games. *arXiv preprint arXiv:2102.08903*, 2021.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Appendix C.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] some assumptions like non-zero state visitation probability are stated earlier (e.g. in the background section) and apply throughout.
   (b) Did you include complete proofs of all theoretical results? [Yes] in the Appendix, with very brief proof sketches in the main paper.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Github repo with code and instructions provided in intro (`https://github.com/Silent-Zebra/POLA`).
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Either in the main paper or in the Appendix (mostly Appendix B).
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Figures 4 and 5.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Appendix B.5.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] Details in Appendix B.4.

(b) Did you mention the license of the assets? [Yes] Details in Appendix B.4.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Github repo with code provided (`https://github.com/Silent-Zebra/POLA`).

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] The only "data" is rollouts from environments.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] The only "data" is rollouts from environments.

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix: Additional Background, Derivations, and Algorithm Details

A.1 LOLA with Direct Update

Throughout this paper, we interpret LOLA as directly differentiating through $L(\pi_{\theta^1}, \pi_{\theta^2 - \Delta\theta^2(\theta^1)})$; we provide an algorithm box in Algorithm 3. We use this formulation of LOLA since it avoids using a first-order approximation around the objective as in Foerster et al. [2018a], and forms the conceptual basis for LOLA-DiCE [Foerster et al., 2018b]. In our experience, we find this formulation generally provides better and more consistent results, and is easier to implement.

Algorithm 3 LOLA direct update 2-agent formulation: exact gradient update for agent 1

Input: Policy parameters $\boldsymbol{\theta} = \{\theta^1, \theta^2\}$, learning rates α, η
Initialize: $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$
$\theta^{2\prime} \leftarrow \theta^{2\prime} - \eta \nabla_{\theta^{2\prime}} L^2(\pi_{\boldsymbol{\theta}'})$ // Preserve Computation Graph
$\theta^{1\prime} \leftarrow \theta^1 - \alpha \nabla_{\theta_1'} L^1(\pi_{\boldsymbol{\theta}'})$ // Differentiate through agent 2's update
Output: $\theta^{1\prime}$

A.2 POLA Invariance Proof

Proof. Consider arbitrary $(\theta^{1a}, \theta^{1b}, \theta^{2a}, \theta^{2b})$ such that $\pi_{\theta^{1a}} = \pi_{\theta^{1b}}$ and $\pi_{\theta^{2a}} = \pi_{\theta^{2b}}$, and consider $u(\theta^{1a}, \theta^{2a})$ and $u(\theta^{1b}, \theta^{2b})$.

$$u(\theta^{1a}, \theta^{2a}) = \arg\min_{\theta^{1\prime\prime}} \left(L^1(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime}(\theta^{1\prime\prime}, \theta^{2a})}) + \beta_{\text{out}} D(\pi_{\theta^{1a}} || \pi_{\theta^{1\prime\prime}}) \right)$$

$$u(\theta^{1b}, \theta^{2b}) = \arg\min_{\theta^{1\prime\prime}} \left(L^1(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime}(\theta^{1\prime\prime}, \theta^{2b})}) + \beta_{\text{out}} D(\pi_{\theta^{1b}} || \pi_{\theta^{1\prime\prime}}) \right)$$

Recall again that we assume all $\arg\min$ to be unique. From Equation 2:

$$\theta^{2\prime}(\theta^{1\prime\prime}, \theta^{2a}) = \arg\min_{\theta^{2\prime\prime}} \left(L^2(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime\prime}}) + \beta_{\text{in}} D(\pi_{\theta^{2a}} || \pi_{\theta^{2\prime\prime}}) \right)$$

$$\theta^{2\prime}(\theta^{1\prime\prime}, \theta^{2b}) = \arg\min_{\theta^{2\prime\prime}} \left(L^2(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime\prime}}) + \beta_{\text{in}} D(\pi_{\theta^{2b}} || \pi_{\theta^{2\prime\prime}}) \right)$$

Since $\pi_{\theta^{2a}} = \pi_{\theta^{2b}}$, we get $\pi_{\theta^{2\prime}(\theta^{1\prime\prime}, \theta^{2a})} = \pi_{\theta^{2\prime}(\theta^{1\prime\prime}, \theta^{2b})}$ because:

$$\arg\min_{\theta^{2\prime\prime}} \left(L^2(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime\prime}}) + \beta_{\text{in}} D(\pi_{\theta^{2a}} || \pi_{\theta^{2\prime\prime}}) \right) = \arg\min_{\theta^{2\prime\prime}} \left(L^2(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime\prime}}) + \beta_{\text{in}} D(\pi_{\theta^{2b}} || \pi_{\theta^{2\prime\prime}}) \right)$$

Combining this with $\pi_{\theta^{1a}} = \pi_{\theta^{1b}}$, we get $u(\theta^{1a}, \theta^{2a}) = u(\theta^{1b}, \theta^{2b})$ because:

$$\arg\min_{\theta^{1\prime\prime}} \left(L^1(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime}(\theta^{1\prime\prime}, \theta^{2a})}) + \beta_{\text{out}} D(\pi_{\theta^{1a}} || \pi_{\theta^{1\prime\prime}}) \right) =$$
$$\arg\min_{\theta^{1\prime\prime}} \left(L^1(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime}(\theta^{1\prime\prime}, \theta^{2b})}) + \beta_{\text{out}} D(\pi_{\theta^{1b}} || \pi_{\theta^{1\prime\prime}}) \right)$$

Thus, $\pi_{u(\theta^{1a}, \theta^{2a})} = \pi_{u(\theta^{1b}, \theta^{2b})}$, so u is invariant to policy parameterization. □

A.3 Proof of Connection Between POLA and LOLA

Here we show that LOLA is a version of POLA that uses first-order approximations to all objectives and an L2 penalty on policy parameters rather than a divergence over policies. We assume the loss is differentiable so first-order approximations are well-defined, and that all $\arg\min$ are unique.

Lemma A.1. *LOLA is equivalent to applying the proximal operator on a first-order approximation (around θ^1) of the modified objective $L^1(\pi_{\theta^1}, \pi_{\theta^2 - \Delta\theta^2(\theta^1)})$.*

Proof. Let $f^1(\theta^1) = L^1(\pi_{\theta^1}, \pi_{\theta^2 - \Delta\theta^2(\theta^1)})$.

Consider the first-order Taylor approximation $\hat{f}^{1(1)}_{\theta^1}$ of the modified objective function (around θ^1):[2]

$$\hat{f}^{1(1)}_{\theta^1}(\theta^{1\prime\prime}) = f^1(\theta^1) + \nabla_{\theta^1} f^1(\theta^1)^T(\theta^{1\prime\prime} - \theta^1)$$

Applying the proximal operator to the above, and using the results from Section 2.4 (with $x = \theta^{1\prime\prime}$ and $v = \theta^1$), we have:

$$\begin{aligned}\text{prox}_{\hat{f}^{1(1)}_{\theta^1}}(\theta^1) &= \arg\min_{\theta^{1\prime\prime}} \left(\hat{f}^{1(1)}_{\theta^1}(\theta^{1\prime\prime}) + \frac{1}{2\lambda}||\theta^1 - \theta^{1\prime\prime}||_2^2 \right) \\ &= \theta^1 - \lambda \nabla_{\theta^1} f^1(\theta^1) = \theta^1 - \lambda \nabla_{\theta^1} L^1(\pi_{\theta^1}, \pi_{\theta^2 - \Delta\theta^2(\theta^1)})\end{aligned}$$

which recovers the gradient update that LOLA does. □

Lemma A.2. *For any arbitrary fixed θ^1, the naive gradient step in LOLA: $\theta^2 - \Delta\theta^2 = \theta^2 - \lambda \nabla_{\theta^2} L^2(\pi_{\theta^1}, \pi_{\theta^2})$ is equivalent to applying the proximal operator on a first-order approximation (around θ^2) of the objective $L^2(\pi_{\theta^1}, \pi_{\theta^2})$.*

The proof is similar to the one above for Lemma A.1:

Proof. Let θ^1 be fixed. Let $f^2(\theta^2) = L^2(\pi_{\theta^1}, \pi_{\theta^2})$.

Apply the first-order Taylor approximation $\hat{f}^{2(1)}_{\theta^2}$ of the objective function (around θ^2), as in the following:

$$\hat{f}^{2(1)}_{\theta^2}(\theta^{2\prime\prime}) = f^2(\theta^2) + \nabla_{\theta^2} f^2(\theta^2)^T(\theta^{2\prime\prime} - \theta^2)$$

Applying the proximal operator to the above, and using the results from Section 2.4 (with $x = \theta^{2\prime\prime}$ and $v = \theta^2$), we have:

$$\begin{aligned}\text{prox}_{\hat{f}^{2(1)}_{\theta^2}}(\theta^2) &= \arg\min_{\theta^{2\prime\prime}} \left(\hat{f}^{2(1)}_{\theta^2}(\theta^{2\prime\prime}) + \frac{1}{2\lambda}||\theta^2 - \theta^{2\prime\prime}||_2^2 \right) \\ &= \theta^2 - \lambda \nabla_{\theta^2} f^2(\theta^2) = \theta^2 - \lambda \nabla_{\theta^2} L^2(\pi_{\theta^1}, \pi_{\theta^2}) = \theta^2 - \Delta\theta^2\end{aligned}$$

thus recovering the gradient update. □

Theorem A.3. *POLA reduces to LOLA when replacing the divergence on policies with the L2 norm on policy parameters and using a first order approximation of both agents' objectives.*

Proof. Recall the expression for POLA (Equations 1 and 2):

$$\begin{aligned}\theta^{1\prime}(\theta^1, \theta^2) &= \arg\min_{\theta^{1\prime\prime}} \left(L^1(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime}(\theta^{1\prime\prime}, \theta^2)}) + \beta_{\text{out}} D(\pi_{\theta^1} || \pi_{\theta^{1\prime\prime}}) \right) \\ \theta^{2\prime}(\theta^{1\prime\prime}, \theta^2) &= \arg\min_{\theta^{2\prime\prime}} \left(L^2(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime\prime}}) + \beta_{\text{in}} D(\pi_{\theta^2} || \pi_{\theta^{2\prime\prime}}) \right)\end{aligned}$$

Replace $L^2(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime\prime}})$ with a first order approximation of $L^2(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^2})$ around θ^2, replace $D(\pi_{\theta^2} || \pi_{\theta^{2\prime\prime}})$ with $||\theta^2 - \theta^{2\prime\prime}||_2^2$, and set $\beta_{\text{in}} = \frac{1}{2\lambda}$. Then by Lemma A.2, $\theta^{2\prime}(\theta^{1\prime\prime}, \theta^2) = \theta^2 - \Delta\theta^2(\theta^{1\prime\prime})$, resulting in the following equation (which is Outer POLA in Section 3.3):

$$\theta^{1\prime}(\theta^1, \theta^2) = \arg\min_{\theta^{1\prime\prime}} \left(L^1(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^2 - \Delta\theta^2(\theta^{1\prime\prime})}) + \beta_{\text{out}} D(\pi_{\theta^1} || \pi_{\theta^{1\prime\prime}}) \right)$$

Replace $D(\pi_{\theta^1} || \pi_{\theta^{1\prime\prime}})$ with $||\theta^1 - \theta^{1\prime\prime}||_2^2$, set $\beta_{\text{out}} = \frac{1}{2\lambda}$, and replace $L^1(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^2 - \Delta\theta^2(\theta^{1\prime\prime})})$ with the first order approximation of $L^1(\pi_{\theta^1}, \pi_{\theta^2})$ around θ^1, $\hat{f}^{1(1)}_{\theta^1}(\theta^{1\prime\prime})$:

$$\theta^{1\prime}(\theta^1, \theta^2) = \arg\min_{\theta^{1\prime\prime}} \left(\hat{f}^{1(1)}_{\theta^1}(\theta^{1\prime\prime}) + \frac{1}{2\lambda}||\theta^1 - \theta^{1\prime\prime}||_2^2 \right)$$

This is exactly $\text{prox}_{\hat{f}^{1(1)}_{\theta^1}}(\theta^1)$, so by Lemma A.1, this reduces to LOLA. □

[2]Note that this is different from the first-order approximation used in the original LOLA paper, as that approximation is a Taylor series expansion of $L^1(\pi_{\theta^1}, \pi_{\theta^2})$, whereas here we use a Taylor series expansion of $L^1(\pi_{\theta^1}, \pi_{\theta^2 - \Delta\theta^2(\theta^1)})$

A.4 LOLA-DiCE

DiCE [Foerster et al., 2018b] introduces an infinitely differentiable estimator for unbiased higher-order Monte Carlo gradient estimates. DiCE introduces a new operator $\boxdot$ which operates on a set of stochastic nodes $\mathcal{W}$, where:

$$\boxdot(\mathcal{W}) = \exp(\tau - \perp(\tau)) \text{ and } \tau = \sum_{w \in \mathcal{W}} \log p(w; \boldsymbol{\theta}). \tag{4}$$

$\perp$ is a stop-gradient operator (detach in Pytorch). The loss for agent 1, on a single rollout using policies $(\pi_{\theta^1}, \pi_{\theta^{2\prime}})$, with LOLA-DiCE is:

$$\mathcal{L}^1_{\boxdot(\pi_{\theta^1}, \pi_{\theta^{2\prime}})} = -\sum_{t=0}^{T} \boxdot(a_{\leq t}) \gamma^t r^1_t \tag{5}$$

where $a_{\leq t}$ denotes the set of actions all agents took at time step t or earlier. Algorithm 4 provides pseudo-code for LOLA-DiCE.

Algorithm 4 LOLA-DiCE 2-agent formulation: update for agent 1

Input: Policy parameters $\boldsymbol{\theta} = \{\theta^1, \theta^2\}$, learning rates α, η, number of inner steps K
Initialize: $\boldsymbol{\theta}' \leftarrow \boldsymbol{\theta}$
for k in $1...K$ **do**
  Rollout trajectories using policies $(\pi_{\theta^1}, \pi_{\theta^{2\prime}})$
  $\theta^{2\prime} \leftarrow \theta^{2\prime} - \eta \nabla_{\theta^{2\prime}} \mathcal{L}^2_{\boxdot(\pi_{\theta^1}, \pi_{\theta^{2\prime}})}$
end for
Rollout trajectories using policies $(\pi_{\theta^1}, \pi_{\theta^{2\prime}})$
$\theta^{1\prime} \leftarrow \theta^1 - \alpha \nabla_{\theta^1} \mathcal{L}^1_{\boxdot(\pi_{\theta^1}, \pi_{\theta^{2\prime}})}$
Output: $\theta^{1\prime}$

LOLA-DiCE [Foerster et al., 2018b] replicated the original LOLA policy gradient results [Foerster et al., 2018a] in a way that was more direct, efficient, and stable, supported by experiments with tabular policies in the one-step memory IPD.

A.5 Loaded DiCE

Loaded DiCE [Farquhar et al., 2019] rewrites the DiCE objective (5) as:

$$\mathcal{L}^1_{\boxdot(\pi_{\theta^1}, \pi_{\theta^{2\prime}})} = -\sum_{t=0}^{T} \gamma^t \left(\boxdot(a_{\leq t}) - \boxdot(a_{<t})\right) \sum_{l=t}^{T} \gamma^{l-t} r^1_l \tag{6}$$

where $a_{<t}$ denotes the set of actions all agents took before time step t.

Farquhar et al. [2019] showed Equation 6 has the same gradients as Equation 5. $\sum_{l=t}^{T} \gamma^{l-t} r^1_l$ in Equation 6 is then replaced with an advantage function: $A^1(s_t, a_t)$. Thus, *loaded DiCE* incorporates a baseline for variance reduction with DiCE.

In all our experiments with rollouts (Sections 4.2 and 4.3), we use *loaded DiCE* (Equation 6) with generalized advantage estimation (see Appendix A.6 and Schulman et al. [2015b]) for the baseline.

A.6 Generalized Advantage Estimation

Generalized advantage estimation [Schulman et al., 2015b], introduces the following advantage estimator for time step t:

$$\hat{A}_t^{GAE(\gamma,\lambda)} \triangleq (1-\lambda)(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \lambda^2 \hat{A}_t^{(3)} + ...)$$

$$\hat{A}_t^{(k)} \triangleq \sum_{l=0}^{k-1} \gamma^l \delta^V_{t+l} = -V(s_t) + r_t + \gamma r_{t+1} + ... + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k})$$

When $\lambda = 0$, the advantage estimator is the one-step Bellman residual, whereas $\lambda = 1$ is equivalent to Monte Carlo estimation (in the finite horizon setting, it extrapolates the Monte Carlo estimation with the final state value). In our finite-horizon experiments, we use a finite sum:

$$\hat{A}_t^{GAE(\gamma,\lambda,T)} \triangleq (1-\lambda)\sum_{t'=1}^{T} \lambda^{t'-1}\hat{A}_t^{(t')}$$

which is the same as what *loaded DiCE* does [Farquhar et al., 2019].

When the value function V is not completely accurate, the advantage estimator is biased. Using a lower value of λ increases bias, but reduces variance.

A.7 Training the Critic

Advantage estimation requires a value function (critic) V. To train the critic, we minimize the mean squared error of the Monte Carlo return extended by the value in the final state:

$$\mathcal{L}_{critic} = \sum_{t=0}^{T}([-V(s_t)] + [r_t + \gamma r_{t+1} + ... + \gamma^{T-t-1} r_{T-1} + \gamma^{T-t} V(s_T)])^2$$

averaged over samples in the batch.

A.8 POLA-DiCE with Repeated Training on the Same Samples

POLA-DiCE can require a lot of environment rollouts. One idea for improving sample efficiency and reducing training time is developing a modification like Schulman et al. [2017] or Kakade and Langford [2002] that allows for repeated training on the same set of samples.

We define a new operator similar to (4) from LOLA-DiCE:

$$\boxdot(\mathcal{W}) = \exp(\tau' - \perp(\tau')) \text{ and } \tau' = \sum_{w \in \mathcal{W}} p(w;\boldsymbol{\theta})/p(w;\boldsymbol{\theta}_{old}) \tag{7}$$

where $\boldsymbol{\theta}_{old}$ denotes policy parameters used on the first rollout (step $k = 1$ for the inner loop and step $m = 1$ for the outer loop).

Our new loss for agent 1 is:

$$\mathcal{L}^1_{\boxdot(\pi_{\theta^1},\pi_{\theta^{2\prime}})} = \sum_{t=0}^{T} \left(\boxdot(a_{\leq t}) - \boxdot(a_{<t})\right) \gamma^t A^1(s_t, a_t)$$

This is the *loaded DiCE* formulation (6), but with probability ratios instead of log probabilities, which lets us make multiple updates on the same batch of rollouts. We provide pseudo-code in Algorithm 5; we call this POLA-DiCE-PPO.

On the first update step ($m = 1$ and $k = 1$), $\theta_{\text{old}} = \perp(\theta)$, so $\nabla_{\theta^{2\prime\prime}}\mathcal{L}^2_{\boxdot(\pi_{\theta^{1\prime\prime}},\pi_{\theta^{2\prime\prime}})} = \nabla_{\theta^{2\prime\prime}}\mathcal{L}^2_{\boxdot(\pi_{\theta^{1\prime\prime}},\pi_{\theta^{2\prime\prime}})}$ and $\nabla_{\theta^{1\prime\prime}}\mathcal{L}^1_{\boxdot(\pi_{\theta^{1\prime\prime}},\pi_{\theta^{2\prime\prime}})} = \nabla_{\theta^{1\prime\prime}}\mathcal{L}^1_{\boxdot(\pi_{\theta^{1\prime\prime}},\pi_{\theta^{2\prime\prime}})}$. Thus, when inner steps $K = 1$, outer steps $M = 1$, and $\beta_{\text{in}}, \beta_{\text{out}} = 0$, POLA-DiCE-PPO is equivalent to LOLA-DiCE.

Algorithm 5 repeats training only on the inner loop. A similar repeated training procedure can be used on the outer loop, for even greater sample efficiency. However, this works only if the higher order gradients calculated from the initial rollout remain accurate as the policy changes.

We present this section only in the Appendix because we empirically found a larger number of outer steps to be more important than a larger number of inner steps, and that repeatedly training on the same samples for the outer loop resulted in learning reciprocity-based cooperation significantly less consistently. In future work, modifications such as periodic rollouts every $x < M$ outer steps may provide improved sample efficiency with less deterioration in performance.

This objective can also be clipped, like the clipped version of PPO [Schulman et al., 2017]; future work could explore this and compare to the KL penalty version.

Algorithm 5 POLA-DiCE-PPO 2-agent formulation: update for agent 1

Input: Policy parameters θ^1, θ^2, learning rates α_1, α_2, penalty hyperparameters $\beta_{\text{in}}, \beta_{\text{out}}$, number of outer steps M and inner steps K
Initialize: $\theta^{1\prime\prime} \leftarrow \theta^1$
for m in $1...M$ **do**
  Initialize: $\theta^{2\prime\prime} \leftarrow \theta^2$
  for k in $1...K$ **do**
    if k = 1 **then**
      Rollout trajectories using policies $(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime\prime}})$. Save states $s^{\text{in}}_{\leq T}$ from the trajectories
    end if
    $\theta^{2\prime\prime} \leftarrow \theta^{2\prime\prime} - \alpha_2 \nabla_{\theta^{2\prime\prime}}(\mathcal{L}^2_{\boxdot(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime\prime}})} + \beta_{\text{in}} D(\pi_{\theta^2}, \pi_{\theta^{2\prime\prime}} | s^{\text{in}}_{\leq T}))$
  end for
  Rollout trajectories with states $s^{\text{out}}_{\leq T}$ using policies $(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime\prime}})$
  $\theta^{1\prime\prime} \leftarrow \theta^{1\prime\prime} - \alpha_1 \nabla_{\theta^{1\prime\prime}}(\mathcal{L}^1_{\boxdot(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime\prime}})} + \beta_{\text{out}} D(\pi_{\theta^1}, \pi_{\theta^{1\prime\prime}} | s^{\text{out}}_{\leq T}))$
end for
Output: $\theta^{1\prime\prime}$

A.9 POLA N-Agent Formulation

Consider again agent 1's perspective. Agent 1 solves for the following policy:

$$\theta^{1\prime}(\theta^1, \theta^2, ..., \theta^N) = \arg\min_{\theta^{1\prime\prime}} \left(L^1(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime}(\theta^{1\prime\prime}, \theta^2, ..., \theta^N)}, ..., \pi_{\theta^{N\prime}(\theta^{1\prime\prime}, \theta^2, ..., \theta^N)}) + \beta_{\text{out}} D(\pi_{\theta^1} || \pi_{\theta^{1\prime\prime}}) \right) \tag{8}$$

where $D(\pi_{\theta^i} || \pi_{\theta^{i\prime\prime}})$ is again shorthand for a general divergence defined on policies. For $\theta^{2\prime}, ..., \theta^{N\prime}$ in Equation 8, we choose the following proximal updates:

$$\theta^{2\prime}(\theta^{1\prime\prime}, \theta^2, ..., \theta^N) = \arg\min_{\theta^{2\prime\prime}} \left(L^2(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime\prime}}, \pi_{\theta^3} ..., \pi_{\theta^N}) + \beta_{\text{in}} D(\pi_{\theta^2} || \pi_{\theta^{2\prime\prime}}) \right)$$

$$\vdots$$

$$\theta^{N\prime}(\theta^{1\prime\prime}, \theta^2, ..., \theta^N) = \arg\min_{\theta^{N\prime\prime}} \left(L^N(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^2}, ..., \pi_{\theta^{N-1}}, \pi_{\theta^{N\prime\prime}}) + \beta_{\text{in}} D(\pi_{\theta^N} || \pi_{\theta^{N\prime\prime}}) \right)$$

In short, agent 1 assumes all other agents i find the argmin of their loss, assuming the policies of other agents $j \neq i$ are fixed (using agent 1's updated policy $\theta^{1\prime\prime}$ and the original policies of all other agents).

A.10 POLA-DiCE N-Agent Formulation

Let $\pi_{\theta^{-1,-i}} \triangleq \{\pi_{\theta^2}, ..., \pi_{\theta^N}\} \setminus \pi_{\theta^i}$ be shorthand for all of the policies except those of agent 1 and agent i. Algorithm 6 provides an N-agent formulation of POLA-DiCE. The idea is similar to Appendix A.9; in the inner loop, each agent updates assuming other agents' policies are fixed.

B Appendix: Experiment and Hyperparameter Details

B.1 One-Step Memory IPD

B.1.1 Exact Loss Calculation

With one step of memory, we can directly build the transition probability matrix $\mathcal{P}$ and starting state distribution $\mathbf{p}_0$ given all agents' policies. Foerster et al. [2018a] in their Appendix A.2 derived the exact loss: $L^i = -\mathbf{p}_0^T (I - \gamma \mathcal{P})^{-1} R^i$. This can then be directly used for gradient updates.

B.1.2 Example Showing how the Cooperation Factor Recovers the IPD Reward Structure

Below we show that $f = 4/3$ recovers the IPD from Foerster et al. [2018a]:

Algorithm 6 POLA-DiCE N-agent formulation: update for agent 1

Input: Policy parameters $\theta^1, \theta^2, ..., \theta^N$, learning rates $\alpha_1, \alpha_2, ..., \alpha_N$, penalty hyperparameters $\beta^2_{\text{in}}, ..., \beta^N_{\text{in}}$, β_{out}, number of outer steps M and inner steps K
Initialize: $\theta^{1''} \leftarrow \theta^1$
for m in $1...M$ **do**
  for i in $2...N$ **do**
    Initialize: $\theta^{i''} \leftarrow \theta^i$
    for k in $1...K$ **do**
      Rollout trajectories with states $s^{\text{in}}_{\leq T}$ using policies $(\pi_{\theta^{1''}}, \pi_{\theta^{i''}}, \pi_{\theta^{-1,-i}})$
      $\theta^{i''} \leftarrow \theta^{i''} - \alpha_i \nabla_{\theta^{i''}}(\mathcal{L}^i_{\boxdot(\pi_{\theta^{1''}}, \pi_{\theta^{i''}}, \pi_{\theta^{-1,-i}})} + \beta^i_{\text{in}} D(\pi_{\theta^i}, \pi_{\theta^{i''}} | s^{\text{in}}_{\leq T}))$
    end for
  end for
  Rollout trajectories with states $s^{\text{out}}_{\leq T}$ using policies $(\pi_{\theta^{1''}}, \pi_{\theta^{2''}}, ..., \pi_{\theta^{N''}})$
  $\theta^{1''} \leftarrow \theta^{1''} - \alpha_1 \nabla_{\theta^{1''}}(\mathcal{L}^1_{\boxdot(\pi_{\theta^{1''}}, \pi_{\theta^{2''}}, ..., \pi_{\theta^{N''}})} + \beta_{\text{out}} D(\pi_{\theta^1}, \pi_{\theta^{1''}} | s^{\text{out}}_{\leq T}))$
end for
Output: $\theta^{1''}$

Let M be the number of agents who cooperate, $D(M)$ be the payoff to each defector, and $C(M)$ be the payoff to each cooperator. Then $C(2) = 1/3$, $C(1) = -1/3$, $D(1) = 2/3$, $D(0) = 0$, and we have the payoff matrix:

P1/P2	C	D
C	(1/3, 1/3)	(-1/3, 2/3)
D	(2/3, -1/3)	(0, 0)

Multiplying all rewards by 3 (equivalent to scaling the learning rate), subtracting 2 from all rewards (which preserves the ordering of policies by expected reward, and is equivalent under an accurately learned value function baseline), then gives the payoff matrix:

P1/P2	C	D
C	(-1, -1)	(-3, 0)
D	(0, -3)	(-2, -2)

which is the 2-player IPD with the reward structure given in Foerster et al. [2018a].

B.1.3 Function Approximation Setup

With function approximation, our state representation is a one-hot vector with 3 dimensions (defect, cooperate, and start state) for each agent's past action. Thus, with two agents, each agent's input is two 3-d one-hot vectors, which we flatten to a single 6-d vector. We use this representation as it has size $\Theta(N)$, whereas a single one-hot vector over all possible combinations of actions has size $\Theta(2^N)$; our representation is conducive to future experiments with large N.

B.1.4 Hyperparameter Settings

Typical neural network weight initializations (e.g. Gaussian) produce policies that are close to random (cooperating with probability close to 0.5 in each state). Our policy probability initializations are close to random throughout; this helps provide comparable results between tabular and neural network policies. Naive learning can find TFT if initialized sufficiently close to it, but never finds TFT with policies initialized close to random.

Empirically, for policies initialized close to random and for sufficiently large η, LOLA always updates the policy toward cooperating *iff* the opponent last cooperated. Thus, with sufficiently large learning rate α, LOLA finds TFT in the tabular setting (for $f > 1$). However, LOLA no longer finds TFT with large learning rates when the policy parameterization is a neural network function approximator or a transformed tabular policy (Figure 3).

For Figure 3, we show results from the best set of hyperparameters (highest % TFT found). We tuned inner and outer learning rates (η, α) for LOLA using a greedy heuristic; we increased or decreased

Table 1: Average Policies (Probability of Cooperation by State) Learned in IPD for Various Policy Parameterizations

Algorithm	Parameterization (Self & Opponent)	DD	DC	CD	CC	Start
Contribution Factor 1.1						
Naive Learning	Tabular	0.00	0.06	0.06	0.16	0.01
LOLA	Tabular	0.00	1.00	0.00	1.00	1.00
LOLA	Neural net	0.00	0.01	0.00	0.03	0.02
LOLA	Pre-condition	0.00	0.00	0.97	0.00	0.00
Outer POLA	Tabular	0.00	0.97	0.45	1.00	0.94
Outer POLA	Neural net	0.00	0.94	0.49	1.00	0.94
Outer POLA	Pre-condition	0.01	0.29	0.26	0.87	0.60
Contribution Factor 1.25						
Naive Learning	Tabular	0.00	0.07	0.07	0.18	0.02
LOLA	Tabular	0.00	1.00	0.00	1.00	1.00
LOLA	Neural net	0.00	0.02	0.01	0.06	0.02
LOLA	Pre-condition	0.00	0.00	0.96	0.00	0.00
Outer POLA	Tabular	0.08	0.97	0.18	1.00	0.95
Outer POLA	Neural net	0.01	0.87	0.47	1.00	0.85
Outer POLA	Pre-condition	0.12	0.97	0.23	1.00	0.77
Contribution Factor 1.33						
Naive Learning	Tabular	0.00	0.07	0.07	0.19	0.02
LOLA	Tabular	0.00	1.00	0.00	1.00	1.00
LOLA	Neural net	0.03	0.35	0.06	0.41	0.15
LOLA	Pre-condition	0.00	0.00	0.96	0.00	0.00
Outer POLA	Tabular	0.13	0.96	0.08	1.00	0.94
Outer POLA	Neural net	0.02	0.85	0.45	0.99	0.68
Outer POLA	Pre-condition	0.18	0.99	0.30	1.00	0.76
Contribution Factor 1.4						
Naive Learning	Tabular	0.00	0.07	0.07	0.20	0.02
LOLA	Tabular	0.00	1.00	0.00	1.00	1.00
LOLA	Neural net	0.25	0.87	0.37	0.88	0.65
LOLA	Pre-condition	0.00	0.00	0.96	0.00	0.00
Outer POLA	Tabular	0.13	0.96	0.12	1.00	0.95
Outer POLA	Neural net	0.02	0.87	0.44	0.99	0.76
Outer POLA	Pre-condition	0.21	0.98	0.33	1.00	0.75
Contribution Factor 1.6						
Naive Learning	Tabular	0.00	0.09	0.09	0.23	0.05
LOLA	Tabular	0.00	1.00	0.00	1.00	1.00
LOLA	Neural net	0.15	0.98	0.41	0.97	0.67
LOLA	Pre-condition	0.10	0.10	0.87	0.10	0.10
Outer POLA	Tabular	0.05	0.90	0.07	1.00	0.91
Outer POLA	Neural net	0.03	0.95	0.28	0.99	0.87
Outer POLA	Pre-condition	0.08	0.98	0.20	1.00	1.00

values until we no longer got better results. For *outer POLA*, we tuned η and β_{out}, and did less tuning on the outer learning rate, which matters only for convergence speed and stability. For the exact set of hyperparameters, see: `https://github.com/Silent-Zebra/POLA`.

B.1.5 Additional Detailed IPD Results

In Table 1, we show the average probability of cooperation in each state of the IPD for each of the algorithms and contribution factors shown in Figure 3. Here, DD denotes both agents last defected,

Table 2: Average Policies (Probability of Cooperation by State) Learned in IPD for Various Opponent Model Policy Parameterizations

Algorithm	Parameterization (Opponent Model)	DD	DC	CD	CC	Start
	Contribution Factor 1.33					
LOLA	Tabular	0.02	0.99	0.17	1.00	0.93
LOLA	Neural net	0.00	0.99	0.03	1.00	0.97
LOLA	Pre-condition	0.00	0.10	0.07	0.23	0.08
POLA	Tabular	0.01	0.97	0.05	1.00	0.97
POLA	Neural net	0.01	0.97	0.05	1.00	0.97
POLA	Pre-condition	0.12	0.94	0.02	1.00	0.85

DC denotes the agent last defected while the opponent last cooperated, CD denotes the agent last cooperated while the opponent last defected, CC denotes both agents last cooperated, and Start is the starting state. Numbers are averaged over 20 runs and averaged over both agents. *Outer POLA* learns reciprocity-based cooperation across contribution factor and policy parameterization settings much more consistently than LOLA.

B.1.6 IPD Experiments with Varying Opponent Model Parameterizations

We revisit the IPD with opponent modeling, considering agents with tabular policies but varying parameterizations of opponent models: tabular, neural network function approximation, and pre-conditioned tabular models. These parameterizations are the same as described previously except for the pre-conditioned tabular model, for which we now use:

$$\mathbf{Q}^1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 & 0 \\ -2 & 0 & 1 & 0 & 0 \\ -2 & 0 & 0 & 1 & 0 \\ -2 & 0 & 0 & 0 & 1 \end{pmatrix}, \mathbf{Q}^2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 & 0 \\ -2 & 0 & 1 & 0 & 0 \\ -2 & 0 & 0 & 1 & 0 \\ -2 & 0 & 0 & 0 & 1 \end{pmatrix}$$

This set of transformations again only changes basis and lets us illustrate the difference between LOLA and POLA. We show results in Table 2. LOLA fails to learn reciprocity-based cooperation for the pre-conditioned tabular opponent model, whereas POLA learns reciprocity-based cooperation across all parameterizations.

LOLA learns reciprocity-based cooperation well even with a neural net opponent policy. We believe this is because the inner player policy update (usually towards defecting in each state) is less sensitive to policy parameterization than the outer player policy update which requires second order gradients and learning reciprocity.

The version of POLA we use here is similar to POLA-DiCE (Algorithm 2) except with exact losses and the uniform distribution KL penalty; we provide pseudocode in Algorithm 7.

Algorithm 7 POLA direct approximation 2-agent formulation: update for agent 1

Input: Policy parameters θ^1, θ^2, learning rates α_1, α_2, penalty hyperparameters $\beta_{\text{in}}, \beta_{\text{out}}$, number of outer steps M and inner steps K
Initialize: $\theta^{1\prime\prime} \leftarrow \theta^1$
for m in $1...M$ **do**
  Initialize: $\theta^{2\prime\prime} \leftarrow \theta^2$
  for k in $1...K$ **do**
    $\theta^{2\prime\prime} \leftarrow \theta^{2\prime\prime} - \alpha_2 \nabla_{\theta^{2\prime\prime}}(L^2(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime\prime}}) + \beta_{\text{in}}(\mathbb{E}_{s\sim U(\mathcal{S})}[D_{KL}(\pi_{\theta^2}(s)||\pi_{\theta^{2\prime\prime}}(s))]))$
  end for
  $\theta^{1\prime\prime} \leftarrow \theta^{1\prime\prime} - \alpha_1 \nabla_{\theta^{1\prime\prime}}(L^1(\pi_{\theta^{1\prime\prime}}, \pi_{\theta^{2\prime\prime}}) + \beta_{\text{out}}(\mathbb{E}_{s\sim U(\mathcal{S})}[D_{KL}(\pi_{\theta^1}(s)||\pi_{\theta^{1\prime\prime}}(s))]))$
end for
Output: $\theta^{1\prime\prime}$

We use 100-200 inner steps for POLA and 1 outer step; since we are only changing the opponent model parameterization, invariance on the inner loop is most important. This is a fairly small number of steps which does not achieve full invariance, causing results to vary slightly across parameterizations. We

assume unlimited batch size for learning the opponent model, which is equivalent to directly learning from the policy, to separate the effects of noise from opponent modeling. Detailed hyperparameters are available in the codebase (`https://github.com/Silent-Zebra/POLA`).

B.2 IPD Full History Details

We use the state representation as discussed in Appendix B.1.3. Agents condition actions on the entire state history up to the current time step. Policies are parameterized by a fully connected input layer with 64 hidden units, a ReLU nonlinearity, a GRU cell, and a fully connected output layer. We use a batch size of 2000 (parallel environment rollouts), discount rate $\gamma = 0.96$, and rollout for $T = 50$ steps in the environment. For both LOLA-DiCE and POLA-DiCE, we use a simple gradient step on the inner loop and the Adam optimizer with default betas on the outer loop, as LOLA-DiCE [Foerster et al., 2018b] does.

For all algorithms, we use *loaded DiCE* (Appendix A.5) with GAE (Appendix A.6) with $\lambda = 1$, an outer learning rate of 0.003, and a learning rate for the critic (value function) of 0.0005. For LOLA-DiCE, we use an inner learning rate of 0.05, but we also tried the values: [0.005, 0.015, 0.02, 0.03, 0.07, 0.1, 0.2] and got similar or worse results. We also tried a few settings with lower and higher outer learning rates and value function learning rates, and 2 inner steps; none learned reciprocity-based cooperation more consistently.

For POLA-DiCE, we use 2 inner steps and 200 outer steps, with $\beta_{\text{in}} = 10$ and $\beta_{\text{out}} = 100$, and an inner learning rate of 0.005. Results are not very sensitive to β_{out}; we got similar results with $\beta_{\text{out}} = 30$ and $\beta_{\text{out}} = 200$. We did not extensively tune hyperparameters for POLA-DiCE in this setting, so it is likely that similar results can be reproduced with fewer outer steps. Results are more sensitive to the inner learning rate and β_{in}, but we expect using more inner steps would lessen this sensitivity. One challenge with taking more inner steps is the memory requirement, which forces a tradeoff with batch size. We update the critic after each policy update on both the inner and outer loop (for the corresponding agent or opponent model).

To learn the opponent model in POLA-OM, we use 200 environment rollouts (of 2000 batch size each) between each set of POLA updates. The opponent model architecture is the same as the agent's. We learn the opponent's value function in the same way as the agent's own value function, but with the reward and states of the other agent. We use learning rates of 0.005 for the policy model and 0.0005 for the value model. For other hyperparameters, we use the same settings for POLA-DiCE with and without opponent modeling.

In Figure 4 we choose the number of outer steps as the x-axis because the outer steps are policy updates that are actually made; inner step updates are not saved, and are used only in the gradient calculation of outer step updates. Each inner step currently requires an environment rollout, though this can be mitigated in future work (e.g. Appendix A.8), another reason why we consider outer steps more representative of sample efficiency. Strictly comparing environment rollouts would horizontally stretch the lines for POLA-DiCE relative to LOLA-DiCE by a factor of 1.5 (3 environment rollouts per outer step for POLA-DiCE vs. 2 for LOLA-DiCE); this would not change the conclusions drawn in the paper.

B.3 Coin Game Details

Our environment implementation adheres to Figure 3 in Foerster et al. [2018a] where two agents that step on the same coin at the same time both collect the coin. In previous experiments, splitting the coin 50-50 between agents gave very similar results.

The LOLA results in Figure 5 cannot be directly compared with those in Foerster et al. [2018a] for several reasons. Our implementation of the coin game environment fixes bugs such as ties always being broken in favour of the red agent (see: https://github.com/alshedivat/lola/issues/9), which make the original results irreproducible. We use LOLA-DiCE instead of the original LOLA-PG formulation. We rollout for fewer steps to reduce computation time and memory requirements. Our policy parameterization has more hidden units and uses a GRU, which should make it more expressive than the originally used RNN, and may also make optimization more difficult.

Same as the IPD with full history, our GRUs use 64 hidden units with a fully connected input layer with ReLU nonlinearity and a linear output layer. We use a batch size of 2000 (parallel environment

rollouts), discount rate $\gamma = 0.96$, and rollout for $T = 50$ steps in the environment. For both LOLA-DiCE and POLA-DiCE, we use a simple gradient step on the inner loop and the Adam optimizer with default betas on the outer loop, as LOLA-DiCE [Foerster et al., 2018b] does.

For all algorithms, we use *loaded DiCE* (Appendix A.5) with GAE (Appendix A.6) with $\lambda = 1$, an outer learning rate of 0.003, and a learning rate for the critic (value function) of 0.0005. For LOLA-DiCE, we use an inner learning rate of 0.003, but we also tried the values: [0.001, 0.005, 0.01, 0.02, 0.05, 0.1, 0.2] and got similar results. We also tried a few settings with lower and higher outer learning rates and value function learning rates, and 2 inner steps; none learned reciprocity-based cooperation more consistently.

For POLA-DiCE, we use 2 inner steps and 200 outer steps, with $\beta_{\text{in}} = 5$ and $\beta_{\text{out}} = 150$, and an inner learning rate of 0.02. We update the critic after each policy update on both the inner and outer loop (for the corresponding agent or opponent model).

To learn the opponent model in POLA-OM, we use 200 environment rollouts (of 2000 batch size each) between each set of POLA updates. The opponent model architecture is the same as the agent's. We learn the opponent's value function in the same way as the agent's own value function, but with the reward and states of the other agent. We use learning rates of 0.005 for the policy model and 0.0005 for the value model. For POLA-OM, we use 4 inner steps with $\beta_{\text{in}} = 10$ and an inner learning rate of 0.01; other hyperparameters are the same as POLA-DiCE without opponent modeling. The additional inner steps and higher β_{in} provide greater invariance to the opponent model, which helps in the coin game. We tried a few settings with even more inner steps but those did not learn reciprocity-based cooperation more consistently. We believe this may be due to memory constraints forcing smaller batch sizes (and thus more noise from environment rollouts) with more inner steps.

In Figure 5 we choose the number of outer steps as the x-axis because the outer steps are policy updates that are actually made; inner step updates are not saved, and are used only in the gradient calculation of outer step updates. Each inner step currently requires an environment rollout, though this can be mitigated in future work (e.g. Appendix A.8), another reason why we consider outer steps more representative of sample efficiency. Strictly comparing environment rollouts would horizontally stretch the lines for POLA-DiCE relative to LOLA-DiCE by a factor of 1.5 (3 environment rollouts per outer step for POLA-DiCE vs. 2 for LOLA-DiCE); this would not change the conclusions drawn in the paper.

B.4 Code Details

Parts of code were adapted from `https://github.com/alexis-jacq/LOLA_DiCE` [Foerster et al., 2018b] and `https://github.com/aletcher/stable-opponent-shaping` [Letcher et al., 2018]. Both use the MIT license, which grants permission free of charge for subsequent use, modification, and distribution.

B.5 Compute Usage

For the IPD with one-step memory (Section 4.1), experiments were run on CPUs provided free of charge by Google Colaboratory. Most experiments required only a small amount of compute (taking minutes to run).

For the IPD with full history (Section 4.2) and the coin game (Section 4.3), experiments were run on GPUs on an internal cluster. GPUs were either NVIDIA Tesla T4 or NVIDIA Tesla P100. Coin game experiments took around 1 full day (24 hours) to run for 1 seed on 1 GPU, whereas the IPD with full history experiments took around 8-10 hours for each seed.

The total amount of compute used, including during the experimentation phase, was significantly higher than that used for Figures 4 and 5.

C Societal Impact

We do not anticipate any immediate societal impact from this work; at the time of writing, there is no direct real world application. That said, we hope this work helps produce socially beneficial outcomes when autonomous learning agents interact, which is critical for future real-world

deployment. However, while opponent shaping helps in the social dilemma settings we tested, it could cause undesirable consequences in other settings. For example, pricing algorithms learning reciprocity-based cooperation would be tantamount to collusion. In such cases, POLA could learn undesirable behaviour in a way that is invariant to policy parameterization. Overall though, we expect the potential positive impact of our work to outweigh the potential negative impact.