Learning with Learning Awareness using Meta-Values

Tim Cooijmans [1] Milad Aghajohari [1] Aaron Courville [1]

Abstract

Gradient-based learning in multi-agent systems is difficult because the gradient derives from a first-order model which does not account for the interaction between agents' learning processes. LOLA (Foerster et al., 2018a) accounts for this by differentiating through one step of optimization. We extend the ideas of LOLA and develop a fully-general value-based approach to optimization. At the core is a function we call the meta-value, which at each point in joint-policy space gives for each agent a discounted sum of its objective over future optimization steps. We argue that the gradient of the meta-value gives a more reliable improvement direction than the gradient of the original objective, because the meta-value derives from empirical observations of the effects of optimization. We show how the meta-value can be approximated by training a neural network to minimize TD error along optimization trajectories in which agents follow the gradient of the meta-value. We analyze the behavior of our method on the Logistic Game (Letcher, 2018) and on the Iterated Prisoner's Dilemma.

1. Introduction

Multi-agent reinforcement learning (Busoniu et al., 2008) has found success in two-player zero-sum games (Mnih et al., 2015; Silver et al., 2017), cooperative settings (Lauer, 2000; Matignon et al., 2007; Foerster et al., 2018b; Panait & Luke, 2005), and mixed settings with intra-team cooperation and inter-team competition (Lowe et al., 2017). General-sum games, however, have proven to be a formidable challenge.

The classic example is the Prisoner's Dilemma, a matrix game in which two players must decide simultaneously whether to cooperate or defect. Both players would prefer for the other player to cooperate, however neither player by themselves wants to cooperate lest they be exploited. In fact, defection is a dominating strategy – it yields higher payoff regardless of what the opponent does – and learning agents will quickly find defect-defect, the only Nash equilibrium of the game.

The game becomes qualitatively different when it is infinitely repeated and players can see what their opponents did in previous rounds: this is known as the Iterated Prisoner's Dilemma (IPD). This introduces the ability to retaliate against defection, which enables players to cooperate with limited risk of exploitation, for they can always switch to defecting if the opponent does not play along. The most well-known retaliatory strategy is tit-for-tat (Axelrod & Hamilton, 1981), which cooperates initially and from then on simply plays whatever the opponent played on the previous round.

The naive application of gradient descent (see Section 2.1) fails to find tit-for-tat on the IPD unless initialized sufficiently close to it (Foerster et al., 2018a). Instead, it converges to always-defect, to the detriment of both players. Similarly, the naive application of Q-learning (Watkins & Dayan, 1992) produces questionable results, often converging to solutions where one agent is systematically exploited by the other (Sandholm & Crites, 1996).

The problem of coordination in general-sum games has received considerable attention over the years (Busoniu et al., 2008; Gronauer & Diepold, 2022). Several works approach the problem through modifications of the objective, e.g. to share reward (Baker, 2020), to explicitly encourage interaction (Jaques et al., 2019) or encourage fairness (Hughes et al., 2018). While these approaches may work, they may achieve cooperation for the wrong reasons. We instead are interested in achieving cooperation only where *self-interest* warrants it. Otherwise our policies may end up exploitable by other self-interested agents. Moreover, an algorithm that yields cooperation for altruistic reasons simply passes the buck: an agent that uses such a learning algorithm may be exploitable by agents that use purely self-interested learning algorithms.

Some approaches (Al-Shedivat et al., 2017; Lu et al., 2022; Kim et al., 2022) seek a meta-policy, i.e. an outer policy for

[1]Mila, Université de Montréal. Correspondence to: Tim Cooijmans <cooijmans.tim@gmail.com>.

Workshop on New Frontiers in Learning, Control, and Dynamical Systems at the International Conference on Machine Learning (ICML), Honolulu, Hawaii, USA, 2023. Copyright 2023 by the author(s).

choosing the inner policy played in the game. We instead seek an algorithm that produces a fixed policy that can be deployed without further learning. Although the method we propose does produce a meta-policy, it serves only to help us find good inner policies. Scoping the problem this way frees us to make assumptions that would otherwise be unrealistic. We will assume control over the entire joint learning process. In particular, during training, agents have access to each other's parameters and learning algorithms. After training, the agents can be deployed outside of this context.

We take inspiration from the recent work Learning with Opponent-Learning Awareness (LOLA (Foerster et al., 2018a;c)), the first general learning algorithm to find tit-for-tat. LOLA mitigates the problems of simultaneous gradient descent by looking ahead: it simulates a naive update and evaluates the objective at the resulting point. By differentiating through the naive update, LOLA effectively uses second-order information to account for the opponent's learning process.

LOLA has a few known issues:

- It fails to preserve the stable fixed points (SFPs) of the game (Letcher et al., 2018). This is of mainly theoretical interest, as the SFPs remain close for small learning rates.
- It is not invariant to policy parameterization; Proximal LOLA (POLA (Zhao et al., 2022)) addresses this by replacing all gradient steps by proximal steps, which however need to be computed by solving an optimization problem.
- It is not consistent, in the sense that the simulated updates do not match the actual updates. Consistent LOLA (COLA (Willi et al., 2022)) gets around this with a model of the update that is trained to satisfy a consistency loss.
- It is not able to look far into the future to accounting for multiple optimization steps. Extrapolating for multiple steps is not effective (Foerster et al., 2018a;c), and neither is increasing the learning rate of the simulated update.

It is this last issue that we aim to address in this work. Our contributions are the following:

- We propose a general framework for learning with learning awareness. Section 2 discusses the surrogate used by LOLA and its derivatives. In Section 3 we propose a slightly different surrogate that is naturally consistent and readily accounts for longer-term and higher-order interactions.
- We demonstrate the importance of looking far ahead in Section 5.1, where our algorithm is the only one to find the Pareto-efficient solution regardless of initialization.
- We show in Section 5.2 that our algorithm finds nonexploitable cooperation, and like LOLA, is able to *shape* naive opponents into exploitable cooperation.

Code to reproduce the experiments will be made available at `https://github.com/MetaValueLearning/MetaValueLearning`.

2. Differentiable Games

We consider differentiable games $f : \mathbb{R}^{2\times n} \longmapsto \mathbb{R}^2$ that map pairs of policies to pairs of expected returns,

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = f\begin{pmatrix} x_1 \\ x_2 \end{pmatrix},$$

or $y = f(x)$ for short. For simplicity, we assume $x_1, x_2 \in \mathbb{R}^n$ are real-valued parameter vectors that represent policies through some fixed parametric class (e.g. a lookup table or a particular neural network architecture).

For notational convenience in working with the concatenated vector x we will define the slicing matrices

$$S_1 = \begin{pmatrix} I_n & 0_n \end{pmatrix}, S_2 = \begin{pmatrix} 0_n & I_n \end{pmatrix},$$

with I_n being the $n \times n$ identity matrix and 0_n the $n \times n$ zero matrix. Now $x_1 = S_1 x, x_2 = S_2 x$ and $x = S_1^\top x_1 + S_2^\top x_2$.

2.1. Naive Learning

Under naive learning (also known as "simultaneous gradient descent"), agents update their policies according to

$$x^{(t+1)} = x^{(t)} + \alpha \bar{\nabla} f(x^{(t)}) \tag{1}$$

where $\bar{\nabla} f(x) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(x) & \frac{\partial f_2}{\partial x_2}(x) \end{pmatrix}^\top$ denotes the simultaneous gradient and α is a learning rate.

Naive learning is the straightforward application of standard gradient descent which works extremely well when optimizing a single objective, e.g. a single-agent system or a supervised learning problem. However, the gradient derives from a local first-order model – for each element of x, it reflects the change in f that can be attained by modifying that element, but only if all other elements remain constant. For optimization problems with a single objective, this assumption can profitably be violated, but when different elements of x optimize different objectives, the gradient generally fails to be a reliable direction of improvement.

2.2. Looking Ahead

A number of approaches in the literature aim to address these issues by "looking ahead", considering not just the current parameter values $x^{(t)}$ but also an extrapolation based on an imagined update. They essentially replace the game f with a surrogate game $\tilde{f}$ that evaluates f after an imagined naive update with learning rate α:[1]

$$\tilde{f}(x) = f(x + \alpha \bar{\nabla} f(x)). \tag{2}$$

This surrogate was to our knowledge first suggested with LookAhead (Zhang & Lesser, 2010), though in computing the associated update $\bar{\nabla} \tilde{f}$, the authors considered $\bar{\nabla} f(x)$ constant with respect to x. LOLA (Foerster et al., 2018a) introduced the idea of differentiating through the imagined update $\bar{\nabla} f(x)$, thus incorporating second-order information that accounts for interactions between the learning of agents.

Unfortunately, this surrogate trades one assumption for another: while it no longer assumes players to stand still, it now assumes players to update according to naive learning. The LOLA authors also proposed Higher-Order LOLA (HOLA (Foerster et al., 2018a)), where HOLA0 assumes opponents are fixed, HOLA1 assumes opponents are naive learners, HOLA2 assumes opponents use LOLA, and so on. Nevertheless there is always a gap where each player assumes it looks one step further ahead than their opponent. To avoid such an assumption, we should like to use a consistent surrogate such as

$$\tilde{f}^\star(x) = f(x + \alpha \bar{\nabla} \tilde{f}^\star(x)), \tag{3}$$

however this is an implicit equation; it is unclear how to obtain the associated update $\bar{\nabla} \tilde{f}^\star(x)$.

COLA (Willi et al., 2022) solves the implicit equation by replacing the surrogate with a model $\hat{g}(x;\theta) \approx \bar{\nabla} \tilde{f}^\star(x)$. The model is trained to satisfy

$$\hat{g}(x;\theta) = \bar{\nabla} f(x + \alpha \hat{g}(x;\theta)) \tag{4}$$

by minimizing the squared error between both sides. When the equation is tight, $\hat{g}(x;\theta) = \bar{\nabla} \tilde{f}^\star(x)$ which lets us train policies according to the consistent surrogate of Equation (3).

3. Meta-Value Learning

We now describe our method. First we introduce the meta-value function, a consistent and far-sighted surrogate, to be used in place of that of Equation 2. Next, we make a connection to reinforcement learning, which yields a straightforward way of approximating the surrogate.

[1]LookAhead (Zhang & Lesser, 2010), LOLA (Foerster et al., 2018a) and follow-up work differ slightly from Equation (2) in that each player extrapolates only their opponent and not themselves. We provide an exact formulation of LOLA and COLA in Appendix B.

3.1. The Meta-Value Function

We propose to use the surrogate

$$V(x) = f(x) + \gamma V(x + \alpha \bar{\nabla} V(x)) \tag{5}$$

which consists of the original objective f plus a discounted sum of objective values at future optimization iterates. Like the popular surrogate from Equation (2), it looks ahead in optimization time, but it does so in a way that is consistent and naturally covers multiple steps.

When agent policies update according to

$$x^{(t+1)} = x^{(t)} + \alpha \bar{\nabla} V(x),$$

then

$$V(x^{(t)}) = \sum_{\tau=t}^{\infty} \gamma^{\tau - t} f(x^{(\tau)}),$$

and hence differentiating $V(x^{(t)})$ corresponds to differentiating through many steps of optimization.

3.2. Connection to Q-Learning

This approach has a straightforward reinforcement learning interpretation: optimization takes place in a continuous spatial environment in which agents move around by making small modifications Δx to their parameters x. The transition is deterministic and simply adds the joint action Δx to the state: the next state is $x' = x + \Delta x$. The reward is deterministically $f(x')$, the expected return on the inner game f.

As the environment is simple and deterministic, we can equate the state-value function $V(x)$ with a pair of state-action Q-functions, one for each player:

$$\begin{pmatrix} Q_1(x, \Delta x_1) \\ Q_2(x, \Delta x_2) \end{pmatrix} = \begin{pmatrix} V_1(x + S_1^\top \Delta x_1) \\ V_2(x + S_2^\top \Delta x_2) \end{pmatrix}.$$

It is uncommon to use state-action values when the actions are continuous, as finding the greedy action involves an argmax that is usually intractible. However, since we are doing local optimization, we are satisfied with a *local* argmax, which is what the simultaneous gradient $\bar{\nabla} V(x)$ gives us.

Our method is thus related to independent Q-learning (Watkins & Dayan, 1992; Busoniu et al., 2008), which we must point out is not known to converge in general-sum games. It nevertheless does appear to converge reliably in practice, and we conjecture that applying it on the level of optimization effectively simplifies the interaction between the agents' learning processes.

3.3. Learning Meta-Values

We do not have direct access to the meta-value function, but we can learn an approximation $\hat{V}(x;\theta)$ with parameters θ.

In its simplest form, the learning process follows a nested loop. In the inner loop, we collect a policy optimization trajectory according to

$$x^{(t+1)} = x^{(t)} + \alpha \bar{\nabla} \hat{V}(x^{(t)}; \theta). \tag{6}$$

Then in the outer loop, we train $\hat{V}$ by minimizing the TD error

$$\sum_t \|\hat{f}(x^{(t)}) + \gamma \hat{V}(x^{(t+1)}; \theta) - \hat{V}(x^{(t)}; \theta)\|^2 \tag{7}$$

along the trajectory. Here $\hat{f}(x^{(t)})$ is typically an empirical estimate of the expected return $f(x^{(t)})$ based on a batch of Monte-Carlo rollouts.

While we write $\hat{V}(x; \theta)$ for convenience, this should be understood to be a separate model for each agent:

$$\hat{V}(x; \theta) = \begin{pmatrix} \hat{V}_1(x; \theta_1) \\ \hat{V}_2(x; \theta_2) \end{pmatrix}, \quad \theta = \begin{pmatrix} \theta_1 \\ \theta_2 \end{pmatrix}.$$

In our experiments we share parameters $\theta_1 = \theta_2$ to exploit the symmetry of the games and optimization processes (see Appendix D).

Once trained, $\hat{V} \approx V$ and we can use $\bar{\nabla}\hat{V}$ in place of the implicit $\bar{\nabla}V$. Thus rather than modeling gradient directly (as COLA does), we model scalars and estimate the gradient of the scalar by the gradient of the estimated scalar. The resulting algorithm is related to Value-Gradient Learning (Fairbank & Alonso, 2012), but applied on the meta-level.

4. Practical Considerations

We use a number of established techniques to improve the dynamics of training value functions (Hessel et al., 2018). The prediction targets in Equation 7 are computed with a target network (Mnih et al., 2015) that is an exponential moving average of the parameters θ. We use distributional reinforcement learning with quantile regression (Dabney et al., 2018). Instead of the fully-bootstrapped TD(0) error, we use λ-returns (Sutton & Barto, 2018) as the targets, computed individually for each quantile. Appendix C lays out the learning process with all the techniques discussed in this section applied.

4.1. Reformulation as a Correction

We use a variant of the method that provides a correction to the naive gradient $\bar{\nabla} f$ rather than replacing it entirely. If we define $U(x) = V(x + \alpha \bar{\nabla} V(x))$ then we have the Bellman equation

$$\begin{aligned} U(x) &= f(x + \alpha \bar{\nabla}(f(x) + \gamma U(x))) \\ &\quad + \gamma U(x + \alpha \bar{\nabla}(f(x) + \gamma U(x))). \end{aligned}$$

Now agents follow the gradient field $\bar{\nabla} f(x) + \gamma \bar{\nabla} U(x)$, and we minimize

$$\sum_t \|\hat{f}(x^{(t+1)}) + \gamma \hat{U}(x^{(t+1)}; \theta) - \hat{U}(x^{(t)}; \theta)\|^2$$

with respect to the parameters θ of our model $\hat{U}(x; \theta)$.

This variant is more strongly grounded in the game f and helps avoid the detachment from reality that plagues bootstrapped value functions. A drawback of this approach is that the naive gradient term $\bar{\nabla} f(x)$ will typically have to be estimated by REINFORCE (Williams, 1992).

4.2. Variable Discount Rates

We also set up the model (be it $\hat{U}$ or $\hat{V}$) to condition on discount rates γ_1, γ_2, so that we can train it for different rates and even rates that differ between the players. This is helpful because it forces the model to better understand the given policies, in order to distinguish policies that would behave the same under some fixed discount rate but differently under another. When the context calls for it, we will make this structure explicit by writing $f(x) + \gamma \odot \hat{U}(x; \theta, \gamma)$ where γ is now a vector and $\odot$ denotes the elementwise product. During training, we draw $\gamma_1, \gamma_2 \sim \text{Beta}\,(1/2, 1/2)$ from the standard arcsine distribution to emphasize extreme values.

Varying γ affects the scale of V, U and hence the scale of our approximations to them. This in turn results in a change in the effective learning rate when we take gradients. To account for this, we can normalize the outputs and gradients of V, U by scaling by $1 - \gamma$ before use. However, this is numerically problematic for values of γ close to 1. Instead, we multiply the meta-reward term $f(x)$ in the Bellman equation by $1 - \gamma$. This ensures our models learn the normalized values instead, which will fall in the same range as $f(x)$. Appendix A has a derivation.

4.3. Exploration

Our model $\hat{V}$ provides a deterministic (meta-)policy for changing the inner policies x. For effective value learning, however, we need exploration as well as exploitation. A straightforward way to introduce exploration into the system is to perturb the greedy transition in Equation 6 with some additive Gaussian noise (Heess et al., 2015). However, this leads to a random walk that fails to systematically explore the state space. Instead of perturbing the actions, we perturb the policy by applying noise to the parameters θ, and hold the perturbed policy fixed over the course of an entire optimization trajectory. Specifically, we randomly flip signs on parameters in the final layer of $\hat{V}$; this results in a perturbed value function that incentivizes different high-level characteristics of the inner policies x.[2] The trajectories so collected are entirely off-policy and serve only to provide a variety of states. In order to train our model on a given state x, we do a short on-policy rollout with the unperturbed parameters θ and minimize TD error there.

5. Experiments

The method is evaluated on two environments. First, we demonstrate the advantage of being able to look farther ahead on a two-dimensional game that is easy to visualize. Next, we turn to the IPD to demonstrate that our method does not simply cooperate unconditionally, but in fact learns to play tit-for-tat.

5.1. Logistic Game

We analyze the behavior of several algorithms on the Logistic Game (Letcher, 2018), a two-player game where each player's policy is a single scalar value. Thus the entire joint policy space is a two-dimensional plane, which we can easily visualize. The game is given by the function[3]

$$f(x) = -\begin{pmatrix} 4\sigma(x_1)(1-2\sigma(x_2)) \\ 4\sigma(x_2)(1-2\sigma(x_1)) \end{pmatrix} - \frac{x_1^2 x_2^2 + (x_1-x_2)^2(x_1+x_2)^2}{10000}. \quad (8)$$

Figure 1 shows the structure of the game. There are two stable fixed points – one in the lower left (A) and one in the upper right (B). Both players prefer B to A, however to get from A to B requires coordination: the horizontal player prefers left if the vertical player plays low and vice versa.

We look at this game through the lens of basins of attraction, and how different algorithms affect them (Figure 2b). Following naive gradients (lola:0.0,hola2:0.0), players converge to whichever solution is nearest; the basins of attraction meet at a diagonal line through the origin. LOLA grows the basin of the preferred solution B, but only slightly and increasing the extrapolation step size α does not help much. HOLA2 grows the basin of B around the edges, but suffers from instabilities around the origin (a saddlepoint). We found HOLA3 to be significantly worse than HOLA2 and did not pursue that direction further. COLA (our implementation) makes significant improvements around the edges and around the origin. Finally, our meta-value approach

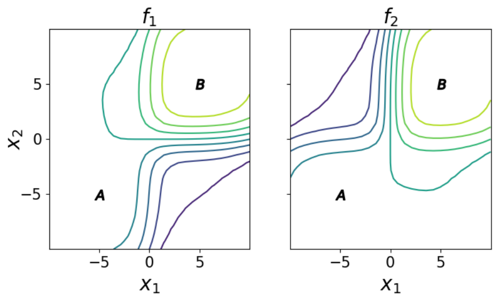


Figure 1: The Logistic Game. The left panel displays the contours of player 1's objective $f_1(x)$, the right panel similarly for player 2. Player 1's policy x_1 is a horizontal position, player 2's policy x_2 is a vertical position. Both players prefer solution B over solution A, but cannot unilaterally go there. Naive learning converges to whichever solution is closest upon initialization.

MeVa is able to make the basin of B arbitrarily large. When $\gamma > 0.9$, it converges to the preferred solution B from anywhere in the surveyed area. We also show some actual optimization trajectories in Figure 2a.

The meta-value approach gives us an additional hyperparameter γ to control the extent to which we look ahead. By increasing γ, we can make the basin of B arbitrarily large. Even if agents initialize close to A, it is worth moving in a direction that immediately decreases f, because they know (through the gradient of the meta-value) that doing so will eventually increase f.

We use hyperparameters $\alpha = 1, M = 16, T = k = 50$. The model is trained for 1000 outer loops using Adam (Kingma & Ba, 2014) with learning rate $\eta = 10^{-3}$ and batch size 128. In this experiment we do not use a target network or λ-returns, nor do we use exploration. More detail can be found in Appendix D.

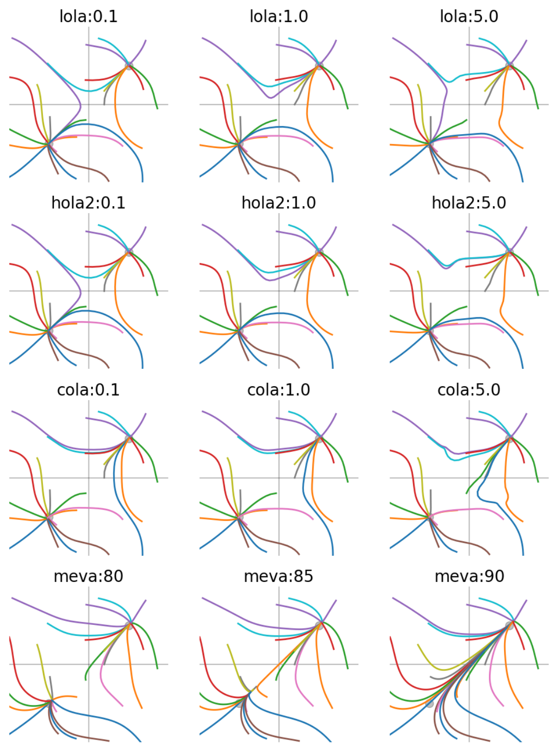
5.2. Iterated Prisoner's Dilemma

The Iterated Prisoner's Dilemma is a repeated matrix game in which two players choose whether to cooperate (C) or defect (D) with one another based on what happened in previous turns. As in (Foerster et al., 2018a), we consider policies with one turn's worth of memory; this class contains tit-for-tat as well as always-defect, with tit-for-tat being the preferred solution that naive learning fails to find. Specifically, a policy consists of five numbers: the log-odds of cooperation in the initial state, and after each of the four possible joint actions (DD,CD,DC,CC). We use the (normalized) exact value function formulation given in (Foerster et al., 2018a) with discount rate 0.96 (not to be confused with our meta-discount rate γ).

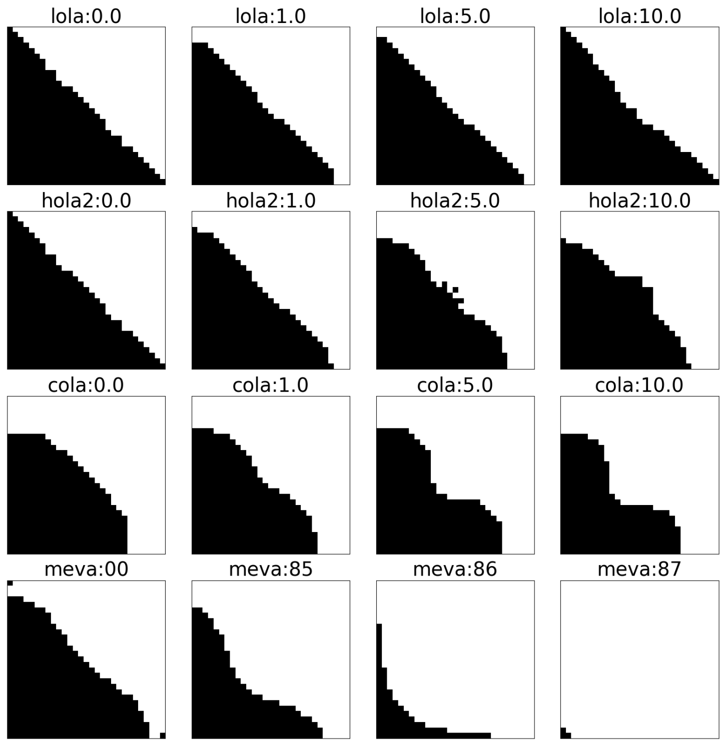
We trained 8 models $\hat{U}$ from different random seeds, and

[2]For the $\tilde{\nabla} f(x) + \gamma \tilde{\nabla} \hat{U}(x;\theta)$ formulation, we find that the effect of the exploration noise is diminished because the $\tilde{\nabla} f(x)$ term is unaffected; for this reason we simply use $\tilde{\nabla} \hat{U}(x;\theta)$ during exploration.

[3](Letcher, 2018) use the divisor 1000 in Eqn (8), however it does not match their plots. Moreover we have flipped the sign to turn this into a maximization problem in accordance with our notation.



(a) Optimization trajectories. We took a random set of policy pairs and, for each panel, optimized them according to the algorithm under consideration. Each curve shows an optimization trajectory, typically finishing in or close to either A or B.



(b) Basins of attraction. For each panel, we took a grid of policy space points and optimized them according to the algorithm under consideration. White cells indicate that the corresponding point ended up in the positive quadrant $x_1, x_2 > 0$, black cells ended up in other quadrants (typically the negative quadrant).

Figure 2: Logistic Game behaviors of different algorithms (rows) with different settings (columns).

for each model produced 256 policy pairs by following gradients of $f(x) + \gamma \hat{U}(x;\theta)$, for a total of 2048 policy pairs. Figure 3a shows our results, averaged over the policy pairs. The leftmost panel shows the expected returns $f(x)$ obtained by the policy pair. At low γ, policies converge to mostly defective strategies and approach the DD payoff -2. As γ increases, cooperation becomes more prevalent, with returns approaching the CC payoff of -1.

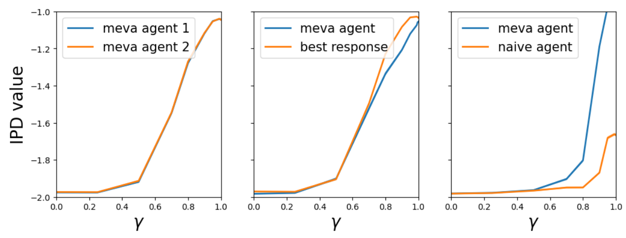
Now that we have exhibited cooperation, we must show that it is not exploitable. For each pair of agents that we trained, we hold one fixed and find the (self-interested) best response policy using standard gradient descent. If our agent were to cooperate or defect unconditionally, the best response would be to defect, in which case our agent would obtain -3 or -2 respectively. Instead, the best response is to cooperate (Figure 3a, middle), which indicates that our agent uses the threat of retaliation to enable mutual cooperation.

Finally, we demonstrate *opponent shaping* (Foerster et al., 2018a). We train a pair of agents, one following meta-value gradients and the other following naive gradients. Since we trained our model with variable values of γ (Section 4.2), following naive gradients is a special case of following meta-value gradients. So to implement this, we can simply use $\bar{\nabla} f(x) + \gamma \odot \bar{\nabla}\hat{U}(x;\theta,\gamma)$ with $\gamma_2 = 0$ and γ_1 set to some fixed value. The right-hand panel of Figure 3a shows our ability to shape the naive opponent into exploitable cooperation as $\gamma_1 \to 1$.

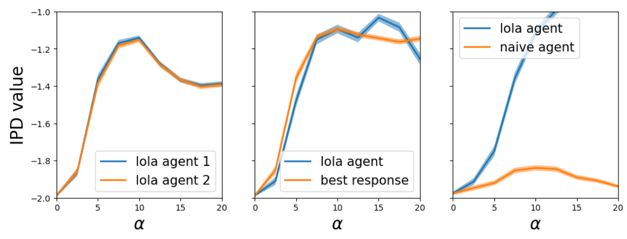
Figure 3b shows LOLA's behavior on the same experiment. LOLA exhibits qualitatively the same results. However, the α hyperparameter can only be increased so far before results deteriorate. MeVa retains the desirable features of LOLA, but is easier to extend farther into the future.

Our demonstration of opponent shaping doubles as a demonstration of how MeVa may be applied in the case where different agents use different learning algorithms. Since we train our model with variable γ, we can immediately use it for any combination of different γs. In general, however, we will need to train the model for the specific downstream situation, because like any other value function, the meta-value function has an implicit dependency on the meta-policy with which it is co-trained.

We used hyperparameters $\alpha = 2, M = 32, T = 100, k =$



(a) Behavior of MeVa on the IPD, as a function of (meta)discount rate γ. Each plot shows mean±standard error of the IPD value (expected return) computed over 2048 policy pairs (256 per model seed).



(b) Behavior of LOLA on the IPD, as a function of extrapolation step size α. Each plot shows mean±standard error of the IPD value (expected return) computed over 256 policy pairs.

Figure 3: Behavior of MeVa (top) and LOLA (bottom) on Iterated Prisoner's Dilemma.
Left: two agents trained simultaneously, both following MeVa (top) or LOLA (bottom). Higher values of γ, α lead to policies that yield higher returns through cooperation.
Middle: after training two agents following MeVa/LOLA (top/bottom), we hold one fixed and train (using the naive gradient) another agent against it to find the best response. The best response is cooperative, which means the agent exhibits nonexploitable cooperation.
Right: two agents trained simultaneously, one following MeVa/LOLA (top/bottom), the other following naive gradients. The naive agent is led to exploitable cooperation.

$10, \rho = 0.99, \lambda = 0.9$. The model is trained for 2000 outer loops, using AdamW (Loshchilov & Hutter, 2017) with learning rate 10^{-4} and batch size 128. During exploration, we flip signs on the last layer of the model (see Section 4.3), such that on average $\frac{1}{16}$ of the units are perturbed. Full details can be found in Appendix E.

6. Limitations

The method requires access to and control over the parameters and learning processes of all agents. This is not necessarily unrealistic – our aim is to devise an algorithm to learn a policy for the game f that can then be deployed in the wild without further training.

The meta-value function is a function over (joint) policies. In practice, policies will often take the form of neural networks, and so will our approximations to the meta-value function. Conditioning neural networks on other neural networks is a major challenge (Harb et al., 2020). In addition, the large parameter vectors associated with neural networks will quickly prohibit handling batched optimization trajectories.

Our discount rate γ, like LOLA's step size α, is hard to interpret. Its meaning changes significantly depending on the learning rate α and the parameterization of both the model $\hat{V}$ and the policies x. Future work could explore the use of proximal updates, like POLA (Zhao et al., 2022) did for LOLA.

It is well known that LOLA fails to preserve the Nash equilibria of the original game f. The method presented here shares this property.

7. Conclusion

We have introduced Meta-Value Learning (MeVa), a new perspective on learning with learning awareness. The meta-value function is naturally consistent and far-sighted, and lends itself to standard reinforcement learning treatment. Our method exhibits LOLA-like behavior on the IPD, including opponent shaping, and surpasses LOLA on the Logistic Game due to its ability to look ahead.

The main weakness of the method as it stands is scalability, particularly to policies that take the form of neural networks. We aim to address this in future work using policy fingerprinting (Harb et al., 2020).

Finally, we note that although we develop our method in the context of multi-agent reinforcement learning, it is a general meta-learning approach that readily applies to optimization problems with a single objective.

Acknowledgements

This research was enabled in part by compute resources provided by Mila. We used the JAX (Bradbury et al., 2018) library for scientific computing. We thank Christos Tsirigotis, Michael Noukhovitch, Juan Augustin Duque and Shunichi Akatsuka for helpful discussion.

References

Al-Shedivat, M., Bansal, T., Burda, Y., Sutskever, I., Mordatch, I., and Abbeel, P. Continuous adaptation via meta-learning in nonstationary and competitive environments. *arXiv preprint arXiv:1710.03641*, 2017.

Axelrod, R. and Hamilton, W. D. The evolution of cooperation. *science*, 211(4489):1390–1396, 1981.

Baker, B. Emergent reciprocity and team formation from randomized uncertain social preferences. *Advances in Neural Information Processing Systems*, 33:15786–15799, 2020.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

Busoniu, L., Babuska, R., and De Schutter, B. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

Dabney, W., Rowland, M., Bellemare, M., and Munos, R. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Fairbank, M. and Alonso, E. Value-gradient learning. In *The 2012 international joint conference on neural networks (ijcnn)*, pp. 1–8. IEEE, 2012.

Foerster, J., Chen, R. Y., Al-Shedivat, M., Whiteson, S., Abbeel, P., and Mordatch, I. Learning with opponent-learning awareness. *International Conference on Autonomous Agents and Multiagent Systems*, 2018a.

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., and Whiteson, S. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018b.

Foerster, J., Farquhar, G., Al-Shedivat, M., Rocktäschel, T., Xing, E., and Whiteson, S. Dice: The infinitely differentiable monte carlo estimator. In *International Conference on Machine Learning*, pp. 1529–1538. PMLR, 2018c.

Gronauer, S. and Diepold, K. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, pp. 1–49, 2022.

Harb, J., Schaul, T., Precup, D., and Bacon, P.-L. Policy evaluation networks. *arXiv preprint arXiv:2002.11833*, 2020.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Heess, N., Wayne, G., Silver, D., Lillicrap, T., Erez, T., and Tassa, Y. Learning continuous control policies by stochastic value gradients. *Advances in neural information processing systems*, 28, 2015.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Hughes, E., Leibo, J. Z., Phillips, M., Tuyls, K., Dueñez-Guzman, E., García Castañeda, A., Dunning, I., Zhu, T., McKee, K., Koster, R., et al. Inequity aversion improves cooperation in intertemporal social dilemmas. *Advances in neural information processing systems*, 31, 2018.

Jaques, N., Lazaridou, A., Hughes, E., Gulcehre, C., Ortega, P., Strouse, D., Leibo, J. Z., and De Freitas, N. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In *International conference on machine learning*, pp. 3040–3049. PMLR, 2019.

Kim, D.-K., Riemer, M., Liu, M., Foerster, J., Everett, M., Sun, C., Tesauro, G., and How, J. P. Influencing long-term behavior in multiagent reinforcement learning. *Advances in Neural Information Processing Systems*, 35:18808–18821, 2022.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Lauer, M. An algorithm for distributed reinforcement learning in cooperative multiagent systems. In *Proc. 17th International Conf. on Machine Learning*, 2000.

Letcher, A. Stability and exploitation in differentiable games. Master's thesis, 2018.

Letcher, A., Foerster, J., Balduzzi, D., Rocktäschel, T., and Whiteson, S. Stable opponent shaping in differentiable games. *arXiv preprint arXiv:1811.08469*, 2018.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Lowe, R., WU, Y., Tamar, A., Harb, J., Pieter Abbeel, O., and Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Lu, C., Willi, T., De Witt, C. A. S., and Foerster, J. Model-free opponent shaping. In *International Conference on Machine Learning*, pp. 14398–14411. PMLR, 2022.

Matignon, L., Laurent, G. J., and Le Fort-Piat, N. Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 64–69. IEEE, 2007.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015.

Panait, L. and Luke, S. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11:387–434, 2005.

Sandholm, T. W. and Crites, R. H. Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems*, 37(1-2):147–166, 1996.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Srivastava, R. K., Greff, K., and Schmidhuber, J. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Watkins, C. J. and Dayan, P. Q-learning. *Machine learning*, 8:279–292, 1992.

Willi, T., Letcher, A. H., Treutlein, J., and Foerster, J. Cola: consistent learning with opponent-learning awareness. In *International Conference on Machine Learning*, pp. 23804–23831. PMLR, 2022.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, pp. 5–32, 1992.

Zhang, C. and Lesser, V. Multi-agent learning with policy prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pp. 927–934, 2010.

Zhao, S., Lu, C., Grosse, R. B., and Foerster, J. Proximal learning with opponent-learning awareness. *Advances in Neural Information Processing Systems*, 35:26324–26336, 2022.

A. Normalized Bellman Equation

We introduce a slight tweak to the usual Bellman equation

$$V(s_t) = r_t + \gamma V(s_{t+1})$$

to address a scaling issue. Suppose the rewards fall in the range $(0, 1)$. Then V will take values in the range $\left(0, \frac{1}{1-\gamma}\right)$. We can multiply through by $1 - \gamma$ to obtain a new equation that will force values $\tilde{V}$ into the same range as the rewards:

$$\begin{aligned}(1-\gamma)V(s_t) &= (1-\gamma)r_t + \gamma(1-\gamma)V(s_{t+1}) \\ \tilde{V}(s_t) &= (1-\gamma)r_t + \gamma\tilde{V}(s_{t+1}).\end{aligned}$$

This is useful in our case because otherwise changing γ would affect the scale of the gradient $\bar{\nabla} V$ and hence change the effective learning rate. We also found it helpful when using distributional RL with a fixed binning – the convex-combination form of the right-hand side guarantees it will fall within the same range as the left-hand side. Finally, we find the normalized values easier to interpret.

B. LOLA, HOLA, COLA

LOLA (Foerster et al., 2018c) (sans Taylor approximation) uses the update

$$\bar{\nabla} f^{\text{LOLA}}(x) = \bar{\nabla} \begin{pmatrix} f_1(x + \alpha S_2^\top S_2 \bar{\nabla} f(x)) \\ f_2(x + \alpha S_1^\top S_1 \bar{\nabla} f(x)) \end{pmatrix}.$$

This differs from what we presented in Equation 2 in that each player only considers their opponent's update, and not their own.

HOLAn (Foerster et al., 2018a;c) (again sans Taylor approximation) applies the LOLA surrogate recursively to itself:

$$\bar{\nabla} f^{\text{HOLA}(n)}(x) = \bar{\nabla} \begin{pmatrix} f_1(x + \alpha S_2^\top S_2 \bar{\nabla} f^{\text{HOLA}(n-1)}(x)) \\ f_2(x + \alpha S_1^\top S_1 \bar{\nabla} f^{\text{HOLA}(n-1)}(x)) \end{pmatrix},$$

with base case $\bar{\nabla} f^{\text{HOLA}(0)}(x) = \bar{\nabla} f(x)$. Notice that HOLA1 recovers LOLA: $\bar{\nabla} f^{\text{HOLA}(1)}(x) = \bar{\nabla} f^{\text{LOLA}}(x)$.

COLA (Willi et al., 2022) considers

$$\bar{\nabla} f^{\text{COLA}}(x) = \bar{\nabla} \begin{pmatrix} f_1(x + \alpha S_2^\top S_2 \bar{\nabla} f^{\text{COLA}}(x)) \\ f_2(x + \alpha S_1^\top S_1 \bar{\nabla} f^{\text{COLA}}(x)) \end{pmatrix},$$

an implicit equation similar to Eqn (3). The equation is solved with a model $\hat{g}^{\text{COLA}}(x;\theta) \approx \bar{\nabla} f^{\text{COLA}}(x)$ of the gradient of the implicit surrogate. The model is trained to satisfy

$$\hat{g}^{\text{COLA}}(x;\theta) = \bar{\nabla} \begin{pmatrix} f_1(x + \alpha S_2^\top S_2 \hat{g}^{\text{COLA}}(x;\theta)) \\ f_2(x + \alpha S_1^\top S_1 \hat{g}^{\text{COLA}}(x;\theta)) \end{pmatrix}$$

by minimizing the squared distance between the two sides of this equation.

C. Full Algorithm

Algorithm 1 details the learning process we use after incorporating the techniques discussed in Section 4. Note that after introducing quantile distributional RL, $\hat{U}(x;\theta,\gamma) \in \mathbb{R}^{2\times M}$ is a matrix with a column for each of the M quantiles. Whenever we take gradients with respect to x, we use the gradients of the mean, denoted $\hat{U}_{\mathbb{E}}(x;\theta,\gamma) = \frac{1}{M}\hat{U}(x;\theta,\gamma)\vec{1}$ where $\vec{1}$ is the vector of ones. The divergence $D(\hat{y}, y)$ is the quantile regression loss (Dabney et al., 2018) between the predictions $\hat{y}$ and the targets y.

Algorithm 1 Meta-Value Learning.

```
**Input:** Learning rates $\eta, \alpha$, exploration trajectory length $T$, stride $k$, bootstrapping rate $\lambda$, target network inertia $\rho$.
Initialize meta-value functions $\theta_1, \theta_2$ and target networks $\bar{\theta} = \theta$.
while $\theta$ has not converged do
  Initialize policies $\tilde{x}^{(0)}$; draw $\tilde{\gamma}$ (Section 4.2); draw $\tilde{\theta}$ (Section 4.3).
  $t \leftarrow 0$
  while $t < T$ do
    for $\tau = t \dots t + k$ do
      $\tilde{x}^{(\tau+1)} = \tilde{x}^{(\tau)} + \alpha \bar{\nabla} \hat{f}(\tilde{x}^{(\tau)}) + \alpha \tilde{\gamma} \odot \bar{\nabla} \hat{U}_{\mathbb{E}}(\tilde{x}^{(\tau)}; \tilde{\theta}, \tilde{\gamma})$
    end for
    $t \leftarrow t + k$
    Let $x^{(0)} = \tilde{x}^{(t)}$; draw $\gamma$ (Section 4.2).
    for $\tau = 0 \dots k - 1$ do
      $x^{(\tau+1)} = x^{(\tau)} + \alpha \bar{\nabla} \hat{f}(x^{(\tau)}) + \alpha \gamma \odot \bar{\nabla} \hat{U}_{\mathbb{E}}(x^{(\tau)}; \theta, \gamma)$
    end for
    for $i \in 1, 2$ do
      $Y_i^{(k)} = \hat{U}_i(x^{(k)}; \bar{\theta}_i, \gamma) \in \mathbb{R}^M$
      for $\tau = k - 1 \dots 0$ do
        $Y_i^{(\tau)} = (1 - \gamma_i) \hat{f}_i(x^{(\tau+1)}) + \gamma_i \left( (1 - \lambda) \hat{U}_i(x^{\tau+1}; \bar{\theta}_i, \gamma) + \lambda Y_i^{(\tau+1)} \right)$
      end for
      $\theta_i \leftarrow \theta_i - \eta \nabla \mathcal{L}_i(\theta_i)$ where $\mathcal{L}_i(\theta_i) = \frac{1}{k} \sum_{\tau=0}^{k-1} D(\hat{U}_i(x^{(\tau)}; \theta_i, \gamma), Y_i^{(\tau)})$.
    end for
    $\bar{\theta} \leftarrow \bar{\theta} + (1 - \rho)(\theta - \bar{\theta})$
  end while
end while
```

D. Logistic Game Details

We use the $\hat{V}(x; \theta, \gamma)$ formulation. While conceptually, each agent maintains their own model $\hat{V}_1(x; \theta_1, \gamma)$, the implementation combines the computation of both models. The structure of the resulting single model can be seen in the following diagram:

$$\begin{tikzcd}
\begin{pmatrix} x_1 \\ \gamma_1 \end{pmatrix} \arrow[r] & \mathsf{MLP}_1 \arrow[r] \arrow[d, dash, dotted] & z_1 \arrow[r] \arrow[dr] & \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \arrow[r] & \mathsf{MLP}_2 \arrow[r] \arrow[d, dash, dotted] & \hat{V}_1 \\
\begin{pmatrix} x_2 \\ \gamma_2 \end{pmatrix} \arrow[r] & \mathsf{MLP}_1 \arrow[r] & z_2 \arrow[r] \arrow[ur] & \begin{pmatrix} z_2 \\ z_1 \end{pmatrix} \arrow[r] & \mathsf{MLP}_2 \arrow[r] & \hat{V}_2
\end{tikzcd}$$

We first feed the x_i, γ_i pairs into a multi-layer perceptron (MLP) to obtain a representation z_i of each agent. Then for each agent we concatenate their own representation with that of their opponent. This is run through a second MLP which outputs the quantile estimates.

The dotted lines in the diagram indicate parameter sharing between the two players ($\theta_1 = \theta_2$), which exploits the symmetry of the games under consideration (specifically $f_1(x_1, x_2) = f_2(x_2, x_1)$) to improve sample efficiency of the learning process.

The MLPs consist of a residual block (Srivastava et al., 2015; He et al., 2016) sandwiched between two layer-normalized GeLu (Hendrycks & Gimpel, 2016) layers. The residual block uses a layer-normalized GeLu as nonlinearity, and uses learned unitwise gates to merge with the linear path. Each layer has 64 units.

We use the same model structure for COLA, albeit without quantile regression. In the case of COLA we pass the inner learning rate α in place of γ, and we trained a single model for values of $\alpha \sim U[0, 10]$.

Policies $\tilde{x}_i^{(0)} \sim U(-8, +8)$ are initialized uniformly on an area around the origin.

We used hyperparameters $\alpha = 1, M = 16, T = k = 50$. In this experiment we do not use a target network or λ-returns (i.e. $\lambda = 0$), nor do we use exploration (i.e. $\tilde{\theta} = \theta$). The model is trained for 1000 outer loops using Adam (Kingma & Ba, 2014) with learning rate $\eta = 10^{-3}$ and batch size 128. Training takes about a minute on a single GPU.

E. Iterated Prisoner's Dilemma Details

For the IPD we use the $\hat{U}(x; \theta, \gamma)$ formulation. We use a similar shared-model structure as for the Logistic Game (see the diagram in the previous section), but we use two residual blocks in each MLP. In the final nonlinear layer, the GeLu is replaced by a hyperbolic tangent – a signed nonlinearity that enables our exploration scheme. During exploration, we flip signs on the units in this layer with Bernoulli probability $^1/_{16}$.

Policy logits are initialized from a standard Normal distribution, i.e. $\tilde{x}_i^{(0)} \sim \mathcal{N}(0, I)$.

We used hyperparameters $\alpha = 2, M = 32, T = 100, k = 10, \rho = 0.99, \lambda = 0.9$. The model is trained for 2000 outer loops, using AdamW (Loshchilov & Hutter, 2017) with learning rate 10^{-4}, batch size 128 and 10^{-2} weight decay. Training the model on the IPD takes about half an hour on a single GPU.