Abstract

A core challenge in policy optimization in competitive Markov decision processes is the design of efficient optimization methods with desirable convergence and stability properties. We propose competitive policy optimization (CoPO), a novel policy gradient approach that exploits the game-theoretic nature of competitive games to derive policy updates. Motivated by the competitive gradient optimization method, we derive a bilinear approximation of the game objective. In contrast, off-the-shelf policy gradient methods utilize only linear approximations, and hence do not capture players’ interactions. We instantiate CoPO in two ways: (i) competitive policy gradient, and (ii) trust-region competitive policy optimization. We theoretically study these methods, and empirically investigate their behavior on a set of comprehensive, yet challenging, competitive games. We observe that they provide stable optimization, convergence to sophisticated strategies, and higher scores when played against baseline policy gradient methods.

1 INTRODUCTION

Reinforcement learning (RL) in competitive Markov decision processes CoMDP [Filar and Vrieze, 2012] is the study of competitive players, sequentially making decisions in an environment. In CoMDPs, the competing agents (players) interact with each other within the environment, and through their interactions, learn how to develop their behavior and improve their policy. In this paper, we consider the rich and fundamental class of zero-sum two-player games.

A core challenge in CoMDP is to design optimization procedures with desirable convergence and stability properties. Policy gradient (PG) is a prominent RL approach that is widely used in single agent optimization and derives policy update using the first order (linear) approximation of the objective function [Robbins and Monro, 1951; Aleksandrov et al., 1968; Sutton et al., 2000]. A straightforward extension of conventional single-agent PG approaches to two-player min-max games results in the gradient descent ascent algorithm. This approximation is linear in agents’ parameters and does not take their interaction into account. Therefore, GDA directly optimizes the policy of each agent, assuming the policy of the opponent is fixed which sometimes leads to divergence even in simple scenarios and hence considered undesirable in competitive optimization.

We propose a new paradigm, competitive policy optimization (CoPO) for solving two-player CoMDPs. CoPO exploits the game-theoretic and competitive nature of CoMDPs, and, inspired by the competitive gradient descent approach [Schaefer and Anandkumar, 2019], deploys a local bilinear approximation of the game objective to derive policy updates. This local bilinear approximation can be viewed as the simultaneous two-player generalization of the local linear approximation used in single-agent policy gradient approaches (holding the other agent’s policy fixed). To compute the policy updates, CoPO computes the Nash equilibrium of the local bilinear approximation of the game objective. In CoPO, each agent derives its update with the consideration of what the other agent’s current move and moves in the future time steps might be. In addition, each agent considers how the environment, as the result of the agents’ current and future moves, evolves in favor of each agent. Therefore, each agent hypothesizes about what the other agent’s and the environment’s responses would be, resulting in the recursive reasoning in game theory [Keynes, 2018] and temporal recursion in CoMDPs.

We instantiate CoPO in two ways to arrive at practical algorithms. We propose competitive policy gradient (CoPG), a novel PG algorithm that exploits value functions and the structure of CoMDPs to efficiently obtain policy updates. We further extend our approach to the case where each agent does not have direct access to the opponent’s policy parameters, and must (approximate) it. We also propose trust region
competitive policy optimization (TRCoPO), a novel trust region based PG method [Schulman et al., 2015]. TRCoPO updates agents’ parameters simultaneously by deriving the Nash equilibrium of a bilinear (in contrast to linear approximation in off-the-shelf trust region methods) approximation to the surrogate objective within a defined trust region in the parameter space.

We empirically validate our approach in several settings. We construct the main empirically study on competitive games, such as Markov soccer, Linear dynamical systems, and Racing cars. We show in all these environments that CoPO leads to superior policies. We observe many cases where standard policy gradient approaches do not exhibit stable learning behavior and can diverge. In our case studies, we show that CoPO can be applied to self-play setting, where an agent is playing against itself. We further show that CoPO can be applied to improve performance in other competitive algorithms such as generative adversarial imitation learning (GAIL) (where one player is the policy and the other is the discriminator) [Ho and Ermon, 2016]. We further extend our case study and show that CoPO remains effective even when one needs to learn a model of the opponent’s policy rather than having direct access.

2 PRELIMINARIES

A two player CoMDP is a tuple of \( (S, A^1, A^2, R, T, P, \gamma) \), where \( S \) is the state space, \( s \in S \) is a state, for player \( i \in \{1, 2\} \), \( A^i \) is the player \( i \)'s action space with \( a^i \in A^i \). \( R \) is the reward kernel with probability distribution \( R(s, a^1, a^2) \) and mean function \( r(s, a^1, a^2) \) on \( \mathbb{R} \). For a probability measure \( P \), \( p \) denotes the probability distribution of initial state, and for the transition kernel \( T \), \( T(s'|s, a^1, a^2) \) is the distribution of successive state \( s' \) after taking actions \( a^1, a^2 \) simultaneously at state \( s \), with discount factor \( \gamma \in [0, 1] \). We consider episodic environments with reachable absorbing state \( s_T \) almost surely in finite time. An episode starts at \( s_0 \sim p \), and at each time step \( k \geq 0 \) at state \( s_k \), each player \( i \) draws its action \( a^i_k \) according to policy \( \pi(a^i_k|s_k; \theta^i) \) parameterized by \( \theta^i \in \Theta^i \), where \( \Theta^i \subset \mathbb{R}^i \) is a compact metric space. Players 1, 2 receive \((r_k, -r_k)\) with \( r_k \sim R(s_k, a^1_k, a^2_k) \), and the environment evolves to a new state \( s_{k+1} \). A realization of this stochastic process is a trajectory \( \tau = ((s_k, a^1_k, a^2_k, r_k)|_{k=0}^{|\tau|-1}, s_{|\tau|}) \), an ordered sequence with random length \(|\tau| \), where \(|\tau| \) is determined by episode termination time and state \( s_{|\tau|} = s_T \). Let \( f(\tau; \theta^1, \theta^2) \) denote the probability distribution of the trajectory \( \tau \) following players’ policies \( \pi(\theta^i) \).

\[
f(\tau; \theta^1, \theta^2) = p(s_0) \prod_{k=0}^{|\tau|-1} \pi(a^1_k|s_k; \theta^1)\pi(a^2_k|s_k; \theta^2) R(r_k|s_k, a^1_k, a^2_k) T(s_{k+1}|s_k, a^1_k, a^2_k).
\]

For \( R(\tau) = \sum_{k=0}^{|\tau|-1} \gamma^k r(s_k, a^1_k, a^2_k) \), the Q-function, V-functions, and game objective are defined,

\[
Q(s_k, a^1_k, a^2_k; \theta^1, \theta^2) = \mathbb{E}_{\tau \sim P(\cdot; \theta^1, \theta^2)} \left[ \sum_{j=k}^{|\tau|-1} \gamma^{j-k} r(s_j, a^1_j, a^2_j)|s_k, a^1_k, a^2_k) \right],
\]

\[
V(s_k; \theta^1, \theta^2) = \mathbb{E}_{\tau \sim P(\cdot; \theta^1, \theta^2)} \left[ \sum_{j=k}^{|\tau|-1} \gamma^{j-k} r(s_j, a^1_j, a^2_j)|s_k) \right],
\]

\[
\eta(\theta^1, \theta^2) = \int_\gamma f(\tau; \theta^1, \theta^2) R(\tau)d\tau
\]

We assume \( V, Q, \) and \( \eta \) are differentiable and bounded in \((\theta^1, \theta^2)\) and for \( f \) on \((\theta^1, \theta^2)\), \( D_{\theta^i} f = \frac{\partial}{\partial \theta^i} f(\theta^1, \theta^2)|_{(\theta^1, \theta^2) = (\theta^1, \theta^2)} \), and \( D_{\theta^1 \theta^2} f = \frac{\partial}{\partial \theta^1 \theta^2} f(\theta^1, \theta^2)|_{(\theta^1, \theta^2) = (\theta^1, \theta^2)} \), for \( i, j \in \{1, 2\} \).

3 COMPETITIVE POLICY OPTIMIZATION

Player 1 aims to maximize the game objective \( \eta \) Eq. (2), and player 2 aims to minimize it, i.e., simultaneously solving for \( \max_{\theta_1^1} \eta(\theta^1, \theta^2) \) and \( \min_{\theta_2^2} \eta(\theta^1, \theta^2) \) respectively with,

\[
\theta^{1*} \in \argmax_{\theta^1 \in \Theta^1} \eta(\theta^1, \theta^2), \text{ and } \theta^{2*} \in \argmin_{\theta^2 \in \Theta^2} \eta(\theta^1, \theta^2).
\]

As discussed in the introduction, a straightforward generalization of PG methods to CoMDP, results in GDA (Alg[4]). Given players’ parameters \((\theta^1, \theta^2)\), GDA prescribes to optimize a linear approximation of the game objective in the presence of a regularization for the policy updates,

\[
\theta^1 \leftarrow \theta^1 + \alpha \argmax_{\Delta \theta^1, \Delta \theta^2 \in \Theta^1 \times \Theta^2} \Delta \theta^1 \top D_{\theta^1} \eta - \frac{1}{2\alpha} ||\Delta \theta^1||^2, \text{ and }
\]

\[
\theta^2 \leftarrow \theta^2 + \alpha \argmin_{\Delta \theta^1, \Delta \theta^2 \in \Theta^1 \times \Theta^2} \Delta \theta^2 \top D_{\theta^2} \eta + \frac{1}{2\alpha} ||\Delta \theta^2||^2, \tag{4}
\]

where \( \alpha \) represent the step size. The parameter updates in Eq.[4] result in greedy updates along the directions of maximum change, assuming the other player stays constant. These updates are myopic, and ignore the agents’ interactions. In other words, player 1 does not take player’s 2 potential move into consideration and vice versa. While GDA might be an approach of interest in decentralized CoMDP, it mainly falls short in the problem of competitive and centralized optimization in a priori unknown CoMDPs, i.e., the focus of this work. In fact, this behaviour is far from optimal and is shown to diverge in many simple cases e.g. plain bilinear or linear quadratic games [Schaef and Anandkumar, 2019, Mazumdar et al., 2019]. While single agent policy gradient methods generalize gradient descent [Robbins and Monro, 1951] to single player RL settings, in this paper, we generalize competitive gradient descent [Schaef and Anandkumar, 2019] to zero-sum RL settings.

We propose competitive policy optimization CoPO, a policy gradient approach for optimization in unknown CoMDPs.
In contrast to standard PG methods, such as GDA, CoPO considers a bilinear approximation of the game objective, and takes the interaction between players into account. Following the competitive gradient updates, CoPO incorporates the game theoretic nature of the CoMDP optimization and derives parameter updates through finding the Nash equilibrium of the bilinear approximation of the game objective,

$$\theta^1 \leftarrow \theta^1 + \arg\max_{\Delta \theta^1} \Delta \theta^1 \mathcal{D}_{\theta^1 \eta} + \Delta \theta^1 \mathcal{D}_{\theta^1 \eta} \Delta \theta^2 - \frac{1}{2\alpha} \| \Delta \theta^1 \|^2$$

$$\theta^2 \leftarrow \theta^2 + \arg\max_{\Delta \theta^2} \Delta \theta^2 \mathcal{D}_{\theta^2 \eta} + \Delta \theta^2 \mathcal{D}_{\theta^2 \eta} \Delta \theta^1 + \frac{1}{2\alpha} \| \Delta \theta^2 \|^2$$  \hspace{1cm} (5)

which has an extra term, the interaction term, in contrast to standard PG viewed as a natural linear generalization of the single agent learning problem. Fig. 1 shows that for a bilinear game, the parameter updates in Eq. 6 and updates (8) can often diverge or cycle indefinitely and never converge [Mertikopoulos et al., 2018a].

**Proposition 1.** Given a CoMDP, players $i, j \in \{1, 2\}, i \neq j$ and the policy parameters $\theta^i, \theta^j$,  

$$D_{\theta^i \eta} = \int f(\tau; \theta^i, \theta^j)g(\tau, \theta^i)R(\tau)d\tau,$$

$$D_{\theta^j \eta} = \int f(\tau; \theta^j, \theta^i)g(\tau, \theta^j)g(\tau, \theta^i)^T R(\tau)d\tau.$$

Proof in Apdx. 10.1. In practice, we use conjugate gradient and Hessian vector product to efficiently compute the updates in Eq. 6 as explained in later sections. A closer look at CoPO shows that this paradigm does not require the knowledge of the environment if sampled trajectories are available. It neither requires full observability of the states, nor any structural assumption on CoMDP, but the Monte Carlo estimation suffers from high variance. In the following, we explicitly take the CoMDP structure into account to develop efficient algorithms.

**3.1 COMPETITIVE POLICY GRADIENT**

We propose competitive policy gradient (CoPG), an efficient algorithm that exploits the structure of CoMDPs to compute the parameter updates. The following is the CoMDP generalization of the single agent PG theorem [Sutton et al., 2000].

For $\tau_{l:s} = (s_k, a_k^1, a_k^2, r_k)_{k=l}^{s}$, the events from time step $l$ to $s$, we have:

**Theorem 1.** Given a CoMDP, players $i, j \in \{1, 2\}, i \neq j$, and the policy parameters $\theta^i, \theta^j$,  

$$D_{\theta^i \eta} = \mathbb{E}_{s, a} \sum_{k=0}^{s-l} \gamma^k f(\tau_{0:k}; \theta^1, \theta^2)D_{\theta^i} (\log \pi(a_k^i | s_k; \theta^i)) Q(s_k, a_k^1, a_k^2; \theta^i, \theta^2)d\tau,$$

$$D_{\theta^j \eta} = \mathbb{E}_{s, a} \sum_{k=0}^{s-l} \gamma^k f(\tau_{0:k}; \theta^1, \theta^2)D_{\theta^j} (\log \pi(a_k^j | s_k; \theta^j)) Q(s_k, a_k^1, a_k^2; \theta^i, \theta^j)d\tau.$$  \hspace{1cm} (7)

Proof in Apdx. 10.2. \(\mathcal{F}_1, \mathcal{F}_2, \text{ and } \mathcal{F}_3\) are described in Table 1. This theorem indicates that the terms in Eq. 6 can be computed using Q function. In Eq. 8 \(\mathcal{F}_1\) is the immediate interaction between players, \(\mathcal{F}_2\) is the interaction of player $i$’s behavior up to time step $k$ with player $j$’s reaction at time step $k$, and the environment, \(\mathcal{F}_3\) is the interaction of player $j$’s behavior up to time step $k$ with player $i$’s reaction at step $k$, and the environment.

CoPG operates in epochs. At each epoch, CoPG deploys $\pi(\theta^1), \pi(\theta^2)$ to collect trajectories, exploits them to estimate the Q, $D_{\theta^1 \eta},$ and $D_{\theta^2 \eta}$. Then CoPG follows the parameter updates in Eq. 6 and updates $\theta$, and this process goes on to the next epoch (Alg. 2). Many variants of PG approach uses baselines, and replace Q with, the advantage function $A(s, a^1, a^2; \theta^1, \theta^2) = Q(s, a^1, a^2; \theta^1, \theta^2) - V(s; \theta^1, \theta^2)$. Monte Carlo estimate of Q-V [Baird, 1993].

(a) CoPO updates point inward, leading to cycling/divergence.

(b) GDA updates point outward, leading to cycling/divergence.

Figure 1: Bilinear games: (a) CoPO updates towards Nash equilibrium. (b) GDA updates point outward, leading to cycling/divergence.

where $I$ is an identity matrix of appropriate size. Note that, the bilinear approximation in Eq. 5 is still linear in each player’s action/parameters. Including any other terms, e.g., block diagonal Hessian terms from the Taylor expansion of the game objective, results in nonlinear terms in at least one player’s parameters. As such, CoPO can be viewed as a natural linear generalization of PG. It is known that GDA-style co-learning approaches can often diverge or cycle indefinitely and never converge [Mertikopoulos et al., 2018a]. Fig. 1 shows that for a bilinear game, the gradient flow of GDA cycles or has gradient flow outward, while the CoPO flow, considering player’s future moves, has gradient flow toward the Nash equilibrium and converges. In Apdx 9 we deploy the Neumann series expansion of the inverses in Eq 5 and show that CoPO recovers the infinite recursion reasoning between players and the environment, while GDA correspond to the first term, and LOLA corresponds to the first two terms in the series. Next, we compute terms in Eq. 8 using the score function $g(\tau, \theta^i) := D_{\theta^i} (\log \prod_{k=0}^{s-l-1} \pi(a_k^i | s_k; \theta^i))$. 

Empirical TD error or generalized advantage function
We propose trust region competitive policy optimization (GAE) \cite{Schulman et al., 2016}. Our algorithm can be extended to this set up and in Apdx. 11 we show how CoPG can be accompanied with all the mentioned variants.

### 3.2 OPPONENT PARAMETER ESTIMATION

In some settings, each learner does not have access to the opponent’s policy. To apply CoPG in such settings, one natural approach is to estimate online the opponent’s policy by the opponent’s state-action pairs, as proposed in \cite{Foerster et al., 2017b}. We thus propose a variant of CoPG that also inferences the opponent’s policy parameters (CoPG-OP), where each agent $i$ also estimates the parameters $\theta^i$ of its opponent $j$’s policy, e.g., using maximum-likelihood estimator,

$$ \hat{\theta}^i = \arg\max_{\theta^i} \mathbb{E}_{\tau \sim f(\cdot, \theta^i, \theta^j)} \sum_{k=0}^{T-1} \log \pi(\theta^i_k | s_k, \theta^j), $$

Then, the agent utilizes $\hat{\theta}^i$ to derive its policy updates in Eq. \ref{eq:policy_update} in place of $\theta^i$. In our empirical study, we observe that CoPG-OP training is as stable as CoPG and the policies learned using CoPG-OP are as competent as CoPG (refer to Apdx. 16). We conclude that opponent parameter learning can be considered effective in online settings, which confirms the observation in \cite{Foerster et al., 2017b}.

### 3.3 TRUST REGION COMPETITIVE POLICY OPTIMIZATION

Trust region based policy optimization methods exploit the local Riemannian geometry of the parameter space to derive more efficient policy updates \cite{Kakade and Langford, 2002, Kakade, 2002, Schulman et al., 2015}. In this section, we propose trust region competitive policy optimization (TRCoPO), the CoPO generalization of TRPO \cite{Schulman et al., 2015}, that exploits the local geometry of the competitive objective to derive more efficient parameter updates.

#### Lemma 1.

Given the advantage function of policies $\pi(\theta^1), \pi(\theta^2), \forall \theta^1, \theta^2 \in \Theta^1 \times \Theta^2$ we have,

$$ \eta(\theta^1, \theta^2) = \eta(\theta^1, \theta^2) + \mathbb{E}_{\tau \sim f(\cdot; \theta^1, \theta^2)} \sum_{k=0}^{T-1} \gamma^k A(s, \alpha^1_k, \alpha^2_k; \theta^1, \theta^2). $$

#### Proof in Apdx. 13.1

Equation (10) indicates that, considering our current policies ($\pi(\theta^1), \pi(\theta^2)$), having access to their advantage function, and also samples from $f(\cdot; \theta^1, \theta^2)$ (without rewards), we can directly compute $\eta(\theta^1, \theta^2)$ and optimize for $(\theta^1, \theta^2)$. However, in practice, we might not have access to $f(\cdot; \theta^1, \theta^2)$ for all $(\theta^1, \theta^2)$ to accomplish the optimization task, therefore, direct use of Eq. (10) is not favorable. Instead, we define a surrogate objective function, $L_{\theta^1, \theta^2}(\theta^1, \theta^2)$,

$$ L_{\theta^1, \theta^2}(\theta^1, \theta^2) = \eta(\theta^1, \theta^2) + \mathbb{E}_{\tau \sim f(\cdot, \theta^1, \theta^2)} \left[ \sum_{k=0}^{T-1} \gamma^k \mathbb{E}_{\pi(a^1_k | s_k, \theta^1), \pi(a^2_k | s_k, \theta^2), \mathbf{A}}(s_k, \alpha^1_k, \alpha^2_k; \theta^1, \theta^2) \right], \tag{11} $$

which can be computed using trajectories of our current policies $\pi(\theta^1), \pi(\theta^2)$. $L_{\theta^1, \theta^2}(\theta^1, \theta^2)$ is an object of interest in PG \cite{Kakade and Langford, 2002, Schulman et al., 2015} since mainly its gradient is equal to that of $\eta(\theta^1, \theta^2)$ at $(\theta^1, \theta^2)$, and as stated in the following theorem, it can be used as a surrogate of the game value. For KL divergence $D_KL((\theta^1, \theta^2), (\theta^1, \theta^2)) := \int_{\tau} f(\tau, \theta^1, \theta^2) \log \left( f(\tau, \theta^1, \theta^2) / f(\tau, \theta^1, \theta^2) \right) d\tau$, we have,

#### Theorem 2.

$L_{\theta^1, \theta^2}(\theta^1, \theta^2)$ approximate $\eta(\theta^1, \theta^2)$ up to the following error upper bound, with constant $\epsilon$

$$ |\eta(\theta^1, \theta^2) - L_{\theta^1, \theta^2}(\theta^1, \theta^2)| \leq \epsilon \sqrt{2} \mathbb{D} K_L((\theta^1, \theta^2), (\theta^1, \theta^2)) \tag{12} $$

$$ \epsilon := \max_{\theta^1, \theta^2} \sum_{k=0}^{T-1} \gamma^k \mathbb{E}_{\pi(a^1_k | s_k, \theta^1), \pi(a^2_k | s_k, \theta^2), \mathbf{A}}(s_k, \alpha^1_k, \alpha^2_k; \theta^1, \theta^2). $$

#### Proof in Apdx. 13.2

This theorem states that we can use $L_{\theta^1, \theta^2}(\theta^1, \theta^2)$ to optimize for $\eta(\theta^1, \theta^2)$ as long as we keep the $(\theta^1, \theta^2)$ in the vicinity of $(\theta^1, \theta^2)$. Similar to single agent TRPO (\cite{Schulman et al., 2015}), we optimize for $L_{\theta^1, \theta^2}(\theta^1, \theta^2)$, while constraining the KL divergence, $D_KL((\theta^1, \theta^2), (\theta^1, \theta^2)) \leq \delta$, i.e.,

$$ \max_{\theta^1 \in \Theta^1, \theta^2 \in \Theta^2} \min_{\theta^1, \theta^2} L_{\theta^1, \theta^2}(\theta^1, \theta^2), \text{with } D_KL((\theta^1, \theta^2), (\theta^1, \theta^2)) \leq \delta. \tag{13} $$

The GDA generalizing of TRPO uses a linear approximation of $L_{\theta^1, \theta^2}(\theta^1, \theta^2)$ to approach this optimization, which again ignores the players’ interactions. In contrast, TRCoPO considers the game theoretic nature of this optimization, and uses a bilinear approximation. TRCoPO operates in epochs.
At each epoch, TRCoPO deploys \((\pi(\theta^1), \pi(\theta^2))\) to collect trajectories, exploits them to estimate \(A\) (or GAE), then updates parameters accordingly, Alg.3 (For more details, please refer to Apx.13.3).

**4 EXPERIMENTS**

We empirically study the performance of CoPG and TRCoPO and their counterparts GDA and TRGDA, on six games, ranging from single-state repeated games to general sequential games, and tabular games to infinite/continuous high dimensional states/action games. They are 1) linear-quadratic(LQ) game, 2) bilinear game, 3) matching pennies (MP), 4) rock paper scissors (RPS), 5) Markov soccer, and 6) car racing. These games are representative enough that their study provides insightful conclusions, and challenging enough to highlight the core difficulties and interactions in competitive games.

We show that CoPG and TRCoPO converge to stable points, and learn opponent aware strategies, whereas GDA’s and TRGDA’s greedy approach shows poor performance and even diverge in bi-linear, MP, and RPS games. For the LQ game, when GDA does not diverge, it almost requires 1.5 times the amount of samples, and is 1.5 times slower than CoPG. In high strategic games, where players’ policies are tightly coupled, we show that CoPG and TRCoPO learn much better interactive strategies. In the soccer game, CoPG and TRCoPO players learn to defend, dodge and score goals, whereas GDA and TRGDA players learn how to score when they are initialized with the ball, and give way to the other player otherwise. In the car racing, while GDA and TRGDA show poor performance, CoPG and TRCoPO produce competing players, which learn to block and fake each other. Overall, we observe CoPG and TRCoPO considerably outperform their counterparts in terms of convergence, learned strategies, and gradient dynamics.

We implemented all algorithms in Pytorch \cite{Paszke2019}, and made the code and the videos public. In our implementation, we deploy Pytorch’s autograd package and Hessian vector product to efficiently obtain gradients and Hessian vector products to compute the bilinear terms in the optimizer. Moreover, we use the conjugate gradient trick to efficiently computed the inverses-matrix vector product in Eq.6 which incurs a minimal computational overhead (see Shewchuk \cite{Shewchuk1994} for more details). To improve computation times, we compute inverse-matrix vector product only for one player strategy, and use optimal counter strategy for other player \(\Delta \theta^2\) which is computed without an inverse matrix vector product. Also, the last optimal strategy can be used to warm start the conjugate gradient method which improves convergence times. We provide efficient implementation for both CoPO- and GDA-based methods, where CoPO incurs 1.5 times extra computation per batch.

**Zero-sum LQ game** is a continuous state-action linear dynamical game between two players, where GDA, with considerably small learning rate, has favorable convergence guarantees \cite{Zhang2019}. This makes the LQ game an ideal platform to study the range of allowable step sizes and convergence rate of CoPG and GDA. We show that, with increasing learning rate, GDA generates erratic trajectories and policy updates, which cause instability (see “○” in Table\ref{table:lq}), whereas CoPG is robust towards this behavior. Fig.\ref{fig:lq} shows that CoPG dynamics are not just faster at the same learning rate but more importantly, CoPG can potentially take even larger steps.

**Bilinear game** is a state-less strongly non-cooperative game, where GDA is known to diverge \cite{Balduzzi2018}. In this game, reward \(r(a^1, a^2) = \langle a^1, a^2 \rangle\) where \(a^1 \sim \mathcal{N}(\mu_1, \sigma_1)\) and \(a^2 \sim \mathcal{N}(\mu_2, \sigma_2)\), and \((\mu_1, \sigma_1), (\mu_2, \sigma_2)\) are policy parameters. We show that GDA diverges for all learning rates, whereas CoPG converges to the unique Nash equilibrium Fig.\ref{fig:bilinear}.

**Matching pennies** and **Rock paper scissors**, are challenging matrix games with mixed strategies as Nash equilibria, demand opponent aware optimization \footnote{Traditionally, fictitious and counterfactual regret minimization approaches have been deployed \cite{Neller2013}.}. We show that CoPG and TRCoPO both converge to the unique Nash equilibrium of MP Fig.\ref{fig:mp} and RPS Fig.\ref{fig:rps} whereas GDA and TRGDA

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure2.png}
\caption{During training CoPG (C) vs GDA (G) on a) LQ game, difference in game objective due to policy update for \(\alpha = 1e-1, 1e-2\) b) Bilinear game, \(\mu^1\) vs \(\mu^2\) c) MP, probability of selecting Head(H) and Tail(T) d) RPS, probability of selecting rock(R), paper(P) and scissors(S).}
\end{figure}
diverges (to a sequence of polices that are exploitable by deterministic strategy). Detailed empirical study, formulation and explanation of these 4 games can be found in Apx. [15]

Markov soccer game, Fig. 4 consists of players A and B that are randomly initialised in the field, that are supposed to pick up the ball and put it in the opponent’s goal [Littman 1994; He et al. 2016]. The winner is awarded with +1 and the loser with -1 (see Apx. [15.5] for more details).

Since all methods converge in this game, it is a suitable game to compare learned strategies. In this game, we expect a reasonable player to learn sophisticated strategies of defending, dodging and scoring. For each method playing against their counterparts, e.g., CoPG against CoPG and GDA, Fig. 3 shows the number of times the ball is seized between the players before one player finally scores a goal, or time-out. In (3a), CoPG vs CoPG, we see agents seize ball, twice the times as compared to (3b), GDA vs GDA (see $A_2$ and $B_2$ in (3a) and (3b)). In the matches CoPG vs GDA (3c), CoPG trained agent could seize the ball from GDA agent ($A_2$) nearly 12 times more due to better seizing and defending strategy, but GDA can hardly take the ball back from CoPG ($B_2$) due to a better dodging strategy of the CoPG agent. Playing CoPG agent against GDA one, we observe that CoPG wins more than 74% of the games Fig. 3d. We observe a similar trend for trust region based methods TRCPO and TRGDA, playing against each other (a slightly stronger results of 85% wins, $A_2$ column in Figs 3g, 3c).

We also compared CoPG with MADDPG [Lowe et al. 2017] and LOLA [Foerster et al. 2017b]. We observe that the MADDPG learned policy behaves similar to GDA, and loses 80% of the games to CoPG’s (Fig. [15.5]). LOLA learned policy, with its second level reasoning, performs better than GDA, but lose to CoPG 72% of the matches. For completeness, we also compared GDA-PG, TRGDA, and TRCPO playing against each other. TRCPO performs best, CoPG was runner up, then TRGDA, and lastly GDA (see Apx. [15.5]).

Car Racing is another interesting game, with continuous state-action space, where two race cars competing against each other to finish the race first [Liniger and Lygeros 2020]. The game is accompanied by two important challenges, 1) learning a policy that can maneuver the car at the limit of handling, 2) strategic interactions with opponents. The track is challenging, consisting of 13 turns with different curvature (Fig. 6). The game is formulated as a zero-sum, with reward $r(k, a_k, a_k^2) = \Delta_p_{car_1} - \Delta_p_{car_2}$, where $\Delta_p = \rho_{k+1} - \rho_k$ and $\rho_k$ is the car’s progress along the track at the $k^{th}$ time step. If a car crosses track boundaries (e.g., hit the wall), it is penalized, and the opponent receives rewards, which encourages cars to play rough and push each other into the track boundaries. When a collision happens, the rear car is penalized, and the car in the front receives a reward; it promotes blocking by the car in front and overtaking by the car in the rear. We study agents trained with all GDA, TRGDA, MADDPG, LOLA, CoPG and TRCPO in this game, and show that even though all players were able to learn to "drive" only CoPG and TRCPO were able to learn how to "race". Using GDA, only one player was able to learn, which manifested in either one player completely failing and the other finishing the track (Fig. 5a), or by an oscillation behavior where one player learns to go ahead, the other agent stays at lower progress (Fig. 5c). Even if one of the players learns to finish one lap at some point, this player does not learn to interact with its opponent (https://youtu.be/rxkGW02GwV). In contrast, players trained with CoPG and TRGDA, both progress, learn to finish the lap, and race (interact with each other) (See Fig 5b and Fig 5d). To test the policies, we performed races between CoPG and GDA, TRCPO and TRGDA, and CoPG and TRCPO. As shown in Table 2, CoPG and TRCPO win almost all races against their counterparts. Overall, we see that both CoPG and TRCPO are able to learn policies that are faster, more precise, and interactive with the other player (e.g., learns to overtake).

Table 2: Trained agents competing against each others in car racing. Ratio of Wins(W), Overtakes(O) and Collisions(C).

<table>
<thead>
<tr>
<th>CoPG v GDA</th>
<th>TRCPO v TRGDA</th>
<th>TRGDA v CoPG</th>
<th>TRCPO v TRGDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td>1</td>
<td>0</td>
<td>0.24</td>
</tr>
<tr>
<td>O</td>
<td>1.28</td>
<td>0.78</td>
<td>1.28</td>
</tr>
<tr>
<td>C</td>
<td>0.17</td>
<td>16.11</td>
<td>0.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.30</td>
</tr>
</tbody>
</table>

4.1 CASE STUDIES

Generative Adversarial Imitation Learning: One can also apply our approach in asymmetric games, such as learning the agent policy and the discriminator in generative adversarial imitation learning (GAIL) [Ho and Ermon 2016]. In GAIL, to imitate the expert, the agent $(\theta, \phi)$ player) plays a game with a discriminator $\mathcal{D}(\phi)$ player), i.e.,

$$\min_{\theta} \max_{\phi} \mathbb{E}_{\tau_0}[\log \mathcal{D}(s,a)] + \mathbb{E}_{\tau_1}[\log(1 - \mathcal{D}(s,a))] - \lambda H(\theta),$$

where $\tau_0$ is a trajectory and $H(\theta)$ is the casual entropy.

We conduct this study on the car racing game with a single car, where the aim is to learn to drive a full lap. Given a long track (Fig. 6), exploration and reward formulation can be challenging. We train the agent using CoPO to learn to drive by imitating an expert. The expert trajectories are collected using a pure pursuit (PP) controller [Coulter 1992] with different sets of parameters Apx [17] We evaluate agent’s policy using lap time ($t_{lap}$), and:

$$\zeta = \mathbb{E}_{\tau_0}\sum_{k}^{[\tau_1]-1} ||a_k - a_k^\ast||^2/|\tau|,$$

(14)
Table 3: Comparing lap time and $\zeta$ (Eq. 14) of the policies learnt using imitation learning and the baseline controllers.

<table>
<thead>
<tr>
<th></th>
<th>$\zeta \times 10^{-2}$</th>
<th>$t^{lap}\text{(s)}$</th>
<th>PP$_{avg}$</th>
<th>CoPO-GAIL</th>
<th>GDA-GAIL</th>
<th>PID</th>
<th>BC</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\zeta$</td>
<td>0</td>
<td>13.07</td>
<td>0.82</td>
<td>1.14</td>
<td>2.35</td>
<td>2.02</td>
<td></td>
</tr>
<tr>
<td>$t^{lap}$</td>
<td>0</td>
<td>13.07</td>
<td>1.14</td>
<td>2.35</td>
<td>14.67</td>
<td>DNF</td>
<td></td>
</tr>
</tbody>
</table>

Opponent learning: We next explore the case where one does not directly have access to the opponent’s parameters and they have to be inferred through interaction with the other agent and the environment. We propose to use CoPO-SP (Sec. 3.2), the opponent learning variant of CoPO.

Fig. 7 shows interaction plots of CoPO-OP conducted on the Markov Soccer game (setting explained in Apx. 15.5). The opponent’s parameters are estimated by observing state-action pairs of the opponent using Eq. 9. The interaction plot of the CoPO-OP agent with the estimated opponent (Fig. 7a) shows that the CoPO-OP player learns to seize the ball and interact with the opponent ($A_2$, $A_4$). Fig. 18b shows the interaction plot of CoPO-OP with CoPG, where we observe that CoPO-OP also learns a policy similar to CoPG, which is able to defend, escape and score goals ($A_3$). When directly competing with CoPG, we observe that CoPO-OP can win 46.5% of the games against CoPG.

The experiment details and numerical results on other setups can be found in Apx. 16. They show that CoPO-OP achieves the performance of CoPG in terms of stability and convergence to sophisticated strategies.

Training by Self-play: In self-play, one player plays against itself using the same policy model for both players. Each player then samples actions from this policy for their respective state and updates the policy using CoPG-SP (Alg. 5) which is a special case of CoPG. We observe that CoPG-SP can successfully learn competing strategies similarly to CoPG. We provide the CoPG-SP algorithm and a detailed empirical study in Apx. 12.
5 RELATED WORK

In tabular CoMDP, Q-learning and actor-critic have been deployed [Littman, 1994, 2001b], [Bowling and Veloso, 2002, Greenwald and Hall, 2003, [Hu and Wellman, 2003, Frénay and Saeens, 2009, Pérolat et al., 2018, Srinivasan et al., 2018], and recently, deep RL methods have been extending to CoMDPs, with focus on modeling agents behaviour [Tampuu et al., 2017, Leibo et al., 2017, Raghu et al., 2017]. To mitigate the stabilization issues, centralized methods [Matignon et al., 2012, Lowe et al., 2017, Forster et al., 2017a], along with opponent’s behavior modeling [Raileanu et al., 2018, He et al., 2016] have been explored. Optimization in multi-agent learning can be interpreted as a game in the parameter space, and the main body of the mentioned literature does not take this aspect directly into account since they attempt to separately improve players’ performance. Hence, they often fail to achieve desirable performance and oftentimes suffer from unstable training, especially in strategic games [Hernandez-Leal et al., 2019, Buşoniu et al., 2010]. In imperfect information games with known rules, e.g., poker [Moravčík et al., 2017], a series of works study algorithmically computing Nash equilibrium [Shalev-Shwartz and Singer, 2007, Koller et al., 1995, Gilpin et al., 2007, Zinkevich et al., 2008, Bowling et al., 2017]. Also, studies in stateless episodic games shows convergence to coarse correlated equilibrium [Hartline et al., 2015, Blum et al., 2008]. In contrast CoPO converges to the Nash equilibrium in such games. In two-player competitive games, self-play is an approach of interest where a player plays against itself to improve its behavior [Tesauro, 1995, Silver et al., 2016]. But, many of these approaches are limited to specific domains [Heinrich et al., 2015, Heinrich and Silver, 2016]. The closest approach to CoPO in the literature is LOLA [Forster et al., 2017b] an opponent aware approach. LOLA updates parameters using a second-order correction term, resulting in gradient updates corresponding to the following shortened recursion: if a player thinks that the other player thinks its strategy stays constant [Schaefer and Anandkumar, 2019], whereas CoPO recovers the full recursion series until the Nash equilibrium is delivered. In contrast to [Forster et al., 2017b] we also provide CoPO extension to value-based, and trust region-based methods, along with
their efficient implementation.

Our work is also related to GANs [Goodfellow et al., 2014], which involves solving a zero sum two-player competitive game (CoMDP with single state). Recent development in nonconvex-nonconcave problems and GANs training show GDA has undesirable convergence properties [Mazumdar et al., 2019] and exhibit strong rotation around fixed points [Balduzzi et al., 2018]. To overcome this rotation behaviour of GDA, various modifications have been proposed, including averaging [Yazici et al., 2019], negative momentum [Gidel et al., 2018] along many others [Mertikopoulos et al., 2018b, Daskalakis et al., 2017, Mescheder et al., 2017, Balduzzi et al., 2018, Gemp and Mahadevan, 2018]. Considering the game-theoretic nature of this problem, competitive gradient descent has been proposed as a natural generalization of gradient descent in two-players instead of GDA for GANs [Schaefer and Anandkumar, 2019]. This method, as the predecessor to CoPO, enjoys stability in training, robustness in choice of hyper-parameters, and has desirable performance and convergence properties.

6 DISCUSSION ON APPLICABILITY

CoPO is the paradigm of competitive policy optimization where the goal is to jointly find policies for agents. In CoPO, the optimization is centralized, and the execution of actions is decentralized. Applications of such setting are: (i) self-play: we train an agent to play against itself; (ii) adversarial robustness, inverse RL, and imitation learning: we aim to find a robust model; (iii) robust control [Zhou et al., 1996]: we train agents to be robust against attackers; (iv) athletic games analysis, e.g., soccer and basketball: we train models of teams in simulation, and let them play against each other to discover tactics and strategies; (v) Robocup World (robots soccer): we train our team in our lab before deploying it in the real match; (vi) AI economist [Zheng et al., 2020]: we run a game between workers along with rule-makers to discover new tax laws, and many more real-world problems.

Our empirical study shows that CoPG and TRCoPO can excel in this setting and have clear advantages compared to existing algorithms. While the centralized optimization setting in CoPO has a vast range of real-world applications, there are problems that require decentralized optimization. We showed that CoPG-OP, a decentralized extension of CoPG, where each agent also learns its opponent’s parameters/model to compute its policy update, comes with the same benefits as CoPG in the centralized setting.

7 CONCLUSION

We presented competitive policy optimization CoPO, a novel PG-based RL method for two player strictly competitive game. In CoPO, each player optimizes strategy by considering the interaction with the environment and the opponent through game theoretic bilinear approximation to the game objective. This method is instantiated to competitive policy gradient (CoPG) and trust region competitive policy optimisation (TRCoPO) using value based and trust region approaches. We theoretically studied these methods and provided PG theorems to show the properties of these model-free RL approaches. We provided efficient implementation of these methods and empirically showed that they provide stable and faster optimization, and also converge to more sophisticated and competitive strategies. We performed case studies for CoPO based approach on self-play, asymmetric mini-max game GAIL, with opponent modelling and further discussed the general applicability of the CoPO paradigm in various real life settings. We dedicated this paper to two player zero-sum games, however, the principles provided in this paper can be used for multi-player general games. In the future, we plan to extend this study to multi-player general-sum games along with efficient implementation of methods. Moreover, we plan to use the techniques proposed in partially observable domains, and study imperfect information games [Azizzadenesheli et al., 2020].

Acknowledgements

The main body of this work took place when M. Prajapat was a visiting scholar at Caltech. The authors would like to thank Florian Schäfer for his support. M. Prajapat is thankful to Zeno Karl Schindler foundation for providing him with a Master thesis grant. K. Azizzadenesheli is supported in part by Raytheon and Amazon Web Service. A. Anandkumar is supported in part by Bren endowed chair, DARPA PAIHR00111890035 and LwLL grants, Raytheon, Microsoft, Google, and Adobe faculty fellowships.

References


Appendix

8 ALGORITHMS

In this section, we briefly present the algorithms discussed in the paper, namely gradient descent ascent (GDA), competitive policy gradient (CoPG), trust region gradient descent ascent (TRGDA) and trust region competitive policy optimization (TRCoPo).

Algorithm 1: Gradient Descent Ascent Policy Gradient

Initialize $(\theta^1, \theta^2)$

for epoch = 1, 2, 3, .. until termination do
  Collect samples under $\pi(.|; \theta^1), \pi(.|; \theta^2)$,
  Estimate $Q$, then $D_{\theta^i} \eta$ in Eq. 7
  Update $\theta^1, \theta^2$ using Eq. 4
end

Algorithm 2: Competitive Policy Gradient

Initialize $(\theta^1, \theta^2)$

for epoch = 1, 2, 3, .. until termination do
  Collect samples under $\pi(.|; \theta^1), \pi(.|; \theta^2)$,
  Estimate $Q$, then $D_{\theta^i} \eta, D_{\theta^j} \theta$ in Eqs. 7, 8
  Update $\theta^1, \theta^2$ using Eq. 6
end

Algorithm 3: Trust Region Gradient Descent Ascent

Initialize $(\theta^1, \theta^2)$

for epoch = 1, 2, 3, .. until termination do
  Collect samples under $\pi(.|; \theta^1), \pi(.|; \theta^2)$,
  Estimate $A, D_{\theta^i} L$ using Eq. 63
  Update $\theta^i$ using Eq. 58 with bilinear term as a null matrix.
end

Algorithm 4: Trust Region Competitive Policy Optimisation

Initialize $(\theta^1, \theta^2)$

for epoch = 1, 2, 3, .. until termination do
  Collect samples under $\pi(.|; \theta^1), \pi(.|; \theta^2)$,
  Estimate $A, D_{\theta^i} L, D_{\theta^j} L$ using Eq. 63
  Update $\theta^1, \theta^2$ using Eq. 58
end

9 RECURSION REASONING

When players play strategic games, in order to maximize their payoffs, they need to consider their opponents’ strategies which in turn may depend on their strategies (and so on), resulting in the well known infinite recursion in game theory. This is the recursive reasoning of the form what do I think that you think that I think (and so on) and arises while acting rationally in multi-player games. In this section we elaborate on the recursion in CoMDP and latter show how CoPO recovers the infinite recursion reasoning Fig. 8. Recursion reasoning in CoMDP works as follows:
• Level 1 recursion: Each player considers how the environment, as the results of the agents’ current and future moves temporally evolves in favor of each agent,

• Level 2 recursion: Considering that the other agent also considers how the environment, as the results of the agents’ current and future moves temporally evolves in favor of each agent,

• Level 3 recursion: Considering that the other agent also considers that the other agent also considers how the environment, as the results of the agents’ current and future moves temporally evolves in favor of each agent,

• .......

Figure 8: Recursion reasoning. The notion of "Level N recursion" in the figure assumes that there is no further recursive thinking (cloud) beyond that level.

In the special case of single state games (non sequential setting), this recursive reasoning boils down to what each player thinks, the other player thinks that the other player thinks, ..... (For readers interested in the recursion reasoning in non-sequential but repeated episodic competitive games (single state (or stateless) episodic CoMDP), stability and convergence of competitive gradient optimization and the derivation of Eq. 6, we refer to [Schaefer and Anandkumar, 2019]).

In the following we provide an intuitive explanation for the updates prescribed by CoPO. Let us restate the Nash equilibrium of the bilinear game in Eq. 5 in its matrix form,

\[
\begin{bmatrix}
\theta^1 \\
\theta^2
\end{bmatrix} \leftarrow \begin{bmatrix}
\theta^1 \\
\theta^2
\end{bmatrix} + \alpha \begin{bmatrix}
I & -\alpha D_{\theta^1 \theta^2 \eta} \\
\alpha D_{\theta^2 \theta^1 \eta} & I
\end{bmatrix}^{-1} \begin{bmatrix}
D_{\theta^1 \eta} \\
-D_{\theta^2 \eta}
\end{bmatrix}.
\] (15)

We rewrite the above mentioned update as

\[
\begin{bmatrix}
\theta^1 \\
\theta^2
\end{bmatrix} \leftarrow \begin{bmatrix}
\theta^1 \\
\theta^2
\end{bmatrix} + \alpha(I - A)^{-1} \begin{bmatrix}
D_{\theta^1 \eta} \\
-D_{\theta^2 \eta}
\end{bmatrix}, \quad \text{with} \quad A = \begin{bmatrix}
0 & \alpha D_{\theta^1 \theta^2 \eta} \\
-\alpha D_{\theta^2 \theta^1 \eta} & 0
\end{bmatrix}.
\] (16)

If it holds that the spectral radius of \(A\) is smaller than one, then we can use the Neumann series to compute the inverse,

\[
(I - A)^{-1} = \lim_{N \to \infty} \sum_{k=0}^{N} A^k = I + A + A^2 + A^3 + \ldots.
\] (17)

Using this expansion and its different orders as an approximation, e.g., up to \(k \in \{0, \ldots, N\}\) for a finite \(N\), we derive policy update rules that have different levels of reasoning. For example, consider the following levels of recursion reasoning:
\[ \eta(\theta^1, \theta^2) = \int_{\tau} f(\tau; \theta^1, \theta^2)R(\tau)d\tau, \]  
(20)

thus, the gradient with respect to \( \theta_i \) is given by

\[ D_{\theta_i} \eta = \int_{\tau} D_{\theta_i} f(\tau; \theta^1, \theta^2)R(\tau)d\tau, \quad i \in \{1, 2\}, \]
(21)

for any \( f(\tau; \theta^1, \theta^2) \neq 0 \) using \( D_{\theta_i} \log f(\tau; \theta^1, \theta^2) = \frac{D_{\theta_i} f(\tau; \theta^1, \theta^2)}{f(\tau; \theta^1, \theta^2)} \) from standard calculus and the definition of \( f(\tau; \theta^1, \theta^2) \) in Eq. [1] we can compute the gradient of the game objective,

\[ D_{\theta_i} \eta = \int_{\tau} f(\tau; \theta^1, \theta^2)(D_{\theta_i} \log(\prod_{k=0}^{\lfloor \tau \rfloor - 1} \pi(a_k|s_k; \theta^1)))R(\tau)d\tau, \]

Let us define the score function as \( g(\tau, \theta^1) = D_{\theta^1}(\log(\prod_{k=0}^{\lfloor \tau \rfloor - 1} \pi(a_k|s_k; \theta^1))) \) which results in,

\[ D_{\theta^1} \eta = \int_{\tau} f(\tau; \theta^1, \theta^2)g(\tau, \theta^1)R(\tau)d\tau. \]
(22)
By definition we know that the second order bilinear approximation of the game objective is $D_{\theta_2 \theta_1} \eta(\theta^1, \theta^2) = D_{\theta_2} D_{\theta_1} \eta(\theta^1, \theta^2)$. Hence,

$$D_{\theta_2 \theta_1} \eta = \int_{\tau} D_{\theta_2} f(\tau; \theta^1, \theta^2) g(\tau, \theta^1)^\top R(\tau) d\tau .$$

$$= \int_{\tau} f(\tau; \theta^1, \theta^2)(D_{\theta_2} \log(\prod_{k=0}^{\tau-1} \pi(a_k|s_k; \theta^2)))g(\tau, \theta^1)^\top R(\tau) d\tau ,$$

$$= \int_{\tau} f(\tau; \theta^1, \theta^2) g(\tau, \theta^1)^\top R(\tau) d\tau .$$

which concludes the proof.

\[\square\]

### 10.2 PROOF FOR THM. 1

The competitive policy gradient theorem (Thm. 1) consists of two parts. We first provide the derivation for the gradient and utilise this result to derive the bilinear term in the subsequent subsection.

#### 10.2.1 Derivation of gradient term $D_{\theta_1} \eta$

**Proof.** Using Eq. 2 which defines $V(s_k; \theta^1, \theta^2)$ and $Q(s_k, a_k^1, a_k^2; \theta^1, \theta^2)$. We can write $V(s_0; \theta^1, \theta^2)$ as,

$$V(s_0; \theta^1, \theta^2) = \int \pi(a_0^1|s_0; \theta^1) \pi(a_0^2|s_0; \theta^2) Q(s_0, a_0^1, a_0^2; \theta^1, \theta^2) da_0^1 da_0^2 ,$$

where, $Q(s_0, a_0^1, a_0^2; \theta^1, \theta^2) = r(s_0, a_0^1, a_0^2) + \gamma \int_{s_1} T(s_1|s_0, a_0^1, a_0^2) V(s_1; \theta^1, \theta^2) ds_1$.

Hence, the gradient of the state-value $V$ is,

$$D_{\theta_1} V(s_0; \theta^1, \theta^2) = \int \frac{\partial \pi(a_0^1|s_0; \theta^1)}{\partial \theta^1} \pi(a_0^2|s_0; \theta^2) Q(s_0, a_0^1, a_0^2; \theta^1, \theta^2) da_0^1 da_0^2$$

$$+ \int \frac{\partial \pi(a_0^1|s_0; \theta^1)}{\partial \theta^2} \pi(a_0^2|s_0; \theta^2) \frac{\partial Q(s_0, a_0^1, a_0^2; \theta^1, \theta^2)}{\partial \theta^1} da_0^1 da_0^2 .$$

The gradient of the state-action value $Q$, in term of the gradient of the state-value $V$ is given by

$$D_{\theta_1} Q(s_0, a_0^1, a_0^2; \theta^1, \theta^2) = \gamma \int_{s_1} T(s_1|s_0, a_0^1, a_0^2) \frac{\partial V(s_1; \theta^1, \theta^2)}{\partial \theta^1} ds_1 .$$

Substituting Eq. 25 in Eq. 24 results in,

$$D_{\theta_1} V(s_0; \theta^1, \theta^2) = \int \frac{\partial \pi(a_0^1|s_0; \theta^1)}{\partial \theta^1} \pi(a_0^2|s_0; \theta^2) Q(s_0, a_0^1, a_0^2; \theta^1, \theta^2) da_0^1 da_0^2$$

$$+ \int \frac{\partial \pi(a_0^1|s_0; \theta^1)}{\partial \theta^2} \pi(a_0^2|s_0; \theta^2) \gamma \int_{s_2} T(s_2|s_1, a_0^1, a_0^2) \frac{\partial V(s_1; \theta^1, \theta^2)}{\partial \theta^1} ds_1 da_0^1 da_0^2 .$$

By exploring the recursive nature of the state-value $V$ and $D_{\theta_1} V(s_0; \theta^1, \theta^2)$, we can unroll Eq. 26 as,

$$D_{\theta_1} V(s_0; \theta^1, \theta^2) = \int \frac{\partial \pi(a_1^1|s_1; \theta^1)}{\partial \theta^1} \pi(a_1^2|s_1; \theta^2) Q(s_1, a_1^1, a_1^2; \theta^1, \theta^2) da_1^1 da_1^2$$

$$+ \int \frac{\partial \pi(a_1^1|s_1; \theta^1)}{\partial \theta^2} \pi(a_1^2|s_1; \theta^2) \gamma \int_{s_2} T(s_2|s_1, a_1^1, a_1^2)$$

$$\left( \int \frac{\partial \pi(a_2^1|s_2; \theta^1)}{\partial \theta^1} \pi(a_2^2|s_2; \theta^2) Q(s_2, a_2^1, a_2^2; \theta^1, \theta^2) da_2^1 da_2^2 \right)$$

$$+ \int \frac{\partial \pi(a_2^1|s_2; \theta^1)}{\partial \theta^2} \pi(a_2^2|s_2; \theta^2) \gamma \int_{s_3} T(s_3|s_2, a_2^1, a_2^2)$$

$$\left( \int \frac{\partial \pi(a_3^1|s_3; \theta^1)}{\partial \theta^1} \pi(a_3^2|s_3; \theta^2) Q(s_3, a_3^1, a_3^2; \theta^1, \theta^2) da_3^1 da_3^2 \right)$$
\[
\frac{\partial V(s_2; \theta^1, \theta^2)}{\partial \theta^1} ds_2 da_1^0 da_2^0 \]

to find the derivative of the expected return, we can use the definition of the expected return \(\eta(\theta^1, \theta^2)\) and Eq. 27, which results in

\[
D_{\theta^1} \eta(\theta^1, \theta^2) = \int p(s_0) D_{\theta^1} V(s_0; \theta^1, \theta^2) ds_0
\]

\[
= \int p(s_0) \int_{a_0^0} \int_{a_0^2} \frac{\partial \pi(a_0^1|s_0; \theta^1)}{\partial \theta^1} \pi(a_0^2|s_0; \theta^2) Q(s_0, a_0^1, a_0^2, \theta^1, \theta^2) da_0^0 da_0^2 ds_0
\]

\[
+ \gamma \int f(\tau_{0:1}, s_1; \theta^1, \theta^2) \frac{\partial \pi(a_1^1|s_1; \theta^1)}{\partial \theta^1} \pi(a_1^2|s_1; \theta^2) Q(s_1, a_1^1, a_1^2, \theta^1, \theta^2) d\tau_{0:1}
\]

\[
+ \gamma^2 \int f(\tau_{0:1}, s_2; \theta^1, \theta^2) \frac{\partial V(s_2; \theta^1, \theta^2)}{\partial \theta^1} ds_2 d\tau_{0:1}. \tag{28}
\]

Similar to the two previous steps, unrolling and marginalisation can be continued for the entire length of the trajectory, which results in

\[
D_{\theta^1} \eta(\theta^1, \theta^2) = \int_{\tau_0}^{\tau_{|1|-1}} \sum_{k=0}^{\tau_{k-1}} \gamma^k f(\tau_{0:k-1}, s_k; \theta^1, \theta^2) \int_{a_1^1}^{a_1^2} \int_{a_2^1}^{a_2^2} \frac{\partial \pi(a_1^1|s_k; \theta^2)}{\partial \theta^1} \pi(a_2^2|s_k; \theta^2) Q(s_k, a_1^1, a_2^2, \theta^1, \theta^2) d\tau 
\]

\[
= \int_{\tau_0}^{\tau} \sum_{k=0}^{\tau_{|1|-1}} \gamma^k f(\tau_{0:k}; \theta^1, \theta^2) \frac{\partial \log \pi(a_1^1|s_k; \theta^1)}{\partial \theta^1} Q(s_k, a_1^1, a_2^2, \theta^1, \theta^2) d\tau. \tag{29}
\]

This concludes our derivation of the gradient term of the game objective for the two agent case.

\[\square\]

### 10.2.2 Derivation of Bilinear term \(D_{\theta^1 \theta^2} \eta\)

**Proof:** By definition we know that \(D_{\theta^1 \theta^2} V(s_0; \theta^1, \theta^2) = D_{\theta^1} (D_{\theta^2} V(s_0; \theta^1, \theta^2))\). Thus we can use \(D_{\theta^2} V(s_0; \theta^1, \theta^2)\) as defined in Eq. 27 and differentiating with respect to \(\theta^1\), thereby we get,

\[
D_{\theta^1 \theta^2} V(s_0; \theta^1, \theta^2) = \int_{a_0^1} \int_{a_0^2} \frac{\partial \pi(a_0^1|s_0; \theta^1)}{\partial \theta^1} \frac{\partial \pi(a_0^2|s_0; \theta^2)}{\partial \theta^2} \pi(a_0^2|s_0; \theta^2) Q(s_0, a_0^1, a_0^2, \theta^1, \theta^2) da_0^1 da_0^2
\]

\[
+ \int_{a_0^1} \int_{a_0^2} \frac{\partial \pi(a_0^1|s_0; \theta^2)}{\partial \theta^1} \pi(a_0^2|s_0; \theta^2) Q(s_0, a_0^1, a_0^2, \theta^1, \theta^2) \frac{\partial \pi(a_0^2|s_0; \theta^1)}{\partial \theta^2} da_0^2 da_0^2
\]

\[
+ \int_{a_0^1} \int_{a_0^2} \pi(a_0^1|s_0; \theta^1) \frac{\partial Q(s_0, a_0^1, a_0^2, \theta^1, \theta^2)}{\partial \theta^1} \pi(a_0^2|s_0; \theta^2) da_0^2 da_0^2
\]

\[
+ \int_{a_0^1} \int_{a_0^2} \pi(a_0^1|s_0; \theta^1) \pi(a_0^2|s_0; \theta^2) \frac{\partial^2 Q(s_0, a_0^1, a_0^2, \theta^1, \theta^2)}{\partial \theta^1 \partial \theta^2} da_0^1 da_0^2. \tag{31}
\]

Exploiting the recursive structure of \(D_{\theta^1 \theta^2} V(s_0; \theta^1, \theta^2)\), we can evaluate \(D_{\theta^1 \theta^2} V(s_1; \theta^1, \theta^2)\) and substitute it back in Eq. 31, we get,

\[
D_{\theta^1 \theta^2} V(s_0; \theta^1, \theta^2) = \int_{a_0^1} \int_{a_0^2} \frac{\partial \pi(a_0^1|s_0; \theta^1)}{\partial \theta^1} \frac{\partial \pi(a_0^2|s_0; \theta^2)}{\partial \theta^2} Q(s_0, a_0^1, a_0^2, \theta^1, \theta^2) da_0^1 da_0^2
\]

\[
+ \int_{a_0^1} \int_{a_0^2} \frac{\partial \pi(a_0^1|s_0; \theta^1)}{\partial \theta^1} \pi(a_0^2|s_0; \theta^2) Q(s_0, a_0^1, a_0^2, \theta^1, \theta^2) \frac{\partial \pi(a_0^2|s_0; \theta^1)}{\partial \theta^2} da_0^2 da_0^2
\]

\[
+ \int_{a_0^1} \int_{a_0^2} \pi(a_0^1|s_0; \theta^1) \frac{\partial Q(s_0, a_0^1, a_0^2, \theta^1, \theta^2)}{\partial \theta^1} \pi(a_0^2|s_0; \theta^2) da_0^2 da_0^2
\]

\[
+ \int_{a_0^1} \int_{a_0^2} \pi(a_0^1|s_0; \theta^1) \pi(a_0^2|s_0; \theta^2) \left( \gamma \int_{s_1} T(s_1|s_0, a_0^1, a_0^2) \right) da_0^1 da_0^2 \tag{32}
\]


When investigating Eq. 32 note that the fourth term of Eq. 31
\begin{align*}
&\int \int \frac{\partial \pi(a_1^1|s_1; \theta^1)}{\partial \theta^1} \left( \frac{\partial \pi(a_2^1|s_1; \theta^2)}{\partial \theta^2} \right)^T Q(s_1, a_1^1, a_2^1; \theta^1, \theta^2) da_1^1 da_2^1 \\
&+ \int \int \frac{\partial \pi(a_1^1|s_1; \theta^1)}{\partial \theta^1} \left( \frac{\partial Q(s_1, a_1^1, a_2^1; \theta^2)}{\partial \theta^2} \right)^T da_1^1 da_2^1 \\
&+ \int \int \pi(a_1^1|s_1; \theta^1) \left( \frac{\partial Q(s_1, a_1^1, a_2^1; \theta^2)}{\partial \theta^2} \right)^T da_1^1 da_2^1 \\
&+ \int \int \pi(a_1^1|s_1; \theta^1) \pi(a_2^1|s_1; \theta^2) \gamma \int T(s_2|s_1, a_1^1, a_2^1) \left( \frac{\partial^2 V(s_2; \theta^1, \theta^2)}{\partial \theta^2 \partial \theta^1} da_1^1 da_2^1 \right) ds_1 da_1^0 da_2^0.
\end{align*}

Exploring the recursive structure, quickly leads to long sequences of terms, to deal with this let us define a short hand notation, the first term in Eq. 31 is denoted by \( b_0 \), second term by \( e_0 \), third term by \( d_0 \), forth term as \( e_0 \), note that the subscript indicates the time step and let \( Pr_k = Pr(s_k \rightarrow s_{k+1}, \theta^1, \theta^2) \) be the state transition probability summed over all actions. When investigating Eq. 32 note that the forth term of Eq. 31 expanded to \( e_0 = \gamma \int s_1 Pr_0(b_1 + c_1 + d_1 + e_1) ds_1 \). Thus, recursively applying this insight and using our short hand notation results in,
\begin{align*}
D_{\theta^1 \theta^2} V(s_0; \theta^1, \theta^2) &= b_0 + c_0 + d_0 + \gamma \int \int Pr_0 \left( b_1 + c_1 + d_1 + \gamma \int \int Pr_1 \left( b_2 + c_2 + d_2 \right) \\
&\quad + \gamma \int \int Pr_2 \left( b_3 + c_3 + d_3 + \gamma \int \int Pr_3 \left( \ldots \right) ds_4 \right) ds_3 \right) ds_2 ds_1 \\
&= b_0 + \gamma \int \int Pr_0 \left( b_1 + \gamma \int \int Pr_1 \left( b_2 + \gamma \int \int Pr_2 \left( b_3 + \gamma \int \int Pr_3 \left( \ldots \right) ds_4 \right) ds_3 \right) ds_2 \right) ds_1 \\
&\quad + c_0 + \gamma \int \int Pr_0 \left( c_1 + \gamma \int \int Pr_1 \left( c_2 + \gamma \int \int Pr_2 \left( c_3 + \gamma \int \int Pr_3 \left( \ldots \right) ds_4 \right) ds_3 \right) ds_2 \right) ds_1 \\
&\quad + d_0 + \gamma \int \int Pr_0 \left( d_1 + \gamma \int \int Pr_1 \left( d_2 + \gamma \int \int Pr_2 \left( d_3 + \gamma \int \int Pr_3 \left( \ldots \right) ds_4 \right) ds_3 \right) ds_2 \right) ds_1.
\end{align*}

Summing over the initial state distribution results in,
\begin{align*}
D_{\theta^1 \theta^2} \eta(\theta^1, \theta^2) &= \int p(s_0) D_{\theta^1 \theta^2} V(s_0; \theta^1, \theta^2) ds_0 \\
&= \int p(s_0) \left( b_0 + \gamma \int \int Pr_0 \left( b_1 + \gamma \int \int Pr_1 \left( b_2 + \gamma \int \int Pr_2 \left( b_3 + \gamma \int \int Pr_3 \left( \ldots \right) ds_4 \right) ds_3 \right) ds_2 \right) ds_1 \right) ds_0 \\
&\quad + \int p(s_0) \left( c_0 + \gamma \int \int Pr_0 \left( c_1 + \gamma \int \int Pr_1 \left( c_2 + \gamma \int \int Pr_2 \left( c_3 + \gamma \int \int Pr_3 \left( \ldots \right) ds_4 \right) ds_3 \right) ds_2 \right) ds_1 \right) ds_0 \\
&\quad + \int p(s_0) \left( d_0 + \gamma \int \int Pr_0 \left( d_1 + \gamma \int \int Pr_1 \left( d_2 + \gamma \int \int Pr_2 \left( d_3 + \gamma \int \int Pr_3 \left( \ldots \right) ds_4 \right) ds_3 \right) ds_2 \right) ds_1 \right) ds_0.
\end{align*}

Eq. 33 is clearly build up by three terms the first depending on \( b_k \), the second on \( c_k \) and the last on \( d_k \), let us denote these terms as \( \mathcal{T}_1, \mathcal{T}_2 \) and \( \mathcal{T}_3 \) respectively. Thus we have \( D_{\theta^1 \theta^2} \eta(\theta^1, \theta^2) = \mathcal{T}_1 + \mathcal{T}_2 + \mathcal{T}_3 \). Given this separation, we can tackle each part separately.

**Solving for the \( \mathcal{T}_1 \):**

To bring \( \mathcal{T}_1 \) in Eq. 33 to the final form in our theorem, let us substitute the terms for \( b_k \) back in,
\begin{align*}
\mathcal{T}_1 &= \int p(s_0) \left( \int \int \frac{\partial \pi(a_1^1|s_0; \theta^1)}{\partial \theta^1} \left( \frac{\partial \pi(a_2^1|s_0; \theta^2)}{\partial \theta^2} \right)^T Q(s_0, a_0^1, a_0^2; \theta^1, \theta^2) da_0^1 da_0^2 \\
&\quad + \gamma \int \int Pr(s_1|s_0; \theta^1, \theta^2) \left( \int \int \frac{\partial \pi(a_1^1|s_1; \theta^1)}{\partial \theta^1} \left( \frac{\partial \pi(a_2^1|s_1; \theta^2)}{\partial \theta^2} \right)^T Q(s_1, a_1^1, a_2^1; \theta^1, \theta^2) da_1^1 da_2^1 \\
&\quad + \int \int \pi(a_1^1|s_1; \theta^1) \pi(a_2^1|s_1; \theta^2) \gamma \int T(s_2|s_1, a_1^1, a_2^1) \left( \ldots \right) ds_2 da_1^1 da_2^1 \right) ds_1 da_1^0 da_2^0 \right) ds_0.
\end{align*}
Similar to Eq. [28] we can unrolling and marginalizing, which allows us to bring it into a compact form,

\[
\mathcal{T}_1 = \int_\tau^{|\tau|-1} \sum_{k=0}^n \gamma^k f(\tau_{0:k-1}, s_k; \theta^1, \theta^2) \frac{\partial \pi(a^1_k|s_k; \theta^1)}{\partial \theta^1} \frac{\partial \pi(a^2_k|s_k; \theta^2)}{\partial \theta^2}^\top Q(s_k, a^1_k, a^2_k; \theta^1, \theta^2) \, d\tau
\]

\[
= \int_\tau^{|\tau|-1} \sum_{k=0}^n \gamma^k f(\tau_{0:k}; \theta^1, \theta^2) \frac{\partial \log \pi(a^1_k|s_k; \theta^1)}{\partial \theta^1} \frac{\partial \log \pi(a^2_k|s_k; \theta^2)}{\partial \theta^2}^\top Q(s_k, a^1_k, a^2_k; \theta^1, \theta^2) \, d\tau. \tag{35}
\]

**Solving for the \( \mathcal{T}_2 \):**

For \( \mathcal{T}_2 \) in Eq. [33] we again substitute the \( c_k \) terms back in, which gets us,

\[
\mathcal{T}_2 = \int_{s_0} p(s_0) \left( \int_{a_0^1} \int_{a_0^2} \frac{\partial \pi(a^1_0|s_0; \theta^1)}{\partial \theta^1} \pi(a^2_0|s_0; \theta^2) \frac{\partial Q(s_0, a^1_0, a^2_0; \theta^1, \theta^2)}{\partial \theta^2} \right)^\top \, da_0^1 da_0^2
\]

\[
+ \int_{a_1^1} \int_{a_1^2} \pi(a^1_1|s_1; \theta^1) \frac{\partial \pi(a^2_1|s_1; \theta^2)}{\partial \theta^2} \left( \int_{s_1} T(s_1|s_0, a^1_0, a^2_0) \right) \frac{\partial Q(s_1, a^1_1, a^2_1; \theta^1, \theta^2)}{\partial \theta^2} \right)^\top \, da_1^1 da_1^2
\]

\[
+ \int_{a_1^1} \int_{a_1^2} \pi(a^1_1|s_1; \theta^1) \pi(a^2_1|s_1; \theta^2) \gamma \int_{s_2} T(s_2|s_1, a^1_1, a^2_1) \left( \frac{\partial Q(s_1, a^1_1, a^2_1; \theta^1, \theta^2)}{\partial \theta^2} \right)^\top \, da_1^1 da_1^2
\]

\[
\left( \int_{a_2^1} \int_{a_2^2} \pi(a^1_2|s_1; \theta^1) \pi(a^2_2|s_1; \theta^2) \gamma \int_{s_2} T(s_2|s_1, a^1_1, a^2_1) \right)^\top \, ds_2 da_1^1 da_1^2 ds_0.
\]

Note that \( \mathcal{T}_2 \) contains \( \frac{\partial Q(s, a^1, a^2; \theta^1, \theta^2)}{\partial \theta^2} \). We can unroll \( \frac{\partial Q(s, a^1, a^2; \theta^1, \theta^2)}{\partial \theta^2} \) using Eq. [25] and Eq. [26] till the next state \( s_1 \), which results in,

\[
\mathcal{T}_2 = \int_{s_0} p(s_0) \left( \int_{a_0^1} \int_{a_0^2} \frac{\partial \pi(a^1_0|s_0; \theta^1)}{\partial \theta^1} \pi(a^2_0|s_0; \theta^2) \int_{s_1} \gamma T(s_1|s_0, a^1_0, a^2_0) \right.
\]

\[
\left( \int_{a_1^1} \int_{a_1^2} \pi(a^1_1|s_1; \theta^1) \frac{\partial \pi(a^2_1|s_1; \theta^2)}{\partial \theta^2} \frac{\partial Q(s_1, a^1_1, a^2_1; \theta^1, \theta^2)}{\partial \theta^2} \right)^\top \, da_1^1 da_1^2
\]

\[
+ \int_{a_1^1} \int_{a_1^2} \pi(a^1_1|s_1; \theta^1) \pi(a^2_1|s_1; \theta^2) \gamma \int_{s_2} T(s_2|s_1, a^1_1, a^2_1) \right)^\top \, da_1^1 da_1^2
\]

\[
\left( \int_{a_2^1} \int_{a_2^2} \pi(a^1_2|s_1; \theta^1) \pi(a^2_2|s_1; \theta^2) \gamma \int_{s_2} T(s_2|s_1, a^1_1, a^2_1) \right)^\top \, ds_2 da_1^1 da_1^2 ds_0.
\]

Using log trick \( (D_{\theta^1} \log \pi(a^1_k|s_k; \theta^1) = D_{\theta^1} \pi(a^1_k|s_k; \theta^1) / \pi(a^1_k|s_k; \theta^1), i \in 1, 2 \) \) and combining terms together, we get,

\[
\mathcal{T}_2 = \int_{s_0} p(s_0) \left( \int_{a_0^1} \int_{a_0^2} \pi(a^1_0|s_0; \theta^1) \pi(a^2_0|s_0; \theta^2) \int_{s_1} \gamma T(s_1|s_0, a^1_0, a^2_0) \int_{a_1^1} \int_{a_1^2} \pi(a^1_1|s_1; \theta^1) \pi(a^2_1|s_1; \theta^2) \right.
\]

\[
\left( \frac{\partial \log \pi(a^1_0|s_0; \theta^1)}{\partial \theta^1} \frac{\partial \log \pi(a^2_1|s_1; \theta^2)}{\partial \theta^2} \right)^\top Q(s_1, a^1_1, a^2_1; \theta^1, \theta^2)
\]

\[
+ \left( \frac{\partial \log \pi(a^1_0|s_0; \theta^1)}{\partial \theta^1} + \frac{\partial \log \pi(a^1_1|s_1; \theta^1)}{\partial \theta^1} \right) \frac{\partial Q(s_1, a^1_1, a^2_1; \theta^1, \theta^2)}{\partial \theta^1}
\]

\[
+ \left( \frac{\partial \log \pi(a^2_1|s_1; \theta^2)}{\partial \theta^2} + \frac{\partial \log \pi(a^2_1|s_1; \theta^2)}{\partial \theta^2} \right) \frac{\partial Q(s_1, a^1_1, a^2_1; \theta^1, \theta^2)}{\partial \theta^2}
\]

\[
+ \gamma \int_{s_2} T(s_2|s_1, a^1_1, a^2_1)
\]

\[
\left( \int_{a_2^1} \int_{a_2^2} \pi(a^1_2|s_1; \theta^1) \pi(a^2_2|s_1; \theta^2) \gamma \int_{s_2} T(s_2|s_1, a^1_1, a^2_1) \right)^\top \, ds_2 da_1^1 da_1^2 ds_0.
\]
We can use the identical method for the remaining \( \frac{\partial Q(s_k, a_k^1, a_k^2; \theta^1, \theta^2)}{\partial \theta} \). Thus unrolling this term for one time step and using the log trick to combine terms. For example unrolling the second step results in,

\[
\mathcal{F}_2 = \int_{\tau} f(\tau_{0:1}; \theta^1, \theta^2) \gamma \left( \sum_{k=1}^{N} \frac{\partial \log \pi(a_0^1|s_0; \theta^1)}{\partial \theta^1} \frac{\partial \log \pi(a_0^2|s_1; \theta^2)}{\partial \theta^2} Q(s_1, a_1^1, a_1^2; \theta^1, \theta^2) \right) Q(s_1, a_1^1, a_1^2; \theta^1, \theta^2) d\tau_{0:1} \]

Finally, on unrolling for \(|\tau|\) steps we get,

\[
\mathcal{F}_2 = \int_{\tau} \sum_{k=1}^{N} \frac{\partial \log (\prod_{k=1}^{N} \pi(a_k|s_k; f_k))}{\partial \theta^1} \frac{\partial \log (\prod_{k=1}^{N} \pi(a_k^2|s_k; f_k))}{\partial \theta^2} Q(s_k, a_k^1, a_k^2; \theta^1, \theta^2) d\tau. \quad (37)
\]

**Solving for the \( \mathcal{F}_3 \):**

In \( \mathcal{F}_3 \) in Eq. (33) similar to the previous two terms we again substitute \( d_k \) back in, which results in,

\[
\mathcal{F}_3 = \int_{s_0} p(s_0) \left( \int_{a_0} \int_{a_0} \pi(a_0^1|s_0; \theta^1) \frac{\partial Q(s_0, a_0^1, a_0^2; \theta^1, \theta^2)}{\partial \theta^1} \frac{\partial \pi(a_0^2|s_0; \theta^2)}{\partial \theta^2} \right) d\theta^1 d\theta^2 \]

\[
+ \int_{a_0} \int_{a_0} \pi(a_0^1|s_0; \theta^1) \frac{\partial Q(s_0, a_0^1, a_0^2; \theta^1, \theta^2)}{\partial \theta^1} \frac{\partial \pi(a_0^2|s_0; \theta^2)}{\partial \theta^2} \gamma \int_{s_1} T(s_1|s_0; a_0^1, a_0^2) \]

\[
\left( \int_{a_1^1} \int_{a_1^2} \pi(a_1^1|s_1; \theta^1) \frac{\partial Q(s_1, a_1^1, a_1^2; \theta^1, \theta^2)}{\partial \theta^1} \frac{\partial \pi(a_1^2|s_1; \theta^2)}{\partial \theta^2} \right) d\theta^1 d\theta^2 \]

\[
+ \int_{a_1^1} \int_{a_1^2} \pi(a_1^1|s_1; \theta^1) \frac{\partial Q(s_1, a_1^1, a_1^2; \theta^1, \theta^2)}{\partial \theta^1} \frac{\partial \pi(a_1^2|s_1; \theta^2)}{\partial \theta^2} \gamma \int_{s_2} T(s_2|s_1, a_1^1, a_1^2) \]

\[
\left( \ldots \right) d\tau_{s_2} d\theta^1 d\theta^2 \right) ds_1 da_0^1 da_0^2 d\theta^1 d\theta^2 \right) ds_0.
\]

Similar to \( \mathcal{F}_2, \mathcal{F}_3 \) contains \( \frac{\partial Q(s_k, a_k^1, a_k^2; \theta^1, \theta^2)}{\partial \theta} \), using the same approach and unfold this term using Eq. 25 and Eq. 26 for one step results in,

\[
\mathcal{F}_3 = \int_{s_0} p(s_0) \left( \int_{a_0} \int_{a_0} \pi(a_0^1|s_0; \theta^1) \int_{s_1} \gamma T(s_1|s_0; a_0^1, a_0^2) \right) \]

\[
\left( \int_{a_1^1} \int_{a_1^2} \frac{\partial \pi(a_1^1|s_1; \theta^1)}{\partial \theta^1} \frac{\partial Q(s_1, a_1^1, a_1^2; \theta^1, \theta^2)}{\partial \theta^2} \right) d\theta^1 d\theta^2 \right) ds_0.
\]
\[
\begin{align*}
&+ \int_{a_1^k} \int_{a_2^k} \int_{a_3^k} \pi(a_0^k|s_0; \theta^1) \pi(a_0^0|s_0; \theta^2) \gamma \int_{s_1} T(s_1|s_0; a_0^k, a_0^0) \\
&\left( \int_{a_1^k} \int_{a_2^k} \int_{a_3^k} \pi(a_1^k|s_1; \theta^1) \frac{\partial Q(s_1, a_1^k, a_2^k; \theta^1, \theta^2)}{\partial \theta^1} \frac{\partial \pi(a_0^0|s_0; \theta^2)}{\partial \theta^2} \right) da_1 da_2 da_3 \\
&\quad + \int_{a_1^k} \int_{a_2^k} \int_{a_3^k} \pi(a_1^k|s_1; \theta^1) \pi(a_2^k|s_1; \theta^2) \gamma \int_{s_2} T(s_2|s_1, a_1^k, a_2^k) \\
&\left( \int_{a_1^k} \int_{a_2^k} \int_{a_3^k} \pi(a_1^k|s_2; \theta^1) \frac{\partial \pi(a_0^0|s_0; \theta^2)}{\partial \theta^2} \right) da_1 da_2 da_3 ds_1 da_0^k da_0^0 ds_0.
\end{align*}
\]

Using log trick \( D_{\theta^1} \log \pi(a_k^i|s_k; \theta^i) = D_{\theta^1} \pi(a_k^i|s_k; \theta^i)/\pi(a_k^i|s_k; \theta^i), i \in 1, 2 \) and combining terms together, we get,

\[
\begin{align*}
\mathcal{F}_3 &= \int_{s_0} p(s_0) \left( \int_{a_1^k} \int_{a_2^k} \int_{a_3^k} \pi(a_0^k|s_0; \theta^1) \pi(a_0^0|s_0; \theta^2) \gamma \int_{s_1} T(s_1|s_0, a_0^k, a_0^0) \int_{a_1^k} \int_{a_2^k} \int_{a_3^k} \pi(a_1^k|s_1; \theta^1) \pi(a_2^k|s_1; \theta^2) \\
&\quad \left( \frac{\partial \log \pi(a_0^k|s_0; \theta^1)}{\partial \theta^1} \frac{\partial \log \pi(a_0^0|s_0; \theta^2)}{\partial \theta^2} \right) Q(s_1, a_1^k, a_2^k; \theta^1, \theta^2) \\
&\quad + \frac{\partial Q(s_1, a_1^k, a_2^k; \theta^1, \theta^2)}{\partial \theta^1} \left( \frac{\partial \log \pi(a_0^0|s_0; \theta^2)}{\partial \theta^2} + \frac{\partial \log \pi(a_2^k|s_1; \theta^2)}{\partial \theta^2} \right) \right) \gamma \int_{s_2} T(s_2|s_1, a_1^k, a_2^k) \left( \cdots \right) ds_2 \right) ds_1 da_0^k da_0^0 ds_0.
\end{align*}
\]

by simplifying the notation we get,

\[
\begin{align*}
\mathcal{F}_3 &= \int_{\tau} f(\tau_0; \theta^1, \theta^2) \gamma \left( \frac{\partial \log \pi(a_0^i|s_0; \theta^1)}{\partial \theta^1} \frac{\partial \log \pi(a_0^0|s_0; \theta^2)}{\partial \theta^2} \right) Q(s_0, a_0^i, a_0^0; \theta^1, \theta^2) \\
&\quad + \frac{\partial Q(s_k, a_k^i, a_k^0; \theta^1, \theta^2)}{\partial \theta^i} \left( \frac{\partial \log \pi(a_0^0|s_0; \theta^2)}{\partial \theta^2} + \frac{\partial \log \pi(a_2^k|s_1; \theta^2)}{\partial \theta^2} \right) \gamma \int_{s_2} T(s_2|s_1, a_1^k, a_2^k) \left( \cdots \right) ds_2 \right) d\tau_{0:1}.
\end{align*}
\]

The remaining terms in the recursive structure again contain terms of the form \( \frac{\partial Q(s_k, a_k^i, a_k^0; \theta^1, \theta^2)}{\partial \theta^i} \), which we simplify in a similar fashion by unfolding for one step, for example unfolding the second term for one step results in,

\[
\begin{align*}
\mathcal{F}_3 &= \int_{\tau} f(\tau_0; \theta^1, \theta^2) \gamma \left( \frac{\partial \log \pi(a_0^k|s_0; \theta^1)}{\partial \theta^1} \frac{\partial \log \pi(a_0^k|s_0; \theta^2)}{\partial \theta^2} \right) Q(s_0, a_0^k, a_0^0; \theta^1, \theta^2) d\tau_{0:1} \\
&\quad + \int_{\tau} f(\tau_0; \theta^1, \theta^2) \gamma^2 \left( \frac{\partial \log \pi(a_1^k|s_2; \theta^1)}{\partial \theta^1} \left( \frac{\partial \log \pi(a_0^0|s_0; \theta^2)}{\partial \theta^2} + \frac{\partial \log \pi(a_2^k|s_1; \theta^2)}{\partial \theta^2} \right) \right) \gamma \int_{s_2} T(s_2|s_1, a_1^k, a_2^k) \left( \cdots \right) ds_2 d\tau_{0:2}.
\end{align*}
\]

Finally on unrolling for \(|\tau|\) steps we get,

\[
\begin{align*}
&= \int_{\tau} \sum_{k=0}^{|\tau|-1} \gamma^k f(\tau_{0:k}; \theta^1, \theta^2) \frac{\partial \log \pi(a_1^k|s_k; \theta^1)}{\partial \theta^1} \frac{\partial \log \left( \prod_{i=0}^{k-1} \pi(a_{2}^{i}|s_{i}; \theta^2) \right)}{\partial \theta^2} \gamma \int_{s_2} T(s_2|s_1, a_1^k, a_2^k) \left( \cdots \right) ds_2 d\tau.
\end{align*}
\]

Combining all 3 terms:

By Eq. (33) we know that the bilinear term of the game objective is given by \( D_{\theta^1, \theta^2} \eta(\theta^1, \theta^2) = \mathcal{F}_1 + \mathcal{F}_2 + \mathcal{F}_3 \), or in other words the sum of the three terms derived above. Thus, we derived that,

\[
D_{\theta^1, \theta^2} \eta(\theta^1, \theta^2) = \int_{\tau} \sum_{k=0}^{|\tau|-1} \gamma^k f(\tau_{0:k}; \theta^1, \theta^2) \frac{\partial \log \pi(a_1^k|s_k; \theta^1)}{\partial \theta^1} \frac{\partial \log \pi(a_2^k|s_k; \theta^2)}{\partial \theta^2} \gamma \int_{s_2} T(s_2|s_1, a_1^k, a_2^k) \left( \cdots \right) ds_2 d\tau.
\]
we extend Thm. 1 to estimate gradient and bilinear terms using the advantage function. A popular choice for a baseline. Given that the advantage function is defined as
\[ Q(s, a_k, a_{k+1}; \theta^1, \theta^2) \]
This is equivalent to
\[ Q(s, a_k, a_{k+1}; \theta^1, \theta^2) \]
The gradient and bilinear terms can be estimated using Eq. 7 and Eq. 8 based on
\[ Q(s, a_k, a_{k+1}; \theta^1, \theta^2) \]
This concludes our derivation of the bilinear term of the game objective.

11 GENERALIZATION OF COPG TO VARIANTS OF RETURN

11.1 ESTIMATION OF GRADIENT AND BILINEAR TERM USING A BASELINE

The gradient and bilinear terms can be estimated using Eq. 7 and Eq. 8 based on \( Q(s, a_k, a_{k+1}; \theta^1, \theta^2) \) the state-action value function. But in practice, this can perform poorly due to variance and missing the notion of relative improvement. During learning, policy parameters are updated to increase the probability of visiting good trajectories and reducing chances of encountering bad trajectories. But, let us assume good samples have zero reward and bad samples have a negative reward. In this case, the policy distribution will move away from the negative samples but can move in any direction as zero reward does not provide information on being better a state-action pair. This causes high variance and slow learning. In another case, let us assume that a non zero constant is added to the reward function for all states-action pairs, ideally this constant factor should not affect the policy update but in the current formulation using \( Q(s, a_k, a_{k+1}; \theta^1, \theta^2) \) it does because the notion of better or worse sample is with respect to zero. Hence, missing the notion of relative improvement.

To mitigate the issues, a baseline \( b \) can be subtracted from \( Q(s, a_k, a_{k+1}; \theta^1, \theta^2) \). The baseline can be any random variable as long as it is not a function of \( a_k \) and \( a_{k+1} \) [Sutton and Barto 2018]. For an MDP, a good baseline \( b \) is a function of the state \( s \), because let say in some states all actions have high values and we need a high baseline to differentiate the higher valued actions from the less highly valued ones; in other states all actions will have low values and a low baseline is appropriate. This is equivalent to \( Q(s, a_k, a_{k+1}; \theta^1, \theta^2) \leftarrow Q(s, a_k, a_{k+1}; \theta^1, \theta^2) - b(s) \) in Eq. 8 and Eq. 7.

11.2 EXTENDING THM. 1 FOR GRADIENT AND BILINEAR ESTIMATION USING ADVANTAGE FUNCTION

A popular choice for a baseline \( b(s) \) is \( V(s, \theta^1, \theta^2) \), which results in the advantage function defined in Sec. 3.1. In Thm. 3 we extend Thm. 1 to estimate gradient and bilinear terms using the advantage function.

Theorem 3. Given that the advantage function is defined as \( A(s, a^1, a^2; \theta^1, \theta^2) = Q(s, a^1, a^2; \theta^1, \theta^2) - V(s; \theta^1, \theta^2) \) and given Eq. 7 then for player \( i, j \in \{ 1, 2 \} \) and \( i \neq j \), it holds that,
\[
D_{\theta^i \theta^j} \eta = \int_{\tau} \sum_{k=0}^{\tau-1} \gamma^k f(\tau_{0:k}; \theta^1, \theta^2) \frac{\partial}{\partial \theta^i} \log(\prod_{l=0}^{k-1} \pi(a_l^i | s_l; \theta^1)) \frac{\partial}{\partial \theta^j} \log(\prod_{l=0}^{k-1} \pi(a_l^j | s_l; \theta^2)) A(s_k, a_k^1, a_k^2; \theta^1, \theta^2) d\tau,
\]
\[
D_{\theta^i \theta^j} \eta = \int_{\tau} \sum_{k=0}^{\tau-1} \gamma^k f(\tau_{0:k}; \theta^1, \theta^2) \frac{\partial}{\partial \theta^i} \log(\prod_{l=0}^{k-1} \pi(a_l^i | s_l; \theta^1)) \frac{\partial}{\partial \theta^j} \log(\prod_{l=0}^{k-1} \pi(a_l^j | s_l; \theta^2)) A(s_k, a_k^1, a_k^2; \theta^1, \theta^2) d\tau
\]
\[
+ \int_{\tau} \sum_{k=1}^{\tau-1} \gamma^k f(\tau_{0:k}; \theta^1, \theta^2) D_{\theta^i \theta^j} \log(\prod_{l=0}^{k-1} \pi(a_l^i | s_l; \theta^1)) D_{\theta^i \theta^j} (\log(\prod_{l=0}^{k-1} \pi(a_l^j | s_l; \theta^2))) A(s_k, a_k^1, a_k^2; \theta^1, \theta^2) d\tau
\]
\[
+ \int_{\tau} \sum_{k=1}^{\tau-1} \gamma^k f(\tau_{0:k}; \theta^1, \theta^2) D_{\theta^i \theta^j} \log(\prod_{l=0}^{k-1} \pi(a_l^i | s_l; \theta^1)) D_{\theta^i \theta^j} (\log(\prod_{l=0}^{k-1} \pi(a_l^j | s_l; \theta^2))) A(s_k, a_k^1, a_k^2; \theta^1, \theta^2) d\tau.
\]

Proof. The RHS of Eq. 40 is,
\[
\int_{\tau} \sum_{k=0}^{\tau-1} \gamma^k f(\tau_{0:k}; \theta^1, \theta^2) D_{\theta^i \theta^j} (\log(\prod_{l=0}^{k-1} \pi(a_l^i | s_l; \theta^1))) A(s_k, a_k^1, a_k^2; \theta^1, \theta^2) d\tau
\]
The last equality holds since the value function $V(s; \theta^1, \theta^2)$ is independent of the actions, and this concludes the proof for $D_{\theta^*} \eta$. Now consider, the RHS of Eq. (41)

$$
\int_{\tau}^{\tau-1} \sum_{k=0}^{\lceil \tau \rceil - 1} \gamma^k f(\tau_{0:k}; \theta^1, \theta^2) D_{\theta^*}(\log \pi(a^k_k|s_k; \theta^1)) D_{\theta^1}(\log \pi(a^k_k|s_k; \theta^2)) (Q(s, a^1, a^2; \theta^1, \theta^2) - V(s; \theta^1, \theta^2)) d\tau
$$

Using definition of Advantage function, we can equivalently write,

$$
\int_{\tau}^{\tau-1} \sum_{k=0}^{\lceil \tau \rceil - 1} \gamma^k f(\tau_{0:k}; \theta^1, \theta^2) D_{\theta^*}(\log \pi(a^k_k|s_k; \theta^1)) D_{\theta^1}(\log \pi(a^k_k|s_k; \theta^2)) (Q(s, a^1, a^2; \theta^1, \theta^2) - V(s; \theta^1, \theta^2)) d\tau
$$

Using Eq. (8) we can equivalently separate the expression in $D_{\theta^*} \eta$ and terms depending on the value function,

$$
D_{\theta^*} \eta - \int_{\tau}^{\tau-1} \sum_{k=0}^{\lceil \tau \rceil - 1} \gamma^k f(\tau_{0:k}; \theta^1, \theta^2) D_{\theta^*}(\log \pi(a^k_k|s_k; \theta^1)) D_{\theta^1}(\log \pi(a^k_k|s_k; \theta^2)) \top V(s_k; \theta^1, \theta^2) d\tau
$$

$$
- \int_{\tau}^{\tau-1} \sum_{k=1}^{\lceil \tau \rceil - 1} \gamma^k f(\tau_{0:k}; \theta^1, \theta^2) D_{\theta^*}(\log \pi(a^k_k|s_k; \theta^1)) D_{\theta^1}(\log \pi(a^k_k|s_k; \theta^2)) \top V(s_k; \theta^1, \theta^2) d\tau
$$

$$
- \int_{\tau}^{\tau-1} \sum_{k=1}^{\lceil \tau \rceil - 1} \gamma^k f(\tau_{0:k}; \theta^1, \theta^2) D_{\theta^*}(\log \pi(a^k_k|s_k; \theta^1)) D_{\theta^1}(\log \pi(a^k_k|s_k; \theta^2)) \top V(s_k; \theta^1, \theta^2) d\tau,
$$

note that for any $p_0(\tau) \neq 0$ using $D_{\theta} \log p_0(\tau) = D_{\theta} p_0(\tau)/p_0(\tau)$, which allows us to further simplify the expression to,

$$
D_{\theta^*} \eta - \int_{\tau}^{\tau-1} \sum_{k=0}^{\lceil \tau \rceil - 1} \gamma^k f(\tau_{0:k-1}, s_k; \theta^1, \theta^2) D_{\theta^*}(\pi(a^k_k|s_k; \theta^1)) D_{\theta^1}(\pi(a^k_k|s_k; \theta^2)) \top V(s_k; \theta^1, \theta^2) d\tau
$$

$$
- \int_{\tau}^{\tau-1} \sum_{k=1}^{\lceil \tau \rceil - 1} \gamma^k f(\tau_{0:k-1}, s_k, a^k_k; \theta^1, \theta^2) D_{\theta^*}(\pi(a^k_k|s_k; \theta^1)) D_{\theta^1}(\pi(a^k_k|s_k; \theta^2)) \top V(s_k; \theta^1, \theta^2) d\tau
$$

$$
- \int_{\tau}^{\tau-1} \sum_{k=1}^{\lceil \tau \rceil - 1} \gamma^k f(\tau_{0:k-1}, s_k, a^k_k; \theta^1, \theta^2) D_{\theta^*}(\pi(a^k_k|s_k; \theta^1)) D_{\theta^1}(\pi(a^k_k|s_k; \theta^2)) \top V(s_k; \theta^1, \theta^2) d\tau
$$

$$
= D_{\theta^*} \eta,
$$
where the last equality holds since $V(s_k; \theta^1, \theta^2)$ is independent of the actions of any player. This concludes our proof.

11.3 APPROACHES TO ESTIMATE ADVANTAGE FUNCTION

There exist multiple approaches to estimate $A(s_k, a^1_k, a^2_k; \theta^1, \theta^2)$, for example Monte Carlo (Eq. 46), 1-step TD residual (Eq. 47) and t-step TD residual (Eq. 48)

$$A(s_k, a^1_k, a^2_k; \theta^1, \theta^2) = \sum_{j=k}^{|\tau|-1} \gamma^{j-k} r(s_j, a^1_j, a^2_j) - V(s_k; \theta^1, \theta^2), \quad (46)$$

$$A(s_k, a^1_k, a^2_k; \theta^1, \theta^2) = r(s_k, a^1_k, a^2_k) + \gamma V(s_{k+1}; \theta^1, \theta^2) - V(s_k; \theta^1, \theta^2), \quad (47)$$

$$A(s_k, a^1_k, a^2_k; \theta^1, \theta^2) = r(s_k, a^1_k, a^2_k) + \gamma r(s_{k+1}, a^1_{k+1}, a^2_{k+1}) + \cdots + \gamma^{t-1} r(s_{k+t-1}, a^1_{k+t-1}, a^2_{k+t-1}) + \gamma^t V(s_{k+t}; \theta^1, \theta^2) - V(s_k; \theta^1, \theta^2). \quad (48)$$

The Monte Carlo approach experience high variance in gradient estimation, particularly with long trajectories, which may result in slow learning or not learning at all. On the other hand 1-step TD residual has bias towards initialization of the value function. t-step TD residual can lower this effect up to an extent. To trade between bias and variance we propose to use generalized advantage estimation (GAE) \cite{Schulman et al. 2016}, which is an exponentially weighted average over t-step estimators. It is given by,

$$A^{GAE}(s_k, a^1_k, a^2_k; \theta^1, \theta^2) = \sum_{i=0}^{|\tau|-1} (\lambda \gamma)^i \delta^V_{i+k},$$

where, $\delta^V_k = r(s_k, a^1_k, a^2_k) + \gamma V(s_{k+1}; \theta^1, \theta^2) - V(s_k; \theta^1, \theta^2)$.

Note that at $\lambda = 0$, $A^{GAE}(s_k, a^1_k, a^2_k; \theta^1, \theta^2)$ reduces to 1-step TD residual and at $\lambda = 1$ it is equivalent to the Monte Carlo approach to estimate the advantage function. We refer the interested reader to \cite{Schulman et al. 2016} for more details.

12 TRAINING BY SELF-PLAY

As proposed in Alg. 6 we use CoPG to learn a policy for each player, however, it is also possible to use the competitive policy gradient algorithm to learn game strategies by self-play. In contrast to two player training where each player has its own policy model, in self-play, we have one model for both players. Each player then samples actions from the same policy model for two different state instances $s^1 = [s^1_l, s^2_l]$ and $s^2 = [s^2_l, s^1_l]$, where $s^1_l$ and $s^2_l$ are the local states of player 1 and player 2 respectively. Let $\tau = ((s^1_k, s^2_k, a^1_k, a^2_k, r_k), k=0, |\tau|)$ be a trajectory obtained in the game. Using Eq. 7 and Eq. 8 we can define the gradient and bilinear terms as $D_{\theta^1 \eta^1}$ and $D_{\theta^2 \eta^2}$, $i, j \in \{1, 2\}, i \neq j$,

$$D_{\theta^1 \eta^1} = \int_\tau \sum_{k=0}^{|\tau|-1} \gamma^k f(\tau_{0:k}; \theta) D\theta (\log \pi(a^1_k | s^1_k; \theta)) Q(s^1_k, a^1_k, a^2_k; \theta) d\tau, \quad (49)$$

$$D_{\theta^2 \eta^1} = \int_\tau \sum_{k=0}^{|\tau|-1} \gamma^k f(\tau_{0:k}; \theta) D\theta (\log \pi(a^2_k | s^2_k; \theta)) D\theta (\log \pi(a^1_k | s^1_k; \theta))^\top Q(s^1_k, a^1_k, a^2_k; \theta) d\tau$$

$$+ \int_\tau \sum_{k=1}^{|\tau|-1} \gamma^k f(\tau_{0:k}; \theta) D\theta (\log \prod_{i=0}^{k-1} \pi(a^1_i | s^1_i; \theta)) D\theta (\log \pi(a^2_k | s^2_k; \theta))^\top Q(s^1_k, a^1_k, a^2_k; \theta) d\tau$$

$$+ \int_\tau \sum_{k=1}^{|\tau|-1} \gamma^k f(\tau_{0:k}; \theta) D\theta (\log \pi(a^2_k | s^2_k; \theta)) D\theta (\log \prod_{i=0}^{k-1} \pi(a^1_i | s^1_i; \theta))^\top Q(s^1_k, a^1_k, a^2_k; \theta) d\tau. \quad (50)$$

Utilizing the gradients and bilinear term in Eq. 49 and Eq. 50 the parameters are updated twice which results in the following update rule for self-play,

$$\theta^t \leftarrow \theta + \alpha (I + \alpha^2 D_{\theta^1 \eta^1} D_{\theta^2 \eta^2})^{-1} (D_{\theta^1 \eta^1} - \alpha D_{\theta^2 \eta^2} D_{\theta^2 \eta^2}),$$
\[ \theta \leftarrow \theta^t - \alpha (I + \alpha^2 D_{\theta \eta}^2 D_{\theta \eta}^1)^{-1} \left( D_{\theta \eta}^2 + \alpha D_{\theta \eta}^2 D_{\theta \eta}^1 \right), \]  
\[ \text{(51)} \]

where \( \theta^t \) is a temporary intermediate variable\[3\]

**Algorithm 5:** Competitive Policy Gradient with self-play

**Input:** \( \pi(\cdot ; \theta), N, B, \alpha \). Choice of \( A(s^1, a^1, a^2; \theta) \) mentioned in Sec 3.1

**for** epoch : \( e = 1 \) to \( N \) **do**

**for** batch : \( b = 1 \) to \( B \) **do**

**for** \( k = 0 \) to \( \lvert \tau \rvert - 1 \) **do**

Sample \( a_k^1 \sim \pi(a_k^1 | s_k^1; \theta), a_k^2 \sim \pi(a_k^2 | s_k^2; \theta). \)

Execute actions \( a_k = (a_k^1, a_k^2) \)

Record \( \{s_k^1, s_k^2, a_k^1, a_k^2, r_k\} \) in \( \tau_b \)

Record \( \tau_b \) in \( M \)

**end**

1. Estimate \( A(s^1, a^1, a^2; \theta) \) with \( \tau \) in \( M \)
2. Estimate \( D_{\theta \eta}^1, D_{\theta \eta}^2, D_{\theta \eta}^1, D_{\theta \eta}^2 \) using Eq. 49 and Eq. 50
3. Update \( \theta \) using Eq. 51

**end**

We tested the self-play algorithm on the Markov soccer game, which is discussed in more detail in Sec. 15.5. We focus in this comparison on the Markov soccer game since it is strategic enough while being representative. The game has two player A and B. The state vector in perspective of player A is \( s^1 = [s^A, s^B] \) and in player B \( s^2 = [s^B, s^A] \), in accordance with the self-play setting described above. The game description is given in Sec. 15.5. The player was trained while competing with itself using Alg. 5. Fig. 9a below shows the interaction plot for CoPG-SP A:CoPG-SP and B:CoPG-SP against agents trained with two different networks (CoPG), we observe that self-play had slight edge in number of game wins. The is probably due to better generalisation of agents in self-play obtained by exploiting symmetricity in the game. We also trained a GDA agent using self-play and saw no clear improvements compared to normal GDA as tested in Sec. 15.5.

Figure 9: a-b) Interaction plots, representing probability of seizing ball in the game between A vs B. X-axis convention, for player A. \( A_1 \): A scored goal, \( A_2 \): A scored goal after seizing ball from B, \( A_3 \): A scored goal by seizing ball from B which took the ball from A and so forth. Vice versa for player B. N: No one scored goal both kept on seizing ball. c) probability of games won.

### 13 TRUST REGION COMPETITIVE POLICY OPTIMISATION

Following our discussion in the Sec. 3.3 on trust region competitive policy optimization, in this section we will go into more details on constraint competitive optimization and formally derive and proof all our results presented in Sec. 3.3. More precisely, in this section we first proof Lem. 1 and Thm. 2, second we will derive how we can efficiently approximate Eq. 13.
using a bilinear approximation of the objective and a second order approximation of the KL divergence constraint and how this allows us to efficiently find an approximate solution.

13.1 PROOF FOR LEM. 1

Proof. The second term on the RHS of Eq. 10 can be simplified as,

\[
E_{\tau \sim f(\cdot; \theta^1, \theta^2)} \left[ \sum_{k=0}^{|\tau|-1} \gamma^k A(s_k, a_1^k, a_2^k; \theta^1, \theta^2) \right]
\]

\[
= E_{\tau \sim f(\cdot; \theta^1, \theta^2)} \left[ \sum_{k=0}^{|\tau|-1} \gamma^k \left( Q(s_k, a_1^k, a_2^k; \theta^1, \theta^2) - V(s_k; \theta^1, \theta^2) \right) \right]
\]

\[
= E_{\tau \sim f(\cdot; \theta^1, \theta^2)} \left[ \sum_{k=0}^{|\tau|-1} \gamma^k \left( r(s_k, a_1^k, a_2^k) + \gamma V(s_{k+1}; \theta^1, \theta^2) - V(s_k; \theta^1, \theta^2) \right) \right]
\]

\[
= E_{\tau \sim f(\cdot; \theta^1, \theta^2)} \left[ \sum_{k=0}^{|\tau|-1} \gamma^k \left( r(s_k, a_1^k, a_2^k) + \gamma V(s_{k+1}; \theta^1, \theta^2) - \gamma^k V(s_k; \theta^1, \theta^2) \right) \right] - E_{\tau \sim f(\cdot; \theta^1, \theta^2)} \left[ V(s_0; \theta^1, \theta^2) \right]
\]

\[
= \eta(\theta^1, \theta^2) - \eta(\theta^1, \theta^2).
\]

This concludes our proof that,

\[
\eta(\theta^1, \theta^2) = \eta(\theta^1, \theta^2) + E_{\tau \sim f(\cdot; \theta^1, \theta^2)} \sum_{k=0}^{|\tau|-1} \gamma^k A(s, a^1, a^2; \theta^1, \theta^2).
\]

\[\blacksquare\]

13.2 PROOF FOR THM. 2

Proof. Let \( \theta^1, \theta^2 \) be the parametrization of the policies, when these parameters are updated to \( \theta'^1, \theta'^2 \), by Lem. 1 we know that the improved game objective \( \eta(\theta'^1, \theta'^2) \) is,

\[
\eta(\theta'^1, \theta'^2) = \eta(\theta^1, \theta^2) + E_{\tau \sim f(\cdot; \theta^1, \theta^2)} \sum_{k=0}^{|\tau|-1} \gamma^k A(s_k, a_1^k, a_2^k; \theta^1, \theta^2).
\]

Furthermore, we defined \( L_{\theta'^1, \theta'^2}(\theta^1, \theta^2) \) which denotes the surrogate game objective where the trajectory depends on policies \( (\theta^1, \theta^2) \) but actions are sampled from \( (\theta'^1, \theta'^2) \).

\[
L_{\theta'^1, \theta'^2}(\theta^1, \theta^2) = \eta(\theta^1, \theta^2) + E_{\tau \sim f(\cdot; \theta^1, \theta^2)} \sum_{k=0}^{|\tau|-1} \gamma^k E_{\pi(a_1^k|s_k; \theta'^1), \pi(a_2^k|s_k; \theta'^2)} A(s_k, a_1^k, a_2^k; \theta^1, \theta^2).
\]

Note that \( L_{\theta^1, \theta^2}(\theta^1, \theta^2) \) is an approximation of true game objective \( \eta(\theta^1, \theta^2) \). Using the average advantage function,

\[
\overline{A}(s; \theta^1, \theta^2) = \sum_{a^1, a^2} \pi(a^1|s; \theta^1) \pi(a^2|s; \theta^2) A(s, a^1, a^2; \theta^1, \theta^2),
\]

one can write Eq. 53 and Eq. 54 as,

\[
\eta(\theta'^1, \theta'^2) - \eta(\theta^1, \theta^2) = E_{\tau \sim f(\cdot; \theta^1, \theta^2)} \left[ \sum_{k=0}^{|\tau|-1} \gamma^k \overline{A}(s; \theta^1, \theta^2) \right],
\]

\[
\eta(\theta^1, \theta^2) = \eta(\theta^1, \theta^2) + E_{\tau \sim f(\cdot; \theta^1, \theta^2)} \sum_{k=0}^{|\tau|-1} \gamma^k A(s_k, a_1^k, a_2^k; \theta^1, \theta^2).
\]

\[\blacksquare\]
\[ L_{\theta^1, \theta^2}(\theta'^1, \theta'^2) - \eta(\theta^1, \theta^2) = E_{r \sim f(\cdot; \theta^1, \theta^2)} \left[ \sum_{k=0}^{\tau-1} \gamma^k A(s; \theta^1, \theta^2) \right]. \tag{57} \]

To finish our proof, we subtract Eq. 57 from Eq. 56 and take the absolute value, this allows to bound \(|\eta(\theta'^1, \theta'^2) - L_{\theta^1, \theta^2}(\theta'^1, \theta'^2)|\). To derive this bound let \(\epsilon = \max_s \sum_{k=0}^{\tau-1} \gamma^k A(s; \theta^1, \theta^2)\), which allows us to perform the following steps,

\[
|\eta(\theta'^1, \theta'^2) - L_{\theta^1, \theta^2}(\theta'^1, \theta'^2)| = \left| E_{r \sim f(\cdot; \theta^1, \theta^2)} \left[ \sum_{k=0}^{\tau-1} \gamma^k A(s; \theta^1, \theta^2) \right] - E_{r \sim f(\cdot; \theta^2)} \left[ \sum_{k=0}^{\tau-1} \gamma^k A(s; \theta^1, \theta^2) \right] \right|
\]

\[
= \int \left| f(\tau; \theta'^1, \theta'^2) - f(\tau; \theta^1, \theta^2) \right| \left| \sum_{k=0}^{\tau-1} \gamma^k A(s; \theta^1, \theta^2) \right| d\tau
\]

\[
\leq \epsilon D_{TV}(\tau \sim (\theta^1, \theta^2), \tau \sim (\theta'^1, \theta'^2))
\]

\[
\leq \epsilon \frac{1}{2} D_{KL}(\tau \sim (\theta^1, \theta^2), \tau \sim (\theta'^1, \theta'^2))
\]

\[
= \epsilon \sqrt{2} \frac{1}{2} D_{KL}((\theta^1, \theta^2), (\theta'^1, \theta'^2)).
\]

This concludes our proof. \(\square\)

### 13.3 Constraint Competitive Optimization of a Trust Region

In this section, we approach the constrained optimization problem of TRCoPO given by Eq. 13 in 3 steps. First, motivated by CGD we seek for a Nash equilibrium of the bilinear approximation of the game objective. Secondly, in contrast to CoPG, where an L2 penalty term is used along with the bilinear approximation denoting limited confidence in parameter deviation, here a KL divergence constraint will influence the gradient direction. Note that we incorporate this KL divergence constraint into the objective using the method of Lagrange multipliers. This results in the game theoretic update rule defined in Eq. 58

\[
\theta^1 \leftarrow \theta^1 + \arg\max_{\Delta \theta^1; \Delta \theta^1 + \theta^1 \in \Theta^1} \Delta \theta^1 D_{\theta^1} L + \Delta \theta^1 D_{\theta^1, \theta^2} L \Delta \theta^2 - \frac{\lambda}{2} (\Delta \theta^1 A_{11} \Delta \theta^1 + \Delta \theta^2 A_{22} \Delta \theta^2 - 2 \delta'),
\]

\[
\theta^2 \leftarrow \theta^2 + \arg\min_{\Delta \theta^2; \Delta \theta^2 + \theta^2 \in \Theta^2} \Delta \theta^2 D_{\theta^2} L + \Delta \theta^2 D_{\theta^2, \theta^1} L \Delta \theta^1 + \frac{\lambda}{2} (\Delta \theta^1 A_{11} \Delta \theta^1 + \Delta \theta^2 A_{22} \Delta \theta^2 - 2 \delta'). \tag{58} \]

Here, \(\lambda\) is the Lagrange multiplier. Note that the same \(\lambda\) is used for both the agents and that we use a quadratic approximation of the KL divergence, which is \(D_{KL}((\theta^1, \theta^2), (\theta'^1, \theta'^2)) \approx \frac{1}{2} \sum_{i,j} A_{ij} \Delta \theta^i \Delta \theta^j\), where \(A_{ij} = D_{\theta^i \theta^j} D_{KL}((\theta^1, \theta^2), (\theta'^1, \theta'^2))\), and it holds that for \(i \neq j\), \(A_{ij} = 0\). Note that we will go into more details later on the approximation of the KL divergence.

**TRCoPO update**: As said the update step in Eq. 58 is given by the Nash equilibrium of the game within Eq. 58. We can find the Nash equilibrium that fulfills the approximated KL divergence constraint efficiently by solving the following equation using a line-search approach,

\[
\Delta \theta = -(B + \lambda A')^{-1} C, \text{ subject to: } \Delta \theta^T A \Delta \theta \leq \delta, \tag{59} \]

where, let \(A', B, C, \delta, \text{ and } \Delta \theta\) be defined as,

\[
A' = \begin{bmatrix} -A_{11} & 0 \\ 0 & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} 0 & D_{\theta^1 \theta^2} L \\ D_{\theta^2 \theta^1} L & 0 \end{bmatrix},
\]

\[
C = \begin{bmatrix} D_{\theta^1} L \\ D_{\theta^2} L \end{bmatrix}, \quad \delta = 2 \delta', \text{ and } \Delta \theta = \begin{bmatrix} \Delta \theta^1 \\ \Delta \theta^2 \end{bmatrix}. \tag{60} \]

**TRCoPO constraint optimization**: After this quick introduction on how to practically update the policy parameters using TRCoPO, let us derive in detail how to get to Eq. 59. As explained above Eq. 59 is the Nash equilibrium of an approximation
of Eq. [13] Before starting to approximate Eq. [13] let us rewrite it by using the definition of \( L_{\theta_1 \theta_2} (\theta_1', \theta_2') \) as given in Eq. [54]. Note that we can neglect the constant term \( \eta(\theta_1', \theta_2') \), which results in the following game,

\[
\max_{\theta_1'} \min_{\theta_2'} \mathbb{E}_{\tau \sim f(\cdot; \theta_2')} \left[ \sum_{k=0}^{\tau-1} \gamma^k \mathbb{E}_{\pi(a_k^1 | s_k; \theta_1')} \pi(a_k^2 | s_k; \theta_2') A(s_k, a_k^1, a_k^2; \theta_1', \theta_2') \right],
\]

subject to: \( D_{KL}((\theta_1', \theta_2'), (\theta_1', \theta_2')) \leq \delta' \).

Furthermore, we can use an importance sampling factor instead of considering the expected value with respect to the policies parametrized with \((\theta_1', \theta_2')\). This results in the following optimization problem,

\[
\max_{\theta_1'} \min_{\theta_2'} \mathbb{E}_{\tau \sim f(\cdot; \theta_2')} \left[ \sum_{k=0}^{\tau-1} \gamma^k \frac{\pi(a_k^1 | s_k; \theta_1') \pi(a_k^2 | s_k; \theta_2')}{\pi(a_k^1 | s_k; \theta_2')} A(s_k, a_k^1, a_k^2; \theta_1', \theta_2') \right],
\]

subject to: \( D_{KL}((\theta_1', \theta_2'), (\theta_1', \theta_2')) \leq \delta' \).

For the game in Eq. [62] we can compute the gradient \( D_{\theta_1} L \) and bilinear term \( D_{\theta_1 \theta_2} L \) given player \( i, j \in \{1, 2\} \), \( i \neq j \) as

\[
D_{\theta_1} L = \mathbb{E}_{\tau \sim f(\cdot; \theta_2')} \left[ \sum_{k=0}^{\tau-1} \gamma^k D_{\theta_1} \pi(a_k^1 | s_k; \theta_1') \pi(a_k^2 | s_k; \theta_2') A(s_k, a_k^1, a_k^2; \theta_1', \theta_2') \right],
\]

\[
D_{\theta_1 \theta_2} L = \mathbb{E}_{\tau \sim f(\cdot; \theta_2')} \left[ \sum_{k=0}^{\tau-1} \gamma^k D_{\theta_1} \pi(a_k^1 | s_k; \theta_1') D_{\theta_2} \pi(a_k^2 | s_k; \theta_2') A(s_k, a_k^1, a_k^2; \theta_1', \theta_2') \right],
\]

which allows us to formulate the following bilinear approximation of the optimization problem,

\[
\max_{\Delta \theta_1: \Delta \theta_1^2 + \Delta \theta_2 + \Delta \theta_2^2 \in \theta_1^2} \min_{\Delta \theta_2: \Delta \theta_2^2 + \Delta \theta_2^2 \in \theta_2^2} \Delta \theta_1^T D_{\theta_1} L + \Delta \theta_1^T D_{\theta_1 \theta_2} L \Delta \theta_2^2 + \Delta \theta_2^T D_{\theta_2} L + \Delta \theta_2^T D_{\theta_1 \theta_2} L \Delta \theta_2 + \Delta \theta_2^T D_{\theta_2} L \Delta \theta_2^2,
\]

subject to: \( D_{KL}((\theta_1', \theta_2'), (\theta_1', \theta_2')) \leq \delta' \).

However, to efficiently solve the optimization problem we also need to approximate the KL divergence for the two agent case, similar to Schulman et al. [2015] we use a quadratic approximation of the constraint.

**Approximation of KL divergence** Let us derive an approximation of the KL divergence for the two player case that can be estimated efficiently using Monte Carlo samples. As defined in Sec. [3.3] the KL divergence between \((\theta_1, \theta_2)\) and \((\theta_1', \theta_2')\) is given by,

\[
D_{KL}((\theta_1, \theta_2), (\theta_1', \theta_2')) = \int_{\tau} f(\tau; \theta_1, \theta_2) \log \left( \frac{f(\tau; \theta_1', \theta_2')}{f(\tau; \theta_1, \theta_2)} \right) d\tau,
\]

Let us define \( \theta = (\theta_1^T, \theta_2^T)^T \), and \( \Delta \theta = \theta' - \theta \), thus, using a Taylor series to approximate \( D_{KL} \) around \( \theta \) we get,

\[
D_{KL}(\theta, \theta') = D_{KL}(\theta, \theta') \bigg|_{\theta' = \theta} + \frac{\partial D_{KL}(\theta, \theta')}{\partial \theta'} \bigg|_{\theta' = \theta} \Delta \theta + \frac{1}{2} \Delta \theta^T A \Delta \theta + \varepsilon(||\Delta \theta||^3),
\]

where,

\[
A = \left[ D_{\theta_1 \theta_1} D_{KL}((\theta_1, \theta_2), (\theta_1', \theta_2')) \quad D_{\theta_1 \theta_2} D_{KL}((\theta_1, \theta_2), (\theta_1', \theta_2')) \quad D_{\theta_2 \theta_1} D_{KL}((\theta_1, \theta_2), (\theta_1', \theta_2')) \quad D_{\theta_2 \theta_2} D_{KL}((\theta_1, \theta_2), (\theta_1', \theta_2')) \right].
\]

In the Taylor expansion Eq. [65] the zero order term \( D_{KL}(\theta, \theta') \big|_{\theta' = \theta} = 0 \) and the first order term can be expanded as,

\[
\frac{\partial D_{KL}(\theta, \theta')}{\partial \theta'} \bigg|_{\theta' = \theta} = D_{\theta_1} D_{KL}((\theta_1, \theta_2), (\theta_1', \theta_2'))^T, \quad D_{\theta_2} D_{KL}((\theta_1, \theta_2), (\theta_1', \theta_2'))^T
\]

When solving for \( D_{\theta_1} D_{KL}((\theta_1, \theta_2), (\theta_1', \theta_2')) \), we get,

\[
D_{\theta_1} D_{KL}((\theta_1, \theta_2), (\theta_1', \theta_2')) \bigg|_{\theta' = \theta} = D_{\theta_1} \left( \int_{\tau} f(\tau; \theta_1, \theta_2) \log \left( \frac{f(\tau; \theta_1', \theta_2')}{f(\tau; \theta_1, \theta_2)} \right) d\tau \right) \bigg|_{(\theta_1', \theta_2') = (\theta_1', \theta_2')}.
\]
where, $A$ only using first order derivatives is in terms of the Fisher matrix. Using Eq. 69, we get,

$$f = \log f(\theta^1, \theta^2),$$

and now using the definition of $A$ using Monte Carlo samples. Note that an interesting alternative to estimate $D_{\theta^2}D_{KL}(\theta^1, \theta^2)$ is 0. Hence, we write that,

$$\frac{\partial D_{KL}(\theta, \theta')}{\partial \theta'}|_{\theta' = \theta} = 0.$$

Ignoring higher order terms in the approximation, we get

$$D_{KL}(\theta^1, \theta^2, (\theta^1, \theta^2)) \approx \frac{1}{2} [\Delta \theta^1 \Delta \theta^2] \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} [\Delta \theta^1 \Delta \theta^2],$$

where, $A_{ij}$ is defined in Eq. 66 Let us now derive all the $A_{ij}$ terms, starting with $A_{11},$

$$A_{11} = D_{\theta^1 \theta^1} \left( \int f(\tau; \theta^1, \theta^2) \log \left( \frac{f(\tau; \theta^1, \theta^2)}{f(\tau; \theta^1, \theta^2)} \right) d\tau \right)$$

$$= \int_{\tau} D_{\theta^1 \theta^1} \left( f(\tau; \theta^1, \theta^2) \log f(\tau; \theta^1, \theta^2) - f(\tau; \theta^1, \theta^2) \log f(\tau; \theta^1, \theta^2) \right) d\tau$$

$$= \int_{\tau} f(\tau; \theta^1, \theta^2) D_{\theta^1 \theta^1} \log f(\tau; \theta^1, \theta^2) d\tau$$

$$= \int_{\tau} f(\tau; \theta^1, \theta^2) D_{\theta^1 \theta^1} \log f(\tau; \theta^1, \theta^2) d\tau,$$

and now using the definition of $f(\tau; \theta^1, \theta^2)$ in Eq. 1, we get,

$$A_{11} = -\int_{\tau} f(\tau; \theta^1, \theta^2) D_{\theta^1 \theta^1} \left( \log(p(s_0) \prod_{k=0}^{\left|\tau\right|-1} \pi(a_k^1|s_k; \theta^1) \pi(a_k^2|s_k; \theta^2) R(r_k|s_k, a_k^1, a_k^2) T(s_{k+1}|s_k, a_k^1, a_k^2)) \right) d\tau$$

$$= -\int_{\tau} f(\tau; \theta^1, \theta^2) D_{\theta^1 \theta^1} \left( \sum_{k=0}^{\left|\tau\right|-1} \log \pi(a_k^1|s_k; \theta^1) + \log \pi(a_k^2|s_k; \theta^2) \right) d\tau$$

$$= -\int_{\tau} f(\tau; \theta^1, \theta^2) D_{\theta^1 \theta^1} \left( \sum_{k=0}^{\left|\tau\right|-1} \log \pi(a_k^1|s_k; \theta^1) \right) d\tau.$$

This formulation allows us to estimate $A_{11}$ using Monte Carlo samples. Note that an interesting alternative to estimate $A_{11}$ only using first order derivatives is in terms of the Fisher matrix. Using Eq. 69, we get,

$$A_{11} = \int_{\tau} D_{\theta^1 \theta^1} \left( -f(\tau; \theta^1, \theta^2) D_{\theta^1} \log f(\tau; \theta^1, \theta^2) \right) d\tau$$

$$= \int_{\tau} D_{\theta^1 \theta^1} \left( -\frac{f(\tau; \theta^1, \theta^2)}{f(\tau; \theta^1, \theta^2)} D_{\theta^1} f(\tau; \theta^1, \theta^2) \right) d\tau$$

$$= \int_{\tau} \frac{f(\tau; \theta^1, \theta^2)}{f(\tau; \theta^1, \theta^2)} D_{\theta^1 \theta^1} f(\tau; \theta^1, \theta^2)$$

$$+ \frac{f(\tau; \theta^1, \theta^2)}{f(\tau; \theta^1, \theta^2)} D_{\theta^1} f(\tau; \theta^1, \theta^2) D_{\theta^1} f(\tau; \theta^1, \theta^2) d\tau.$$

Similar to Eq. 67 using the above equation we can write $A_{11}$ at $\theta^1 = \theta^1, \theta^2 = \theta^2$ as,

$$A_{11}|_{\theta' = \theta} = \int_{\tau} f(\tau; \theta^1, \theta^2) D_{\theta^1} f(\tau; \theta^1, \theta^2) D_{\theta^1} f(\tau; \theta^1, \theta^2)^\top d\tau.$$
Which by definition is the Fisher information matrix. Using the definition of \( f(\tau; \theta^1, \theta^2) \) in Eq. 1 and further reformulations, we get the following estimate for \( D \)

Thus, the constraint on the KL divergence

Using the same derivation we also get

in Eq. 70, we can find the

\[ A_{11} \bigg|_{\theta' = \theta} = \mathbb{E}_{f(\tau; \theta^1, \theta^2)} \left[ D_{\theta^1} \log(f(\tau; \theta^1, \theta^2)) D_{\theta^1} \log(f(\tau; \theta^1, \theta^2)^T) \right]. \]

Note that for our implementation, we use the first approach Eq. 70 but the second approach has the advantage that it avoids second order derivatives, which can potentially improve computation times. Using the same ideas as in the derivation of \( A_{11} \) in Eq. 70, we can find the \( A_{22} \) term, which is

Next, in order to estimate the Hessian of the KL divergence, we want to derive a formulation for \( A_{12} \). Similar to Eq. 70, \( A_{12} \) can be written as,

Thus, the constraint on the KL divergence \( D_{KL}((\theta^1, \theta^2), (\theta'^1, \theta'^2)) \leq \delta' \) can be approximated by the following quadratic inequality constraint,

which allows us to efficiently find an approximate solution for Eq. 13

**Derivation of the TRCOPO Update:** Given the bilinear approximation of the objective Eq. 63 and the quadratic approximation of the KL divergence constraint Eq. 71, we can use the method of Lagrange multiplier to reformulate the problem as the following optimization problem:

\[
\begin{align*}
\max_{\Delta \theta^1: \Delta \theta^1 + \theta^1 \in \Theta^1} & \quad \Delta \theta^1 \top D_{\theta^1} L + \Delta \theta^1 \top D_{\theta^1} g \Delta \theta^2 - \frac{\lambda}{2} (\Delta \theta^1 \top A_{11} \Delta \theta^1 + \Delta \theta^2 \top A_{22} \Delta \theta^2) - 2\delta', \\
\min_{\Delta \theta^2: \Delta \theta^2 + \theta^2 \in \Theta^2} & \quad \Delta \theta^2 \top D_{\theta^2} L + \Delta \theta^2 \top D_{\theta^2} g \Delta \theta^1 + \frac{\lambda}{2} (\Delta \theta^1 \top A_{11} \Delta \theta^1 + \Delta \theta^2 \top A_{22} \Delta \theta^2) - 2\delta').
\end{align*}
\]

Note that in Eq. 58 the same game was used, and thus finding a Nash equilibrium \((\Delta \theta^1, \Delta \theta^2, *)) to Eq. 72 allows to define the update rule as \((\theta^1, \theta^2) = (\theta^1, \theta^2) + (\Delta \theta^1, *, \Delta \theta^2, *)\). The Nash equilibrium \((\Delta \theta^1, *, \Delta \theta^2)\) of the game, for a fixed \( \lambda \), can be obtained by differentiating Eq. 72 with respect to \( \theta^1, \theta^2 \) and setting the derivatives to zero:

\[
\begin{align*}
D_{\theta^1} L + D_{\theta^1} g \Delta \theta^2 - \lambda (A_{11} \Delta \theta^1) &= 0, \\
D_{\theta^2} L + D_{\theta^2} g \Delta \theta^1 + \lambda (A_{22} \Delta \theta^2) &= 0.
\end{align*}
\]

To solve this system of equations, we can formulate Eq. 73 in matrix form, using \( \Delta \theta = [\Delta \theta^1, \Delta \theta^2] \top \) and \( \delta = 2\delta' \), which results in

\[
\begin{bmatrix}
D_{\theta^1} L \\ D_{\theta^2} L
\end{bmatrix} + \begin{bmatrix}
0 & D_{\theta^1} g L \\
D_{\theta^2} g L & 0
\end{bmatrix} \Delta \theta + \lambda \begin{bmatrix}
-A_{11} & 0 \\
0 & A_{22}
\end{bmatrix} \Delta \theta = 0.
\]
This gives us a one dimensional manifold of solutions for $\Delta \theta$ when varying $\lambda$. Using the notation in Eq. 60, we can reformulate the system of equation as,

$$C + B \Delta \theta + \lambda A^T \Delta \theta = 0,$$

which for non-singular $B + \lambda A^T$ has the following solution,

$$\Delta \theta = -(B + \lambda A^T)^{-1} C$$

For any $\lambda$, in case of a unique solution, we obtain a point $\Delta \theta$. In the end we are interested in a $\Delta \theta$ solution that satisfies the approximate KL divergence constraint Eq. 71 or in other words,

$$\Delta \theta^T A_{11} \Delta \theta^1 + \Delta \theta^2 T A_{22} \Delta \theta^2 \leq 2\delta'$$

All $\lambda$ which satisfy this constraint are legitimate solutions, but we prefer the one which assigns higher value to the left hand side of this equation. Therefore, considering $\frac{1}{\lambda}$ as step size, we select a $\lambda$ that satisfy this constraint with a high value for the left hand side using line search. The solution is given by computing $\Delta \theta = -(B + \lambda A^T)^{-1} C$ inside a line search, and choosing the solution that satisfies the following constraint with the biggest margin,

$$C^T (B + \lambda A^T)^{-1} T A(B + \lambda A^T)^{-1} C \leq \delta.$$

This concludes our derivation.

## 14 IMPLEMENTATION OF ALGORITHMS

In this section we talk about final setup of CoPG and TRCoPO algorithms and their efficient implementation.

### 14.1 COMPETITIVE POLICY GRADIENT

**Algorithm 6:** Competitive Policy Gradient

**Input:** $\pi(\cdot|\cdot; \theta^1), \pi(\cdot|\cdot; \theta^2), V_\phi(s), N, B, \alpha$, Choice of $A(s, a^1, a^2; \theta^1, \theta^2)$ mentioned in Sec 3.1

**for epoch :** $e = 1$ to $N$ do

**for batch :** $b = 1$ to $B$ do

**for $k = 0$ to $|\tau| - 1$ do**

- Sample $a_k^1 \sim \pi(a_k^1|s_k; \theta^1), a_k^2 \sim \pi(a_k^2|s_k; \theta^2)$.
- Execute actions $a_k = (a_k^1, a_k^2)$
- Record $\{s_k, a_k^1, a_k^2, r_k^1, r_k^2\}$ in $\tau_b$

**end**

- Record $\tau_b$ in $M$

**end**

1. Estimate $A(s, a^1, a^2; \theta^1, \theta^2)$ with $\tau$ in $M$
2. Estimate $D_{\theta^1} \eta, \ D_{\theta^2} \eta, \ D_{\theta^1 \theta^2} \eta, \ D_{\theta^2 \theta^1} \eta$ using Thm. 1
3. Simultaneously update $(\theta^1, \theta^2)$ using Eq. 6

**end**

Based on our discussion in Sec. 11 we propose Alg. 6 which is an extension of Alg. 2 to utilise advantage function estimation. In this algorithm, simultaneous update of $(\theta^1, \theta^2)$ using Eq. 6 requires computing inverse matrix product twice, one for each agent. Instead we can compute infinite recursion strategy using Eq. 6 for one player only and apply the counter strategy for the other player as proposed in Schaefer and Anandkumar [2019]. From Eq. 6 for player 1 $\Delta \theta^1 = \alpha (Id + \alpha^2 D_{\theta^1 \theta^2} \eta D_{\theta^2 \theta^1} \eta)^{-1} (D_{\theta^1} \eta - \alpha D_{\theta^1 \theta^2} \eta D_{\theta^2} \eta)$. Using this, one can derive an optimal counter strategy $\Delta \theta^2$ for player 2 by solving the argmin in Eq. 5. This results in,

$$\Delta \theta^2 = -\alpha (D_{\theta^2 \theta^1} \eta \Delta \theta^1 + D_{\theta^2} \eta),$$
which is computed without evaluating an inverse. Similarly, if one evaluates $\Delta \theta^2$ from Eq. [6], the optimal counter strategy $\Delta \theta^1$ can be obtained by solving the $\text{argmax}$ in Eq. [5], which is given by,

$$\Delta \theta^1 = \alpha(D_{\theta^1, \theta^2} \eta \Delta \theta^2 + D_{\theta^1} \eta).$$

This reduces the requirement of computing inverse to only once per epoch. Further, CoPG has been extended to utilise RMSProp to iteratively adapt the step size as proposed in Schäfer et al. [2019]. In RMSProp, the learning rate is adapted based on the moving average of the square of gradients. The code for CoPG along with RMSProp is also available in the code repository.

### 14.2 TRUST REGION COMPETITIVE POLICY OPTIMIZATION

Based on our discussion in Sec. 13.3, we propose the trust region competitive policy optimization algorithm given in Alg. 7.

**Algorithm 7: Trust Region Competitive Policy Optimisation**

**Input:** $\pi(\cdot|\cdot; \theta^1), \pi(\cdot|\cdot; \theta^2), N, B, \delta$, Choice of $A(s, a^1, a^2; \theta^1, \theta^2)$ mentioned in Sec 3.1

**for epoch :** $e = 1$ to $N$

**for batch :** $b = 1$ to $B$

**for** $k = 0$ to $|\tau| - 1$

- Sample $a_k \sim \pi(a_k|s_k; \theta_1), a_k^2 \sim \pi(a_k^2|s_k; \theta_2)$.
- Execute actions $a_k = (a_k^1, a_k^2)$
- Record $\{s_k, a_k^1, a_k^2, r_k^1, r_k^2\}$ in $\tau_b$

**end**

Record $\tau_b$ in $M$

**end**

1. Estimate $A(s, a^1, a^2; \theta^1, \theta^2)$ with $\tau$ in $M$
2. Estimate $D_{\theta^1} L, D_{\theta^2} L, D_{\theta^1, \theta^2} L, D_{\theta^2, \theta^1} L$ using Eq. [63]
3. Solve for $\Delta \theta = [\Delta \theta^1 T, \Delta \theta^2 T]^T$ in Eq. [59] using line search, while $\Delta \theta^T A \Delta \theta < \delta$

   - $\lambda \leftarrow 2 \lambda$
   - $\Delta \theta \leftarrow -(B + \lambda A')^{-1} C$
   - $\theta \leftarrow \theta + \Delta \theta$

**end**

In the experiments, we used Eq. [70] to estimate $A'$. As given in the Alg. 3, we suggest to fulfill the constraint using a line search method. Start with small a value of $\lambda = \lambda_0$, and solve for $\Delta \theta = -(B + \lambda A')^{-1} C$. Check for $\Delta \theta^T A \Delta \theta < \delta$. If not satisfied change $\lambda$ to $2\lambda$ and repeat it until the constraint is satisfied.

There could be multiple heuristics to initialize the $\lambda$ value for a fast convergence. We propose to solve for $\lambda_0$ assuming 2 agents solve for their objective independently (TRGDA). The $\lambda$ value can be obtained in closed form using single-agent TRPO [Schulman et al., 2015]. Let us say we obtain $\lambda_{\text{min}}$ and $\lambda_{\text{max}}$ corresponding to the minimizing and maximizing player. We propose to use $\lambda_0 = \min\{\lambda_{\text{min}}, \lambda_{\text{max}}\}$ as an initialisation.

Similar to single-agent TRPO, often the resulting step size from the algorithm might be large and thus probably does not benefit either of the agents with policy updates. During such cases, we can further refine the line search iteratively for a smaller step size and update the policies only if both players gain an advantage.

### 15 EXPERIMENT DETAILS

In this section, we provide all the details and explanations of the experiment platforms used in this work. First, we start with the games with a known closed-form Nash equilibrium. Later we talk about the Markov Soccer game and the car racing game.
15.1 LINEAR QUADRATIC (LQ) GAME

The zero sum LQ game is defined by:

\[
J(K^1, K^2) = \max_{K^1} \min_{K^2} E \sum_{k=0}^\infty \gamma^k \left( s_k^T Q s_k + a_k^1^T R_{11} a_k^1 + a_k^2^T R_{22} a_k^2 \right),
\]

(74)

where: \( s_{k+1} = A s_k + B_1 a_k^1 + B_2 a_k^2 \),
\[
a_{k+1}^1 = -K^1 s_k, a_{k+1}^2 = -K^2 s_k,
\]

where, \( s_k \in \mathbb{R}^n \) is the system state vector, \( a^1 \in \mathbb{R}^{d_1} \) and \( a^2 \in \mathbb{R}^{d_2} \) are the control inputs of player 1 and 2 respectively. The matrix \( A \in \mathbb{R}^{n \times n} \), \( B_1 \in \mathbb{R}^{n \times d_1} \) and \( B_2 \in \mathbb{R}^{n \times d_2} \) describe the system dynamics. Here, \( K^1 \) and \( K^2 \) are the control policies. We considered a simple game where, \( A = 0.9, B_1 = 0.8, B_2 = 1.5, R_{11} = 1, R_{22} = 1 \). These environment parameters are not known to the players. The policy of each player is linear in the state and \( \sigma \) is randomly initialised. We collected batch of 1000 trajectories. The experiment was performed for learning rates between 0.001 and 0.05 for 8 random seeds. The main paper features results of CoPG vs GDA, but even for the trust region approach we observe a similar contrast between TRCoPO and TRGDA. Starting from some random policy initialization, TRCoPO converges to the unique Nash equilibrium of the game whereas TRGDA diverges for all \( \delta \). Fig. [11] shows such behavior for \( \delta \) of 0.001.

<table>
<thead>
<tr>
<th>learning rate</th>
<th>0.1</th>
<th>0.05</th>
<th>0.01</th>
<th>0.005</th>
<th>0.001</th>
<th>0.0001</th>
</tr>
</thead>
<tbody>
<tr>
<td>GDA</td>
<td>⊗</td>
<td>⊗</td>
<td>113</td>
<td>139</td>
<td>223</td>
<td>611</td>
</tr>
<tr>
<td>CoPG</td>
<td>75</td>
<td>87</td>
<td>105</td>
<td>127</td>
<td>195</td>
<td>590</td>
</tr>
</tbody>
</table>

Table 4: Number of iterations to converge to optimal policies

![Figure 10: Difference in game objective due to policy updates for LQ game at different learning rates](image)

15.2 BILINEAR GAME

In a bilinear game both players play actions simultaneously and receive rewards based on the product of their actions,
\[
r^1(a^1, a^2) = a^1 a^2, \quad r^2(a^1, a^2) = -a^1 a^2,
\]

(75)

where \( a^1 \sim \mathcal{N}(\mu_1, \sigma_1) \) and \( a^2 \sim \mathcal{N}(\mu_2, \sigma_2) \) are actions of player 1 and player 2 respectively. In our experimental setup, a player's policy is modelled as a Gaussian distribution with \( \mu_i, \sigma_i, i \in \{1, 2\} \) as mean and variance respectively. The policy is randomly initialised. We collected batch of 1000 trajectories. The experiment was performed for learning rates between 0.5 and 0.05 for 8 random seeds. The main paper features results of CoPG vs GDA, but even for the trust region approach we observe a similar contrast between TRCoPO and TRGDA. Starting from some random policy initialization, TRCoPO converges to the unique Nash equilibrium of the game whereas TRGDA diverges for all \( \delta \). Fig. [11] shows such behavior for \( \delta \) of 0.001.
15.3 MATCHING PENNIES

The matching pennies game is played between two players, each player holds a coin and hence has two possible actions {Head, Tail}. Both the players secretly turn their coin to either head or tail. If the pennies of both the players matches then player 1 wins and player 2 looses. Otherwise, if pennies do not match then player 2 wins and player 1 loses. The game matrix is given in Table 5 captures win and lose of players for every possible action pair.

<table>
<thead>
<tr>
<th></th>
<th>H</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>1,-1</td>
<td>-1,1</td>
</tr>
<tr>
<td>T</td>
<td>-1,1</td>
<td>1,-1</td>
</tr>
</tbody>
</table>

Table 5: Game matrix for matching pennies game

Players’ policy is modeled with a two-class categorical distribution, which is randomly initialized. They sample actions from a softmax probability function over the categorical distribution and receive rewards for their actions according to the game matrix. We collect a batch of 1000 trajectories, to estimate gradient and bilinear terms in every epoch. CoPG converges to the Nash equilibrium of the game $(H, T) = (\frac{1}{2}, \frac{1}{2})$. The experiment was performed for learning rates between 0.5 and 0.05 for 8 random seeds. The main paper features plot of CoPG vs GDA, but even for trust region approach we observe a similar contrast between TRCoPO and TRGDA. Starting from random policy initialization, TRCoPO converges to the unique Nash equilibrium of the game $(H, T) = (\frac{1}{2}, \frac{1}{2})$, whereas TRGDA diverges for all KL-divergence bounds $\delta$. Fig. 12 shows this behaviour for a $\delta$ of 0.01. This experiment was performed also performed for 8 different random seeds and the results where consistent.
15.4 ROCK PAPER SCISSORS

The rock paper scissors game is played between two players, each player has three possible actions \{Rock, Paper, Scissors\}. Both players will simultaneously display one of three actions: a rock, a paper, or a scissors. Rock beats scissors, scissors beat paper by cutting it, and paper beats rock by covering it. This is captured in the game matrix given by Table 6. Players’ policies are modeled with 3 class categorical distribution which is randomly initialized. Players sample action from a softmax probability function over categorical distribution. They receive a reward for their actions according to the game matrix. We collect a batch of 1000 trajectories. COPG converges to the unique Nash equilibrium of the game, where probability of playing \((R, P, S) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})\). The experiment was performed for learning rates between 0.5 and 0.05 for 8 random seeds where COPG converged in all cases and GDA diverged. Similarly for the trust region approach, the TRCoPO agent converges to a unique Nash equilibrium of the game \((R, P, S) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})\), where as TRGDA diverges. Fig. 13 shows this behavior for \(\delta\) of 0.001. The experiment was performed for 8 different random seeds.

<table>
<thead>
<tr>
<th></th>
<th>Rock</th>
<th>Paper</th>
<th>Scissors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rock</td>
<td>0.0</td>
<td>-1.1</td>
<td>1.0</td>
</tr>
<tr>
<td>Paper</td>
<td>1.1</td>
<td>0.0</td>
<td>-1.1</td>
</tr>
<tr>
<td>Scissors</td>
<td>-1.1</td>
<td>-1.1</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Table 6: Game matrix for rock paper scissors game

15.5 MARKOV SOCCER GAME

The soccer game setup is shown in Fig. 4. It is played between two players A and B, both are randomly initialized in one of the 4x5 grid cells. The ball is also randomly initialised in one of the 4x5 grid cells, this contrast to the previous studies, where one of the players was randomly initialised with the ball. Players are supposed to pick up the ball and place it in the opponent’s goal. The players are allowed to move up, down, left, right or stand still. The player without the ball seizes the ball, if both players move simultaneously to the same grid cell. The game ends if a player scores a goal or none of them scores within 1000 steps. The winner receives a +1 reward and -1 is awarded to the losing player, hence the game is formulated as a zero sum game. This game does not have an optimal deterministic policy. This game depicts heavy interactions, where strategy depends on what the other player plays. Potentially, a good player can learn defending, dodging and scoring.

The state vector of a player P with respect to opponent O is \(s^P = [x_{GO}, y_{GO}, x_{ball}, y_{ball}, x_O, y_O]\), where \(P, O \in \{A, B\}\). \(G_O\) is goal of the opponent, \(x, y\) refers to the relative position from the player to the subscript, e.g., \(x_O, y_O\) is the relative position to the opponent \(O\). The state vector of the game used during training is \(s = [s^A, s^B]\), \(s \in \mathbb{R}^{12}\), it captures the position of each player relative to the goal, the ball and relative to the opponent. The players’ policy maps state vector of the game to a categorical distribution with 5 categories using a network with two hidden layers one with 64 and other with 32 neurons. Players sample actions from a softmax probability function over the categorical distribution. The players were trained using GDA and CoPG. In both the algorithms, players were trained for roughly 30,000 episodes until the number of wins against any good player saturates. In each epoch we collected a batch which consists of 10 trajectories. All the parameters were the same throughout the training for CoPG and GDA. The experiment was tested with 6 different random seeds. We used learning rate of 0.01 and GAE for advantage estimation with \(\lambda\) of 0.95 and \(\gamma\) of 0.99.
Similarly, for the trust region approach the same game setting was used, with a maximum KL divergence of 0.0001. The players were trained with TRCoPO or TRGDA. Both the agents were trained for 5,000 episodes, each episode denotes a batch update where each batch consists of 10 trajectories. For comparison, we played 10,000 games between agents trained using TRCoPO and with that using TRGDA, in which for 50% of the games TRCoPO was player A and for remaining TRGDA was player A. TRGDA generates unequal players, on competing with the stronger TRGDA agent, the TRCoPO agent still won more than 80% games against TRGDA as shown in Fig. 14. TRCoPO was able to learn better tactics for snatching the ball and dodging and defending its goal post. The histogram in Fig. 14 depicts a comparison of GDA, CoPG, TRGDA, and TRCoPO in the soccer game playing against each other.

![Win Probability](image)

Figure 14: Win probability in soccer game played between GDA, CoPG, TRGDA and TRCoPO

15.6 THE CAR RACING GAME

Our final experimental setup is a racing game between two miniature race cars, where the goal is to finish the one lap race first. This involves both, learning a policy that can control a car at the limit of handling, as well as strategic interactions with the opposing car. Only if a player has the correct relative position to the opponent, this player is able to overtake or to block an overtaking. Our simulation study mimics the experimental platform located at ETH Zurich, which use miniature autonomous race cars. Following [Liniger et al., 2015] the dynamics of one car is modeled as a dynamic bicycle model with Pacejka tire models [Bakker et al., 1987]. However, compared to [Liniger et al., 2015] we formulate the dynamics in curvilinear coordinates [Vázquez et al., 2020] where the position and orientation are represented relative to a reference path. This change in coordinates significantly simplifies the definition of our reward, and simplified the policy learning. The resulting state of a single car is given as \( z = [\rho, d, \mu, V_x, V_y, \psi]^T \), where \( \rho \) is the progress along a reference path, \( d \) is the deviation from a reference path, \( \mu \) is the local heading with respect to the reference path, \( V_x \) and \( V_y \) are the longitudinal and the lateral velocity respectively in car frame and \( \psi \) is the yaw rate of the car. The inputs to the car are \( [D, \delta]^T \), where \( D \in [-1, 1] \) is duty cycle input to the electric motor varying from full braking at \(-1\) to full acceleration at \(1\) and \( \delta \in [-1, 1] \) is the steering angle. The test track which consists of 13 turns with different curvature can be seen in Figure 6. For the reference path we used the X-Y path obtained by a laptime optimization tool [Vázquez et al., 2020], note that it is not necessary to use a pre-optimize reference path, but we saw that it helped the convergence of the algorithm. Finally, to get a discrete time MDP we discretize the continuous time dynamics using an RK4 integrator with a sampling time of 0.03s.

To formulate the racing game between two cars, we first define the state of the game as the concatenated state of the two players, \( s = [z^1, z^2] \). Second we convert the objective of finishing the race first without an accident into a zero sum game. Therefore, we define the following reward function using reward shaping. First to model our no accident constraints we use a freezing mechanism: (i) If the car is leaving the track we freeze it until the episode ends, (ii) if the cars collide, the rear car (car with the lower progress \( \rho \)) is stopped for 0.1s, which corresponds to a penalty of about two car lengths. Note that this gives an advantage to the car ahead, but in car racing the following car has to avoid collisions. Furthermore, an episode ends if both cars are either frozen due to leaving the track or the first car finished the lap. Finally, to reward the players receive at every time step is \( r(s_k, a_{k+1}^1, a_{k+1}^2) = \Delta\rho_{car1} - \Delta\rho_{car2} \), where \( \Delta\rho = \rho_{k+1} - \rho_k \). This reward encourages the player to get as far ahead of the of the opponent player as possible. Note that the reward is zero sum, and the collision constraints do only indirectly influence the reward.

For the training we started the players on the start finish line (\( \rho = 0 \)) and randomly assigned \( d \in \{0.1, -0.1\} \). We also limited one episode to 700 time steps, which is about twice as long as it takes an expert player to finish a lap. For each policy gradient step we generated a batch of 8 game trajectories, and we run the training for roughly 20,000 epochs until the player consistently finish the lap. This takes roughly 62 hours of training for a single core CPU. To increase the performance and
robustness of learning we adapted the learning rate with RMS prop for both GDA and CoPG and used a slow learning rate of $5 \cdot 10^{-5}$ for both players. For our experiments we run 8 different random seeds and report the numbers of the best seed. As a policy we use a multi-layer perceptron with two hidden layers, each with 128 neurons, we used ReLU activation functions and a Tanh output layer to enforce the input constraints. GAE is used for advantage estimation with $\lambda$ of 0.95 and $\gamma$ of 0.99. The same setting was also used to compare performance of TRCoPo and TRGDA. The maximum KL divergence $\delta$ was set between $1e^{-4}$ to $1e^{-5}$ and the training was performed for 8 different random seeds. The statistics of the experiments are presented in the main part of the paper Sec. 4.

15.7 COMPARISON WITH MADDPG AND LOLA

Finally, we give more details on our comparison of CoPG with MADDPG and LOLA. We performed this comparison on the Markov soccer and the racing game. We start with the comparison for the Markov soccer game. To compare the performance of MADDPG on discrete action spaces (such as in the Markov soccer game), we used policies that can produce differentiable samples through a Gumbel-Softmax distribution, similar to [Iqbal and Sha, 2019]. The rest of the experimental setup was kept as similar as explained in [15.5] and we trained both MADDPG and CoPG. Fig. 15a shows that a policy learned by MADDPG performs similar to that of GDA. But when the MADDPG player competes against CoPG, the CoPG agent wins 80% of the games. Second we studied LOLA, we observed that LOLA learns better policies than GDA potentially due to the second level reasoning, but LOLA still loses 72% of the games when playing against CoPG. Fig. 15 show the number of games won by CoPG when competing against MADDPG and LOLA.

![Figure 15: Comparison with LOLA and MADDPG on the Markov soccer game](image)

Next we evaluated MADDPG and LOLA in the game of Car Racing, where we kept all the setting identical to [13.6] Fig. 5e show the progress of an MADDPG agent during learning, and we can see that it performs similar to GDA. The car learns to drive up to 30% of the track, but after that the agents show oscillatory behaviour in training. We also performed this experiment with LOLA, but due to the variance in estimating the bilinear term and more importantly the non adaptive learning rate of LOLA it does not learn to drive around the track.

16 OPPONENT PARAMETER ESTIMATION

In the following section we conducted experiments comparing CoPG, with its opponent parameter estimation counterpart CoPG-OP as explained in Sec. 13.2

**Matching pennies** The game setting is explained in Apx. 15.3. The opponents’ parameters are estimated by maximizing the likelihood using Eq. 9. The Fig. 16a shows the policy trajectory of player 1 and the estimation of the trajectory of player 2 by player 1 and vice versa in Fig. 16b for player 2. We observe that both the trajectories of player 1 and player 2 converge to the unique Nash equilibrium of the game which is $p(H,T) = \left( \frac{1}{2}, \frac{1}{2} \right)$.

**Rock paper scissors** The game setting is explained in Apx. 15.4. The opponents’ parameters are estimated by maximizing likelihood using Eq. 9. The Fig. 17a, 17b shows the policy trajectory of player 1 and the estimation of the trajectory of player 2 by player 1 and vice versa in Fig. 17c, Fig. 17d for player 2. We observe that both the trajectory of player 1 and player 2 converge to the unique Nash equilibrium of the game which is $p(R,P,S) = \left( \frac{1}{3}, \frac{1}{3}, \frac{1}{3} \right)$.

**Markov Soccer** We next tested the opponent modelling approach on the Markov soccer game, which is discussed in more detail in Sec. 15.5. We focus in this comparison on the Markov soccer game since it is strategic enough while being representative. The game has two player A and B. Player A maintains a record of policy of player B (B Est A)

![Figure 16: Comparison with LOLA and MADDPG](image)
Figure 16: CoPG-OP policy trajectory of player P1 (P2) and its opponents’ estimation P2E1 (P1E2) in matching pennies.

Figure 17: CoPG-OP policy trajectory of players and opponents’ estimation in rock paper scissors.

which is estimated online by observing state-action pairs of the player B. The state vector in perspective of player A is $s^3 = [s^A, s^{BestA}]$ and in player B $s^2 = [s^{AestB}, s^B]$, in accordance with the game setting explained in Apx. 15.5. In contrast to the CoPG experiment, in the case of CoPG-OP the batch size is increased to 40 to reliably estimate the opponent’s parameters. The player estimates it’s opponent’s parameters by maximizing likelihood Eq. 9 with observed state-action pairs of the opponent. The player is trained while competing with the estimated opponent using Alg. 6. Fig. 18a below show the interaction plot for CoPG-OP agents with it’s estimated opponent. We see that the CoPG-OP player learns to seize ball/interact($A_3, A_4$) with the estimated opponent. Fig. 18b shows the interaction plot of CoPG-OP with CoPG, where we observe that CoPG-OP can also learns a policy similar to CoPG, where it learns to defend, escape and score the goal ($A_3$).

When directly competing with CoPG, we observed that CoPG-OP can win 46.5% of the games against CoPG (Fig. 18c). CoPG has slight edge in the number of game wins, which is probably due to variance in the estimation of the opponent’s parameter.

Figure 18: Interaction plots evaluated by playing 5000 games. Matches played between a) CoPG-OP player A and player B estimated by A b) CoPG-OP and CoPG c) Probability of games won between CoPG vs CoPG-OP.
The GAIL case study is conducted on the Car Racing game Apx.\footnote{15.6} with a single car, where \( s = \{z\} \). The aim is to learn to drive around the track Fig.\footnote{15.6} by imitating experts. For expert trajectories, we collected a batch of 80 trajectories using four pure pursuit controllers each having a different set of tuning parameters. This ensures that the batch of expert trajectory is collected across various speeds and with a variation in the action set. The set of parameters are
\[
\begin{align*}
pp_1 & : \{k_p = 10, v_{tar} = 1.45, l_d = 26\}, \\
pp_2 & : \{k_p = 10, v_{tar} = 1.3, l_d = 15\}, \\
pp_3 & : \{k_p = 10, v_{tar} = 1.2, l_d = 20\} \text{ and } pp_4 : \{k_p = 8, v_{tar} = 1.35, l_d = 15\}.
\end{align*}
\]
Here, \( k_p \) is the proportional gain of the velocity control, \( l_d \) is the look ahead distance and \( v_{tar} \) is the target velocity.

In the GAIL training, one episode was limited to 700 time steps, which is about twice as long as it takes an expert player to finish a lap. For each competitive policy gradient step, we generated a batch of 8 game trajectories, and we run the training for roughly 4000 epochs until the player consistently finishes the lap. This takes roughly 2 hours of training for a single-core CPU. For our experiments, we run 8 different random seeds and report the numbers of the best seed. As a policy we use a multi-layer perceptron with two hidden layers, each with 128 neurons, we used ReLU activation functions and a Tanh output layer to enforce the input constraints. The agent receives a reward according to the GAIL framework Alg.2 in \cite{Ho2016}. GAE is used for advantage estimation with a \( \lambda \) of 0.95 and a \( \gamma \) of 0.99. The agent’s policy and discriminator are updated simultaneously using CoPO, and the critic to CoPO update ration is 1:1. We use a slow learning rate of 3e-4 for CoPO and 8e-3 for the critic update.