Cheating Detection in Zero-Sum Games for Mixed Multi-Agent Environments

Aishik Pyne
A0250592E

Harshavardhan Abichandani
A0250610X

Niharika Shrivastava
A0254355A

Abstract

Multi-agent reinforcement learning (MARL) is a prevalent learning paradigm for solving stochastic games. Specifically, imperfect information games model strategic interactions between multiple agents with only partial information which can be modelled as a POMDP [3]. In most MARL studies, agents in a game are defined as teammates or enemies beforehand and these relationships among the agents are known to all. However, in real-world problems, the agent relationships are commonly unknown in advance. Therefore, training policies for such situations in the face of imperfect information and ambiguous identities is a significant problem that needs to be addressed. In this project, we aim to model the scenario for multi-agent partially-observable zero-sum games (e.g., poker, red-10) where M agents play collaboratively against $(N - M)$ competitive agents. However, the $(N - M)$ agents unknowingly assume a fully competitive setting. With such a secret arrangement between the collaborative agents to split the reward equally after a game, we wish to observe if the collaborative agents can develop a strategy to beat the competitive agents on expectation. We also provide a discriminator that observes each agent's entire gameplay and detects cheating (a collaboration between M agents).

1 Introduction

Most AI systems nowadays are based on a single agent tackling a task, or in the case of adversarial models, multiple agents compete against each other to improve the overall behaviour of a system [7]. However, many real-world scenarios require a mixed setting where multiple agents act either collaboratively or competitively with each other at different timesteps to get the best returns for themselves (e.g., driving a car to reach the destination in the shortest time while avoiding traffic). In this aspect, multi-agent reinforcement learning (MARL) is a prevalent learning paradigm for solving stochastic games. Imperfect-information games model strategic interactions between multiple agents with only partial information (e.g., poker, blackjack, red-10). Typically in such games, one seeks a Nash equilibrium [6], in which no player can improve by deviating from the equilibrium. Currently, the most successful algorithms for partially observable zero-sum games are Deep CFR [2], NFSP [4], and ReBel [1].

It is interesting to note that in most multi-agent works, whether an agent is acting collaboratively or competitively with other agents is known by everyone from the beginning which affects everyone's policies. However, in a real-life scenario, such an assumption may not always hold (e.g., a game of poker where 2 agents have teamed up against all other players with the intent of cheating). Therefore, we aim to model such a partially-observable mixed environment where individual agents attempt to maximize their reward assuming a fully competitive setting, but the teammates acting collaboratively are expected to win most games on average by maximizing their combined reward, at the expense of lower individual rewards. Such cheating scenarios are common in highly-dynamic, complex environments such as casinos. Therefore, we also train a cheating detector system that is able to

Preprint. Under review.

observe the entire game state at all times along with the game-plays of every agent and is able to classify if there was any foul play during the game. Our contributions are as follows:

1. We train poker agents where the policy for each agent is learned in a fully-competitive setting using NFSP [4].
2. We train poker agents where two agents play collaboratively against N competitive players in order to beat them in most games. Just as in a real-life case, each player's game space is still partially observable but the expected team reward is maximized.
3. Model a discriminator that learns to distinguish between these two styles of play (competitive vs collaborative) by comparing the actions taken by each player with our trained policies.

2 Related Work

There are many approaches developed for finding nash equilibrium in games with imperfect information [1, 2, 4, 5]. We will be focusing on the two most popular approaches, Neural Fictitious Self Play and Deep Counterfactual Regret Minimization.

Neural Fictitious Self Play (NFSP) [4] is one of the algorithms used to train agents for imperfect information games (eg. poker) which combines fictitious self play with neural networks. It is a two stage process, which can be thought of as an actor-critic framework. There is a critic which is trained using DQN. Critic stores all the $< S, A, R, S' >$ tuples in it's buffer and it is used to predict the value of the actions taken by the agent. The second part is an actor, which is trained using supervised learning. It's buffer is populated with $< S, A >$ pairs only when we sample the best response action from the critic. It learns the average policy and is used during evaluation.

Deep Counterfactual Regret Minimization (Deep CFR) [2], it combines CFR with neural networks. It builds on CFR which traverses the search space and accumulates the regrets at each stage. Strategy is chosen based on algorithms which work on the principle of regret minimization. CFR cannot be used to for games with large state space, to solve this Deep CFR uses neural networks to approximate the rest of the state space. In Deep CFR at every iteration the game tree is partially traversed K times and the paths are sampled using MCCFR. After the traversals, a neural network is used to predict the value of exploring that node. This can be thought of as instantaneous regrets in the CFR algorithm.

3 Background

3.1 Poker Rules

Limit Texas Hold'em Poker consists of N players playing for four rounds. Each player get two hand cards from a deck of 52 shuffled cards, and at each round new community cards are revealed. In each round a player can choose between four actions (call, check, raise, fold). In Limit Texas Hold'em the number of raises is capped to four in each rounds, and the max raise amount is 1.5x the big blind. After four rounds, the best hand wins and that player gets the total chips in the pot. Since it is a zero sum game, the sum of rewards of all the players should sum to zero. This means that there will be players with negative rewards.

3.2 POMDP

We model Poker as a Partially Observable Markov Decision Process (POMDP) which is described as the tuple $\langle S, A, T, R, O, \Omega\rangle$ where:

- S - finite set of states of the environment (game space). For Limit Texas Hold'em Poker, the game space is of the order 10^{14}.
- A - finite set of actions each agent can take at each turn (Raise, Fold, Call, Check).
- $T : S$ x $A \rightarrow \Delta(S)$ - state-transition function giving a distribution over states of the environment, given a starting state and an action performed by the agent.
- R - the reward function, giving a real-values expected immediate reward, given a starting state and an action performed by the agent. (money/chips after every game).
- Ω - finite set of observations the agent can experience.

- $O : S$ x $A \rightarrow \Delta(\Omega)$ - the observation function, giving a distribution over possible observations, given a starting state and an action performed by the agent.

4 Methodology

The section is divided into three parts. In 4.1 we show how we encode the state space, evaluate and train our agents, and in 4.2 we discuss how we model the communication. Finally, 4.3 elaborates on how we take advantage of the tractable action space to discriminate between two agents.

4.1 Competitive Agents

We encode the space space of limit hold'em poker into seventy two bits. The first fifty two bits contain the hand cards and the community cards. The last twenty bits contains the number of raises in each round. Limit hold'em has four rounds and each round can have a maximum of four raises. Therefore, we one hot encode this information into twenty bits. This allows us to make the state space independent of the number of players, and also includes the information about the actions taking place in the game. This way of encoding the state space is taken from the RLCard [8] library.

For our setting we consider a three player setting. In this setting all the agents are playing to maximize their own reward i.e. the chips they win at the end of the round. We initialize a policy network which we will refer to as "Brain 1". We first simulate a game with all the three players and generate the trajectories. Then we feed this trajectories to the neural network and train Brain 1 using the NFSP [4] algorithm.

4.2 Collaborative Agents

This setting contains three agents, out of which two agents are collaborating to maximize their collective reward. To model the communication between two agents we explore two scenarios. First, the agents simply know if their friend has raised in the round. This simulates real life setting where collaborating agents need to infer from the actions. Second, we give the hand cards of the other agent along with the previous information.

In this setting we set up rewards in the following ways.

where c_i is the number of chips won by player i after action a performed in state s.

$$\mathcal{R}(s,a) = \begin{cases} \max c_1, c_2, & c_3 \leq 0 \\ \dfrac{c_1 + c_2}{2}, & c_3 > 0 \end{cases}$$

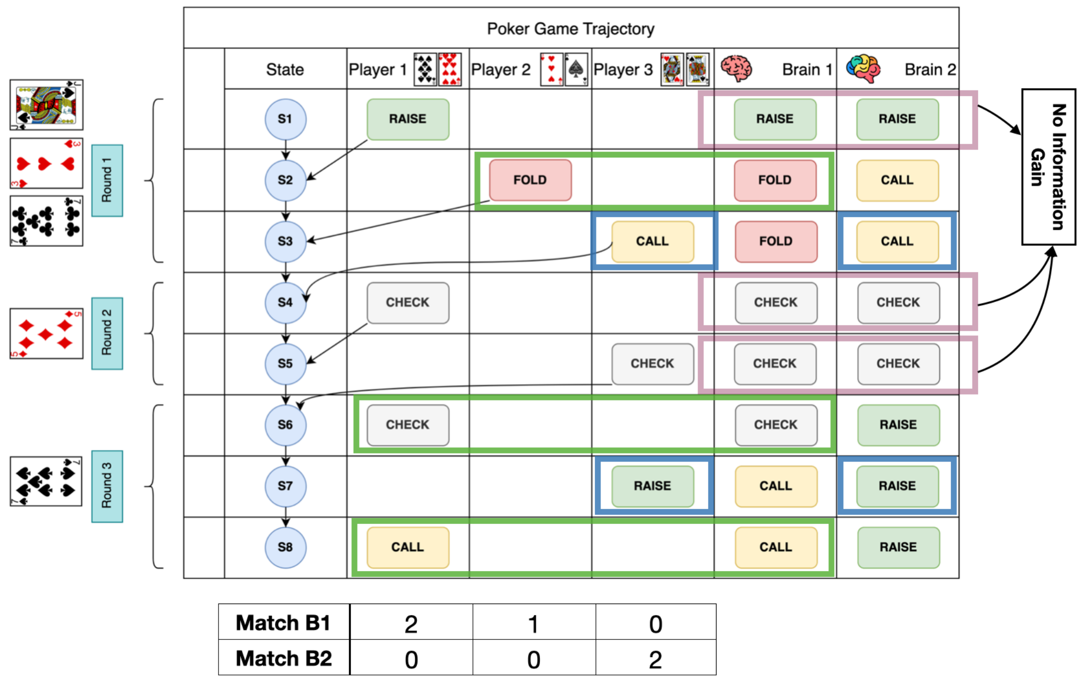
where c_i is the number of chips won by player i after action a performed in state s.

Using the above reward function we train two agents using a policy which we will call "Brain 2". During training we are freezing the weights of the competitive agent (Brain 1) and initializing the collaborative agent with the weights of Brain 1.

4.3 Discriminator

The action space of the game is extremely small compared to the state space (4 vs 10^{14}). Therefore, we take advantage of the small action space and create a discriminator that doesn't need to enumerate the state space to detect cheating. The discriminator analyses the entire trajectory of an agent and predicts if the behaviour is closely related to Brain1 or Brain2. In Figure 1, we see that only enumerating the actions that match with either of the Brains and not both (green and blue boxes) is sufficient to determine the behaviour of the agent on expectation over multiple games.

Specifically, there is no information gain in terms of behaviour deduction if both brains choose the same action for a given state (pink boxes). However, for state spaces that provide different actions for both brains (green and blue boxes), we sum the number of times the agent matches the actions with a specific brain. Over a large number of games, the agent's true behaviour would closely imitate the behaviour of either brain. Therefore, the discriminator would be able to detect if an agent is cheating or playing fairly over an expectation.



Match B1	2	1	0
Match B2	0	0	2

Figure 1: The pink boxes show no information gain since both the Brains exhibit the same behaviour. The green and blue boxes depict the actions of each agent that match Brain1 and Brain2 respectively.

5 Experiments

This section will discuss the parameters and the results of the following experiments.

5.1 Training Results for Competitive agent

We trained the three agents for $85k$ games and evaluated the agents against rule based poker agent and random agent. Brain 1 (policy network) consisted of a simple two layer mlp with 64 neurons and was trained using the nfsp [4] algorithm.

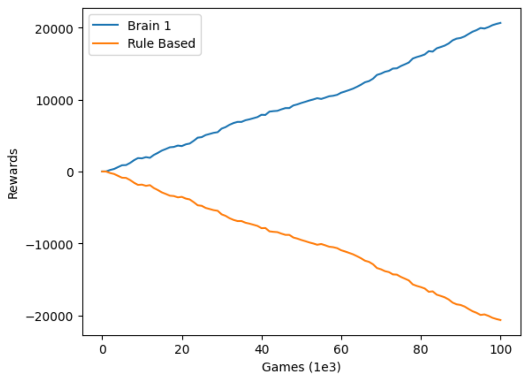


Figure 2: Brain 1 vs Rule Based agents accumulated reward over number of games.

We compare Brain 1 with a rule based poker agent, which takes actions by calculating the probability of winning based on the hand and community cards. We do this to see if our policy is making complex strategies which do not rely on simple probabilities. Brain 1 achieves a win ratio of 3.66 over the rule based agent, and we can see that in figure 2 Brain 1 is accumulating more rewards over time.

Since our framework does not depend on the type of algorithm we use to train, we can confidently say that Brain 1 has at least learned some complex strategy to defeat naive poker algorithms.

5.2 Training Results for Multi agent

In this setting we kept the policy network size same, to maintain consistency. We trained the collaborative agents for $85k$ games and evaluated based on the win ratio and rewards accumulated over time.

As we can see in Figure 3 that collaborative agents over time learn how to consistently defeat the competitive agent. This means that just by adding the information if the collaborative agent raised in that round, is enough to make it outperform. We also simulate $100k$ independent games and find that collaborative agents have a win ratio of 1.14 over the competitive agent. Which means that on average it is winning 14% more games.

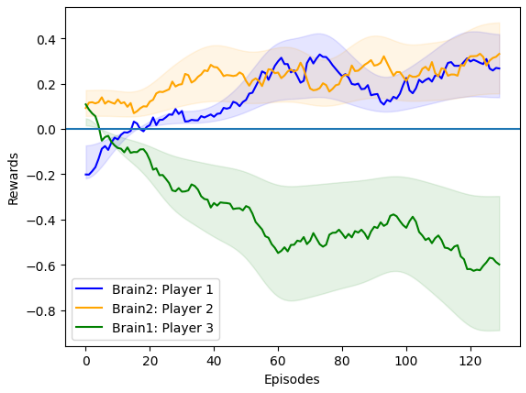


Figure 3: Training graph of collaborative setting.

We find out the marginal action distribution of Brain 1 vs Brain 2. Since state space is intractable, we are computing trajectories for $100k$ independent games and keeping track of the actions taken by the policies. In figure 4 we can see that Brain 2 is performing more number of raises. Which makes intuitive sense, since collaborative agents have more chances of winning, it can afford to take more risks.

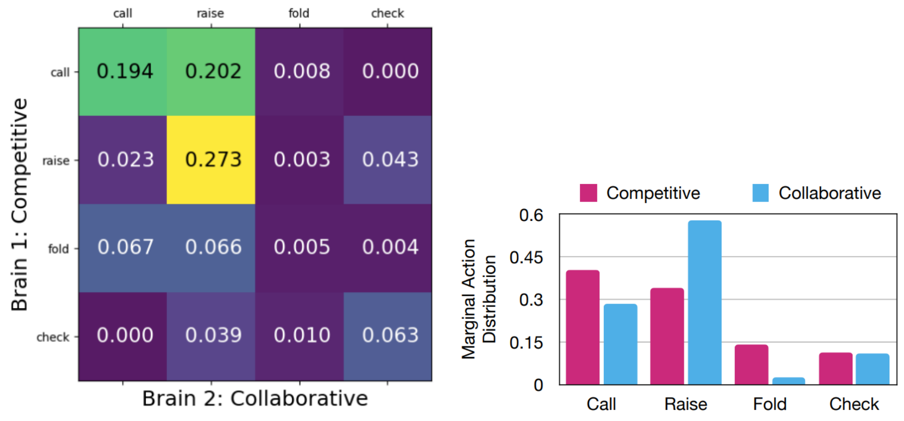


Figure 4: Action distribution of Brain 1 vs Brain 2, $p(B_1, B_2)$

5.3 Discriminator

The plots in Figure 5 show that a rule-based agent is similar to a competitive (fair) player, whereas both the collaborative agents demonstrate high foul play behaviours over an increasing number of

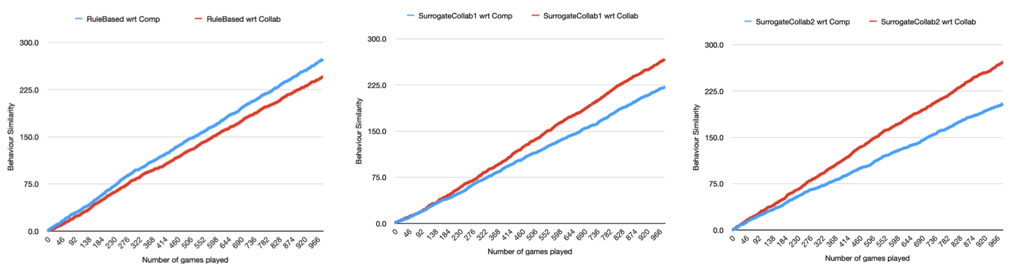


Figure 5: Discriminator: Similarity plots of (a) rule-based (b) collaborative teammate 1, and (c) collaborative teammate 2 agents with respect to fair and foul plays.

games. This is an intuitive result since a rule-based agent has no information about any other player's hand or the motivation for making any raises (bets), and plays only to maximize their own reward.

6 Conclusion

In this project, we have successfully trained multi-agents in a fully competitive and mixed environment to play a zero-sum game. We also modelled a strategy that enables the collaborative agents to win over the competitive agents on average and analysed this performance using multiple evaluation metrics. Finally, we provided an easy method to model a discriminator that can classify the behaviour of the agents as either fair or foul by analysing their individual gameplays.

7 Future Works & Discussions

Current evaluation and training methods are done on independent games, which makes the problem easier. So there is no long term strategy employed by the agents to win the game. We plan to train the agents on dependent games, where the winning of one game will impact the next game. Moreover, it is not intuitive that just by adding the information of which player raised makes the collaborative agent outperform the competitive agent. While there is strict information gain, it doesn't lead to any semantic meaning in real life setting.

References

[1] Bakhtin A. Lerer A. Gong Q. Brown, N. Combining deep reinforcement learning and search for imperfect-information games. *ArXiv. /abs/2007.13544*, 2020.

[2] Lerer A. Gross S. Sandholm T. Brown, N. Deep counterfactual regret minimization. *ArXiv. /abs/1811.00164*, 2018.

[3] Michael Hahsler and Hossein Kamalzadeh. Pomdp: Introduction to partially observable markov decision processes. *https://cran.r-project.org/web/packages/pomdp/vignettes/POMDP.html*, 2021.

[4] Silver D. Heinrich, J. Deep reinforcement learning from self-play in imperfect-information games. *ArXiv. /abs/1603.01121*, 2016.

[5] Matej Moravč ík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, may 2017.

[6] Martin J. Osborne and Ariel Rubinstein. A course in game theory. 1994.

[7] Jesus Rodriguez. How modern game theory is influencing multi-agent reinforcement learning systems. *Medium: Dataseries*, 2020.

[8] Lai K. Cao Y. Huang S. Wei R. Guo J. Hu X. Zha, D. Rlcard: A toolkit for reinforcement learning in card games. *ArXiv. /abs/1910.04376*, 2019.