Université de Montréal

Differentiable Best Response Shaping

par

Milad Aghajohari

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Discipline

July 10, 2023

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

Differentiable Best Response Shaping

présenté par

Milad Aghajohari

a été évalué par un jury composé des personnes suivantes :

Pierre-Luc Bacon
(président-rapporteur)

Aaron Courville
(directeur de recherche)

Glen Berseth
(membre du jury)

Résumé

Cette thèse est structurée en quatre sections. La première constitue une introduction au problème de la formation d'agents coopératifs non exploitables dans les jeux à somme non nulle. La deuxième section, soit le premier chapitre, fournit le contexte nécessaire pour discuter de l'étendue et des outils mathématiques requis pour explorer ce problème. La troisième section, correspondant au deuxième chapitre, expose un cadre spécifique, nommé Best Response Shaping, que nous avons élaboré pour relever ce défi. La quatrième section contient les conclusions que nous tirons de ce travail et nous y discutons des travaux futurs potentiels.

Le chapitre introductif se divise en quatre sections. Dans la première, nous présentons le cadre d'apprentissage par renforcement (Reinforcement Learning) afin de formaliser le problème d'un agent interagissant avec l'environnement pour maximiser une récompense scalaire. Nous introduisons ensuite les Processus Décisionnels de Markov (Markov Decision Processes) en tant qu'outil mathématique pour formaliser le problème d'apprentissage par renforcement. Nous discutons de deux méthodes générales de solution pour résoudre le problème d'apprentissage par renforcement. Les premières sont des méthodes basées sur la valeur qui estiment la récompense cumulée optimale réalisable pour chaque paire action-état, et la politique serait alors apprise. Les secondes sont des méthodes basées sur les politiques où la politique est optimisée directement sans estimer les valeurs. Dans la deuxième section, nous introduisons le cadre d'apprentissage par renforcement multi-agents (Multi-Agent Reinforcement Learning) pour formaliser le problème de plusieurs agents tentant de maximiser une récompense cumulative scalaire dans un environnement partagé. Nous présentons les Jeux Stochastiques comme une extension théorique du processus de décision de Markov pour permettre la présence de plusieurs agents. Nous discutons des trois types de jeux possibles entre agents en fonction de la structure de leur système de récompense. Nous traitons des défis spécifiques à l'apprentissage par renforcement multi-agents. En particulier, nous examinons le défi de l'apprentissage par renforcement profond multi-agents dans des environnements partiellement compétitifs, où les méthodes traditionnelles peinent à promouvoir une coopération non exploitable. Dans la troisième section, nous introduisons le Dilemme du prisonnier itéré (Iterated Prisoner's Dilemma) comme un jeu matriciel simple utilisé comme

exemple de jouet pour étudier les dilemmes sociaux. Dans la quatrième section, nous présentons le Coin Game comme un jeu à haute dimension qui doit être résolu grâce à des politiques paramétrées par des réseaux de neurones.

Dans le deuxième chapitre, nous introduisons la méthode Forme de la Meilleure Réponse (Best Response Shaping). Des approches existantes, comme celles des agents LOLA et POLA, apprennent des politiques coopératives non exploitables en se différenciant grâce à des étapes d'optimisation prédictives de leur adversaire. Toutefois, ces techniques présentent une limitation majeure car elles sont susceptibles d'être exploitées par une optimisation supplémentaire. En réponse à cela, nous introduisons une nouvelle approche, Forme de la Meilleure Réponse, qui se différencie par le fait qu'un adversaire approxime la meilleure réponse, que nous appelons le "détective". Pour conditionner le détective sur la politique de l'agent dans les jeux complexes, nous proposons un mécanisme de conditionnement différenciable sensible à l'état, facilité par une méthode de questions-réponses (QA) qui extrait une représentation de l'agent basée sur son comportement dans des états d'environnement spécifiques. Pour valider empiriquement notre méthode, nous mettons en évidence sa performance améliorée face à un adversaire utilisant l'Arbre de Recherche Monte Carlo (Monte Carlo Tree Search), qui sert d'approximation de la meilleure réponse dans le Coin Game.

Ce travail étend l'applicabilité de l'apprentissage par renforcement multi-agents dans des environnements partiellement compétitifs et offre une nouvelle voie vers l'amélioration du bien-être social dans les jeux à somme non nulle.

Mots clés : apprentissage par renforcement multi-agents, jeux à somme non nulle, mise en forme de la meilleure réponse

Abstract

This thesis is organized in four sections.The first is an introduction to the problem of training non-exploitable cooperative agents in general-sum games. The second section, the first chapter, provides the necessary background for discussing the scope and necessary mathematical tools for exploring this problem. The third section, the second chapter, explains a particular framework, Best Response Shaping, that we developed for tackling this challenge. In the fourth section, is the conclusion that we drive from this work and we discuss the possible future works.

The background chapter consists of four section. In the first section, we introduce the *Reinforcement Learning* framework for formalizing the problem of an agent interacting with the environment maximizing a scalar reward. We then introduce *Markov Decision Processes* as a mathematical tool to formalize the Reinforcement Learning problem. We discuss two general solution methods for solving the Reinforcement Learning problem. The first are Value-based methods that estimate the optimal achievable accumulative reward in each action-state pair and the policy would be learned. The second are Policy-based methods where the policy is optimized directly without estimating the values. In the second section, we introduce *Multi-Agent Reinforcement Learning* framework for formalizing multiple agents trying to maximize a scalar accumulative reward in a shared environment. We introduce *Stochastic Games* as a theoretical extension of the Markov Decision Process to allow multiple agents. We discuss the three types of possible games between agents based on the setup of their reward structure. We discuss the challenges that are specific to Multi-Agent Reinforcement Learning. In particular, we investigate the challenge of multi-agent deep reinforcement learning in partially competitive environments, where traditional methods struggle to foster non-exploitable cooperation. In the third section, we introduce the *Iterated Prisoner's Dilemma* game as a simple matrix game used as a toy-example for studying social dilemmas. In the Fourth section, we introduce the *Coin Game* as a high-dimensional game that should be solved via policies parameterized by neural networks.

In the second chapter, we introduce the Best Response Shaping (BRS) method. The existing approaches like LOLA and POLA agents learn non-exploitable cooperative policies by differentiation through look-ahead optimization steps of their opponent. However, there

is a key limitation in these techniques as they are susceptible to exploitation by further optimization. In response, we introduce a novel approach, Best Response Shaping (BRS), which differentiates through an opponent approximating the best response, termed the "detective." To condition the detective on the agent's policy for complex games we propose a state-aware differentiable conditioning mechanism, facilitated by a question answering (QA) method that extracts a representation of the agent based on its behaviour on specific environment states. To empirically validate our method, we showcase its enhanced performance against a Monte Carlo Tree Search (MCTS) opponent, which serves as an approximation to the best response in the Coin Game.

This work expands the applicability of multi-agent RL in partially competitive environments and provides a new pathway towards achieving improved social welfare in general sum games.

Keywords: Multi-Agent Reinforcement Learning, General-Sum Games, Best Response Shaping

Table des matières

Liste des tableaux

Liste des figures

Remerciements

My sincere thanks to Mohammad Reza Aghajohari and Nahid Rastghalam, my cherished father and mother. Their love has been the wind beneath the wings of my curiosity. I still hold dear our moments at the bookstore, engrossed in the enlightening 'why and how' books. Their enduring support in my decisions and the heartfelt sentiments shared each time I set forth on a new path mean the world to me.

I extend my profound gratitude to Prof. Aaron Courville. His relentless enthusiasm for pursuing fundamental questions and unwavering support significantly propelled my studies forward. Our collective effort over nearly three years led to an in-depth understanding of non-exploitable cooperativeness. Without his passion for addressing challenging inquiries, we might have abandoned this project in its early stages.

Chin-Wei Huang deserves my sincerest appreciation for his unhesitating mentorship. His profound mathematical wisdom and pragmatic approach have been instrumental in enhancing my research perspective and deepening my understanding of the field.

I've been consistently inspired by Tim Coojiman's deep dive into autodifferentiation frameworks and his analytical approach to pivotal ML questions. The depth of his knowledge in reinforcement learning, highlighted in our engaging discussions, has significantly amplified my grasp of deep learning frameworks.

Working with Juan Duque has been a privilege. His infectious enthusiasm, combined with an exceptional flair for conceptualizing innovative research, deserves heartfelt appreciation.

My initial months in Montreal were made brighter and easier due to the steadfast support of Mohammad and Reyhaneh. Their kindness to someone they had just met is a testament to their incredible character, and for that, I am profoundly thankful.

I wish to express my heartfelt gratitude to Kamran Chitsaz, a cherished friend of over 14 years. Because of him, Montreal became as dear to me as our hometown, Isfahan. Our walks and talks at the La Fontaine Parc will forever hold a special place in my heart. I'd also like to express my appreciation to Parsa for being so accommodating when I frequently stayed over at their place.

A huge thanks to Amirhossein Kazemnejad. Our countless deep conversations during those late nights at Mila truly enriched my experience. His unwavering commitment to

excellence and detail played a pivotal role in my growth. And those foosball lessons during our breaks? Absolutely unforgettable! Cheers to all the wonderful times we shared!

A special note of gratitude goes to Saba Ahmadi. Her welcoming nature, coupled with our memorable tea gatherings, was a source of immense joy. Her infectious optimism continually reminded me to look on the brighter side. Here's to her continued success in both research and life's endeavors.

Beheshteh Halimi deserves special mention for our endless tea rendezvous and the exceptional chocolates she always seems to find. Her tranquility has a way of touching everyone she knows.

A warm thank you to Erfan, my roommate through half of my master's studies. Our nightly exchanges were a source of inspiration and reflection.

Deep appreciation goes to Faezeh and Mehran. Our friendship, which extended beyond our short tenure as roommates, was a beacon of light during my initial year of immigration amidst the COVID quarantine.

I'm profoundly grateful to Amin Memarian for his clear insights into UdeM's intricate processes. His readiness to impart knowledge speaks volumes about his generous nature.

I extend my heartfelt thanks to Joey Bose for his support in paper-writing and for entertaining my myriad questions. Parkash Panagaden deserves special mention for his profound mathematical wisdom and his comforting mastery of theory.

I am deeply grateful to Christos Tsirigotis for our invaluable cafe conversations during the challenging times of COVID. My appreciation also extends to Michael Noukhovitch for steering the Aaron group's meetings, and to Samuel Lavoie for the insightful research discussions we shared during my master's at Mila.

I am deeply grateful to Caroline, whose assistance to Aaron proved invaluable in overcoming many challenges. A special mention to Celine for promptly addressing my queries about UdeM.

I express my profound gratitude to Dr. Sharifi Zarchi, not only for his outstanding guidance as my supervisor during my initial research journey but also for being a remarkable human being. His influence significantly enriched my bachelor's journey.

My sincere appreciation goes to Prof. Tom Henzinger. Joining IST as an intern under his mentorship provided me with my first substantial academic experience. Furthermore, his steadfast support during the admission processes was invaluable.

I am profoundly thankful to the team at Mila Cluster. In particular, I wish to express my gratitude to Olexa for his extensive expertise in clusters, linux, PyTorch, and nearly everything!

Last but not least, to Amirhossein, Kamran, Saba, Rozhin, Aarash, Mehrnaz, Reza, Mehran, Mandana, Beheshteh, and the wonderful community of friends at Mila: thank you. Your warmth and camaraderie turned Mila into a haven for me.

Introduction

Recent advancements in reinforcement learning (RL) have enabled us to train agents that can solve complex tasks, such as playing Go [17] and StarCraft [20].

It might be tempting to assume that we can leverage these agents to interact with each other and humans to solve important tasks. For instance, one could imagine that if we can train a single self-driving car, we should be able to populate the streets with multiple such cars and that would lead to maximized traffic flow and efficiency. Similarly, agents performing agricultural tasks could collaborate to maximize crop yield. Unfortunately, this is not necessarily the case. When agents focus on maximizing their rewards, they often overlook how other agents will adapt in response. This can lead to social dilemmas, where agents become trapped in suboptimal outcomes.

One example of these failures is observed by [10] where two agents trained by naive RL converge to a suboptimal Nash Equilibrium (NE) in the Iterated Prisoner's Dilemma (IPD). While learning a non-exploitable cooperative policy would provide higher returns for both agents, naive RL training does not discover this strategy.

Without adequate methods to address this challenge, the potential for a world enhanced by RL agents to improve human productivity remains limited. It is crucial to note that discovering cooperation-inducing policies, which encourage other agents to work together, is not a simple task, particularly when no human data is available. While imitation learning can be employed in games like Diplomacy [9] to achieve an agent that cooperates with humans for its own benefit, developing such agents from scratch in more complex tasks is far from trivial.

[10] develops a method for learning reciprocation-based cooperation by assuming that the other agent is taking a single naive step of training. [23] builds on this by assuming the opponent is taking multiple proximal updates. [2] achieves reciprocation-based agents by training them in various reward-sharing scenarios and with varying levels of knowledge certainty regarding the reward-sharing schema. During testing, these agents do not rely on reward sharing and have complete certainty of the reward-sharing schema.

[13] models the optimization dynamics in a meta-game, where each action represents policy movements in the policy space, and the rewards are generated based on these policy movements within the game. In other words, it models the meta-game of two agents, adapting their policies while attempting to play the underlying game.

[10] and [23] are limited by their assumptions of a few gradient updates on the opponent's optimization side, causing an imprecise approximation of the agent's optimization process. They assume that the agent will only make local policy changes in response to the other agent's update, which may not always hold true. [13] learns a meta-policy of the optimization dynamics, requiring numerous inner trajectories and thus restricting the number of optimization steps that can be considered.

Our detective-based approach involves training an auxiliary agent called the "detective" to approximate the best response against a given agent's policy. The detective serves as a training harness and is discarded after training. The final agent, trained against the detective and backpropagating through the conditioning procedure of the detective, is the focus of our method.

Our detective is not limited in its optimization against the primary agent and can adapt its responses drastically against different agents. By providing different means of access to agent's policy, we can control the rationality of the detective.

We also demonstrate the limitations of assuming training against a best response opponent, as the agent might exploit the optimal response of the opponent. We address this by training the agent to achieve high returns against itself through self-play with shared policies, i.e. assuming optimizing the collective reward of all agents where all agents are exactly the same copy of the same policy. This self-play with reward sharing regularizes the policy search space in favor of reciprocation-based cooperative policies and discourages policies that exploit the rationality of the best response opponent.

Chapitre 1

Background

1.1. Reinforcement Learning

Reinforcement Learning tackles the challenge of an agent trying to maximize a scalar value, i.e. accumulative reward or the return, when interacting inside an environment. The agent observes the state of the environment and makes a decision to take an action based on that state. Consequently, the agent receives a reward based on the state and the action. The only control the agent has over the return it receives is via the choice of actions to take. The agent learns a policy determining its behaviour in various states of the game. This policy is learned based on its previous interactions with the environment. The agent needs to learn from experience, while trading off exploration of new states and new actions and exploiting its knowledge to get high rewards. RL provides a computational framework for precise definition of this challenge based on Markov Decision Processes (MDP). In this section we briefly explain the Markov Decision Process definition, and we shortly elaborate on the two primary categories of RL solutions, namely Value-based solutions and Policy-based solution.

1.1.1. Markov Decision Process

The RL problem in the single agent case can be described formally as a Markov Decision Process, i.e. MDP. A mdp is described by a tuple

$$\langle \mathbb{S}, \mathbb{A}, P, R, \gamma \rangle$$

.

- $\mathbb{S}$: the set of all possible states of the environment.
- $\mathbb{A}$: the set of all possible actions that the agent can take.
- $P : \mathbb{S} \times \mathbb{A} \to \Delta(\mathbb{S})$: transition probabilities given the actions of the agent $a \in \mathbb{A}$. This mapping describes the environment dynamics and its response to the agent's actions.

- $R : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \to \mathbb{R}$: the reward function that maps each state transition to a scalar value called reward. That is, it assigns a single scalar value to $(s, a) \to s'$. Note that a is the action taken by the agent.
- $\gamma \in [0,1]$: The discount factor.

1.1.2. Value-based Solutions

If the agent has access to a function that assesses the optimal value of taking an action at each specific time, it could take the highest value action at any state reaching an optimal policy. Value-based methods first assign a random value to action-state pairs of the environment. These assigned values are called action-value estimates. A policy is defined by a function that outputs an action based on these action-value estimates. For example, a policy that takes the action with the highest value at any state is called a greedy policy based on those action-value estimates. The action-values will be refined later on given the interactions of the agent with the environment. If the action-value estimates are optimal, the greedy policy would be optimal. Formally:

$$\pi^*(s) = \arg\max_a Q_\psi(s,a) \tag{1.1.1}$$

This way of learning is the core of Q-learning. Let $Q_\psi(s,a)$ be the function that returns the value of an state-action pair, i.e the action-value. A policy π is defined over these values. π takes the action with the highest estimated value $1-\epsilon$ fraction of times and takes a random action ϵ fraction of times. This is called a ϵ-greedy policy based on the Q_ψ which is written formally as:

$$\pi(a|s) = \begin{cases} \text{random action}, & \text{with probability } \epsilon \\ \arg\max_a Q_\psi(s,a), & \text{with probability } 1-\epsilon \end{cases} \tag{1.1.2}$$

After the policy interacts with the environment, the action-value estimates are updated by the following:

$$Q_\psi(s,a) \leftarrow Q_\psi(s,a) + \alpha \left[R(s,a) + \gamma \max_{a'} Q_\psi(s',a') - Q_\psi(s,a) \right] \tag{1.1.3}$$

Equation 1.1.3 updates the estimated action-value of the taking action a at state s on the basis of the new interaction with the environment at state s taking action a assuming the agent will follow forever after the optimal actions w.r.t the current estimated action-values.

1.1.3. Policy-based Solutions

A policy assigns a probability distribution over actions at any specific state. The agent samples from this policy to interact with the environment. Let $\pi_\theta(a|s)$ be the probability

of taking action a at state s. The policy can be optimized via the gradient of the expected return of the agent. This can be written as:

$$\theta_{k+1} = \theta_k + \alpha \cdot \nabla_\theta \mathop{\mathbb{E}}_{\tau \sim \Pr_\mu^\theta} [R(\tau)] \tag{1.1.4}$$

This gradient is computed via REINFORCE gradient estimator [19] which estimates this gradient based on the sampled trajectories of the policy's interaction with the environment. The following is the formulation of the policy's update via REINFORCE:

$$\theta_{k+1} = \theta_k + \alpha \cdot \mathop{\mathbb{E}}_{\tau \sim \Pr_\mu^\theta} \sum_{t=0}^{T} \nabla_\theta \log(\pi_\theta(a_t|s_t)) \cdot R(\tau) \tag{1.1.5}$$

1.2. Multi-Agent Reinforcement Learning

1.2.1. Formalizing a Stochastic Game

MDP framework does not readily model multi-player games as it assumes there is a single action and single reward at each time-step. However, it can be easily extended to formalize the multi-player case by modeling the actions and rewards as actions and rewards for N agents. Stochastic Games [21] is one of these extensions. Stochastic Games are represented by a tuple:

$$\left\langle N, \mathbb{S}, \left\{\mathbb{A}^i\right\}_{i\in\{1,\dots,N\}}, P, \left\{R^i\right\}_{i\in\{1,\dots,N\}}, \gamma \right\rangle$$

With the following meaning:

- N: the number of agents.
- $\mathbb{S}$: the set of all possible states of the environment.
- $\mathbb{A} := \mathbb{A}^1 \times \cdots \times \mathbb{A}^N$ where $\mathbb{A}^i$ denotes the set of actions of agent i as each agent may have different set of actions.
- $P : \mathbb{S} \times \mathbb{A} \to \Delta(\mathbb{S})$: transition probabilities given the joint actions of the agents $\boldsymbol{a} \in \mathbb{A}$. This mapping describes the environment dynamics and its response to the agent's actions.
- $R^i : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \to \mathbb{R}$: the reward function that maps each state transition to a scalar value called reward. That is, it assigns a single scalar value to $(s, \boldsymbol{a}) \to s$. Note that $\boldsymbol{a}$ is the joint action describing the joint action of all agents.
- $\gamma \in [0,1]$: The discount factor.

Note that in case of $N = 1$ the Stochastic Game will be reduced to Single-Agent RL as expected. The result of MARL training is a joint policy $\boldsymbol{\pi}$ where $\pi^i : \mathbb{S} \to \Delta(\mathbb{A}^i)$, that is it maps each state to a probability distribution over all possible actions. We also use π^{-i} to denote all policies except the agent i, i.e. the policies of all other agents.

1.2.2. MARL categories

MARL games can be categorized into three categories of games based on the design of rewards of the agents relative to each other.

1.2.2.1. **Fully Cooperative**. A game is considered entirely cooperative when all participants are awarded the same reward, similar to Single-Agent Reinforcement Learning (RL). In other words, $\forall i,j \in N, R^i(\tau) = R^j(\tau)$ for all possible trajecotries of the game. Notable examples of games exhibiting this characteristic include Hanabi [4], Overcooked [5], and Traffic Light Control [7]. In the game of Hanabi, each player possesses cards that must be arranged on the table in a specific sequence to achieve a high team score. Interestingly, while players can view the numbers on other players' cards, they are unable to see the numbers on their own cards. As such, players must communicate and coordinate through the game's limited actions to help other players discern their card numbers or prompt them to take appropriate actions. The players at the end receive the same return, i.e they are in a team and the score is assigned to the whole team's performance. In Overcooked [5], players participate in cooking and preparing meals from raw ingredients. The planning and distribution of tasks must be determined and coordinated by the players themselves. Every time a complete meal is prepared on time all of the team members receive the same exact positive reward without taking into consideration their contribution in preparing the meal.

1.2.2.2. **Fully Competitive or Constant-Sum Games**. A game is fully competitive if the sum of the returns of the agents is constant. That is $\sum_{i=1}^{N} R^i = C, C \in \mathbb{R}$ for all possible trajectories of the game. These games were the first games to be tackled by deep RL as many games that humans play for entertainment fit in this category. Especially, the two-player version can be simplified further because the return of the other agent can be modeled as $R^{\text{agent}} = C - R^{\text{opponent}}$. Chess [18] and Go [17] are two famous examples. In these games, the termination of play results in one of two scenarios: a single player achieves victory, or the game concludes in a draw with rewards being evenly distributed among participants. When the number of players is greater than or equal to three ($N \geq 3$), the complexity of the game increases, as players have the potential to form transient alliances to secure an advantage. For instance, as demonstrated in [14], no player can attain victory without initially being supported from other participants, despite the fact that in most cases [1] ultimately, the game results in a single player's win and other player's loss.

1.2.2.3. **Cooperative-Competitive (General-Sum Games)**. This category of games are games that are neither fully cooperative nor fully competitive. In other words, there is no constraint on the reward structure. Iterative Prisoner's Dilemma 1.3.1, Coin Game [**?**] 1.4 are examples of such games that are studied in the field. These category of games is

[1]There is also the possibility of a draw in some games like chess.

the closest to most real life scenarios as there are few interactions between humans that are purely cooperative or competitive. In most cases, the real life scenarios resemble a general-sum game. This is true in individual, local, and global social situations. We provide three examples of each:

Local Farmers. Let's assumes there are two local farmers utilizing the same water reservoir and ground for their plants. While both farmers want to to consume the highest amount of water in order to plant more vegetables, they also need to maintain a fair usage, therefore, the reservoir has the time to refill. In this case, it is a competitive cooperative scenario. The returns of the farmers are not aligned in a full-cooperative setting. That would need both farmers to profit the same amount from the products of each other. Also, it is not a fully competitive setting. As if the water source runs out, both farmers lose their whole production. In other words, the failure of one farmer does not automatically lead to the success of the other farmer.

Science Laboratories. Let's assume the science laboratories working on discovering drugs for cancer. Each one of these laboratories want to be the first in discovering the new cures. However, if none of them publishes their papers publicly, the progress of the whole field slows down significantly and they all stall in making progress. Therefore, while every laboratory wants to be the first in curing the disease therefore there is some incentive to hide progresses, they also need to share, as if no laboratory does, all labs will stall.

1.2.3. Challenges of MARL

1.2.3.1. **Unclear Objective**. [22] In fully cooperative games, the goal is to maximize the collective reward of all agents within the game environment. In this context, the objective is straightforward, similar to single-agent reinforcement learning (RL). However, in fully competitive games, and particularly in cooperative-competitive games, the objective becomes much more nuanced. One possible objective is to identify a NE, which entails finding a set of strategies for each agent such that no individual agent has an incentive to unilaterally deviate from its strategy. This concept can be formally defined: A joint policy $\boldsymbol{\pi}$ is considered to be a NE if:

$$\forall i \in N \quad \forall \pi_i' \quad V^i(\pi^i, \boldsymbol{\pi}_{-i}) \geq V^i(\pi_i', \boldsymbol{\pi}_{-i})$$

where V^i is the value of agent i, $\boldsymbol{\pi}_{-i}$ represents the policies of all agents except agent i, and π_i' represents any alternative policy for agent i. However, in games like IPD there are multiple NEs. In this situation the desired solution is not to converge to a NE but to converge to an NE with high returns. Furthermore, the evaluation of a MARL agent's performance is influenced by the distribution of other agents. Consequently, it is generally not feasible to assess the agents without assuming a certain distribution regarding the policy of other agents.

1.2.3.2. **Non-Stationarity**. [22] In the training of MARL agents, each agent's state value and action value is dependent on the policies of other agents, which presents challenges for both policy-based and value-based methods. For instance, Q-Learning, a representative value-based method, employs a neural network to approximate the action value function for each agent. However, past experiences cannot be fully trusted, and the experience replay buffer may not be as effective as anticipated in single-agent RL. This is because the reward functions are reliant on the actions of other agents, and as their policies change, their actions in identical states alter, consequently affecting the distribution of rewards for the (S_t, a_t, S_{t+1}, r_t) sequence. As a result, past experiences do not provide an accurate estimation of current performance.

Also, policy gradient methods are known to experience high variance issues when multiple agents are involved [12]. This is a significant drawback, as high variance is generally the primary problem with the REINFORCE training signal. To address this issue, [12] proposes an actor-critic setup that utilizes a centralized critic while employing decentralized actors. Consequently, the agents can operate in a decentralized fashion during inference time.

1.3. IPD Game

The Prisoner's Dilemma (PD) serves as a widely recognized game used to investigate social dilemmas [10]. In PD, two prisoners are placed in separate cells without any means of communication throughout the duration of the game. They are presented with the choice to either confess and betray the other prisoner (Defect) or remain silent (Cooperate).

For this game to be classified as a Prisoner's Dilemma, specific conditions must be met regarding the payoff matrix. The payoff matrix depicted in table 2.1, determines the respective outcomes for each player based on their chosen action. The parameters R, S, T, and P should respect three conditions:

(1) $T > R$: This condition ensures that if it is assumed that the opponent is cooperating, "Defect" becomes the dominant action.

(2) $P > S$: This condition ensures that if it is assumed that the opponent is defecting, "Defect" becomes the dominant action.

(3) $P < R$: This condition guarantees that when both agents act greedily and choose the "Defect" option, the outcome is lower compared to when they cooperate.

The variant of the Prisoner's Dilemma that is being used throughout this thesis incorporates the following values: $R = -1, T = 0, S = -3, P = -2$. These values correspond to the following table:

Given these conditions on the payoff matrix, defecting is the dominant strategy in the Prisoner's Dilemma for each player regardless of the other player's decision. Let's assume player 1 is taking action. If Player 2 has chosen to defect, player 1 receives a payoff of P if

Player 2 \ Player 1	Cooperate	Defect
Cooperate	R R	T S
Defect	S T	P P

Tableau 1.1. Payoff matrix for the prisoner's dilemma game

Player 2 \ Player 1	Cooperate	Defect
Cooperate	−1 −1	0 −3
Defect	−3 0	−2 −2

Tableau 1.2. Example Payoff matrix for the Prisoner's Dilemma

it defects and it receives a payoff of S if it cooperates. Because $P < R$ it is better to defect. If player 2 has chosen to cooperate, player 1 receives a payoff of T if it defects and receives a payoff of R if it cooperates. Because $T > R$ it is better to defect. Therefore, defect results in a higher return regardless of the action chosen by the other. Consequently, defecting is the best choice regardless of the opponent's action. In other words, defecting is the dominant strategy.

1.3.1. Iterative Prisoner's Dilemma

As Prisoner's Dilemma is a single-turn game, it does not represent a cooperative-competitive environment where agents engage in long-term interactions. To address this limitation, the iterated Prisoner's Dilemma (IPD) is introduced. In this version, the game is repeated infinitely. The agents are allowed to change their strategy based on past interactions. Unlike the the classical Prisoner's Dilemma where dominant strategy is mutual defection, it is not a strictly dominant strategy at IPD.

However, the mutual always defect strategy remains a Nash Equilibrium in IPD. In other words, if both agents defect in all time-steps, no agent can get a higher return by changing its policy, cooperating, in one of those time-steps. However, always defect is no longer the best strategy regardless of the opponent's strategy. Consider the TFT strategy which is to initially cooperate and subsequently mirror the opponent's action from the previous time-step [2]. Always Defect agent (AD) is not the best strategy against a tit-for-tat (TFT) opponent.

[2]To provide a more intuitive explanation of the TFT player, the TFT strategy can be described by specifying its action based on the previous time-step. The algorithm for TFT in the iterated Prisoner's Dilemma is presented in 1.

Algorithm 1 Tit-for-Tat Strategy for IPD game: This algorithm evaluates the actions of both agents in the IPD game and returns the action for Agent 2 based on the Tit-for-Tat strategy.

Input: Previous time actions of agent 1 and agent 2 denoted as agent_1 and agent_2
Output: Action for Agent 2 according to the Tit-for-Tat strategy
if (agent_1 = cooperate)&&(agent_2 = cooperate) **then**
  return cooperate {Cooperate if both agents cooperated}
else if (agent_1 = cooperate)&&(agent_2 = defect) **then**
  return defect {Defect if Agent 1 cooperated and Agent 2 defected}
else if (agent_1 = defect)&&(agent_2 = defect) **then**
  return defect {Defect if both agents defected}
else if (agent_1 = defect)&&(agent_2 = cooperate) **then**
  return cooperate {Cooperate if Agent 1 defected and Agent 2 cooperated}
end if

AD does not yield the highest possible return against a tit-for-tat (TFT) agent. When AD plays against a TFT opponent, first the AD defects and TFT cooperates and the rest of the game is mutual defection. However, an always cooperate agent against the TFT opponent results in mutual cooperation. Consequently, the always cooperate agent accumulates a higher total payoff against the TFT opponent compared to AD agent against the TFT opponent.

It is crucial to note that when the game length is fixed at N, the strictly dominant strategy becomes mutual always defect. The proof is by induction. It is advantageous to defect at the final round, $t = N$. Therefore, knowing the other will defect at time $t = N$, defecting at time $t = N-1$ is advantageous. By employing an inductive reasoning approach, we can prove that defecting is the strictly dominant strategy in all time-steps. However, in the case of playing the game indefinitely the dynamic changes. For this reason, in this thesis, we study policies that are able to look up a single step of the history. Therefore, they cannot distinguish the final time-step and the inductive cascade of defecting is prevented.

1.3.2. Social Dilemmas and Iterated Prisoner's Dilemma

In the context of general sum games, social dilemmas emerge when individual agents striving to optimize their personal rewards inadvertently undermine the collective outcome or social welfare. This phenomenon is most distinct when the collective result is inferior to the outcome that could have been achieved through full cooperation. Theoretical studies, such as the Prisoner's Dilemma, illustrate scenarios where each participant, though individually better off defecting, collectively achieves a lower reward compared to cooperating.

However, in the Iterated Prisoner's Dilemma (IPD), unconditional defection ceases to be the dominant strategy. For instance, against an opponent following a tit-for-tat (TFT) strategy, perpetual cooperation results in a higher return for the agent. It might be expected

that MARL, designed to maximize each agent's return, would discover the TFT strategy, as it enhances both collective and individual returns, and provides no incentive for policy change, embodying a Nash Equilibrium. Yet, empirical observations reveal that standard RL agents, trained to maximize their own return, typically converge to unconditional defection.

This exemplifies one of the key challenges of multi-agent RL in general sum games: during training, agents often neglect the fact that other agents are also in the process of learning. To address this issue, and if social welfare is the primary consideration, one could share the rewards among the agents during training. For instance, training both agents in an IPD setup to maximize the collective return would lead to a constant cooperation. However, this approach is inadequate if the goal is to foster reciprocation-based cooperation. A policy is sought that incites the opponent to cooperate in order to maximize their own return. While TFT is one such policy, manually designing a similar TFT policies in other domains is neither desirable nor feasible, underscoring the necessity to develop novel training algorithms that can discover these policies.

1.4. Coin Game

The Coin Game is a high-dimensional general-sum game for evaluating the performance of methods for learning retaliatory policies parameterized by neural networks. The first description of the general concept of the coin game is in [10]. However, a slightly different variation of the game has been used in the recent literature [23]. We follow the [23]'s variant of the coin game.

In the [23] variation, the Coin Game is a two-player general sum game that takes place on a grid. The game involves two players: red player and blue player. At each episode, a coin, either red or blue, spawns somewhere in the grid and players compete to pick it up. The color of the spawning coin changes after each episode. If a player picks any coin, they receive a positive reward of $+1$. If the coin corresponding to their color is picked by the other player, they are punished with a negative reward of -2. An example trajectory between two random agents is shown in Figure 1.1. Ideally, players should cooperate by taking only the coins of their associated color in fear of future retaliation from the other agent.

1.4.1. Always Defect Policy

Let us assume that both agents take the shortest path to reach the current coin on the grid, acting greedily. This policy is called *Always Defect* (AD) throughout the thesis. The Always Defect policy written in pseudo-code in Algorithm 2. The coin game is intentionally designed to ensure that if both agents act greedily by taking all the coins, regardless of their color, the expected return for both agents will be zero. Let us consider the game from the perspective of the red player. In half of the instances, the red player will successfully take

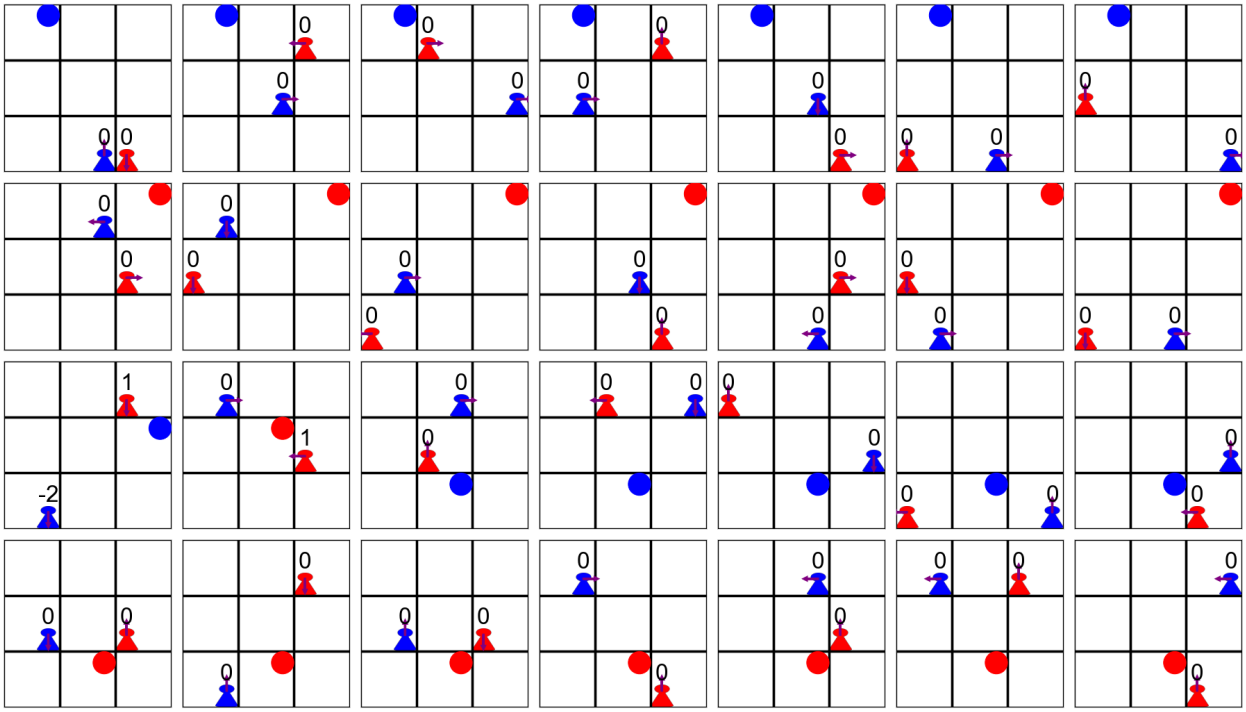

Fig. 1.1. Sampled trajectories between two random agents on the coin game. Each row is an independent game trajectory.

Algorithm 2 Always Defect Policy: This algorithm evaluates a state and returns the action to always move towards the coin, regardless of its color.

Input: State S
Output: Action to move towards the coin
return move_towards_coin {Always move towards the coin}

the coin, resulting in a reward of $+1$. In the remaining half, the blue player takes the coin, and out of those instances, half of the time, the taken coin is red. Thus, a quarter of the time, the red player receives a reward of -2. Overall, when considering the expected value, the red player's rewards can be calculated as follows:

$$\frac{1}{2} \times (+1) + \frac{1}{4} \times (-2) = 0 \tag{1.4.1}$$

Therefore, the expected return of the red player would be zero if both players act greedily. By symmetry, it is also true for the blue player.

The Always Defect policy is considered non-exploitable because no other agent can deceive or manipulate it into cooperation when the agent would actually benefit more by simply taking all the coins for itself. However, the Always Defect policy is not the best response to every possibly opponent. In other words, it is not the strictly dominant strategy. For example, if an opponent cooperates as long as the agent does not take its coin, in that case the best response is to not always defect but to cooperate.

Algorithm 3 Always Cooperate Policy: This algorithm evaluates a state and returns the appropriate action based on the color of the coin.

Input: State S with coin color denoted as coin_color
Output: Action based on the color of the coin
if coin_color = red **then**
  return move_towards_coin {Move towards red coin}
else
  if coin_color = blue **then**
    return avoid_coin_action {Choose action not to land on blue coin}
  end if
end if

However, if both agents adopt the naïve RL training, i.e. player 1 maximizing R^1 and player 2 maximizing R^2, empirically it is observed that they learn the Always Defect policy [10] [23]. Both agents assume their opponent is static with a fixed policy. This blindness to opponent's optimization inclines both agents to adopt an always defect strategy.

1.4.2. Always Cooperate Policy

One possible solution for agents solving the coin game is to implement an *Always Cooperate* (AC) policy. This entails that both agents never take a coin with the same color as the other player, letting the other player take the coin. This is written algorithmically in Algorithm 3. If both agents adopt this strategy, then both returns would be positive. Note that their exact return depends on the size of the board and how long it takes for an agent to reach the coin, in expectation. However, the returns would be positive and the social return, .i.e the sum of the rewards, would assume its highest possible value.

However, the Always Cooperate policy is generally considered undesirable due to its vulnerability to exploitation. For example, the Always Cooperate agent will be exploited by the Always Defect opponent. The Always Cooperate strategy has no retaliatory mechanism to incentivize the opponent to cease defection behaviour.

Training MARL agents to reach Always Cooperate policy is straightforward. If the rewards of both agents are shared with each other, i.e. both maximizing $R^1 + R^2$ instead of R^1 for player 1 and R^2 for player 2.

1.4.3. Tit-for-Tat Policy

Similar to IPD, a Tit-for-Tat (TFT) policy can be described for playing the coin game. The agent first starts by cooperation. It takes the shortest path towards the coins with its own color. Also, it avoids picking up the coin of the other player. However, if the opponent starts taking its coin, the agent changes behaviour to take the coin of the opponent. If the

Algorithm 4 Policy based on behavior of last coin takers: This algorithm evaluates a state and returns the appropriate action based on the behavior of the last red and blue coin takers.

Input: State S with the behavior of the last red and blue coin takers denoted as last_red_coin_taker and last_blue_coin_taker
Output: Action based on the behavior of the last coin takers
if last_red_coin_taker = red **then**
  return cooperative_action {Initial Cooperation}
else
  if last_blue_coin_taker = red **then**
    return cooperative_action {Forgiveness}
  else
    return defective_action {Retaliation}
  end if
end if

opponent refrained from taking the agent's coin, the agent switches back to cooperation. Algorithm 4 is one materialization of a Tit-for-Tat-like behaviour.

This policy is considered non-exploitable because of its retaliatory behaviour and the best response to this policy is cooperation. Therefore, Tit-for-Tat is considered a desirable policy for the Coin Game. However, it should be mentioned that Tit-for-Tat is not the best response to an Always Cooperate policy. Tit-for-Tat cooperates with the Always Cooperate Policy, but actually defecting against the Always Cooperate opponent yields higher return.

1.4.4. Notes on Studying the Coin Game

The details of the Coin Game are important in the experiments. It is desirable to study the Coin Game with infinite steps. However, this is not possible. Theoretically, in the finite length coin game the dominant strategy is Always Defect. The reasoning would be similar to a finite length Iterated Prisoner's Dilemma. It is always the dominant action to take every coin at the last time-step. By induction, it is proved that the dominant strategy is to play Always Defect. In order to prevent this from happening there are two methods. It is possible to employ strong measures to prevent this from happening by limiting the agent's access to time. For example, one can just provide the last C steps of the history to the agent where C is a fixed constant. However, as observed in practice by [23], this cascade does not tend to happen. It may be because of the inductive bias of the neural networks to learn a general policy instead of a time-dependent policy. Also, in all experiments a discount factor of 0.96 is used which makes the last states of lesser importance.

1.5. Related Work

LOLA [10] attempts to shape the opponent by taking the gradient of the value with respect to a one-step look ahead of the opponent's parameters. Instead of considering

the expected return under the current policy parameter pair, $V^1(\theta_i^1,\ \theta_i^2)$, LOLA optimizes $V^1(\theta_i^1,\ \theta_i^2 + \Delta\theta_i^2)$ where $\Delta\theta_i^2$ denotes a naive learning step of the opponent. To make a gradient calculation of the update $\Delta\theta_i^2$, LOLA considers the surrogate value given by the first order Taylor approximation of $V^1(\theta_i^1,\ \theta_i^2 + \Delta\theta_i^2)$. Since for most games the exact value cannot be calculated analytically, the authors introduce a policy gradient formulation that relies on environment roll-outs to approximate it. This method is able to find tit-for-tat strategies on the Iterated Prisoner's Dilemma.

POLA [23] introduces an idealized version of LOLA that is invariant to policy parameterization. To do so, each player attempts to increase the probability of actions that lead to higher returns while penalizing the Kullback-Leibler divergence in policy space relative to their policies at the previous time step. Similar to the proximal point method, each step of POLA constitutes an optimization problem that is solved approximately through gradient descent. Like LOLA, POLA uses trajectory roll-outs to estimate the value of each player and applies the reinforce estimator to compute gradients. POLA effectively achieves non exploitable cooperation on the IPD and the Coin Game improving on the shortcomings of its predecessor.

[13] consider a meta-game where at each meta-step a full game is played and the meta-reward is the return of that game. The agent is then a meta-policy that learns to influence the opponent's behaviour over these rollouts. M-FOS changes the game and is not comparable to our method which considers learning a single policy. [2] changes the structure of the game where each agent is sharing reward with other agents. The agents are aware of this grouping of rewards via a noisy version of the reward sharing matrix. In the test time, the representation matrix is set to no reward sharing and no noise is added to this matrix.

Diplomacy is a zero-sum game, but locally the agents need to form a retaliatory alliance in order to gain support for winning the game. Previous approaches like [15] and [9] utilize human data to learn this behavior. However, self-play without human data, as shown by [3], leads to significantly different policies where agents struggle to cooperate with human players effectively.

Policy Evaluation Networks (PVN) conditions a neural network on a policy by considering the policy's behavior on a set of learned states from the environment [11]. This idea is used in the detective opponent that conditions on the agent's policy. However, the PVN representation is not dependent on the current state.

Chapitre 2

Best Response Shaping (BRS): Achieving Cooperation of the Best Response Opponent

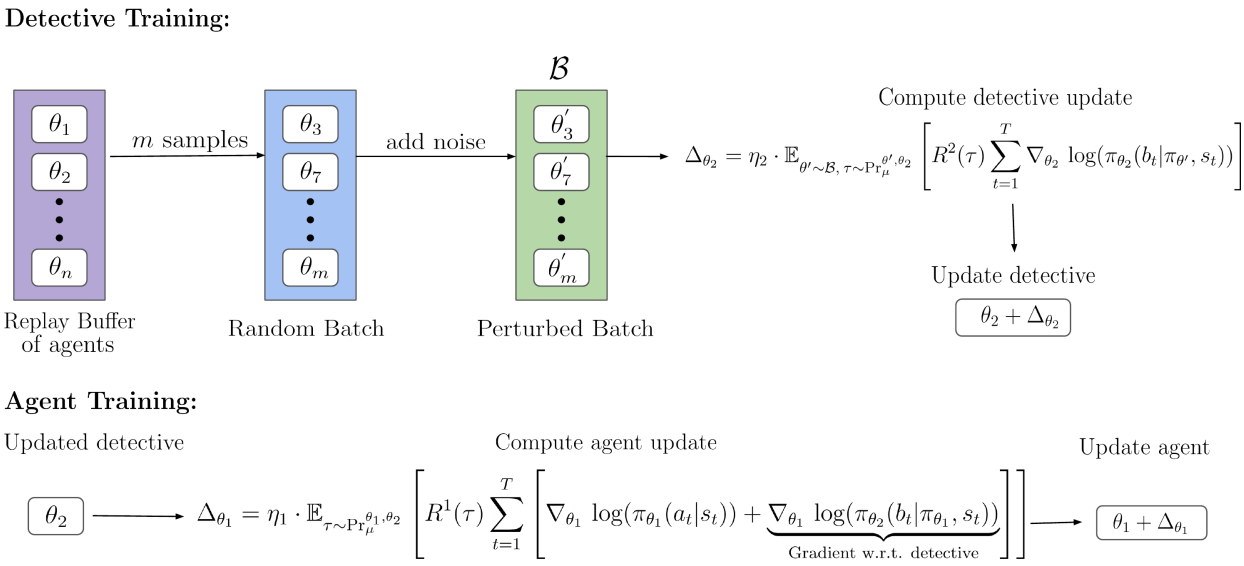

Fig. 2.1. The detective is trained using agents sampled from a replay buffer, which contains agents encountered during training. Additional noise is incorporated to broaden the range of agents encountered by the detective. The detective's training focuses on conditioning based on the agent's policy, aiming to estimate the optimal response to each policy to maximize its own return. Following this, the agent's training is conducted through backpropagation, through the detective's conditioning mechanism.

2.1. Best Response Shaping

Our Best Response Shaping (BRS) algorithm trains an agent by differentiating through an approximation to the best response opponent (as described in Section 2.1.1). This opponent, called the *detective*, conditions on the agent's policy via a question answering mechanism to select its actions (Section 2.1.2). Subsequently, we train the agent by differentiating through the detective using the REINFORCE gradient estimator [19] (Section 2.2.7). For situations where a cooperative policy is desired, we propose Self-Play with reward sharing as a regularization method, encouraging the agent to explore cooperative policies. We refer to this method as BRS with Self-Play (BRS-SP). The pseudo-code for BRS and BRS-SP is provided in Algorithm 5.

2.1.1. Best Response Agent to the Best Response Opponent

Our notation and definitions follow from [1], we denote τ as a trajectory whose distribution, $\Pr_\mu^{\theta_1, \theta_2}(\tau)$, with initial state distribution μ is given by

$$\Pr_\mu^{\theta_1, \theta_2}(\tau) = \mu(s_0)\pi_{\theta_1}(a_0|s_0)\pi_{\theta_2}(b_0|\pi_{\theta_1}, s_0)P(s_1|s_0, a_0, b_0)\dots$$

The best response opponent is the policy that gets the highest expected return against a given agent. Formally, given θ_1, the best response opponent policy θ_2^* solves for the following:

$$\theta_2^* = \arg\max_{\theta_2} \mathbb{E}_{\tau \sim \mathrm{Pr}_\mu^{\theta_1,\theta_2}} \left[R^2(\tau) \right] \tag{2.1.1}$$

Subsequently, we train the agent's policy to get the highest expected return against the best response agent. This training of the agent's policy is solving for the following:

$$\theta_1^{**} = \arg\max_{\theta_1} \mathbb{E}_{\tau \sim \mathrm{Pr}_\mu^{\theta_1,\theta_2^*}} \left[R^1(\tau) \right] \tag{2.1.2}$$

We hypothesize that the agent $\pi_{\theta_1}^{**}$ exhibits characteristics of a non-exploitable agent, as it learns retaliatory strategies in response to a defecting opponent, thereby creating incentives for a rational opponent to cooperate. Note that $(\theta_1^{**}, \theta_2^*)$ is a Nash Equilibrium by definition.

2.1.2. Detective Opponent Training

In deep reinforcement learning, the training of agents relies on the utilization of gradient-based optimization. Consequently, we need a differentiable opponent approximating a best response opponent. We call this opponent the *detective*. The detective's policy conditions on the agent's policy in addition to the state of the environment, which we denote $\pi_{\theta_2}(a|\pi_{\theta_1}, s)$. We train the detective to maximize its own return against various agents. Formally, the detective is trained by the following gradient step:

$$\nabla_{\theta_2} \mathbb{E}_{\theta_1 \sim \mathcal{B}} \mathbb{E}_{\tau \sim \mathrm{Pr}_\mu^{\theta_1,\theta_2}} \left[R^2(\tau) \right] \tag{2.1.3}$$

where $\mathcal{B}$ represents a distribution of diverse policies for agent 1.

2.1.3. Benefits of the Detective Compared to LOLA and POLA

Both LOLA and POLA [10], [23] train agents by differentiating through the opponent's optimization. However, they assume a few policy updates for the opponent. Consequently, the assumed optimization of the opponent is limited by its current policy.

BRS also differentiates through the opponent's optimization. However, BRS assumes the opponent is the Best Response opponent. In other words, it can be seen as assuming the opponent is optimized to the best response.

We hypothesize that LOLA and POLA would be exploitable against the best response opponent. While they are non-exploitable by local improvements of the opponent's policy, because the assumed opponent's optimization is restricted, the agents are not trained against full optimization to the best response opponent. In contrast to LOLA and POLA, the BRS is trained against the best response opponent. Therefore, it would not be exploitable against it.

Backpropagation through the full optimization is a challenging task. Unlike LOLA and POLA it is intractable to build the computational graph of the many optimization steps needed to reach the best response opponent and differentiate through it. In the following Section 2.2 we discuss various methods to alleviate this issue. In particular, we discuss how the amortization of this optimization conditioned on the agent via a neural network detective is used to solve the Coin Game.

2.2. Designing a Detective

The detective is an opponent that approximates the best response to the agent's policy. There are multiple design choices. It is possible to find the best response to a policy by tree-search methods. One possible solution is to use a Monte-Carlo-Tree-Search (MCTS) opponent. The MCTS finds the best response to the opponent's policy given enough samples. However, differentiating through the MCTS can be done via REINFORCE which is known to has a high variance. It becomes prohibitive when the number of samples to approximate the best response increases. We experiment with a variant of tree search detectives on the IPD game. For more complex games like the Coin Game, we need to use a neural network policy for the detective to approximate the best response. This neural network should be conditioned on the agent's policy.

Conditioning the detective's policy on agent's policy which is parameterized by a neural network is a challenging task [11]. If the agent's policy has few parameters, the conditioning can be done on the concatenation of the parameters into a single vector. This obscures the information about the structure of the neural network and possibly is a sample inefficient way of conditioning on a policy.

In cases where the number of parameters is huge, this approach does not scale. An alternative would be to condition the detective's policy on the behaviour of the agent's policy. In general, the detective can query the agent's policy in a set of states, and based on that information, query the agent's policy on a new set of states. We call this approach question-answering, where the questions are the states that the detective query the agent's policy on and the answers are the agent's behaviour on those set of states. However, extracting a single representation of a policy by querying its behaviour on a limited set of states needs lots of samples.

Next, to alleviate this issue, we propose a context based question-answering module which extract a representation of the agent's policy suited towards the current state of the game. This is done by extracting the representation by observing the agent's behaviour on multiple continuations of the game starting from the current state.

2.2.1. Monte Carlo Tree Search Detective

Monte-Carlo-Tree-Search [8], discovers the optimal action in a state by constructing a search tree consisting of future states where the branches are the actions. The value of each node is estimated via simulations of the game. Each simulation updates the values corresponding to each state. The next action of the MCTS is chosen in such a way to maintain a desirable exploration exploitation trade-off when traversing the search tree.

A Monte-Carlo-Tree-Search Detective (MCTS) can be constructed where the agent's actions are sampled from the agent's policy. In other words, the MCTS traverses the search tree given the agent's policy. Knowing the policy of the agent, the MCTS chooses the best response to the agent. Therefore, given enough samples, the MCTS Detective finds the best response to the agent's policy.

The MCTS side-steps the issue of conditioning a neural network policy on the agent's policy due to is non-parametric nature. It conditioning on the agent by using the current agent's policy in simulating the game. In other words, it functions as the agent's policy is a part of the environment. However, the MCTS may need a lot of samples in complex games. Also, differentiating through the MCTS is done via REINFORCE, facing the MCTS as a black-box algorithm. REINFORCE is known to have high variance and the fact that lots of samples are needed make this issue worse. Due to these limitations, the MCTS Detective is not used in our experiments[1]. The MCTS Detective is used for assessing the non-exploitability of the trained agents of different methods against the best response on the Coin Game in 2.5.2.

2.2.2. Tree Search Detective (TSD)

In this section, we describe the Tree Search Detective (TSD). We use TSD in our IPD experiments 2.5.1.The intuition behind TSD is that by simulating all possible trajectories based on the agent's policy, the opponent can select the path that maximizes its own returns. Consequently, the agent achieves the return associated with that specific path.

TSD implements this idea. TSD builds a tree structure in which the agent's actions are directly sampled from its policy. When it comes to TSD's action, a branch is formed for each action to explore the potential outcomes of that specific action.

The agent will treat TSD as a black-box algorithm that queries the agent's policy on a set of states and returns a single return, i.e. the return that corresponds to the agent's return in the path that yielded the highest return for the TSD. This black-box can be differentiated through via policy gradient estimators. It is worth noting that when calculating the policy

[1]Our initial experiments used the MCTS Detective. While it worked on the Coin Game on the 2x1 board with 2 actions, we could not make the agent training work against the MCTS on larger boards possibly due to high variance of the REINFORCE estimator of the gradient.

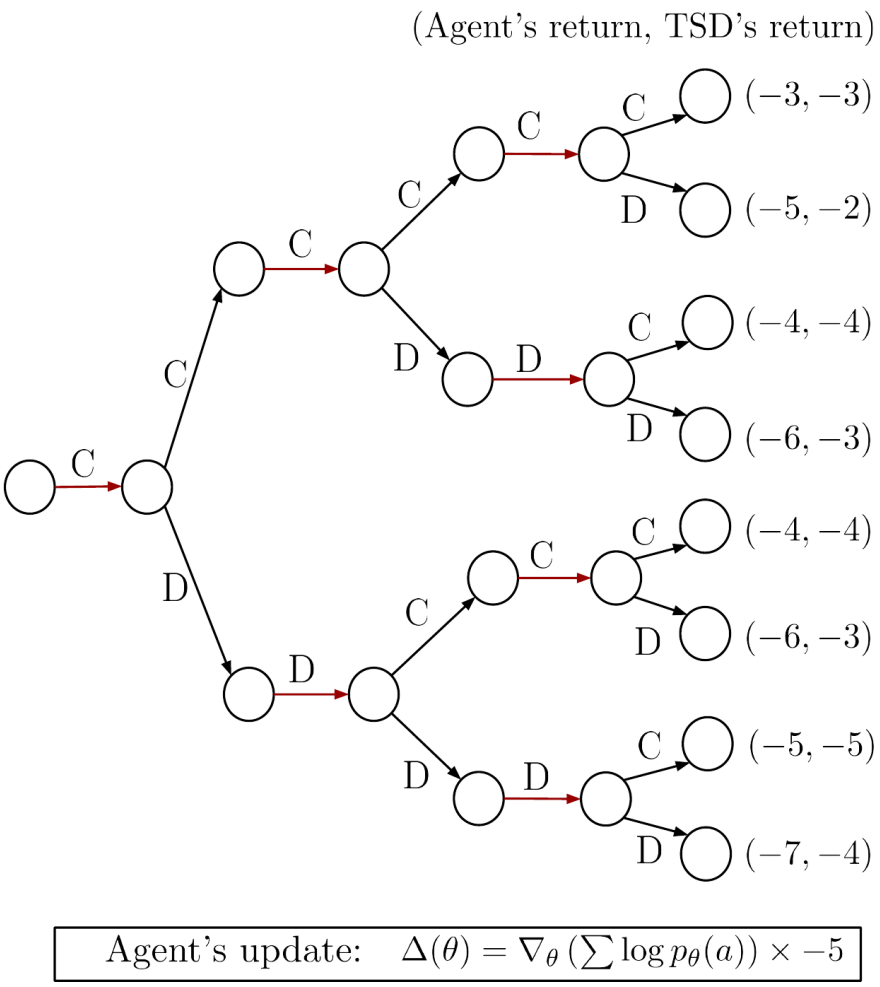

Fig. 2.2. This figure illustrates the training of the IPD agent against the TSD. TSD samples from the agent's policy, represented by red arrows in the plot, while exploring all possible actions when considering its own actions, represented by black arrows in the plot. The agent treats the TSD as a black-box algorithm and differentiates through it via REINFORCE. Note that the summation is over all log probabilities and not only over the log probabilities present in the path.

gradient loss, the sum of all log probabilities should be considered, not just the ones present in the chosen path. This is crucial because the agent's actions in states outside of the selected path are significant in TSD's decision-making process for selecting that particular path. This idea has been depicted in Figure 2.2.

2.2.3. Neural QA Detective

The detective can be a neural network policy that receives the parameters of the agent's policy as the input alongside the state of the game. In other words, its policy is $\pi_{\theta_{\text{Detective}}}(a_t|s_t, \theta_{\text{agent}})$. However, conditioning a neural network policy on another policy is a challenging task. Simple concatenating the parameters of the policy and feeding it to the

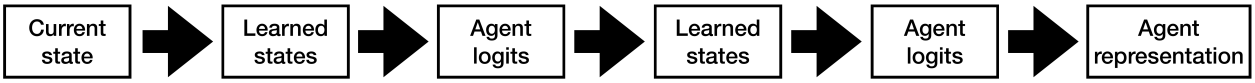

Fig. 2.3. Neural QA schematic

detective's neural net works for simple policies but does not scale to complex architectures. The Neural QA Detective side-steps this issue by Question-Answering rounds. The detective first queries the agent's policy in a set of states. Then, based on the behaviour of the agent on those states, it generates a new set of states and query the agent's policy on those synthetic states. We call the generated states the *Questions* and we call the agent's log probabilities of taking different actions the *Answers*. After all these question-answering rounds, the results can be aggregated to form a representation of the policy of the agent. The schematic of the approach is shown in Figure 2.3

2.2.4. Simulation-Based-QA Detectives

One way to do the question answering is based on simulating real games between the agent and an opponent and feeding information about those games to detective. This simulation removes the need for the detective to train a module to generate fake observations, as in 2.2.3, as the simulation just uses real states of the game. Additionally, it removes the assumption that imperfect game states generated by the detective can be fed into the agent's policy, as the detective merely queries states that happen in the real game.The variant of the simulation-based-qa module we use in our experiments on the coin game makes the agent play against a random opponent and then feed some statistics of those games to the detective.

2.2.5. gathered statistics

Multiple statistics of the game can be input to the detective. However, one should note that we also need to backpropagate through these statistics to the agent's policy for training the agent. One reasonable choice is the return of the sequence of rewards as both can be backpropagated via REINFORCE.

2.2.6. returns of the random agent

The behavior of the agent in possible continuations of the game starting from state s holds valuable information. More specifically, we can assess the behavior of the agent against a random agent starting from game state s. Formally Let δ_A be defined as the following where τ is a trajectory starting from state s at time t:

$$\delta_A := \mathop{\mathbb{E}}_{\tau \sim \mathrm{Pr}_{\mu}^{\theta_1, \theta_r}} \Big[R^2(\tau) | s_t = s \Big] \tag{2.2.1}$$

where π_{θ_r} is an opponent that chooses action A at time t and afterwards samples from a uniform distribution over all possible actions:

$$\pi_{\theta_r}(a_i = A|s_i) = \begin{cases} \frac{1}{|\mathcal{A}|} & \text{if } i > t \\ 1_{\{a_i=A\}} & \text{if } i = t \end{cases} \tag{2.2.2}$$

Detective estimates δ_A by monte-carlo rollouts of the game to a certain length between the agent and the random opponent, π_{θ_r}. We denote the estimate of δ_A by $\hat{\delta}_A$. Then we define $\mathcal{Q}^{\text{simulation}} = [\hat{\delta}_{A_1}, \hat{\delta}_{A_2}, \cdots, \hat{\delta}_{A_{|\mathcal{A}|}}]$. The number of samples used to estimate the returns of the game and the length of the simulated games are considered hyperparameters of $\mathcal{Q}^{\text{simulation}}$ QA. Note that the $\mathcal{Q}^{\text{simulation}}$ can be differentiated with respect to agent's policy parameters via REINFORCE [19] term. Specifically, we use the DICE operator [10].

2.2.7. Agent training

2.2.7.1. Differentiating Through the Detective. The agent's policy is trained to maximize its return against the detective opponent via REINFORCE gradient estimator. However, because the detective's policy is taking the agent's policy as input, the REINFORCE term will include an additional detective-backpropagation term over the usual REINFORCE term:

$$\mathop{\mathbb{E}}_{\tau\sim \text{Pr}_{\mu}^{\theta_1,\theta_2}} \left[R^1(\tau) \sum_{t=1}^{T} \left[\nabla_{\theta_1} \log(\pi_{\theta_1}(a_t|s_t)) + \underbrace{\nabla_{\theta_1} \log(\pi_{\theta_2}(b_t|\pi_{\theta_1}, s_t))}_{\text{detective-backpropagation term}} \right] \right] \tag{2.2.3}$$

This extra term can be thought of as the direction in policy space in which changing the agent's parameters encourages the detective to take actions that increase the agent's own return.

2.2.7.2. Cooperation Regularization via Self-Play with Reward Sharing. Agents that are trained against rational opponents tend to rely on the assumption that the opposing agent is lenient towards their non-cooperative actions. This reliance on rational behavior allows them to exploit the opponent to some extent. Consequently, they may not effectively learn to cooperate with their own selves. In scenarios where the objective is to foster more cooperative behavior, particularly encouraging the agent to cooperate with itself, a straightforward approach is to train the agent in a self-play setting, assuming that the opponent's policy mirrors the agent's policy. Formally, we update the agent using the following update rule:

$$\nabla_{\theta_1} \mathop{\mathbb{E}}_{\tau\sim \text{Pr}_{\mu}^{\theta_1,\theta_1}} \left[R^1(\tau) \right] \tag{2.2.4}$$

We prove that in symmetric games like IPD and Coin Game, this is equivalent to training an agent with self-play with reward sharing (see proof in 2.3). This training brings out the cooperative element of general-sum games. In zero-sum games, this update will have no

effect as the gradient would be zero (see proof in 2.3). We refer to this regularization loss term as Self-Play with reward sharing throughout the thesis.

2.3. Self-Play

Lemma D.1. *Denote $o \in \mathcal{S}$ to be the state $s \in \mathcal{S}$ from the perspective of the opponent. For a symmetric game, if it holds that $\mu(s_0) = \mu(o_0)$ for all $s_0, o_0 \in \mathcal{S}$, then*

$$\mathop{\mathbb{E}}_{\tau \sim \Pr_\mu^{\theta_1,\theta_1}} \left[R^1(\tau)\right] = \mathop{\mathbb{E}}_{\tau \sim \Pr_\mu^{\theta_1,\theta_1}} \left[R^2(\tau)\right]$$

where $R^2 := \sum_{t=0}^{\infty} \gamma^t r^2(o_t, b_t, a_t)$ and r^2 denotes r^1 from the perspective of the opponent.

Proof. Denote $\bar{\tau} = o_0, b_0, a_0, o_1, \ldots$, then notice that

$$\begin{aligned}
\mu(s_0)\pi^1_{\theta_1}(a_0|s_0)\pi^1_{\theta_1}(b_0|o_0)P(s_1|s_0,a_0,b_0)\ldots &= \mu(o_0)\pi^1_{\theta_1}(b_0|o_0)\pi^1_{\theta_1}(a_0|s_0)P(o_1|o_0,b_0,a_0)\ldots \\
\iff \Pr_\mu^{\theta_1,\theta_1}(\tau) &= \Pr_\mu^{\theta_1,\theta_1}(\bar{\tau})
\end{aligned}$$

now by symmetry we have that $r^1(s_t, a_t, b_t) = r^2(o_t, b_t, a_t)$, therefore

$$\begin{aligned}
\mathbb{E}_{\tau \sim \Pr_\mu^{\theta_1,\theta_1}} \left[R^1(\tau)\right] &= \mathbb{E}_{\tau \sim \Pr_\mu^{\theta_1,\theta_1}} \left[\sum_{t=0}^{\infty} \gamma^t r^1(s_t, a_t, b_t)\right] \\
&= \sum_{\tau} \Pr_\mu^{\theta_1,\theta_1}(\tau) \sum_{t=0}^{\infty} \gamma^t r^1(s_t, a_t, b_t) \\
&= \sum_{\bar{\tau}} \Pr_\mu^{\theta_1,\theta_1}(\bar{\tau}) \sum_{t=0}^{\infty} \gamma^t r^2(o_t, b_t, a_t) \\
&= \mathbb{E}_{\tau \sim \Pr_\mu^{\theta_1,\theta_1}} \left[R^2(\tau)\right]
\end{aligned}$$

where we just rename $\bar{\tau}$ in the last equality. ■

Proposition D.2 states that the gradient in Equation 2.2.4 is equivalent to that of self-play with reward-sharing.

Proposition D.2. *For a symmetric game,*

$$\nabla_{\theta_1} \mathop{\mathbb{E}}_{\tau \sim \Pr_\mu^{\theta_1,\theta_1}} \left[R^1(\tau)\right] \propto \left[\nabla_{\theta_1} \mathop{\mathbb{E}}_{\tau \sim \Pr_\mu^{\theta_1,\theta_2}} \left[R^1(\tau) + R^2(\tau)\right] + \nabla_{\theta_2} \mathop{\mathbb{E}}_{\tau \sim \Pr_\mu^{\theta_1,\theta_2}} \left[R^1(\tau) + R^2(\tau)\right]\right]_{\theta_2=\theta_1}.$$

Proof. We write the gradient as follows:

$$\begin{aligned}
\nabla_{\theta_1}\mathbb{E}_{\tau\sim\mathrm{Pr}_\mu^{\theta_1,\theta_1}}\left[R^1(\tau)\right] &= \sum_\tau R^1(\tau)\nabla_{\theta_1}\mathrm{Pr}_\mu^{\theta_1,\theta_1}(\tau) \\
&= \sum_\tau R^1(\tau)\mathrm{Pr}_\mu^{\theta_1,\theta_1}(\tau)\nabla_{\theta_1}\log\left[\mathrm{Pr}_\mu^{\theta_1,\theta_1}(\tau)\right] \\
&= \sum_\tau R^1(\tau)\mathrm{Pr}_\mu^{\theta_1,\theta_1}(\tau)\nabla_{\theta_1}\log\left[\mu(p_0)\pi^1_{\theta_1}(a_t|s_t)\pi^1_{\theta_1}(b_t|o_t)\ldots\right] \\
&= \sum_\tau R^1(\tau)\mathrm{Pr}_\mu^{\theta_1,\theta_1}(\tau)\sum_{t=0}^{\infty}\left[\nabla_{\theta_1}\log\,\pi^1_{\theta_1}(a_t|s_t)+\nabla_{\theta_1}\log\,\pi^1_{\theta_1}(b_t|o_t)\right] \\
&= \mathbb{E}_{\tau\sim\mathrm{Pr}_\mu^{\theta_1,\theta_1}}\left[R^1(\tau)\sum_{t=0}^{\infty}\left[\nabla_{\theta_1}\log\,\pi^1_{\theta_1}(a_t|s_t)+\nabla_{\theta_1}\log\,\pi^1_{\theta_1}(b_t|o_t)\right]\right].
\end{aligned}$$

Now by symmetry and Lemma D.1. we have

$$\mathop{\mathbb{E}}_{\tau\sim\mathrm{Pr}_\mu^{\theta_1,\theta_1}}\left[R^1(\tau)\right] = \mathop{\mathbb{E}}_{\tau\sim\mathrm{Pr}_\mu^{\theta_1,\theta_1}}\left[R^2(\tau)\right]$$

so we can write

$$\begin{aligned}
\nabla_{\theta_1}\mathbb{E}_{\tau\sim\mathrm{Pr}_\mu^{\theta_1,\theta_1}}\left[R^1(\tau)\right] &= \frac{1}{2}\mathbb{E}_{\tau\sim\mathrm{Pr}_\mu^{\theta_1,\theta_1}}\left[R^1(\tau)\sum_{t=0}^{\infty}\left[\nabla_{\theta_1}\log\,\pi^1_{\theta_1}(a_t|s_t)+\nabla_{\theta_1}\log\,\pi^1_{\theta_1}(b_t|o_t)\right]\right] \\
&\quad+\frac{1}{2}\mathbb{E}_{\tau\sim\mathrm{Pr}_\mu^{\theta_1,\theta_1}}\left[R^2(\tau)\sum_{t=0}^{\infty}\left[\nabla_{\theta_1}\log\,\pi^1_{\theta_1}(a_t|s_t)+\nabla_{\theta_1}\log\,\pi^1_{\theta_1}(b_t|o_t)\right]\right] \\
&= \frac{1}{2}\mathbb{E}_{\tau\sim\mathrm{Pr}_\mu^{\theta_1,\theta_1}}\left[\Big(R^1(\tau)+R^2(\tau)\Big)\sum_{t=0}^{\infty}\nabla_{\theta_1}\log\,\pi^1_{\theta_1}(a_t|s_t)\right] \\
&\quad+\frac{1}{2}\mathbb{E}_{\tau\sim\mathrm{Pr}_\mu^{\theta_1,\theta_1}}\left[\Big(R^1(\tau)+R^2(\tau)\Big)\sum_{t=0}^{\infty}\nabla_{\theta_1}\log\,\pi^1_{\theta_1}(b_t|o_t)\right].
\end{aligned}$$

This is equivalent to the training update of two agents optimizing shared rewards. ■

Corollary D.3. *For a symmetric, zero-sum game it holds that*

$$\nabla_{\theta_1}\mathop{\mathbb{E}}_{\tau\sim\mathrm{Pr}_\mu^{\theta_1,\theta_1}}\left[R^1(\tau)\right] = 0$$

Proof. By definition of zero-sum game, we have that

$$\begin{aligned}
& r^1(s_t,a_t,b_t)+r^2(s_t,b_t,a_t)=0 \\
\implies & \sum_{t=0}^{\infty}\gamma^t\left(r^1(s_t,a_t,b_t)+r^2(s_t,b_t,a_t)\right)=0 \\
\iff & R^1(\tau)=-R^2(\tau)\text{ for all }\tau
\end{aligned}$$

Algorithm 5 BRS/BRS-SP pseudo code: a single iteration

Input: Replay Buffer of Agent Parameters $\mathcal{B}$, Agent parameters θ^1, Detective parameters θ_2, learning rates $\alpha_1, \alpha_2, \alpha_3$, Do Self-Play with Reward Sharing flag SP, Standard Error of Noise σ
Detective Training Step:
Sample agent parameter $\theta_1\prime$ from $\mathcal{B}$
$\theta_1 \leftarrow \theta_1 + z$, where $z \sim \mathcal{N}(0, \sigma)$
Rollout trajectory τ_2 using policies $(\pi_{\theta_1\prime}, \pi_{\theta_2})$
$\theta_2 \leftarrow \theta_2 + \alpha_2 R^2(\tau_2) \sum_{t=1}^{T} \nabla_{\theta_2} \log(\pi_{\theta_2}(a_t|\pi_{\theta_1\prime}, s_t))$
Agent Training Step:
Rollout trajectory τ_1 using policies $(\pi_{\theta_1}, \pi_{\theta_2})$
$\theta_1 \leftarrow \theta_1 + \alpha_1 R^1(\tau) \sum_{t=1}^{T} \nabla_{\theta_1} \log(\pi_{\theta_1}(a_t|s_t)) + \nabla_{\theta_1} \log(\pi_{\theta_2}(b_t|\pi_{\theta_1}, s_t))$
Self Play Reward Sharing Step:
if SP = True **then**
  Rollout trajectory τ_3 using policies $(\pi_{\theta_1}, \pi_{\theta_1})$
  $\theta_1 \leftarrow \theta_1 + \alpha_3 R^1(\tau_3) \sum_{t=1}^{T} \nabla_{\theta_1} [\log(\pi_{\theta_1}(a_t|s_t)) + \log(\pi_{\theta_1}(b_t|s_t))]$
end if
Push θ_1 to $\mathcal{B}$
Output: θ_1, θ_2

From proposition D.2. we get

$$\begin{aligned}
\nabla_{\theta_1} \mathbb{E}_{\tau \sim \mathrm{Pr}_{\mu}^{\theta_1,\theta_1}} \left[R^1(\tau)\right] &= \frac{1}{2} \mathbb{E}_{\tau \sim \mathrm{Pr}_{\mu}^{\theta_1,\theta_1}} \left[\underbrace{\left(R^1(\tau) + R^2(\tau)\right)}_{=0} \sum_{t=0}^{\infty} \nabla_{\theta_1} \log \pi_{\theta_1}^1(a_t|s_t) \right] \\
&\quad + \frac{1}{2} \mathbb{E}_{\tau \sim \mathrm{Pr}_{\mu}^{\theta_1,\theta_1}} \left[\underbrace{\left(R^1(\tau) + R^2(\tau)\right)}_{=0} \sum_{t=0}^{\infty} \nabla_{\theta_1} \log \pi_{\theta_1}^1(b_t|o_t) \right] \\
&= 0
\end{aligned}$$

completing the proof. ■

2.3.1. Intuition behind BRS

In this section, we provide an intuitive explanation about why we hypothesize that training an agent by backpropagating through the detective leads to Tit-for-Tat-like behaviour.

Consider the detective we use for the Coin Game approximating the best response to the agent. Let us also assume the detective computes these returns for a state where the detective is one step away from the agent's coin. Determining the best action in this scenario requires understanding the agent's policy retaliatory behaviour. This is done via the question answering mechanism. Specifically, this detective conditions on the returns of a random opponent against the agent. If the agent is retaliatory, it is expected the return of the simulated trajectories, starting from the opponent taking the agent's coin, will be lower

compared to the scenario when the opponent does not take the agent's coin. In such a scenario, it indicates after the agent's coin is taken it retaliates by picking up the opponent's coin as a way to punish the opponent for its defection. However, if the detective observes that the agent continues cooperative behaviour even after its coin is taken, the detective would conclude that the agent is non-retaliatory and therefore exploitable.

Now with the introduction of the detective that conditions on the agent, let us analyze the consequences of the agent's differentiation through this detective. The agent adapts and learns to reduce the simulated opponent's returns when the agent's coin is being taken as the agent knows by lowering this return it signals the detective that its policy is retaliatory. As a result, the detective, being aware of the agent's retaliatory behavior, will start avoiding taking the agent's coin. This dynamic allows the agent to discover and adopt a retaliatory policy.

2.4. Evaluation of the Agents

The final product of our training is the agent and we need to evaluate if the agent adheres to our expectations. The final test of such agents would be to test them in the environment that they are going to be used. For example, the final test for an agent that is designed to be used in solving the diplomacy game [15] is to be tested in games against humans.

This is especially true for competitive cooperative games. In context of the coin game, if the environment is full of always cooperate agents, a TFT-like agent is not the best policy as always defect would yield a higher return. The same is true for an always defect environment as even the first cooperative actions of the TFT-like agent decrease its returns to lower values than an always defect agent. While the performance of any agent designed for a multi-agent environment depends on the distribution of the policies of the opponents it is going to face, there are a few criteria that are of significant important.

We evaluate the agents trained on the coin game against five category of opponents.

Always Cooperate. This opponent always takes the shortest path to its own coins, but avoids picking up the coins of the agent. The performance against this opponent indicates the cooprative-ness of the agent.

Always Defect. This opponent always takes the shortest path towards any coin that is present currently in the board. It does not behave differently between its own coin and the agent's coin. The performance of the agent against this opponent indicates the retaliatory behaviour of the agent against a defecting opponent.

Trained. This opponent is trained to maximize its own return against the agent. The performance of the agent against this opponent indicates the degree to which the agent's policy incentivizes the opponent to cooperate. In other words, if the Trained opponent

cooperates with the agent, it means the agent's policy is such a policy that the opponent could get a higher return via cooperation, compared to defecting.

MCTS. This opponent is a Monte-Carlo-Tree-Search opponent that rolls out the game given the agent's policy multiple times before taking an action. This opponent, given enough samples, approximates the best response. The performance of the agent against this opponent indicates whether the best response to the agent's policy is coooeration. In other words, it indicates whether the best response to the agent's policy is cooperation. If the MCTS cooperates with the agent, it means the agent's policy is retaliatory in such a way that it is better to cooperate with it. This metric is very important, as if the best response is not cooperation, it would indicate that the agent's policy is exploitable against the best response.

Self. This opponent is exactly the same policy as the agent. It can be seen as the agent playing against its own policy. Note that in the implementation of this agent extra care should be taken so the observation that is fed into the agent, is the same as if the opponent is observing the game from the perspective of the agent. This can just be done for symmetric games where the number associated with the players has no importance in the game.

2.5. Experiments

2.5.1. Iterated Prisoner's Dilemma

Following LOLA [10], we study Iterated Prisoner's Dilemma (IPD) game where the agents observe the last actions taken by the agents. Therefore, all possible agent observations would be $\mathcal{S} = \{\mathrm{C}, \mathrm{CC}, \mathrm{CD}, \mathrm{DC}, \mathrm{DD}\}$, where C is the initial state, and each agent's policy can be described by indicating the probability of cooperation for each $s \in \mathcal{S}$. We consider the IPD game that is six steps long. As shown in [10, 23], training two naïve-learning agents leads to strategies that always defect. Although this is a Nash Equilibrium, both agents will receive negative returns.

We test our method by training the agent against a tree search detective. The tree search detective constructs a tree, commencing from the current state. During this process, the agent's actions are sampled from the agent's policy, while the tree branches explore all possible choices for the detective's actions. The detective selects the actions that maximize its return, i.e. the actions that construct the best response path within the tree. The agent receives the return that corresponds to this particular path (see §2.2.3 for details). Our agent is a two-layer MLP that receives the five possible states and outputs the probability of cooperation. We choose an MLP to showcase the possibility of training neural networks via BRS. We update our agent policy via policy gradient.

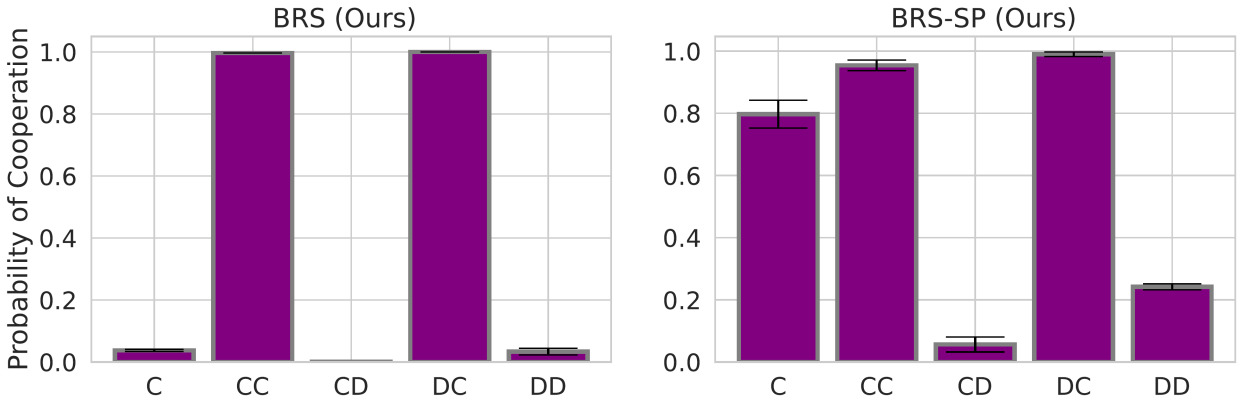

Fig. 2.4. Illustration of the policies of agents trained with BRS and BRS-SP in a finite Iterated Prisoner's Dilemma game of length 6. The agents are trained against a tree search detective maximizing its own return. BRS-SP agents learn tit-for-tat, a policy that cooperates initially and mirrors the opponent's behavior thereafter. BRS agents learn cynic-tit-for-tat (CTFT), they defect initially but mirror the opponent's behavior thereafter. Therefore, at the initial state, BRS agents exploit the rationality of the opponent.

In Figure 2.4, we show the result of training the BRS agent against a tree search detective opponent. The agents learn a variant of tit-for-tat that defects initially but has the same

probability of cooperation as tit-for-tat in $\{CC, CD, DC, DD\}$. We name this policy cynic-tit-for-tat (CTFT). This is because the agent assumes the opponent is a rational opponent that chooses the best response w.r.t to the agent's policy. The best response to a cynic-tit-for-tat in an infinite IPD game is always cooperating because if the opponent defects initially, the agent will defect in the next turn. In other words, the BRS agent learns to exploit the fact that the opponent (detective) is rational.

As shown in Figure 2.4 we also train the BRS with Self-Play (BRS-SP) agent against the same opponent. The trained agents learn tit-for-tat policy. Unlike CTFT, TFT cooperates in the first step, thus it cooperates with the best response opponent and with itself. The downside is that it is exploited by the always-defect opponent at the first step.

2.5.2. Coin Game

We follow [23] in training a GRU [6] agent on a 3×3 sized Coin Game with a game length of 50 and a discount factor of 0.96. The detective opponent is also a GRU agent with an MLP that conditions on the result of the QA (for more details see §2.8).

We evaluate our agent against four policies: an opponent that always takes the shortest path towards the coin regardless of the coin's color (Always Defect), an opponent that takes the shortest path towards its associated coin but never picks up the agent's associated coin (Always Cooperate), a Monte Carlo Tree Search opponent that evaluates multiple rollouts of the game against the agent in order to take an action (MCTS), an opponent that is trained to maximize its own return against the current agent (Trained), and itself (Self). Note that the MCTS will approximate the best response opponent given enough samples. Figure 2.5 visually presents the evaluation metrics for the BRS, BRS-SP, and POLA agents. In the subsequent paragraphs, we present a comprehensive analysis and interpretation of these results.

Is cooperation the best response? MCTS opponent and the Trained opponent approximate the best response opponent. The cooperation exhibited by these opponents towards the agents indicates the extent to which the agents have fostered retaliatory behavior. Comparing returns, BRS and BRS-SP outperform POLA against MCTS, and BRS-SP surpasses both BRS and POLA against the Trained opponent (Figure 2.5). [2]

Does Always Cooperate regret its cooperation with the agent? Ideally, an agent should be retaliatory in such a way that the best response would be always cooperation. In other words, the Always-Cooperate should not regret cooperating with the agent. As shown in Figure 2.5, the Always Cooperate's return is close the MCTS opponent's return against

[2]Note that our MCTS opponent, with 1000 rollouts at each state, provides a significantly stronger approximation of the best response compared to the Trained agent which is a GRU agent trained using policy gradient for 3000 steps.

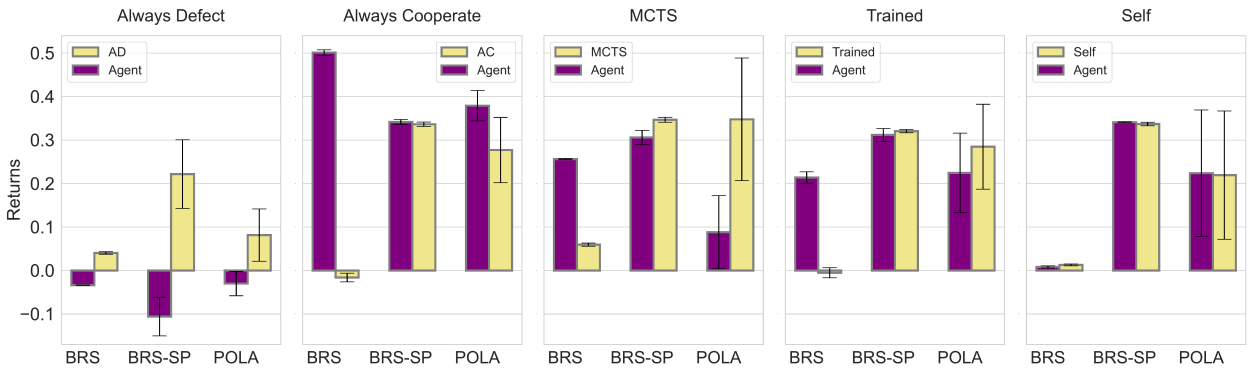

Fig. 2.5. Comparison of Agent's trained with BRS, BRS-SP, and POLA on a 3×3 sized Coin Game. We evaluate the agent's returns versus different opponents: Always Defect opponent (AD) that takes the shortest path to all coins; Always Cooperate opponent (AC) that takes the shortest path to its own coin but does not take the agent's coin; A Monte Carlo Tree Search opponent (MCTS) that do one thousand rollouts using the agent's policy at each state; a trained opponent (Trained) that is trained vs the agent to maximize its own return; and agent's performance against itself (Self). The POLA agents are exploited by the MCTS, which approximates the best response. It means, a rational opponent maximizing its return, harms POLA's return unintentionally. In contrast, BRS and BRS-SP get a higher return against the MCTS. Although BRS agents are not inherently cooperative among themselves, they effectively exploit the AC opponent in a rational manner. On the other hand, BRS-SP agents exhibit greater levels of cooperation among themselves but refrain from exploiting the AC opponent.

the BRS-SP. However, the MCTS return against the BRS and POLA is significantly higher than the Always-Cooperate's return.

Does always defect exploit the agent? As shown in Figure 2.5 BRS agents and POLA agents get near zero return against Always Defect. However, the return of the Always Defect against POLA is higher. BRS-SP agent gets a lower return compared to BRS and POLA indicating it is exploited more by the AD policy (Note that BRS-SP still retaliates but to a lesser degree).

Does the agent cooperate with itself? As shown in Figure 2.5 BRS agents get near zero return against themselves. Indicating that they are not cooperating with themselves. BRS-SP agents always cooperate with themselves. The POLA agents cooperate with themselves less than BRS-SP and more than BRS and also with higher variance.

BRS, BRS-SP, and POLA As shown in Figure 2.9, BRS and POLA agents exploit BRS-SP agents and BRS exploits POLA agents on average.

2.6. Evaluation Metrics of Various Agents

In Figure 2.6 we plotted the evaluation metrics for all the agents against the evaluation opponents. Note that for BRS, we used two seeds. For BRS-SP we used three seeds, and for

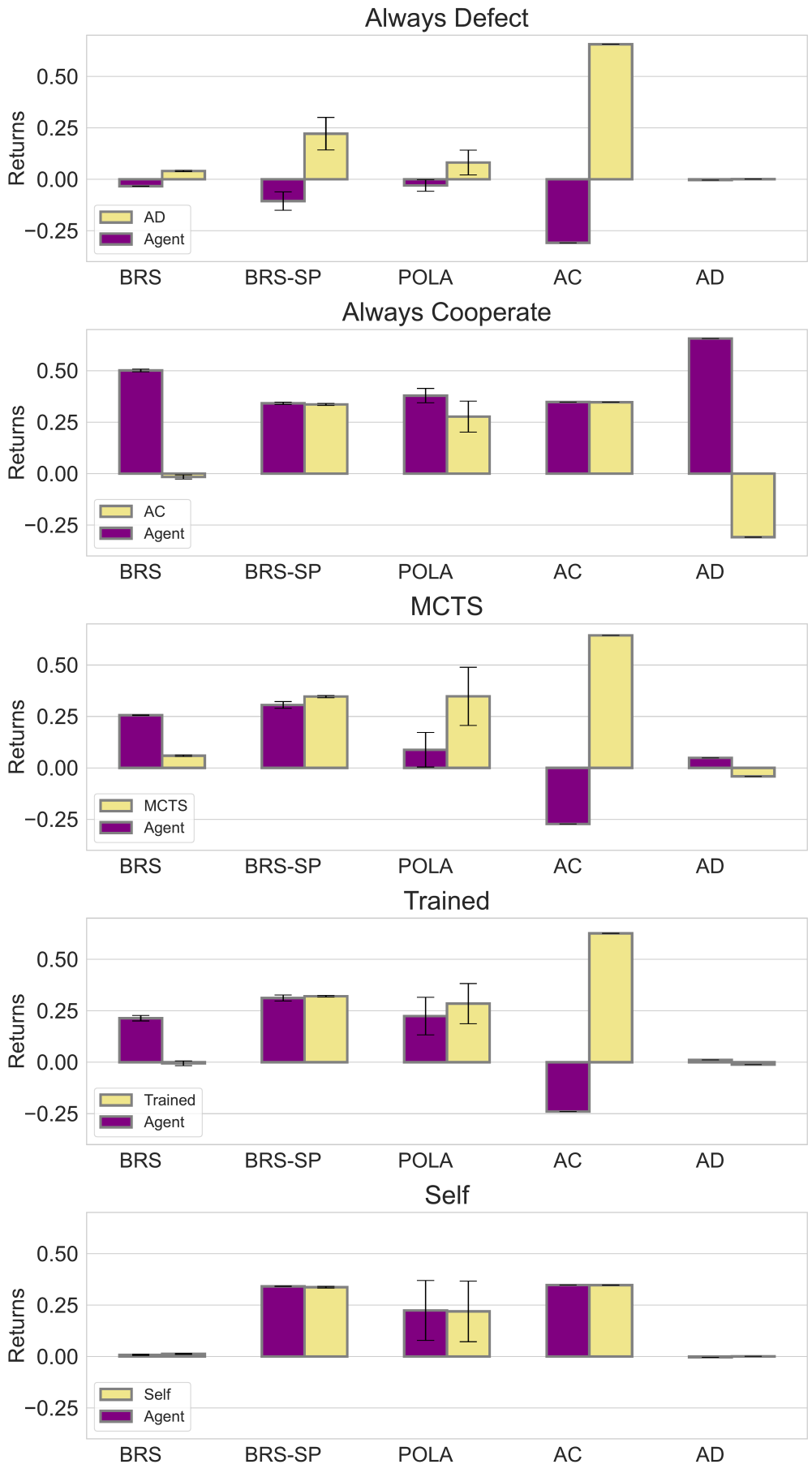

Fig. 2.6. This figure illustrates the performance of all the agents (Including Always Cooperate and Always Defect) against the evaluation metrics.

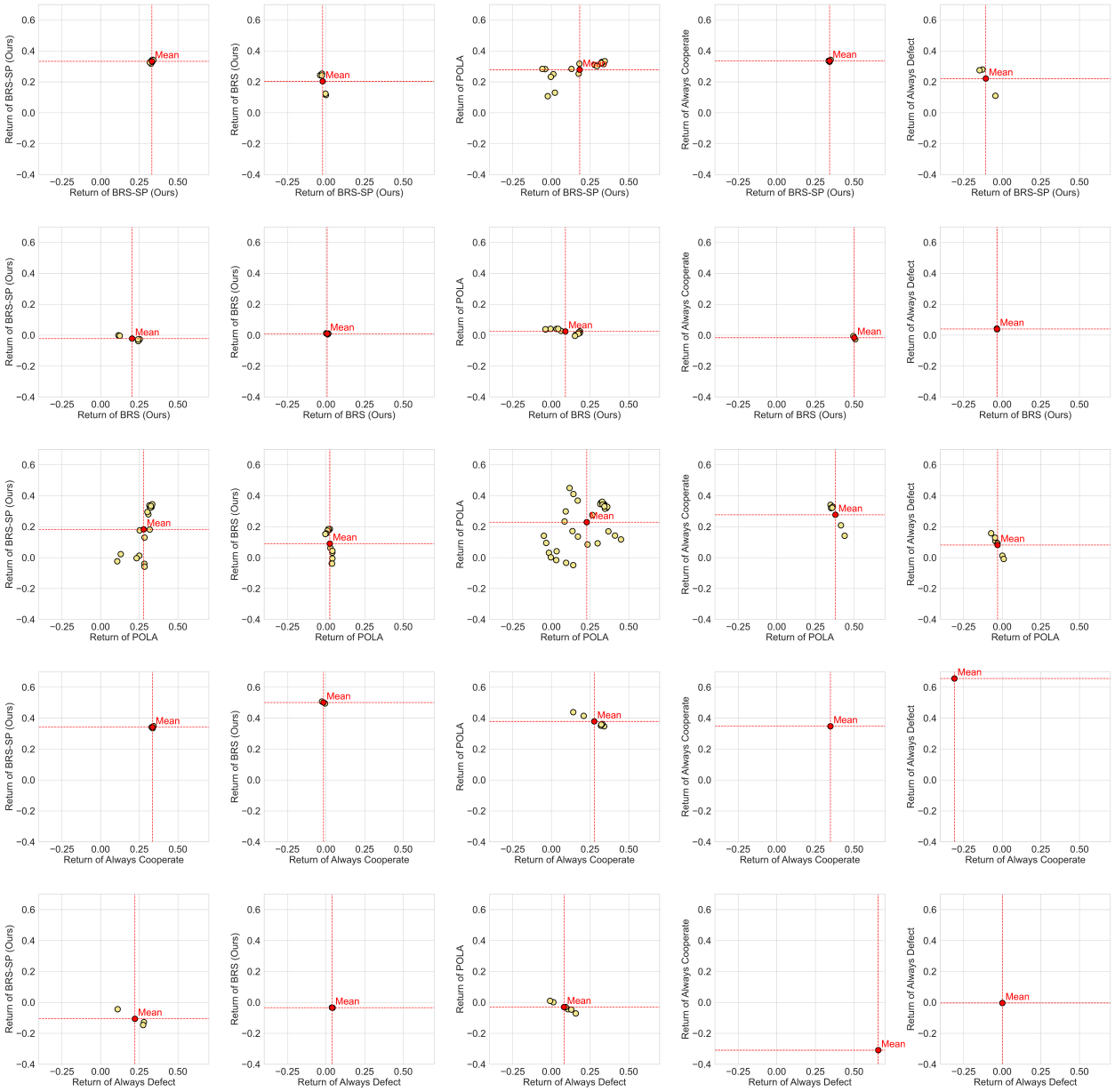

Fig. 2.7. This figure illustrates the performance of all the agents (Including Always Cooperate and Always Defect) against each other.

POLA we used six seeds. Indeed, the POLA agents have more variance in their performance in contrary to the fact that we used more seeds to compute the error bars for them. In Figure 2.7 we visualized the average result of 32 games between different agents.

2.7. League Results

In order to visualize the results of our training in complete detail, in Figure 2.8 we visualize a matrix, in the format of a heatmap, of the returns of various agents against each

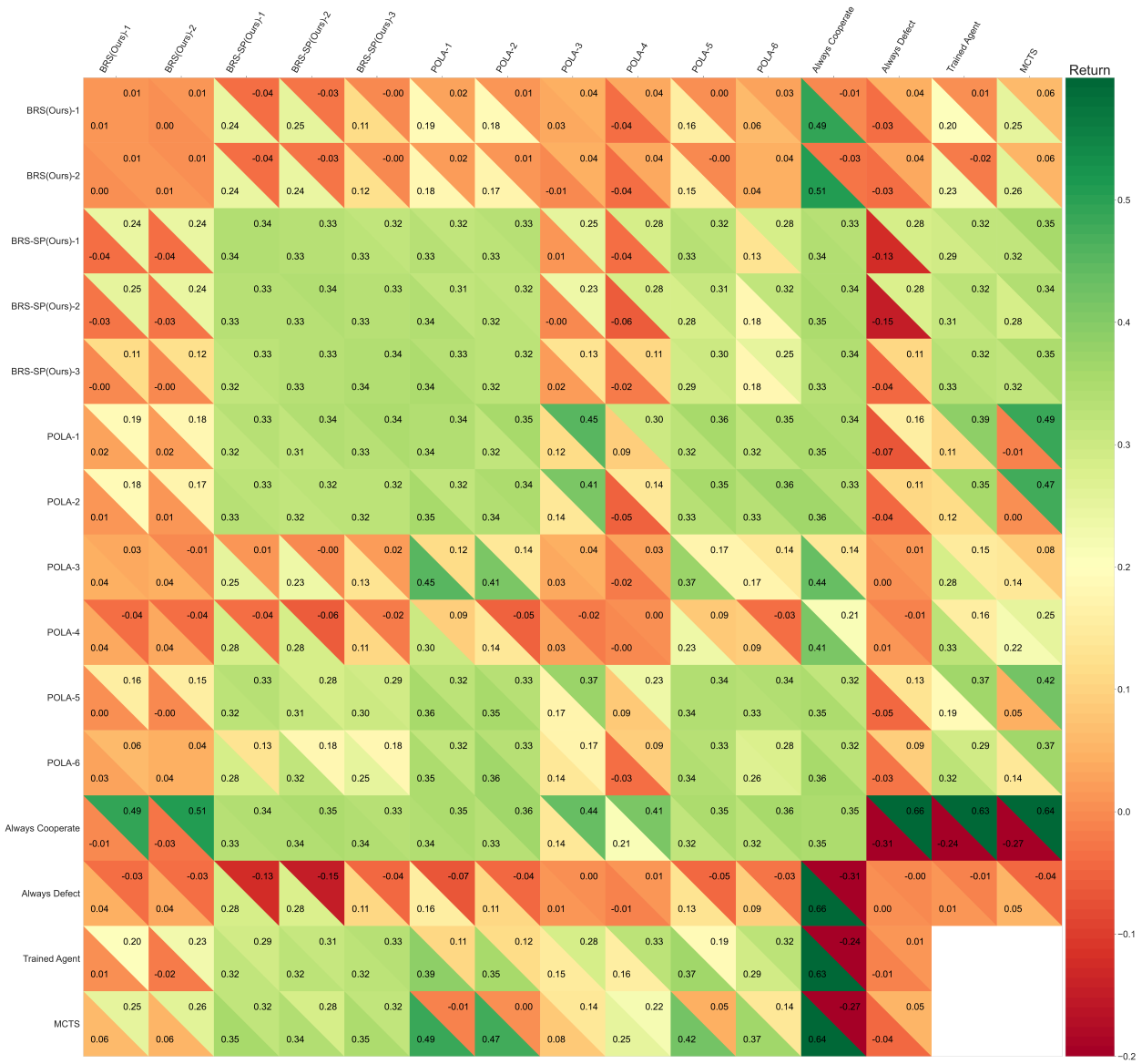

Fig. 2.8. The presented figure illustrates the outcomes of 1-vs-1 Coin games lasting 50 rounds, involving a range of agents. The return achieved by each agent is documented within the corresponding cell. The reported returns are an average across 32 independent games. The numerical suffixes following the agent names signify agents trained using distinct initialization seeds. It is important to note that there are no games recorded between the MTCS agent and the trained agent due to a lack of clarity regarding its meaning.

other. All the results are averaged over 32 independent games between the corresponding agents. The game is the Coin game of length 50. [3]

[3]Note that there is no meaning to test the trained agent against a trained agent. It is because the trained agent assumes a fixed agent and trains against it. Similarly, there is no meaning to train a trained agent against MCTS. Because the MCTS is not a fixed policy assigning probability to actions given different trajectories. Also, there is no meaning to train MCTS against MCTS because the MCTS needs to roll-out the agent's policy to choose an action. However, MCTS against MCTS implies an infinite loop of rolling out the other agent's policy

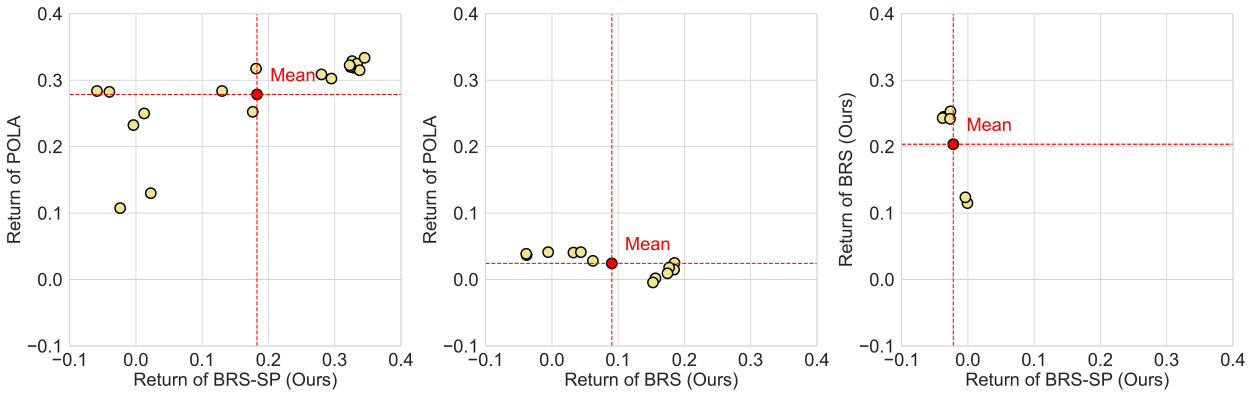

Fig. 2.9. Evaluating BRS, BRS-SP, and POLA agents against each other. The red dot shows the average return of each agent against the average return of the other agent. Left: Both agents get a positive return on average. However, POLA gets a higher return than BRS-SP agent on average, indicating it is exploited. Middle: POLA agents and BRS agents do not cooperate. BRS is able to get a higher return than POLA on average indicating that POLA is exploited. Right: BRS agents get a higher return than the BRS-SP agents. Effectively BRS agent is exploiting the BRS-SP agent.

2.8. Experimental Details

2.8.1. IPD

In IPD experiments, we are experimenting on IPD with 6 steps and discount factor of 1., i.e. no discount factor. The payoff matrix of the IPD game is shown in 2.1.

Player 2 \ Player 1	Cooperate	Defect
Cooperate	-1, -1	0, -3
Defect	-3, 0	-2, -2

Tableau 2.1. Payoff matrix for the prisoner's dilemma game

Our agent's policy is parameterized by a two-layer MLP (Multi-Layer Perceptron) with a tanh non-linearity. The choice of tanh non-linearity is motivated by its smoothing effect and its ability to prevent large gradient updates.

During training, the agent is trained against the Tree Search Detective (TSD) (see Appendix 2.2.3) using a policy gradient estimator. We employ a learning rate of $3e-4$ with the SGD (Stochastic Gradient Descent) optimizer. In the BRS-SP experiments, the Self-Play with reward sharing loss is optimized using SGD with the same learning rate of $3e-4$. To reduce variance, the policy gradients incorporate a baseline.

For replicating the exact results presented in the paper, we provide the code in Appendix 2.9. Running the code on an A100 GPU is expected to take approximately an hour. The plots and error bars are averaged over 10 seeds for both BRS and BRS-SP. The hyperparameter search was conducted by iterating over various learning rates including $(1e-4, 3e-4, 1e-3)$, and the optimizers were explored between SGD and Adam.

2.8.2. Coin Game

The game. Our coin game implementation exactly follows the POLA implementation in [23]. Similar to POLA, we also experiment with the game length of 50 and a discount factor of 0.96.

Agent's architecture. In the coin game, we have an actor-critic setup. The policy of our agent is parameterized by a GRU (Gated Recurrent Unit) architecture, following the approach outlined in the POLA repository (source). However, we introduce a modification compared to POLA by including a two-layer MLP on top of the observations before they are fed into the GRU instead of a single-layer MLP. Additionally, we utilize two linear heads to facilitate separate learning for policy and value estimation.

Detective's architecture. The architecture of the detective is as follows: The sequence of observations is fed into a GRU (Gated Recurrent Unit), which is the same architecture used by the agent. At each time step, the agent's representation is extracted using the QA (Question-Answering) module of the detective. In our experiments, we employed 16 samples of continuing the game for the next 4 steps from the current state.

Environment Simulations. The QA for the BRS experiments changes the environment seed each time simulating the environment. However, in the BRS-SP runs we are using the same environment seed for the simulation which reduces the variance in simulations further but fixes the place that the next coin will appear given the same sequence of events.

Subsequently, the output of the QA module and the GRU are concatenated and passed through a two-layer MLP with ReLU non-linearities. The resulting output from this MLP is then fed into a linear layer for estimating the value (critic), and a linear layer for determining the policy (actor).

Separate optimizers for the two terms. The agent uses separate optimizers for the two terms in the policy gradient. That is, it uses two separate optimizers for the two terms indicated in 2.8.1.

$$\mathop{\mathbb{E}}_{\tau\sim \mathrm{Pr}_{\mu}^{\theta_1,\theta_2}} \left[R^1(\tau) \sum_{t=1}^{T} \left[\underbrace{\nabla_{\theta_1} \log(\pi_{\theta_1}(a_t|s_t))}_{\text{Term 1}} + \underbrace{\nabla_{\theta_1} \log(\pi_{\theta_2}(b_t|\pi_{\theta_1}, s_t))}_{\text{Term 2}} \right] \right] \tag{2.8.1}$$

Losses and optimizers. The value functions in our setup are trained using the Huber loss. On the other hand, the policies are trained using the standard policy gradient loss with Generalized Advantage Estimation (GAE) introduced in [16]. However, it is important to

note that our hyperparameter search led us to set the GAE parameter, λ, to 1, which results in an equivalent estimation of the advantage using the Monte-Carlo estimate. This choice is similar to the hyperparameters reported by POLA (source).

In the BRS experiments, the agent's policy is trained using a learning rate of $1e-3$, while in the BRS-SP experiments, an Adam optimizer with a learning rate of $3e-4$ is utilized. The value functions of both the agent and the detective in all experiments are trained using Adam with a learning rate of $3e-4$. Similarly, the detective's policy is trained using Adam with a learning rate of $3e-4$ in all experiments.

Replay buffer of previous agents. During the training, we keep a replay buffer of previous agents seen during the training. In BRS experiments we keep 2048 previous agents and in BRS-SP experiments we keep the last 512 agents. For training the detective, we sample a batch from this replay buffers uniformly. We add a normal noise with variance of 0.01 to the parameters of these agents to ensure the detective is trained against a diverse set of agents.

Hyperparameter search. We conducted a hyperparameter search using random search over the configurations explained Table 2.2. the entropy coefficient β, which is multiplied by the entropy of the log probabilities associated with the actions of the corresponding player, is added to the policy gradient loss of the corresponding player for controlling the exploration-exploitation trade-off.

Plots and error bars. The results on the paper are computed over two seeds for the BRS, three seeds for the BRS-SP and six seeds for POLA.[4] It is worth noting that the error bars are calculate over seeds, i.e. checkpoints. The result of games between each pair of agents is averaged over 32 independent games between those two agents.

Hyperparameter	**Values**
inner game length in QA	4, 8, 12, 16
samples in QA	16, 64, 256, 1024
replay buffer of agent's size	10, 512, 4096, 16384
value learning algorithm	TD-0, Monte-Carlo
GAE λ	0.9, 0.96, 0.99, 0.999, 1.0
agent policy gradient learning rate	0.001, 0.0003
agent entropy β	0.0, 1.0, 2.0, 5.0, 10.0
detective entropy β	0.0, 1.0, 2.0, 5.0, 10.0

Tableau 2.2. Hyperparameter search options

Compute. Our BRS/BRS-SP runs are run for 48 hours on a single A100 GPU with 40 Gigabytes of RAM[5].

[4]The reason we have fewer seeds for BRS and BRS-SP than POLA is computational constraints due to conference deadlines. Especially, one of our BRS checkpoints was not pickled properly and could not be loaded in the last minute. That is why we don't have three BRS seeds equal to BRS-SP

[5]A single A100 gpu is 80 Gigabyte, but it can be split into two equivalent 40 Gigabyte equivalents and we train on one of these splits

Batch size. We use a batch size of 128.
POLA agent's training. To evaluate the POLA agents, we trained them by executing the POLA repository here [23].

2.9. Reproducing Results

2.9.1. IPD

To replicate the results on IPD (Iterated Prisoner's Dilemma), please refer to the instructions available at here. By running the provided Colab notebook, you will obtain the IPD plot that is included in the paper.

2.9.2. Coin Game

To replicate the outcomes of the coin game, please refer to the instructions available at here. In essence, the provided guidelines encompass training scripts designed for the purpose of training agent checkpoints. Subsequently, there is an exporting phase in which these checkpoints are transformed into their lightweight counterparts. Finally, a script is provided to facilitate the execution of a league involving multiple agents.

Conclusion

Motivated by differentiating through optimization steps of our opponent in order to achieve non-exploitable cooperation, we introduced BRS that differentiates through an approximated best response opponent. We evaluated BRS agents in detail, examining the implications of such an assumption. In particular, we show that BRS agents are not exploited by an MCTS agent. We also introduced BRS-SP for situations in which more cooperative policies are needed. The BRS-SP agent reaches a policy where always cooperate is the best response to that policy. We hope this work helps improving the scalability and non-exploitability of agents in Multi Agent Reinforcement Learning enabling agents that learn reciprocation-based cooperation in complex games.

2.10. Limitations

This thesis focuses on the implementation of our proposed idea in two-player games. However, extending this approach to games with more than two players presents a non-trivial challenge. Additionally, the detective agent approximates the best response opponent, and the accuracy of this approximation is crucial for obtaining correct gradients during the agent's training process. It is particularly important to train the detective against a diverse set of agents in order to ensure the correct gradient estimation. In this study, we introduce a replay buffer that contains a noisy version of agents encountered during training, which serves as a proxy for a diverse agent set. Nevertheless, for more complex settings, this level of diversity may be insufficient. While we employ simulation-based question answering against a random agent to condition the detective in the Coin Game, it should be noted that a random agent might not provide adequate conditioning for more intricate games.

Références bibliographiques

[1] Alekh Agarwal, Nan Jiang, Sham Kakade, and Wen Sun. *Reinforcement Learning: Theory and Algorithms*. 2021.

[2] Bowen Baker. Emergent reciprocity and team formation from randomized uncertain social preferences, 2020.

[3] Anton Bakhtin, David Wu, Adam Lerer, and Noam Brown. No-press diplomacy from scratch. *Advances in Neural Information Processing Systems*, 34:18063–18074, 2021.

[4] Nolan Bard, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H. Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G. Bellemare, and Michael Bowling. The hanabi challenge: A new frontier for ai research. *Artificial Intelligence*, 280:103216, 2020.

[5] Micah Carroll, Rohin Shah, Mark K. Ho, Thomas L. Griffiths, Sanjit A. Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-ai coordination, 2020.

[6] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics.

[7] Tianshu Chu, Jie Wang, Lara Codecà, and Zhaojian Li. Multi-agent deep reinforcement learning for large-scale traffic signal control, 2019.

[8] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games: 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers 5*, pages 72–83. Springer, 2007.

[9] Meta Fundamental AI Research Diplomacy Team (FAIR)†, Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, et al. Human-level play in the game of diplomacy by combining language models with strategic reasoning. *Science*, 378(6624):1067–1074, 2022.

[10] Jakob N. Foerster, Richard Y. Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness, 2018.

[11] Jean Harb, Tom Schaul, Doina Precup, and Pierre-Luc Bacon. Policy evaluation networks. *arXiv preprint arXiv:2002.11833*, 2020.

[12] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[13] Chris Lu, Timon Willi, Christian Schroeder de Witt, and Jakob Foerster. Model-free opponent shaping, 2022.

[14] Philip Paquette, Yuchen Lu, Seton Steven Bocco, Max Smith, Satya O-G, Jonathan K Kummerfeld, Joelle Pineau, Satinder Singh, and Aaron C Courville. No-press diplomacy: Modeling multi-agent gameplay. *Advances in Neural Information Processing Systems*, 32, 2019.

[15] Philip Paquette, Yuchen Lu, Seton Steven Bocco, Max Smith, Satya O-G, Jonathan K Kummerfeld, Joelle Pineau, Satinder Singh, and Aaron C Courville. No-press diplomacy: Modeling multi-agent gameplay. *Advances in Neural Information Processing Systems*, 32, 2019.

[16] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018.

[17] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[18] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm, 2017.

[19] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[20] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[21] Yaodong Yang and Jun Wang. An overview of multi-agent reinforcement learning from game theoretical perspective, 2021.

[22] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms, 2021.

[23] Stephen Zhao, Chris Lu, Roger Baker Grosse, and Jakob Nicolaus Foerster. Proximal learning with opponent-learning awareness. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.