Investigating relaxed probability updating games

Master Thesis

March, 2022

Author: Collin Aldaibis

First supervisor: Thijs van Ommen
Second supervisor: Gerard Vreeswijk

Utrecht University

Computing Science
Graduate School of Natural Sciences
Utrecht University

Contents

Abstract

Digital data is everywhere; it is the backbone of science and our modern society. But data is sometimes incomplete. A complex form of incomplete data is when data is coarse. Many coarse data problems cannot be solved with standard conditioning. The problem can be reformulated as a probability updating game: a zero-sum game between a host and a contestant. An instance of a probability updating game is made from rewriting the Monty Hall problem as a game. It is proved that if the host plays a strategy that satisfies the RCAR condition, it plays worst-case optimally and the probabilities can be updated robustly for the contestant. We study whether RCAR still characterises Nash equilibria when the zero-sum constraint or the one-shot constraint of these games are removed. We found that if RCAR characterises optimality for a zero-sum, one-shot probability updating game, it also characterises optimality for the finitely repeated game. Moreover, we conclude from empirical analysis that if RCAR characterises optimality for a zero-sum probability updating game, it may also characterise optimality for a moderately competitive non-zero-sum game.

1 Introduction

Coarse data problems Presently, digital data is everywhere and has a lot of value. Data can be of many forms and enormous sizes, which can make it difficult to retrieve the most interesting or important parts. It becomes even harder when the data is incomplete. The term "incomplete" has several definitions. One example of incomplete data is missing data. Another one is that of *coarse* data. There is a vast amount of scientific literature that aims to manage missing data points. Conversely, there is significantly less known about solving coarse data problems, while being potentially more complex.

To elucidate, we will provide an example. Suppose you are trying to predict the outcome of a random variable, of which the distribution is known. At some point, you receive new information; you observe a subset of possible outcomes and are told that the true outcome is an element of this subset. How do you update your distribution in light of this information?

This exemplifies a coarse data problem. Instead of seeing a single, true outcome, we observe data that is *coarsened*. This coarsening has a specific structure, and an unknown *coarsening procedure* determines which message, i.e. a subset of outcomes, is sent to us.

Generally, for statistical inference problems, the default method for knowledge updating is to condition on the given probabilities. We will call this method *standard conditioning* from now. For coarse data problems, this method does not always work.

Heitjan and Rubin (1991) showed that any coarse data problem for which the coarsening procedure is *coarsened at random (CAR)*, standard conditioning gives the correct answers. However, there are known to be many problems that do not satisfy CAR. The Monty Hall problem, a famous probability puzzle, is among them. For such problems, standard conditioning does not suffice. It does not consider the structure of the coarse data, which can affect probabilities heavily. The result: incorrect predictions on the outcomes.

Zero-sum probability updating games Van Ommen et al. (2016) proposed a method for correctly updating probabilities for coarse data problems where standard conditioning gives incorrect answers. This method uses game-theoretic concepts, modelling the coarse data problem as a *probability updating game*. It is a simultaneous zero-sum game between a *host* and a *contestant*.

In this game, a true outcome is sampled according to some known distribution. The host knows the true outcome and sends a set of possible outcomes to the contestant. The host does that using its coarsening procedure, i.e. a joint distribution of the messages and outcomes. The contestant makes a prediction, i.e. a conditional distribution on outcomes given a particular message. Finally, the true outcome is revealed and the contestant's prediction quality is measured using a loss function.

Naturally, the contestant aims to minimise the loss, therefore optimising its predictions. In contrast, the host aims to maximise it, making it as hard as possible to predict the true outcome. The coarsening procedure corresponds to the host strategy, while the prediction corresponds to the contestant strategy.

Van Ommen et al. showed that for a subset of probability updating games, worst-case optimal strategies are characterised by a condition we call the *reverse* CAR, or *RCAR* condition: when

the host plays a strategy that satisfies the RCAR condition, it is a worst-case optimal strategy. Subsequently, the strategy for the contestant can be found quite easily. Hence we can *robustly* update the probabilities. Or in other words, we set an upper bound to the expected loss we will incur for a worst-case coarsening procedure. A full description is given in chapter 3.

Relaxed probability updating games Van Ommen et al. argue that a decision maker will not make optimistic assumptions on the coarsening procedure, as it has no knowledge whatsoever on the procedure. In such a case, a worst-case optimal approach is a correct analysis method. Yet in practice, it is quite rare that the decision maker has no knowledge of the procedure. Therefore, we argue that a worst-case optimal approach may not always be required. We ask, what if the procedure is not "against" the decision maker? Out of many possible procedures for a given message structure, the true procedure might be quite advantageous for the decision maker.

Also, the host and the contestant might interact multiple times. Game-theoretic literature tells us that different equilibria may arise when a game is played *repeatedly*, in which agent rewards are at least as good as in the *one-shot* instance of the game.

It is not known what Nash equilibria would emerge when either the zero-sum- or the one-shot-constraint are relaxed. Can we find and describe the best response strategies that form Nash equilibria for relaxed probability updating games? And how do they relate to Nash equilibria for zero-sum, one-shot games? How do they relate to RCAR?

We analyse relaxed probability updating games experimentally. We investigate several classes of probability updating games, among which graph and matroid games (see section 3). In contrast to zero-sum one-shot games, we can not determine Nash equilibria analytically, nor can we compute them efficiently. Therefore, we use *multi-agent reinforcement learning* (MARL) to learn optimal strategies approximately. Consequently, we investigate and describe the resulting optimal strategies.

This document constitutes a master thesis on the investigation of relaxed probability updating games. We propose our research question in chapter 2. Then, we review the related literature in chapter 3. Chapter 4 describes our approach thoroughly. Subsequently, chapter 5 captures the results of our research. We discuss our research in chapter 6 and we make our final conclusions in chapter 7.

2 Aims

The purpose of this research project is to study player strategies in *relaxed* probability updating games. Zero-sum probability updating games assert that the host is *malicious* towards the contestant. But as mentioned in section 1, such situations are rare in practice. We know that RCAR characterises Nash equilibria for zero-sum graph and matroid games, but we have no theoretical guarantees when relaxing the zero-sum constraint. Therefore, we aim to investigate optimal strategy pairs in non-zero-sum probability updating games. We formulate our first two questions:

1. *How are Nash equilibria described in non-zero-sum graph and matroid games?*
2. *How is RCAR related to Nash equilibria in non-zero-sum games?*

When relaxing the zero-sum constraint, the minimax theorem no longer applies, i.e. worst-case optimal strategy pairs (P^*, Q^*) are no longer guaranteed to be a Nash equilibrium. Thus, Nash equilibria will have to be found in other ways. We explain this in section 4.1.

Furthermore, we have an interest in *repeated* probability updating games. In repeated games, players play the *single-shot* game, which is the regular game, for a possibly infinite amount of repetitions. In traditional game theory, *folk theorems* prove that socially optimal strategy profiles of the constituent game emerge as Nash equilibria in the repeated game. For probability updating games, this may cause the strategies for the players to look different from the one-shot game. Considering this, we formulate our next question:

3. *How are Nash equilibria described in repeated zero-sum probability updating games?*

We investigate both graph and matroid games together, as we believe that the results will be equivalent for these two types of games. Eventually, *if* we find any interesting patterns in the Nash equilibria, we aim to investigate those, to see whether there are similarities to the RCAR condition.

Ideally, one would also want to characterise these non-zero-sum Nash equilibrium strategies formally if possible. However, this would require rigorous mathematical substantiation that would include proofs. This part will be omitted given the time constraints for this project.

3 Literature review

In this chapter, we review the general coarse data problem and all of the relevant scientific literature regarding coarse data, probability updating games and repeated games and finding Nash equilibria.

3.1 Coarse data problems

Suppose a decision maker is trying to predict the value of a random variable with finite *outcome space* $\mathcal{X}$. Consider a known strictly positive outcome distribution p on $\mathcal{X}$, where p_x denotes the (positive) probability of $x \in \mathcal{X}$ being sampled. Furthermore, let $\mathcal{Y}$ be the set of *messages* that may be revealed to the decision maker. Any message $y \in \mathcal{Y}$ is a subset of possible outcomes: $y \subseteq \mathcal{X}$.

First, outcome $x \in \mathcal{X}$ is drawn and hidden from the decision maker. We have an unknown coarsening procedure P, which is a joint distribution on $\mathcal{X} \times \mathcal{Y}$ that chooses message y with probability $P(y \mid x)$. We require of P the following:

- P is *marginal consistent*; the marginal distribution of $P(x)$ is equal to p_x.
- P may not lie about the true outcome, so we have $P(y \mid x) = 0$ for all $x \notin y$. Otherwise, it could reveal a message that does not contain the true outcome.

Note that if the decision maker would know $P(y \mid x)$, it could compute the desired $P(x \mid y)$ rather trivially. Instead, we have *prediction* $Q \in \Delta_{\mathcal{X}}$, which is a conditional distribution that picks x on observing y with probability $Q(x \mid y)$. We can constrain Q by the following:

- $Q(x \mid y) = 0$ for all $x \notin y$. It is not rational to assign positive probability to excluded outcomes.

3.1.1 Coarsening at random

In a best-case scenario, the decision maker makes perfect predictions, i.e. $Q^*(x \mid y) = P(x \mid y)$. In some cases, one can make assumptions on the procedure based on the data. It turns out that when P is assumed to be coarsened at random (CAR), we can still update probabilities correctly with standard conditioning (Heitjan and Rubin (1991)).

Definition 1 (CAR)**.** *Suppose coarsening procedure P. For all $x, x' \in y$, we have*

$$P(y \mid x) = P(y \mid x')$$

If P satisfies the above, we say that procedure P is coarsened at random, or CAR.

The question of whether or not the data is CAR has proved to be highly relevant in several statistical areas. One of those areas is missing data (Rubin (1976)). Often, such data sets lead to CAR selection procedures. But also in survival analysis, CAR is an established tool for solving statistical problems (Gill et al. (1997)).

Still, CAR is not the answer to every coarse data problem. Grünwald and Halpern (2003) studied the condition exhaustively and found that CAR is only guaranteed to hold if and only if $\mathcal{Y}$ is a partition of $\mathcal{X}$, meaning that there can be no x that is contained by more than one message.

Grünwald and Halpern (2003) characterised when CAR might still hold and when CAR is guaranteed to be violated. As a result, they argue that CAR imposes very strict conditions on the data and that the condition can rarely be assumed.

3.1.2 The Monty Hall problem

One instance of a problem where CAR cannot be assumed is the famous Monty Hall probability puzzle, as illustrated below:

> *I show you three closed doors. One door hides a car, while the other two hide a goat. You pick a door. Instead of opening that door, I open one of the other two doors, and a goat is revealed. Now, I ask you, before making your decision final, if you'd like to switch from the door you initially chose.*
>
> *Is it a good idea to switch to the other door?*

Originally issued by Selvin (1975) as a game between a quizmaster and a contestant and popularised later in Savant (1990), the Monty Hall problem created quite the discussion between mathematicians. The correct answer here is to switch to the other door: an agent that switches doors has a winning probability of $\frac{2}{3}$, while the agent that stays only wins with probability $\frac{1}{3}$. This goes against human intuition, but also against the rational arguments of a statistician using standard conditioning.

The Monty Hall problem can be reframed as a coarse data problem: we have $\mathcal{X} = \{x_1, x_2, x_3\}$, and we assume the car is hidden uniformly at random behind one of the three doors, i.e. p is uniform on $\mathcal{X}$. By symmetry, it does not matter which door is initially picked by the contestant. So, we can remove this initial event fully and assume that the contestant always picks x_2. Coarsening procedure P is controlled by the quizmaster, which we will call the *host* from now. As x_2 is always picked initially, the host is either:

- (if x_1 is true) forced to open door x_3 (thus leaving doors x_1 and x_2 closed)
- (if x_3 is true) forced to open door x_1 (thus leaving doors x_2 and x_3 closed)
- (if x_2 is true) free to open either door x_1 or door x_3

On opening a door, the two remaining doors represent the message; the set of possible outcomes. This gives us message set $\mathcal{Y} = \{y_1, y_2\}$, with $y_1 = \{x_1, x_2\}$ and $y_2 = \{x_2, x_3\}$.

If the true outcome is x_2, P might show either y_1 or y_2. Usually, we do not know P, but as the Monty Hall problem is a game show, we assume the host assigns *equal* probability to both messages, as there has been given no reason to assume otherwise (although, P could theoretically assign different probabilities to y_1 and y_2, which would result in different probabilities predicted by the contestant). An example strategy is depicted in table 3.1.

$P^*(y \mid x)$	x_1	x_2	x_3
y_1	1	0.5	-
y_2	-	0.5	1

Table 3.1: A worst-case optimal strategy P^* for the Monty Hall problem

There exists no P that is CAR for this problem. (The only exception is when we assume that p_{x_2} is either 0 or 1 (see Grünwald and Halpern (2003), example 3.3), essentially making $\mathcal{Y}$ a partition

on $\mathcal{X}$ if we filter out the outcomes with probability zero). In other words, standard conditioning on the probabilities is not a proper tool for solving Monty Hall.

3.1.3 Loss functions

The decision maker wants to optimise its prediction $Q(\cdot \mid y)$ on observing message y. So, when outcome x is revealed to the decision maker, we use loss function $L(x, Q(\cdot \mid y))$ that assesses the decision maker's prediction quality.

In choosing L, we need to study the specific problem at hand. Different problems require appropriate loss functions. Van Ommen et al. (2016) mainly focused on the following three loss functions:

Randomised 0-1 loss $L(x, Q(\cdot \mid y)) = 1 - Q(x \mid y)$

Brier loss $L(x, Q(\cdot \mid y)) = (1 - Q(x \mid y))^2 + \sum_{x' \in \mathcal{X}, x \neq x'} Q(x' \mid y)^2$

Logarithmic loss $L(x, Q(\cdot \mid y)) = -\log Q(x \mid y)$

Some loss functions L satisfy specific conditions that allow us to assert certain guarantees on how optimal strategies would look like. Definitions of the important conditions for L are given below.

Symmetry Let $Q^{x_1 \leftrightarrow x_2}$ denote an alternative distribution to Q where the probabilities assigned to x_1 and x_2 are exchanged:

$$Q^{x_1 \leftrightarrow x_2}(x \mid y) = \begin{cases} Q(x_2 \mid y), & \text{for } x = x_1 \\ Q(x_1 \mid y), & \text{for } x = x_2 \\ Q(x \mid y), & \text{for otherwise} \end{cases}$$

Loss function $L(x, Q(\cdot \mid y))$ is *symmetric* between x_1 and x_2 if we have $L(x_1, Q(\cdot \mid y)) = L(x_2, Q^{x_1 \leftrightarrow x_2}(\cdot \mid y))$, for any $Q(\cdot \mid y) \in \Delta_{\mathcal{X}}$.

In other words, exchanging probabilities assigned for x_1 and x_2 does not affect the loss function. A loss function is *symmetric with respect to exchanges in* $\mathcal{Y}$ if L is symmetric between any pair of outcomes $x_1, x_2 \in \mathcal{X}$ such that messages $y_1, y_2 \in \mathcal{Y}$ exist with $y_1 \setminus y_2 = \{x_1\}$ and $y_2 \setminus y_1 = \{x_2\}$. This property is important, as can be seen in section 3.2.3. Randomised 0-1 loss, Brier loss and logarithmic loss are all symmetric.

Proper If loss function L is *proper*, we have

$$P(\cdot \mid y) \in \arg\min_{Q \in \Delta_{\mathcal{X}}} \sum_{x \in \mathcal{X}} P(\cdot \mid y) L(x, Q(\cdot \mid y)) \text{ for all } P(\cdot \mid y) \in \Delta_{\mathcal{X}}$$

In other words, a proper loss function L is truly minimised when prediction $Q(\cdot \mid y) = P(\cdot \mid y)$. Randomised 0-1 loss is *not* proper, while Brier and logarithmic loss are.

Local Let Q^{x_1} denote any alternative distribution to Q for which we have $Q(x \mid y) = Q^{x_1}(x \mid y)$ for *only* x_1, and $Q(x \mid y) \neq Q^{x_1}(x \mid y)$ for any other x.

If loss function L is *local*, we have $L(x_1, Q(\cdot \mid y)) = L(x_1, Q^{x_1}(\cdot \mid y))$. That is, the loss depends on the probability assigned by the prediction to the outcome x, but not on the probabilities assigned to outcomes that do not occur. Logarithmic loss is an example of a local loss function. In fact, it is the only loss function known to be both *proper* and *local*.

3.2 Zero-sum probability updating games

Tackling the question of how to update probabilities correctly for non-CAR coarse data, Van Ommen et al. (2016) addressed a generalisation of the Monty Hall problem and calls them *probability updating games*, therefore providing a game-theoretic viewpoint towards the problem.

Van Ommen et al. argued that the 'sender' of y could be a rational agent or just some unknown process. Either way, the decision maker does not know the (probabilistic) coarsening procedure that selects message y. However, if we do not want to make any assumptions about the coarsening procedure, then choosing the *worst-case* (or minimax) optimal prediction guarantees that we incur at most some fixed expected loss, while any other prediction may lead to a larger expected loss depending on the coarsening procedure. Thus, it is best to assume that the sender is a rational agent that makes it as hard as possible for the decision maker to make a correct prediction.

We consider a zero-sum game that is played between the host and the contestant, named for their roles in the Monty Hall problem. Similar to the Monty Hall problem, the decision maker is equivalent to the contestant as the sender is to the host. As is common in game-theoretic literature, we call P and Q the *strategies* of the corresponding players.

Let probability updating game $\mathcal{G} = \langle \mathcal{X}, \mathcal{Y}, p, L \rangle$ denote a quadruple, where $\mathcal{X}$ denotes an outcome space, $\mathcal{Y}$ a message space, p a marginal distribution on $\mathcal{X}$ and L a loss function as defined above. Both the host and the contestant know p. They also simultaneously announce their strategies, so there is no knowledge of the other player. Prior to choosing strategies, we can determine the contestant's expected loss:

$$\sum_{y \in \mathcal{Y}} \sum_{x \in y} P(x, y) L(x, Q(\cdot \mid y))$$

We allow L to take the value ∞. If this value occurs with positive probability, then the expected loss is infinite. However, if $L = \infty$ occurs with zero probability, we define $0 \cdot \infty = 0$

3.2.1 A minimax theorem

Ideally, the contestant aims to *minimise* the expected loss, while the host aims to *maximise* the term. But both players don't know their opponent's strategy. However, the contestant *can* set an upper bound to the expected loss, by imagining a worst-case scenario. It assumes the host to be an omniscient being. In this case, the host picks strategy P such that it maximises the expected loss:

$$\max_{P \in \mathcal{P}} \sum_{y \in \mathcal{Y}} \sum_{x \in y} P(x, y) L(x, Q(\cdot \mid y))$$

Strategy Q^* is called worst-case optimal if it *minimises* the worst-case expected loss.

And vice versa, the host wants to set a lower bound on the expected loss. It can do that by assuming the contestant to be omniscient. In this case, the contestant picks strategy Q such that for each $y \in \mathcal{Y}$, it minimises the expected loss for predicting x, given y:

$$H_L(P(\cdot \mid y)) = \inf_{Q(\cdot \mid y) \in \Delta_{\mathcal{X}}} \sum_{x \in y} P(x) L(x, Q(\cdot \mid y))$$

This term is equivalent to the *generalised entropy* for loss function L (Grünwald and Dawid (2004)). For logarithmic loss, this term reduces to the Shannon entropy. Averaging over every $y \in \mathcal{Y}$, we

get the expected generalised entropy:

$$\sum_{y \in \mathcal{Y}} P(y) H_L(P(\cdot \mid y))$$

Strategy P^* is called worst-case optimal if it *maximises* the expected generalised entropy.

Van Ommen et al. proved that for many loss functions, among which logarithmic loss, the minimum worst-case expected loss *equals* the maximum expected generalised entropy. This constitutes a *minimax theorem* (v. Neumann (1928)) for zero-sum probability updating games. It implies that *any* pair of worst-case optimal strategies (P^*, Q^*) are a Nash equilibrium (Nash (1951)).

Consequently, they seek to find the worst-case optimal P^* for the host, for it is shown that the conditions that characterise P^* are easier to satisfy than for Q^*. In addition, they show that for games where a Nash equilibrium exists, worst-case optimal Q^* can be derived easily once we know worst-case optimal P^*.

3.2.2 Reverse coarsening at random

To update probabilities in a *robust* manner (i.e. minimising loss for a worst-case scenario), Van Ommen et al. (2016) aimed to find worst-case optimal strategies P^*. However, previously it seemed unclear what such strategies should look like. They proposed a condition that *characterises* worst-case optimal strategies, for some zero-sum probability updating games.

Definition 2 (RCAR)**.** *Suppose subset $\mathcal{Y}^+ \subseteq \mathcal{Y}$, where each $y \in \mathcal{Y}^+$ has positive probability $P(y) > 0$. For all $y, y' \in \mathcal{Y}^+$, $x \in \mathcal{X}$ with $x \in y, x \in y'$, we have*

$$P(x \mid y) = P(x \mid y')$$

And for all $y \in \mathcal{Y}$

$$\sum_{x \in y} P(x \mid y) \leq 1$$

When P satisfies the above two conditions, we say that strategy P is *reverse coarsened at random*, or RCAR. The probabilities in the equality from the first condition are reversed to the CAR condition, hence the name.

3.2.3 Worst-case optimality

Proper, local loss Van Ommen et al. have shown that for *any* probability updating game, if strategy P^* satisfies the RCAR condition, then it is worst-case optimal for all L *proper* and *local*.

Among all loss functions that are 'smooth' for all $x \in \mathcal{X}$, logarithmic loss is the only proper, local loss function (Bernardo (1979)). Logarithmic loss is frequently mentioned in the literature regarding coarse data problems and the RCAR condition proves its usefulness for such games. Though, when modelling a probability updating game, we should also consider other classes of games, as logarithmic loss might not be the most appropriate loss function for the problem at hand.

Additionally, Van Ommen et al. found other games in which, independent on L, RCAR characterises worst-case optimality: graph and matroid games.

Graph games When looking at probability updating games, we may view $\mathcal{X}$ and $\mathcal{Y}$ as a *hypergraph*. A hypergraph $G = (\mathcal{X}, \mathcal{Y})$, consists of a set of vertices $\mathcal{X}$ and a set of edges $\mathcal{Y}$. Unlike a graph, in a hypergraph each edge $y \in \mathcal{Y}$ may contain an arbitrary amount of vertices.

A *graph game* is defined as a probability updating game where each message contains at most two outcomes. After removing singleton messages, which are either dominated or are decomposable from the rest of the game (see Van Ommen et al. (2016)), we have $|y| = 2$ for all $y \in \mathcal{Y}$. Van Ommen et al. found that for any graph game where L is symmetric with respect to exchanges in $\mathcal{Y}$, RCAR characterises worst-case optimal strategies.

Matroid games Another way of classifying probability updating games is when looking for *matroid* structures in the outcomes and messages. A probability updating game with outcome space $\mathcal{X}$ and message structure $\mathcal{Y}$ is a *matroid game* if we have that $\mathcal{X}$ and $\mathcal{Y}$ form a matroid. We define a matroid $M = (\mathcal{X}, \mathcal{Y})$ with $\mathcal{X}$ a finite set and $\mathcal{Y}$ a nonempty family of subsets of $\mathcal{X}$. Furthermore, $\mathcal{Y}$ the set of bases of the matroid: for all $y_1, y_2 \in \mathcal{Y}$ and $x_1 \in y_1 \setminus y_2$, we have

$$(y_1 \setminus \{x_1\}) \cup \{x_2\} \in \mathcal{Y} \text{ for some } x_2 \in y_2 \setminus y_1$$

In other words, for any pair of messages (y_1, y_2) and for any outcome from y_1 that is not in y_2, if we remove that outcome from y_1, there exists an outcome from y_2 that is not in y_1, such that if we add it to the result, we obtain a different but existing message in $\mathcal{Y}$.

Similar to the graph games, Van Ommen et al. found that for any matroid game where L is symmetric with respect to exchanges in $\mathcal{Y}$, RCAR characterises worst-case optimal strategies.

3.2.4 Robust probability updating

In summary, Van Ommen et al. have shown that RCAR characterises worst-case optimal strategies for zero-sum probability updating games with

- A proper, local loss function L (3.2.3)
- A *graph* message structure $\mathcal{Y}$ (3.2.3)
- A *matroid* message structure $\mathcal{Y}$ (3.2.3)

It should be noted that for the graph and matroid classes, a symmetric L is required. However, symmetry is a weak condition in which most loss functions used should satisfy.

Furthermore, Van Ommen et al. (2016) have shown that in games for which a worst-case optimal strategy P^* exists, worst-case optimal strategy Q^* can be derived easily.

This means that for any graph and matroid games, we only have to find an RCAR host strategy P^* to robustly update our probabilities to a worst-case optimal prediction. There appears to be an invariance of the loss function being used. For these games, we can think of determining worst-case optimal strategies as a generalisation of standard conditioning, namely, robust probability updating. The difference is that worst-case optimality regards message structures other than partitions (where CAR is guaranteed to hold). In standard conditioning, we would only need message y to determine $P(x \mid y)$. In robust probability updating, we need the whole message structure $\mathcal{Y}$.

3.3 Traditional game theory

When investigating Nash equilibria for games, we need some essential game-theoretic definition and terminology. Suppose a strategic simultaneous n-player game $\mathcal{G} = \langle N, \mathbf{A}, \mathbf{u} \rangle$, where

- $N = \{1, \ldots, n\}$ is a finite set of players;
- $\mathbf{A} = A_1 \times \cdots \times A_n$ is a (infinite) set of *action profiles* (i.e. vectors $\mathbf{a} = (a_1, \ldots, a_n)$). A_i here denotes the set of actions available to player i;
- $\mathbf{u} = (u_1, \ldots, u_n)$ is a profile of *payoff functions*: $u_i : \mathbf{A} \mapsto \mathbb{R}$.

We categorise two types of strategies for the players. Player $i \in N$ might choose action $a_i \in A_i$, which is called a *pure* strategy. i might also play according to a probability distribution on A_i, in which case it plays a *mixed* strategy. Let strategy $s_i \in S_i$ be such a probability distribution on A_i. The set of all strategy profiles is accordingly denoted by $\mathbf{S} = S_1 \times \cdots \times S_n$, where S_i for player i is the set of probability distributions on A_i. Note that when s_i puts all mass on a single action a_i, then we call it a *pure* strategy. Let $\mathbf{s} = (s_1, \ldots, s_n)$ denote the *strategy profile* of all players. Let $\mathbf{s}_{-i}$ denote the *counter profile* of player i, i.e. the strategies played by the opponents of i.

Given counter profile $\mathbf{s}_{-i}$, strategy s_i of player i is a *best response* to $\mathbf{s}_{-i}$ if

$$u_i(s_i, \mathbf{s}_{-i}) \geq u_i(s_i', \mathbf{s}_{-i}) \text{ for all } s_i' \in S_i$$

There may be more than one best response. Let $B(\mathbf{s}_{-i})$ denote the set of the player's best responses against opponent strategy $\mathbf{s}_{-i}$.

Definition 3 (Nash equilibrium)**.** *Strategy profile* $\mathbf{s}$ *forms a Nash equilibrium if all strategies in it are best responses, i.e.* $s_i \in B(\mathbf{s}_{-i})$ *for all* i*. It is a pure Nash equilibrium if each* $s_i \in \mathbf{s}$ *is a pure strategy. Otherwise, it is a mixed Nash equilibrium.*

The Nash equilibrium is originally characterised by Nash (1951). This concept and its numerous refinements are the primary solution concepts for competitive games.

3.4 Repeated games

In this section, we review the literature on repeated games and discuss the theorems related to Nash equilibria in these games. While reviewing this literature, we managed to answer a research question regarding the characterisation of Nash equilibria for zero-sum probability updating games. As a consequence, we can dismiss further experimental investigation on repeated games.

3.4.1 Introduction to repeated games

Much of the following theory has been gathered from Muthoo et al. (1996) chapter 8. This chapter comprises most of the fundamental theory regarding repeated games. Let $\mathcal{G} = \langle N, \mathbf{A}, \mathbf{u} \rangle$ be a game as defined before. This game is also called a one-shot game, stage game or base game; it is played only once and there is no future or past knowledge. But in reality, much of the interactions that people, organisations or other entities have are more similar to long-term games. This is conflicting with the notion of a one-shot game, where each player is myopic. Repeated games capture the concept that players consider their opponents' behaviour as well as their future behaviour. Nash equilibria for repeated games are characterised by certain theorems known as *folk theorems*, which we will give describe below. It should be emphasised that these folk theorems are slightly different for *finitely* and *infinitely* repeated games.

Suppose a T-period repeated game $\mathcal{G}_T = \langle N, \mathbf{A}, \mathbf{u}, \mathcal{H} \rangle$ based on one-shot game $\mathcal{G}$. For infinitely repeated games, $T = \infty$. Hence, we obtain infinite sequences. $\mathcal{H}$ denotes the *history* of the game. Let $\mathcal{H}^t \in \mathcal{H}$ be the history until round $t \in T$, i.e. the sequence of action profiles $\mathbf{a}$ that are played, until round t. As an example, we have

$$\mathcal{H}^0 = \emptyset$$
$$\mathcal{H}^1 = \mathbf{A}$$
$$\mathcal{H}^t = \mathcal{H}^{t-1} \times \mathbf{A}$$

The behaviour of players in (infinitely) repeated games can be represented by a *Moore machine*, which is a simplified finite-state machine in which behaviour is only dependent on the current state. It is described in Moore (2016). This facilitates the structure of equilibria that can be found. For now, the details will be omitted.

As described in Muthoo et al. (1996), a massive amount of Nash equilibria for repeated games are shown and characterised by folk theorems. Remarkably, the origin of folk theorems is not known. The first folk theorem appeared in the fifties. Afterwards, this knowledge spread quickly throughout the scientific community, even though no one had published it. Hence the name; 'folk' theorem. Eventually, Friedman (1971) proved characterisations of equilibria for repeated games. It became the first published paper to do so.

From now, we will restrict attention to games in which all action sets $A_i \in \mathbf{A}$ are compact and all payoff function $u_i \in \mathbf{u}$ are continuous. Indeed, this entails that the theory below applies for repeated probability updating games with continuous L (e.g. randomised 0-1 loss, Brier loss, logarithmic loss).

3.4.2 Feasible and enforceable strategy profiles

Let $\mathbf{v} \in \mathbb{R}^N$ denote the *payoff profile* that is associated with an existing action profile $\mathbf{a}$, i.e. $\mathbf{v} = \mathbf{u}(\mathbf{a})$. $\mathbf{v}$ is a *feasible* payoff profile if it is a convex combination of possible payoff profiles in one-shot game $\mathcal{G}$; that is, if

$$\mathbf{v} = \sum_{\mathbf{a} \in \mathbf{A}} \alpha_{\mathbf{a}} \mathbf{u}(\mathbf{a})$$

where the coefficients $\alpha_{\mathbf{a}}$ satisfy $\alpha_{\mathbf{a}} \geq 0$ for all $\mathbf{a} \in \mathbf{A}$ and $\sum_{\mathbf{a} \in \mathbf{A}} \alpha_{\mathbf{a}} = 1$. From another but equivalent perspective, the coefficients are a distribution on the actions. The rationale behind a payoff profile being *feasible* is that the payoff in an (infinitely) repeated game has to be attainable in some way. It must be formed from a (weighted) combination of payoffs in one-shot game $\mathcal{G}$.

Suppose a *minimax* action profile $\hat{\mathbf{a}}$:

$$u_i(\hat{\mathbf{a}}) = \min_{\mathbf{a}_{-i} \in \mathbf{A}_{-i}} \max_{a_i \in A_i} u_i(a_i, \mathbf{a}_{-i}) \text{ for each player } i$$

That is, all players play a worst-case optimal strategy. Moreover, let $\hat{\mathbf{v}}$ denote the minimax payoff profile $\hat{\mathbf{v}} = \mathbf{u}(\hat{\mathbf{a}})$. We call $\mathbf{a}$ an *enforceable* action profile if $u_i(\mathbf{a}) \geq u_i(\hat{\mathbf{a}})$ for all i. It is *strictly enforceable* if $u_i(\mathbf{a}) > u_i(\hat{\mathbf{a}})$ for all i. Equivalently, $\mathbf{v}$ is an enforceable payoff profile if $v_i \geq \hat{v}_i$ for all i, and is strictly enforceable if $v_i > \hat{v}_i$ for all i.

3.4.3 Folk theorems: characterising Nash equilibria

Practically, the folk theorems tell us that any *feasible*, *enforceable* payoff profile for one-shot game $\mathcal{G}$ is a Nash equilibrium payoff profile for $\mathcal{G}_T$. This implies that the total set of Nash equilibria

for repeated games generally is very large. For the following results, suppose one-shot game $\mathcal{G}$ and its T-period repeated game $\mathcal{G}_T$.

We restrict attention towards games where T is finite. In finitely repeated games, the calculation of the players utility is the most straightforward. Let $\mathbf{h_t} \in \mathcal{H}^\infty$ denote the played action profiles at round t. The collective utility of player i is given by

$$U_i = \frac{1}{T} \sum_{t=0}^{T} u_i(\mathbf{h_t})$$

The propositions are due to Benoit and Krishna (1987).

Proposition 3.1. *If the payoff profile in every Nash equilibrium of $\mathcal{G}$ is equal to the minimax payoff profile $\hat{\mathbf{v}}$, then for all Nash equilibria $(\mathbf{a}^1, \ldots, \mathbf{a}^T)$ of $\mathcal{G}_T$, every action profile $\mathbf{a}^t \in (\mathbf{a}^1, \ldots, \mathbf{a}^T)$ is a Nash equilibrium of $\mathcal{G}$, for all $t = 1, \ldots, T$.*

If the one-shot game has a Nash equilibrium that is a minimax action profile, then any Nash equilibrium for the repeated game is just a repetition of these one-shot game Nash equilibria. This implies that for zero-sum probability updating games where RCAR characterises worst-case optimal strategies, RCAR strategies will also form Nash equilibria for such finitely repeated probability updating games.

Proposition 3.2. *(Nash folk theorem for finitely repeated games) If $\mathcal{G}$ has a Nash equilibrium $\mathbf{a}$ that exceeds the minimax payoff profile (i.e. $u_i(\mathbf{a}) > \hat{v}_i$, for all i) then for any strictly enforceable action profile $\mathbf{a}^*$ there is a Nash equilibrium in $\mathcal{G}_{T \geq T^*}$ in which the payoff is ϵ-close to $u_i(\mathbf{a}^*)$ for all i, for any $\epsilon > 0$ and some minimum T^*.*

If the one-shot game has a strictly enforceable Nash equilibrium, then for any strictly enforceable action profile in the finitely repeated game can be approximated, there is a Nash equilibrium that approximates the payoff of that action profile, given enough rounds of play. Note that this will apply for non-zero-sum probability updating games.

For infinitely repeated games, similar patterns arise. It should be emphasised that the literature shows several criteria that describe how players' preference relation may be modelled in infinitely repeated games. Two key criteria are called *limit of means* and *discounted* infinitely repeated games.

We start with the limit of means infinitely repeated game. Let $\mathbf{h_t} \in \mathcal{H}^\infty$ denote the played action profiles at round t again. The collective utility of player i is given by

$$U_i = \lim_{T \to \infty} \inf \frac{1}{T} \sum_{t=0}^{T} u_i(\mathbf{h_t})$$

Muthoo et al. (1996) showed that for the limit of means infinitely repeated game $\mathcal{G}_\infty$, every Nash equilibrium payoff profile is an enforceable payoff profile of one-shot game $\mathcal{G}$. Furthermore, Aumann and Shapley (2013) proved the following proposition for the limit of means criterion.

Proposition 3.3. *(Nash folk theorem for the limit of means criterion) Every feasible, enforceable payoff profile of one-shot game $\mathcal{G}$ is a Nash equilibrium payoff profile of the limit of means infinitely repeated game $\mathcal{G}_\infty$.*

This proposition creates an even stronger argument for the existence of a large class of Nash equilibria for repeated games. As long as a given payoff profile is feasible and enforceable, which are generally easy restrictions to satisfy, the resulting payoff is equal to that of a Nash equilibrium in the repeated game.

In contrast to limit of means games, for discounted games hold that players do not know whether the next round will be the last or not. Discount factor $\delta \in (0, 1)$ indicates how patient they are; a low δ means that players are more myopic, whereas with a high δ players are more farsighted. The utility of this infinitely repeated game is given by

$$U_i = (1 - \delta) \sum_{t \geq 0} \delta^t u_i(\mathbf{h_t})$$

Muthoo et al. (1996) showed that for any $\delta \in (0, 1)$ of the δ-discounted infinitely repeated game $\mathcal{G}_\infty$, every Nash equilibrium payoff profile is an enforceable payoff profile of one-shot game $\mathcal{G}$. Furthermore, Fudenberg and Maskin (1986) proved the following proposition for δ-discounted games:

Proposition 3.4. *(Nash folk theorem for the discounting criterion) Every feasible strictly enforceable payoff profile of $\mathcal{G}$ is ϵ-close to a Nash equilibrium payoff profile of the discounting infinitely repeated game for any $\delta > \delta_t$ and any ϵ.*

Or as long as δ is high enough, any feasible strictly enforceable payoff profile will be very close to that of a Nash equilibrium in the repeated game. This proposition suggests that even for discounted infinitely repeated games, there probably are a lot of Nash equilibria.

3.4.4 Discussion

The most significant finding from the literature on repeated games is that for finitely repeated zero-sum games, for any Nash equilibrium in the one-shot game, the Nash equilibrium in the corresponding repeated game is just the one-shot Nash equilibrium, repeated T times. We can therefore conclude that RCAR strategies will also characterise Nash equilibria in repeated (zero-sum) probability updating games. However, we cannot make the same conclusions for infinitely repeated games. Also, it is not known how Nash equilibria would arise when relaxing *both* the zero-sum and the one-shot constraint of the probability updating game. In section 6, we discuss this finding and its potential limitations. We also outline future research regarding repeated probability updating games.

3.5 Computing Nash equilibria

Prior to finding Nash equilibria, one should ask whether it might be possible to find an exact solution analytically. Computationally speaking, the simplest class of games are two-player zero-sum games. Finding a Nash equilibrium for two-player zero-sum games is in complexity class **P**; v. Neumann (1928) proposed the minimax theorem for two-player zero-sum games that leads to a linear programming problem. Subsequently, algorithms like the simplex method or the ellipsoid method can find Nash equilibria for such games and have a worst-case and smoothed polynomial running time, respectively. For probability updating games specifically, Van Ommen (2015) proposed polynomial-time algorithms for finding worst-case optimal strategies for *graph* and *matroid* games.

Non-zero-sum games are more difficult in that aspect. There may exist no polynomial-time algorithm for this case. Chen et al. (2009) proved that computing a Nash equilibrium for *general-sum n*-player games is at least **PPAD**-complete, where **PPAD**, or polynomial parity arguments on directed graphs, is a class of computational search problems. In short, finding Nash equilibrium strategies for *general-sum* games is *intractable*, as no polynomial-time method seems to exists that can solve such games.

It is generally believed that **PPAD** is not contained in P, and therefore plausibly harder. **PPAD** contains many problems for which there have been thorough attempts at developing fast algorithms. On the other hand, there is still little proof of the actual hardness of **PPAD** (Johnson (2007)).

In summary, finding possible Nash equilibria or other equilibria requires an empirical approach. For this, we use multi-agent reinforcement learning (MARL) methods in which both agents optimise their strategies iteratively until convergence emerges. We consequently aim to find patterns in the resulting equilibria. This is further elaborated in section 4.1.

4 Methodology

This chapter comprises the constituents of our approach and the significant decisions that were made for the empirical investigation. We discuss reinforcement learning algorithms, reinforcement learning frameworks and important implementation details.

4.1 MARL: multi-agent reinforcement learning

Below, we explain how we use multi-agent reinforcement learning (MARL) methods to optimise player strategies for probability updating games. We review essential theory regarding these learning methods. The theory that we cover on general reinforcement learning (RL) and policy gradient methods is due to Sutton and Barto (2018).

4.1.1 Classical multi-agent learning

Multi-agent learning (MAL) lies at the intersection between multi-agent systems and machine learning. It differs from single-agent (reinforcement) learning, as the learning is done in the presence of other agents that learn. This learning is done in a multi-agent environment in which an agent can affect the outcome of any other agent. Established MAL methods or families of methods are among the following:

- Q-learning
- Fictitious play
- No-regret learning
- Gradient dynamics
- Evolutionary learning

Generally, the goal is to converge to some equilibrium. Some methods reach a *correlated equilibrium*, which is related to the Nash equilibrium but less restrictive. For zero-sum games, many MAL methods provide theoretical guarantees on the convergence to Nash (or related) equilibria. However, for general-sum games, theoretical guarantees are usually less understood or limited. For example, Hu and Wellman (2003) proved that for general-sum games, Q-learning can reach a Nash equilibrium if and only if three quite strict conditions in the game are met. A better faring method is Infinitesimal Gradient Ascent (IGA), in the family of gradient dynamics. Singh et al. (2000) proved that IGA always converges to the expected payoff of some Nash equilibrium, which is, depending on the goal, almost as good as reaching an actual Nash equilibrium.

Nevertheless, one major problem that arises with these traditional MAL methods is that the vast majority of them are designed for games with a discrete action set. This is an issue because probability updating game action sets are continuous. The host and the contestant both pick a probability distribution as their pure strategy. It should be noted that their action sets are also *compact*, as both players act on closed intervals $[0, 1]$ to properly represent distributions P and Q. The literature on learning methods for games with continuous action spaces is lacking. In contrast, learning methods for discrete action sets are considerably more common. Still, we need to apply a learning algorithm that works with probability updating games *as is*; transforming

the structure of the game is not desirable, because that may come with complications further down the line.

In contrast to multi-agent learning, *single-agent* learning brings a lot more attention towards environments with continuous action (and state) spaces. Below, we briefly illuminate the area, and we show how we use it to find equilibria for relaxed probability updating games.

4.1.2 Single-agent reinforcement learning

Single-agent RL (or just reinforcement learning) received a lot of attention in recent years. State-of-the-art deep learning methods proved to easily outperform many other methods and surpass humans in complex games. For instance, Silver et al. (2017) introduced AlphaGo Zero, showing superiority in the complex game Go against previous methods through self-play, without human guidance or domain knowledge.

Generally, RL methods try to solve Markov Decision Processes (MDPs). A (discounted) MDP is defined as $M = \langle \mathcal{S}, \mathcal{A}, P, \mathcal{R}, \gamma \rangle$, with

- $\mathcal{S}$ a set of states
- $\mathcal{A}$ a set of actions
- P the state transition probability, where $P(s' \mid s, a)$ is the probability of transitioning into state s' from s by action a. All state transitions satisfy the *Markov property*, which means that given s and a, $P(s' \mid s, a)$ is conditionally independent of all previous states and actions.
- R the reward function, where $R(s, a, s')$ gives the expected received reward after transitioning from s to s' by action a. Many RL methods do not know the transition probabilities beforehand, and learn them iteratively.
- $\gamma \in [0, 1]$ the discounting factor. The uncertainty of state transitions (and therefore, expected rewards) causes a larger uncertainty in later rewards. γ serves as a penalty to the uncertainty of future rewards.

The problem statement is as follows: without knowledge of the model (i.e. without knowing transition function P or reward function R), *maximise* the total rewards. We do this by learning the optimal *policy* π, determining what action a to take when in state s. The policy can be either deterministic ($\pi : \mathcal{S} \mapsto \mathcal{A}$) or stochastic ($\pi : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$).

We define the *return* G_t as the total sum of future rewards from time step t towards $t \to \infty$. This future reward is discounted using γ. Or formally:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

To learn means to predict future rewards. In order to predict future rewards, one has to evaluate past experiences. To evaluate past experiences, it is necessary to determine the quality of certain states and state-action pairs. We can measure them in terms of expected future return. We have *state value function* $V^\pi(s)$ for state s and *action value function* $Q^\pi(s, a)$ for state-action pair (s, a):

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi[G_t \mid S_t = s] \\ Q^\pi(s, a) &= \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] \end{aligned}$$

In addition, the state value $V_\pi(s)$ of state s can be derived by

$$V^\pi(s) = \sum_{a \in \mathcal{A}} Q^\pi(s, a)\pi(a \mid s)$$

Finally, $A^\pi(s, a)$ denotes the *advantage function* and is defined as:

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$$

It is considered an alternative to $Q^\pi(s, a)$ as estimates $A^\pi(s, a)$ have lower variance as a result of using $V^\pi(s)$ as a baseline.

Monte Carlo methods and Temporal-Difference (TD) learning (Sarsa, Q-learning) are examples of techniques that estimate these value functions and optimise the agent's policy π accordingly. These methods and their extensions have proved to be useful for varying RL problems. But, many of these techniques stumble upon the *curse of dimensionality* when the problem contains a very large or continuous state and action space. As a consequence, it would make them a poor method of choice for learning probability updating game strategies.

Alternatively, *policy gradient methods* do allow for continuous control. Instead of learning $V^\pi(s)$ and $Q^\pi(s)$, policy gradient methods learn a *parameterised policy* directly. Below, we give a brief overview of policy gradient methods up until the method of use: Proximal Policy Optimisation (PPO). We also discuss alternative methods that we inspected, but eventually did not select.

Additionally, a single-agent approach gives rise to a new problem; applying a single-agent algorithm to a multi-agent game. We implemented a fully decentralised learning approach, which allows us to learn many policies $\boldsymbol{\pi} = \{\pi_1, \ldots, \pi_N\}$ without any sharing of information between the learning agents. See section 4.1.8.

4.1.3 Policy Gradient methods

Policy gradient methods try to learn the agent's policy directly. They are particularly interesting for robotics applications, as they often require continuous control of sensors and actuators.

We parameterise the policy with policy parameter vector $\boldsymbol{\theta}$. Let $\pi_{\boldsymbol{\theta}}$ denote a policy parameterised by $\boldsymbol{\theta}$. Suppose a stochastic policy, then we denote $\pi_{\boldsymbol{\theta}}(a \mid s) = P(A_t = a \mid S_t = s, \boldsymbol{\Theta}_t = \boldsymbol{\theta})$ as the probability that action a is selected when in state s at time t given parameter $\boldsymbol{\theta}$. Let $\mathcal{J}(\boldsymbol{\theta})$ be the performance measure of parameter vector $\boldsymbol{\theta}$. For our case of interest, namely the episodic case, the performance measure is defined as

$$\mathcal{J}(\boldsymbol{\theta}) = V^{\pi_\theta}(S_0)$$

where S_0 is the initial state and V^{π_θ} defines the true state value function for $\pi_{\boldsymbol{\theta}}$.

In every policy gradient method, we attempt to maximise $\mathcal{J}(\boldsymbol{\theta})$ w.r.t. policy parameter $\boldsymbol{\theta}$. This leads to a stochastic gradient ascent algorithm such as:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla \mathcal{J}(\boldsymbol{\theta}_t)$$

Computing $\nabla V^{\pi_\theta}(S_t)$ is however infeasible, as it depends on the distribution of states in which the policy is applied, which is unknown. Nevertheless, the *policy gradient theorem* shows that there are other ways to maximise $\mathcal{J}(\boldsymbol{\theta})$. It states the following:

$$\nabla \mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}_{s,a \sim \pi_{\boldsymbol{\theta}}} \left[Q^{\pi_\theta}(s, a) \nabla \ln \pi_{\boldsymbol{\theta}}(a \mid s) \right]$$

where $Q^{\pi_\theta}(s, a)$ defines the true state-action value function for $\pi_{\boldsymbol{\theta}}$. This expression does *not* involve the derivative of the state distribution, which allows us to apply gradient descent methods. Therefore, the policy gradient theorem lays the foundation on which any policy gradient algorithm is built. One instance of such an algorithm is the REINFORCE algorithm by Williams (1992). It implements the following policy gradient:

$$\nabla \mathcal{J}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[\nabla \ln \pi_{\boldsymbol{\theta}}(a \mid s) G_t \right]$$

where G_t is the estimated return, i.e. the total sum of future rewards as seen in section 4.1.2.

As mentioned before, policy gradients can easily be extended to deal with continuous action spaces. For instance, suppose action space $\mathcal{A} \subseteq \mathbb{R}$. Then we let $\boldsymbol{\theta}$ parameterise a continuous distribution from which we can subsequently sample actions. As an example, suppose we divide $\boldsymbol{\theta}$ into $\boldsymbol{\theta} = [\boldsymbol{\theta}_\mu, \boldsymbol{\theta}_\sigma]^\top$. Assume we use the normal distribution. We parameterise it with mean $\mu(s, \boldsymbol{\theta}_{\boldsymbol{\mu}})$ and standard deviation $\sigma(s, \boldsymbol{\theta}_{\boldsymbol{\sigma}})$ for each state s. Our resulting policy would look like the following:

$$\pi_{\boldsymbol{\theta}}(\alpha \mid s) = \frac{1}{\sigma(s, \boldsymbol{\theta}_{\boldsymbol{\sigma}})\sqrt{2\pi}} \exp\left(-\frac{(a - \mu(s, \boldsymbol{\theta}_{\boldsymbol{\mu}}))^2}{2\sigma(s, \boldsymbol{\theta}_{\boldsymbol{\sigma}})^2} \right)$$

4.1.4 Actor-critic methods

Many RL methods can be differentiated by value-based methods (e.g. Q-learning, Sarsa) or policy-based methods (e.g. REINFORCE). At the intersection of the two lie *actor-critic* methods.

Actor-only methods like REINFORCE produce gradient estimates with high variance: this significantly hinders the learning efficiency. By adding a critic that can assist the actor in estimating the policy, we can reduce the gradient variance. An actor-critic algorithm consists of the following two components:

- the *actor*, which estimates policy $\pi_{\boldsymbol{\theta}}$ by updating parameter vector $\boldsymbol{\theta}$ in the direction suggested by the critic;
- the *critic*, which estimates a value function w.r.t. parameter vector $\mathbf{w}$. This is estimated through MC or TD learning.

For instance, we can have an actor-critic method that learns policy $\pi_{\boldsymbol{\theta}}$ as well as state-action value function $Q_{\mathbf{w}}(s, a)$. Suppose we approximate $Q_{\mathbf{w}}$ by a basic TD learning method. Now, we get the following policy gradient:

$$\nabla \mathcal{J}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[\nabla \ln \pi_{\boldsymbol{\theta}}(a \mid s) Q_{\mathbf{w}}(s, a) \right]$$

After the actor selects action a_t, the critic evaluates new state s_{t+1} and it determines the *TD-error*, i.e. the difference between the estimated reward and the actual reward. It is computed as:

$$\delta_t = R_t + \gamma Q_{\mathbf{w}_t}(s_{t+1}, a_{t+1}) - Q_{\mathbf{w}_t}(s_t, a_t)$$

where R_t is the received reward in time step t and $Q_{\mathbf{w}}$ is the estimated state-action value function by the critic. δ_t tells us whether the state-action pair was better than expected or not. We use this to update $\mathbf{w}$ in the direction that *strengthens* the actions that turn out better, i.e.:

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha_{\mathbf{w}} \delta_t \nabla Q_{\mathbf{w}_t}(s_t, a_t)$$

where $\alpha_{\mathbf{w}}$ is the particular step size of the gradient step for $\mathbf{w}$.

The actor-critic algorithm above is still quite basic. In contrast, state-of-the-art algorithms possess many sophisticated modifications and tricks that improve the learning process even further. Also, any modern actor-critic method implements two neural networks to approximate the value function as well as the policy. Proximal Policy Optimisation (PPO) is among the most promising algorithms, which we have selected as the main learning method for our experimental research. We also experimented and ran the following actor-critic methods

- trust region policy optimisation (TRPO);
- deep deterministic policy gradient (DDPG);
- twin delayed deep deterministic policy gradient (TD3);
- soft actor critic (SAC);
- and advantage actor critic (A2C).

The above methods (including PPO) are all from 2015 and forth. Developments in this area are very recent and might therefore drastically change in even a few years.

4.1.5 MADDPG: multi-agent deep deterministic policy gradient

Another notable actor-critic method is *multi-agent deep deterministic policy gradient* (MADDPG). Lowe et al. (2017) introduced this actor-critic method as a multi-agent extension of DDPG. A primary problem in MARL is that the environment becomes non-stationary: each agent's policy continually changes as time passes. This could drastically reduce the stability of learning and significantly increase variance for policy gradient methods.

MADDPG adopts the notion of a *centralised critic*, allowing the critic to observe the policies of all agents. In practice, this implies that we estimate a centralised state-action value function $Q_i^{\pi}(\mathbf{x}, a_1, \ldots, a_N)$ that takes some state information (like agent observations) $\mathbf{x}$ and agent actions $a_1, \ldots, a_N$ and outputs the state-action value for agent i. The *execution*, or action selection, is done in a decentralised manner. Further details are described by Lowe et al. (2017).

From the theory, it seems like the perfect algorithm for learning probability updating games. However, in practice it leaves a lot to be desired: the implementation by the original authors is mainly intended and tested for the *discrete* MPE environments (Mordatch and Abbeel (2018)). Although this original implementation has been adapted by the Ray project, The Ray Team (2021) states that it remains difficult to effectively run it on standard multi-agent environments. They advise not to use it to learn in a multi-agent environment, but instead to use it for research and reproducing purposes.

4.1.6 TRPO: trust region policy optimisation

Basic policy gradient methods suffer from high variance gradient estimates, which are solved by the addition of a critic, forming actor-critic methods. Schulman et al. (2015) introduced TRPO as a solution to two other limitations of vanilla policy gradient methods. Those are:

- step sizes. It is hard to choose the right step size α for the gradient update, as the input data in environments is non-stationary as a result of changing policies. Too large step sizes can be catastrophic for future rewards, as bad policies result in bad observations and reward models;
- sample efficiency. Some dimensions in the gradient landscape might have larger scales than other dimensions, which hinders learning. Also, stochastic gradient descent performs one step per environment sample, which can be improved upon.

As a solution to these problems, Schulman et al. (2015) opted for a constrained numeric optimisation approach. In this approach, we enforce a *trust region* constraint by limiting the distance between $\pi_{\boldsymbol{\theta}_{\text{old}}}$ and $\pi_{\boldsymbol{\theta}}$. This distance is computed by the *Kullback-Leibler divergence* (or *KL-divergence*) $D_{\text{KL}}(P \mid\mid Q)$, measuring a statistical dissimilarity between two distributions P and Q. It can naturally be used to measure the dissimilarity between policy distributions $\pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot \mid s)$ and $\pi_{\boldsymbol{\theta}}(\cdot \mid s)$.

TRPO updates $\boldsymbol{\theta}$ by maximising the approximated advantage, measuring how good policy $\pi_{\boldsymbol{\theta}}$ performs relative to $\pi_{\boldsymbol{\theta}_{\text{old}}}$. Let $r(\boldsymbol{\theta})$ denote the probability ratio of taking action a given state s between the old and new policy: $r(\boldsymbol{\theta}) = \frac{\pi_{\boldsymbol{\theta}}(a|s)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(a|s)}$. The optimisation problem is then formulated as

$$\begin{aligned} \underset{\boldsymbol{\theta}}{\text{maximize}} \quad & \hat{\mathbb{E}}_t \left[r_t(\boldsymbol{\theta}) \hat{A}_t(s,a) \right] \\ \text{subject to} \quad & \hat{\mathbb{E}}_t \left[D_{\text{KL}}(\pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot \mid s) \mid\mid \pi_{\boldsymbol{\theta}}(\cdot \mid s)) \right] \leq \delta \end{aligned}$$

where $\hat{A}_t(s,a)$ is the estimated advantage function, estimated by some value function estimation method. The constraint enforces a limit on the averaged KL-divergence between the old and new policies by parameter δ.

4.1.7 PPO: proximal policy optimisation

Only two years after TRPO's emergence, Schulman et al. (2017) proposed PPO as its direct successor. It improves upon TRPO by simplifying its optimisation problem while achieving better empirical performance. In fact, it is among the most commonly used reinforcement learning algorithms at the moment. As OpenAI (2017) proclaims, it performs similarly or better than many state-of-the-art methods, while being easier to apply and tune.

TRPO's optimisation procedure suffers from the added complexity due to the trust region constraint. PPO drops the explicit constraint and instead implicitly includes the constraint in the objective function itself. If we would force $r(\boldsymbol{\theta})$ to stay within a particular region, we essentially limit the difference between $\pi_{\boldsymbol{\theta}_{\text{old}}}$ and $\pi_{\boldsymbol{\theta}}$. PPO manages to do this by clipping:

$$\mathcal{J}^{\text{CLIP}}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\boldsymbol{\theta}) \hat{A}_t(s,a), \text{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon) \hat{A}_t(s,a) \right) \right]$$

Here, $\hat{A}_t$ is again the estimated advantage function. The clip function clips the probability ratio within domain $[1-\epsilon, 1+\epsilon]$. By taking the minimum between the original policy objective and the clipped version, we enforce that

- if action a is relatively advantageous ($\hat{A}_t(s,a) > 0$), then probability $\pi_{\boldsymbol{\theta}}(a \mid s)$ increases to only a maximum of $1+\epsilon$
- if action a is less advantageous than other actions ($\hat{A}_t(s,a) < 0$), then probability $\pi_{\boldsymbol{\theta}}(a \mid s)$ decreases to only a minimum of $1-\epsilon$

In summary, $\mathcal{J}^{\text{CLIP}}(\boldsymbol{\theta})$ allows us to set a trust region while still using stochastic gradient ascent. Now, instead of computing one gradient update per environment sample, PPO takes K steps of mini-batch stochastic gradient ascent. The full approach is detailed by Schulman et al. (2017).

Schulman et al. (2017) showed experimentally that PPO mostly outperforms TRPO and A2C. Furthermore, Larsen et al. (2021) performed a rigorous continuous control benchmark between where PPO competed against DDPG, TD3 and SAC and concluded that PPO significantly

outperformed its competitors. Additionally, Henderson et al. (2018) found that PPO, as well as TRPO, show the most robustness during runs, compared to DDPG and ACKTR.

In section 5, we show that PPO has a significant advantage over other methods in probability updating games, which is why we decided to use PPO for our strategy analysis.

4.1.8 From single-agent to multi-agent

The literature mentions multiple approaches towards learning multi-agent environments using a single-agent learning method. Among the options are

- a fully centralised approach called *parameter sharing*;
- a semi-centralised approach, using a centralised critic but a decentralised actor (Lowe et al. (2017) use this for MADDPG);
- a fully decentralised approach, which we call *independent learning*.

In parameter sharing, one policy is learned using the observations from *all* agents. Formally, we define such a policy as $\pi_{\theta}(a \mid (\{o_1, \ldots, o_N\}, i))$, where i is the agent index and o_i denotes the observations of agent i. We add agent index i to the state, such that the actor can distinguish which agent the actor picks an action for. Terry, Hari, et al. (2020) proved that an optimal policy can be learned by a single policy when adding an agent index to the state in such a way.

We briefly reviewed the centralised critic approach in section 4.1.5.

The MARL field predominantly prefers parameter sharing over the other two, as parameter sharing shows superior empirical performance: Gupta et al. (2017) showed that parameter sharing outperformed the other methods for cooperative multi-agent environments. Furthermore, Terry, Hari, et al. (2020) applied both parameter sharing and independent learning to 11 single-agent RL methods and showed that parameter sharing consistently outperformed independent learning.

However, for our investigation on probability updating games, parameter sharing is not an appropriate method: with parameter sharing, we maximise the *sum* of agent rewards, rather than the rewards of individual agents. This is due to having a centralised policy. Hence, the learning performance diminishes drastically for competitive games. Indeed, learning becomes impossible for zero-sum games. Although we want to examine non-zero-sum games, we will want to investigate games that are cooperative in one aspect, but competitive in another. We, therefore, conclude that using a fully independent learning approach is superior. Additionally, independent learning is very easy to implement because we do not need to alter observation spaces or share information through network layers.

4.2 From zero-sum to general-sum

In the zero-sum case, the contestant aims to minimise L and the host aims to maximise L. Suppose that the host wants to *help* the contestant. We obtain a game where the host aims to minimise L, just like the contestant. We can generalise this further by providing the host with a standalone, independent loss function L_H. Suppose *general-sum* probability updating game $\mathcal{G} = \langle \mathcal{X}, \mathcal{Y}, p, L_H, L_C \rangle$, where $\mathcal{X}$, $\mathcal{Y}$ and p remain unaltered, and L_H and L_C denotes the loss functions for the host and the contestant, respectively. Note that when $L_H = -L_C$, the game reduces to a zero-sum probability updating game. Furthermore, if $L_H = L_C$, the game reduces to a game where the host tries to *help* the contestant.

Loss function L_C remains unaltered. It still measures the prediction quality of the contestant. L_H cannot be measuring a prediction from the host, though, as the host does not represent a decision maker that aims to predict some random variable (in the probability updating game, the *prediction* is represented by Q, whereas P merely represents a joint distribution on the outcomes and messages). Intuitively, we say that the host attempts to *steer* the contestant into a certain prediction. Depending on its loss function L_H, it either tries to worsen it, improve it or maybe it wants the contestant to desire a specific outcome. This suggests that L_H depends on contestant strategy Q. Indeed, we have $L_H : \mathcal{X} \times \boldsymbol{Q}$.

4.3 Frameworks and libraries for MARL development

The state-of-the-art deep reinforcement learning algorithms are sophisticated, complex and contain numerous small implementation-specific tricks that cause them to perform as well as they do in benchmarks. It is therefore generally a bad idea to create custom implementations if learning performance is an important factor. Instead, it would be significantly more practical to use existing, proven implementations; preferably implementations that have been benchmarked in studies.

There is a convenient way to do this. The vast majority of cutting-edge RL algorithms have been re-implemented in standardised frameworks. These implementations are heavily optimised, reviewed and benchmarked by researchers as well as enthusiasts. Hence, they form a strong foundation for advanced learning, reproducing research results and evaluation. Below, we describe essential frameworks that we used to develop a customised RL environment for probability updating games. These frameworks fall within two different categories: one that provides learning *environments* and another that provides state-of-the-art algorithms.

OpenAI Gym Brockman et al. (2016) developed OpenAI Gym, an open-source toolkit that defines a standard way to create RL *environments*. These environments represent partially observable Markov decision processes (POMDP), i.e. a generalisation of the Markov decision process (MDP) in which the agent cannot directly observe the state. All Gym environments are accessible through a standardised API. This ensures that algorithm implementations can easily be reused for different environments. Once an environment has been fully defined (i.e. the action space, observation space, state space, any end states and initialisation behaviour), any compatible algorithm that is implemented to communicate with the standard Gym API can train an agent in that environment.

Note that the probability updating game is a stochastic game, which is a multi-agent extension of the MDP. The probability updating game has only one state. After the agents have acted, the game reaches the terminal state.

PettingZoo Probability updating games are two-player games. The OpenAI Gym library is primarily for single-agent RL environments and does not support stochastic games natively. For multi-agent environments, Terry, Black, Grammel, et al. (2020) created *PettingZoo*, a MARL environment framework that follows the standardised API of OpenAI Gym very closely. We use PettingZoo along with the useful utility library called *SuperSuit* (Terry, Black, and Hari (2020)). Together with OpenAI Gym, these libraries allow us to easily define multi-agent environments that represent probability updating games and subsequently learn models in those environments.

Ray Essentially, Ray is an API for building distributed applications. On top of this API, several libraries have been developed for specific use cases. We used the following two libraries to assist our empirical investigation.

- Ray RLlib (Liang et al. (2017)): a large framework for distributed reinforcement learning;
- Tune (Liaw et al. (2018)): a standardised research platform for model selection and training.

Below, we further describe how we used the above libraries.

Ray RLlib Besides providing a convenient API in which one can train an agent for a certain environment, the open-source Ray RLlib library contains many re-implementations of state-of-the-art algorithms. Policy gradient methods like PPO, A2C, DDPG, TD3 and SAC are all implemented by this framework in a standardised manner. On using such an algorithm without adding any configurations, it will run with a set of predefined, known hyperparameters. As opposed to manually implementing these algorithms, or using algorithms from different third-party libraries, it is orders of magnitude easier to use and compare these algorithms. Under the hood of RLlib, deep learning frameworks PyTorch and TensorFlow provide the required neural network computations.

Ray is primarily a framework for distributed computing. RLlib demonstrates this in two ways. First of all, it can divide the computational workload over the number of CPUs on the machine. Secondly, it parallelises learning by vectorising the environment. This allows the environments to be batched together, significantly increasing the speed of the action computing forward pass.

After running an environment with a certain RL method, the trained model is saved to the file system, so one can easily access to trained model afterwards (e.g. for evaluation). Moreover, a progress report is saved comprising data of the entire run. This is very useful to investigate the state or the agents during training.

Finally, RLlib is despite its complexity quite customisable. In section 4.4.2, we show how we customise certain aspects of the learning method to favour the performance of probability updating games.

Ray Tune Ray Tune can be viewed as the final wrapper around the entire learning process. It provides a large array of configuration options. Using these configuration options, one can change e.g. the level of parallelisation for a specific run. It also allows for custom metric tracking, which we utilise as described in section 4.5.1. Additionally, it allows for adding certain stopping criteria, to enable us to finish a run automatically if certain conditions are met.

Moreover, it provides us with a toolbox for hyperparameter tuning for the algorithms. Using this toolbox, and multiple hyperparameter search algorithms that are provided, we were able to compare different hyperparameter configurations easily. Therefore, we were able to independently optimise PPO, A2C, DDPG, TD3 and SAC for probability updating games before comparing their performance to one another.

4.4 MARL for probability updating games

Recall a general-sum probability updating game $\mathcal{G} = \langle \mathcal{X}, \mathcal{Y}, p, L_H, L_C \rangle$. In this game, the host and the contestant both act once by providing conditional distributions $P(\cdot \mid x) : \mathcal{Y} \to \mathbb{R}$ for all $x \in \mathcal{X}$ and $Q(\cdot \mid y) : \mathcal{X} \to \mathbb{R}$ for all $y \in \mathcal{Y}$ respectively, after which the expected loss is computed for both. Additionally, we can calculate the expected generalised entropy of the contestant. We

know that for zero-sum games, Nash equilibria can arise if the minimum worst-case expected loss equals the maximum expected generalised entropy. For fully cooperative games (i.e. $L_H = L_C$), Nash equilibria can arise if the minimum worst-case expected loss equals the *minimum* expected generalised entropy. For the general-sum case, the expected generalised entropy term might not be very meaningful.

4.4.1 The action space

In traditional game theory, a game is largely defined by its action space. The action space defines the number of actions and the type of actions a player can take. E.g. in the traditional prisoner's dilemma, the action space is discrete and has two elements: *stay silent*, or 0, and *betray*, or 1. We denote the action space as $S : \{0, 1\}$.

Probability updating games are vastly more complex. The action space is continuous and it is always more than one continuous variable. In addition, these variables together should represent probability distributions. To describe the action space of a probability updating game, we provide the following example. Suppose Monty Hall from section 3.1.2 again. The message structure $\mathcal{Y}$ looks as follows:

$$\begin{aligned} y_1 = &\quad \{x_1 \quad x_2 \quad\quad \} \\ y_2 = &\quad \{\quad\quad x_2 \quad x_3\} \end{aligned}$$

Note that y_1 does not contain x_3, just like y_2 does not contain x_1. We re-enumerate the constraints from section 3.1 for a valid probability updating strategy:

- The host may not lie to the contestant: $P(y \mid x) = 0$ for all $x \notin y$.
- The contestant is rational: $Q(x \mid y) = 0$ for all $x \notin y$.
- P is a (conditional) probability distribution: we have $\sum_{y \in \mathcal{Y}} P(y \mid x) = 1$, for all $x \in \mathcal{X}$. This means that if x is only contained by one message, $P(y \mid x) = 1$ is forced.
- Idem for Q.

For Monty Hall, the above constraints restrict a considerable amount of variables for the host and contestant. Let $N_H(x)$ denote the number of continuous variables the host needs to provide for each x. Similarly, let $N_C(y)$ denote the number of continuous variables needs to provide for each y. From table 4.1, we have $N_H(x_2) = 2$ for the host and $N_C(y_1) = N_C(y_2) = 2$ for the contestant. One might convert this to the action spaces $S_H : [0, 1]^2$ and $S_C : [0, 1]^2 \times [0, 1]^2$.

At second sight at table 4.1, one may notice that the degrees of freedom for the players is given by $D_H(x) = N_H(x) - 1$ and $D_C(u) = N_C(y) - 1$. This could naturally reduce the action spaces to $S_H : [0, 1]$ and $S_C : [0, 1] \times [0, 1]$.

$P(y \mid x)$	x_1	x_2	x_3
y_1	1	**1/2**	0
y_2	0	**1/2**	1

$Q(x \mid y)$	y_1	y_2
x_1	**2/3**	0
x_2	**1/3**	**1/3**
x_3	0	**2/3**

Table 4.1: Example strategies P and Q for Monty Hall. The variables are indicated in bold. The constants are forced to be either 0 or 1.

4.4.2 The action distribution

During the implementation process of the multi-agent reinforcement learning aspect of the experimental research, we went through a few iterations on how the action space and the associated *action distribution* was implemented. This subsection will elaborate on the various iterations and ideas we raised in the pursuit of efficiently learning probability updating games.

First, consider a neural network that classifies a categorical variable. If we send a new observation, this observation will appear in the input layer of the neural network. Through a forward pass computation, in which one or more hidden layers might be passed, we obtain the "answer" in the output layer. However, the output layer values do not present any answer just yet. For categorical variables, a softmax activation function is usually applied. This normalises the values to a probability distribution. Subsequently, we can take the arg max to obtain the classification with the highest probability.

For reinforcement learning, we mentioned in section 4.1.3 that policy gradient methods usually learn continuous variables stochastically by parameterising a normal distribution $\mathcal{N}(\boldsymbol{\theta}_\mu, \boldsymbol{\theta}_\sigma)$. Here, $\boldsymbol{\theta}_\mu$ and $\boldsymbol{\theta}_\sigma$ are the values that are interpreted from the output layer. Additionally, if the continuous variable has a particular domain (e.g. $[0, 1]$), the policy value is squashed into the specified range using a sigmoid function. Not every policy gradient algorithm uses a stochastic policy for learning continuous variables. Specifically, DDPG and TD3 learn a deterministic policy whereas PPO, A2C and SAC use a stochastic policy.

For Monty Hall, parameterising a normal distribution in this way would return valid probability distributions. Indeed, we have for all games $\mathcal{G}$ in which

$$|\{x \in \mathcal{X} \mid x \in y\}| \leq 2 \text{ for all } y \in \mathcal{Y}$$
$$|\{y \in \mathcal{Y} \mid y \ni x\}| \leq 2 \text{ for all } x \in \mathcal{X}$$

there is a maximum of 1 degree of freedom for each outcome/message to condition on. Therefore, an action distribution like $\mathcal{N}(\boldsymbol{\theta}_\mu, \boldsymbol{\theta}_\sigma)$, after squashing with a sigmoid function, generates valid distributions, without the need for probability normalisation.

However, for games that do not comply with the above constraints, this approach may become problematic. We propose three unique approaches that aim to solve this problem. We implemented, evaluated and compared these three techniques empirically, after which we concluded to perform our main analysis using the Dirichlet action distribution. The results can be seen in section 5.1.

1. Learn variables independently, punish invalid distribution

For ease of explanation, we will assume that we use PPO (or another algorithm that learns a stochastic policy using normal distribution $\mathcal{N}(\theta_\mu, \theta_\sigma)$). Suppose we want to provide values for the variables for P. Recall that $D_H(x) = N_H(x) - 1$ denotes the degrees of freedom for x. For outcome x, we sample the variables

$$\mathbf{p}_x = \left[p_x^1 \sim \mathcal{N}(\theta_{x,\mu}^1, \theta_{x,\sigma}^1), \ldots, p_x^{D_H(x)} \sim \mathcal{N}(\theta_{x,\mu}^{D_H(x)}, \theta_{x,\sigma}^{D_H(x)})\right]$$

The remaining variable, $p_x^{N_H(x)}$, is obtained by

$$p_x^{N_H(x)} = 1 - \sum_{n=1}^{D_H(x)} p_x^n$$

We repeat this procedure for each x.

For the contestant, we apply this procedure equivalently, but for $D_C(y)$ and for each y. Now, if the game we try to learn does not satisfy the above constraints (i.e. it has more than 1 degree of freedom for a certain outcome/message), this approach may produce invalid distributions. If the resulting strategy is not a probability distribution, we return ∞ loss. We will from now refer to this technique as **Independent&Punish**.

2. Learn variables independently, normalise to distribution with softmax

Instead of learning only learning ($D_H(x)$ variables and determining the last on the basis of the others, we learn all of the $N_H(x)$ variables independently. Formally, given outcome x, we sample the variables

$$\mathbf{p}_x = \left[p_x^1 \sim \mathcal{N}(\theta_{x,\mu}^1, \theta_{x,\sigma}^1), \ldots, p_x^{N_H(x)} \sim \mathcal{N}(\theta_{x,\mu}^{N_H(x)}, \theta_{x,\sigma}^{N_H(x)})\right]$$

Now, we normalise $\mathbf{p}_x$ using the softmax function:

$$p_x^n = \frac{e^{p_x^n}}{\sum_{i=1}^{N_H(x)} e^{p_x^i}}$$

This procedure is repeated for each x.

For the contestant, we apply this procedure equivalently, but for $N_C(y)$ and for each y. We will from now refer to this technique as **Independent&Softmax**.

3. Learning the distribution directly using Dirichlet

Instead of relying on the normal distribution to sample our continuous variables and then normalising, we can sample P and Q directly from Dirichlet distributions, denoted by $\mathrm{Dir}(\boldsymbol{\alpha})$. Here, vector $\boldsymbol{\alpha} \in \mathbb{R}^K$ is called the concentration parameter. If we sample from such a distribution, we obtain a K-dimensional categorical distribution. Moreover, the Dirichlet distribution is the conjugate prior distribution of the categorical distribution. This is important because this means that we can consecutively update our knowledge of $\boldsymbol{\alpha}$, e.g. one stochastic gradient descent step at a time.

So, for outcome x, we sample the variables

$$\mathbf{p}_x \sim \mathrm{Dir}(\boldsymbol{\theta}_x)$$

Here, we have $\boldsymbol{\theta}_x \in \mathbb{R}^{N_H(x)}$. The result is that $\mathbf{p}_x$ is a valid, $N_H(x)$-dimensional categorical distribution. We repeat this procedure for each x.

For the contestant, we apply this procedure equivalently, but for $N_C(y)$ and for each y. We will from now refer to this technique as **Dirichlet**.

An alternate approach

Another approach would be to apply the softmax function in a more direct manner than **Independent&Softmax**. The following approach is one that we did not implement. Instead of parameterising $\boldsymbol{\theta}_x = [\boldsymbol{\theta}_{x,\mu}, \boldsymbol{\theta}_{x\sigma}]$ for each x and sampling from a normal distribution, we take the values from the output layer as-is:

$$\mathbf{p}_x = \boldsymbol{\theta}_x$$

Subsequently, we apply a softmax function to $\mathbf{p}_x$ to obtain the probabilities for the host. Again, we apply the equivalent procedure for the contestant. Interestingly, this approach closely resembles a deterministic version of the **Dirichlet** approach.

For PPO (and other algorithms that inherently learn a stochastic policy), we expect such a deterministic approach to be inferior. The problem lies in the exploration aspect of the learning process. For stochastic approaches, we can utilise stochastic sampling for exploration. With stochastic sampling, the amount of exploration is adjusted naturally by the parameterisation of the distribution, which is learned over time. Therefore, we can expect the variance of the model to decrease as it improves. For a deterministic approach such as this one, however, we would have to resort to epsilon-greedy, which does not organically change the exploration rate based on the model. We would then need another hyperparameter to adjust the exploration rate, but also a hard-coded mechanism if we want to dynamically change this exploration rate over time.

4.4.3 Observation space

Usually, RL environments contain potentially rich and complex observation spaces. An instance of such an environment would be the (simulated) world of a robot with some visual sensors. However, in many cases, these sensors are not perfect and can give a certain amount of (stochastic) noise. In this case, the agent observes the state but has no certainty of the actual state. Hence we call it the observation space.

For probability updating games, the observation space is quite simple. Consider probability updating game $\mathcal{G} = \langle \mathcal{X}, \mathcal{Y}, p, L_H, L_C \rangle$ again. Both players know $\mathcal{X}$, $\mathcal{Y}$ and outcome distribution p with certainty. However, we notice that after both players have taken action, the calculation of their expected losses is deterministic given the state. Moreover, the valid action spaces of both players can be deduced as explained in section 4.4.1. This implies that the agents do not *need* to see $\mathcal{X}$, $\mathcal{Y}$ or p to produce actions. Indeed, probability updating games have an *empty* observation space.

Still, neural networks do require some form of input. Therefore, we fill the observation with one constant value: 0. This is essentially equivalent to having no observation space, while it does allow for the neural network to receive observations through the input layer (of exactly 1 unit).

4.4.4 Expected loss

Once the contestant and host play their strategies, we determine their rewards by taking the negative of the expected losses of both players, i,e, $-\mathbb{E}[L_C]$ and $-\mathbb{E}[L_C]$. Reiterating section 3.2, the expected loss is computed as

$$\mathbb{E}[L] = \sum_{y \in \mathcal{Y}} \sum_{x \in y} P(x, y) L(x, Q(\cdot \mid y))$$

where $L(x, Q(\cdot \mid y))$ denotes the loss for the contestant or the host, given message y, true outcome x and contestant strategy Q.

After the host and the contestant set their actions, their reward is calculated.

In table 4.2, we list the loss functions that we used for evaluating the performance of learning methods as well as for the main analysis.

Name	$L(x, Q(\cdot \mid y)$
Randomised 0-1	$1 - Q(x \mid y)$
Brier	$(1 - Q(x \mid y))^2 + \sum_{x' \in \mathcal{X}, x' \neq x} Q(x' \mid y)^2$
logarithmic	$-\log Q(x \mid y)$
Randomised matrix	$\sum_{x' \in \mathcal{X}} Q(x' \mid y) A_{x,x'}$

Table 4.2: Loss functions used for evaluation and strategy analysis.

Name	$H_L(P(\cdot \mid y))$
Randomised 0-1	$1 - \max_{x \in \mathcal{X}} P(x \mid y)$
Brier	$1 - \sum_{x \in \mathcal{X}} P(x \mid y)^2$
logarithmic	$\sum_{x \in \mathcal{X}} -P(x \mid y) \log P(x \mid y)$
Randomised matrix	$\min_{x \in y} \sum_{x' \in \mathcal{X}} P(x' \mid y) A_{x',x}$

Table 4.3: Generalised entropy functions corresponding to the loss functions used for evaluation and strategy analysis.

As is clear from the name, logarithmic loss computes the loss using a logarithm operation. During the implementation of our MARL methods, we aimed to follow Van Ommen et al. (2016) and set $\log 0 = \infty$. However, this raised fatal errors in the learning methods, as they were not designed to handle infinities. As a solution, we take $\min(C, \log(x))$ and set $C = 5$. Now, if the contestant predicts $Q(x \mid y) \leq e^{-5}$ (we consider $e^{-5} \approx 0.00673$ to be a very small probability), then logarithmic loss returns the maximum loss $C = 5$.

Randomised Matrix loss

Randomised matrix loss is a versatile loss function, in which we can explicitly specify a loss that is returned when the contestant picks a particular outcome x' instead of true outcome x. Given a matrix $\mathbf{A} \in \mathbb{R}^{\mathcal{X} \times \mathcal{X}}$, randomised matrix loss is defined as in table 4.2. The generalised entropy for randomised matrix loss is displayed in table 4.3.

Randomised matrix loss generalises randomised 0-1 loss. The matrix below corresponds to the equivalent randomised 0-1 loss:

$$A_{x,x'} = \begin{cases} 0 & \text{for } x = x' \\ 1 & \text{for } x \neq x' \end{cases}$$

Van Ommen et al. (2016) proved that a Nash equilibrium exists for games with randomised matrix loss. This fact is important for our analysis, since this loss function gives us the opportunity to investigate very specific configurations and edge cases. It should however be noted that this loss function is not necessarily symmetric with respect to exchanges in $\mathcal{Y}$. It is also not local or proper. We re-emphasise that these properties matter, because at least for zero-sum games, they affect whether RCAR characterises worst-case optimality (and thus Nash equilibria) or not. While we cannot make randomised matrix loss proper or local, we can however, depending on the structure of the given game, create matrices that are symmetric w.r.t. exchanges in $\mathcal{Y}$. A matrix

is symmetric between a pair of outcomes (x_1, x_2) if and only if $A_{x_1,x_1} = A_{x_2,x_2}$, $A_{x_1,x_2} = A_{x_2,x_1}$, $A_{x',x_1} = A_{x',x_2}$, $A_{x_1,x'} = A_{x_2,x'}$ for all $x' \in \mathcal{X} \setminus \{x_1, x_2\}$.

We use randomised matrix loss in a considerable amount of probability updating samples. Naturally, different matrices would result in very different games and therefore greatly different Nash equilibria. To make our analysis more systematic, we established some constraints for the matrices.

We only look at matrices in which the diagonals are all zeroes: $A_{x,x} = 0$ for all $x \in \mathcal{X}$. This represents a loss function in which correct predictions by the contestant do indeed return zero loss. Additionally, we categorise the following type of matrices:

- Positive matrices: $A_{x,x'} \geq (0, 1]$ for all $x, x' \in \mathcal{X}$ with $x \neq x'$. If the matrices assigned to both players are positive, the game would be more cooperative.
- Negative matrices: $A_{x,x'} \geq [-1, 0)$ for all $x, x' \in \mathcal{X}$ with $x \neq x'$. If the matrices assigned to the contestant are positive, but to the host negative, the game would be more competitive.

Instead of randomly generating these matrices, we constructed a predefined set of matrices that we expect may return interesting strategies. These matrices were carefully selected for each game sample. Section 4.5 lists all the predefined matrices for each sample.

4.5 Game samples

We aim to understand the landscape of strategies and their rewards in an efficient way. Therefore, we will look at specific edge cases, rather than randomly generating games. Edge cases may give us far greater insight into the mechanics of general-sum strategies. Van Ommen et al. (2016) describe a few games that would qualify as such. In this subsection, we enumerate and explain the game samples used for analysis.

Game	$\lvert\mathcal{X}\rvert$	$\lvert\mathcal{Y}\rvert$	Graph?	Matroid?	p
Fair Die (Example A)	6	2	No	No	uniform
Monty Hall (Example B)	3	2	Yes	Yes	uniform

Table 4.4: Probability updating games used for the algorithm comparison. The games and their names are according to Van Ommen et al. (2016).

Game	$\lvert\mathcal{X}\rvert$	$\lvert\mathcal{Y}\rvert$	Graph?	Matroid?	p
Monty Hall (Example B)	3	2	Yes	Yes	uniform
Example C	4	3	Yes	No	$(\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{2}{5})$
Example D	4	2	No	No	$(\frac{1}{3}, \frac{1}{3}, \frac{1}{6}, \frac{1}{6})$
Example E	4	2	No	No	$(\frac{9}{20}, \frac{1}{20}, \frac{5}{20}, \frac{5}{20})$
Example F	3	3	Yes	Yes	uniform
Example G	4	4	Yes	Yes	uniform
Example H	3	3	Yes	Yes	$(\frac{1}{5}, \frac{3}{5}, \frac{1}{5})$

Table 4.5: Probability updating games used for our strategy analysis. The games and their names are according to Van Ommen et al. (2016).

Monty Hall This is the smallest game in which there exists no strategy that is coarsened at random (CAR). It is a graph game, as well as a matroid game. The game is structured like the following:

$$\begin{aligned} \mathcal{X} &= \{x_1 \quad x_2 \quad x_3\} \\ p_x &= (1/3 \quad 1/3 \quad 1/3) \\ y_1 &= \{x_1 \quad x_2 \quad \} \\ y_2 &= \{ \quad x_2 \quad x_3\} \end{aligned}$$

Fair Die The host throws a six-sided die. The host may look at the die but the contestant may not. The host either tells the contestant (truthfully) that the result lies within $[1, 4]$ or $[3, 6]$, after which the contestant picks the number that it thinks the die landed on. We have:

$$\begin{aligned} \mathcal{X} &= \{x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6\} \\ p_x &= (1/6 \quad 1/6 \quad 1/6 \quad 1/6 \quad 1/6 \quad 1/6) \\ y_1 &= \{x_1 \quad x_2 \quad x_3 \quad x_4 \quad \quad \} \\ y_2 &= \{ \quad \quad x_3 \quad x_4 \quad x_5 \quad x_6\} \end{aligned}$$

It is neither graph nor matroid. For randomised 0-1, Brier and logarithmic loss, a worst-case optimal strategy for the host is to randomise uniformly over $\mathcal{Y}$ for both x_3 and x_4.

Example C In this example, we have four outcomes and three messages. The game structure is defined as:

$$\begin{aligned} \mathcal{X} &= \{x_1 \quad x_2 \quad x_3 \quad x_4\} \\ p_x &= (1/5 \quad 1/5 \quad 1/5 \quad 2/5) \\ y_1 &= \{x_1 \quad x_2 \quad \quad \} \\ y_2 &= \{ \quad x_2 \quad x_3 \quad \} \\ y_3 &= \{ \quad \quad x_3 \quad x_4\} \end{aligned}$$

It is a graph game, but not a matroid game. For randomised 0-1, Brier and logarithmic loss, the unique worst-case optimal strategy for the host is to never send message y_2 to the contestant: $P(y_2) = 0$.

Example D It contains four outcomes and two messages. This game is structured as:

$$\begin{aligned} \mathcal{X} &= \{x_1 \quad x_2 \quad x_3 \quad x_4\} \\ p_x &= (1/3 \quad 1/3 \quad 1/6 \quad 1/6) \\ y_1 &= \{x_1 \quad x_2 \quad \quad \} \\ y_2 &= \{ \quad x_2 \quad x_3 \quad x_4\} \end{aligned}$$

It is neither a graph- nor a matroid game. Van Ommen et al. (2016) used this sample to demonstrate that the worst-case optimal strategy depends on the loss function. This game has no unique worst-case optimal strategy for randomised 0-1 loss. A worst-case optimal strategy for logarithmic loss as well as randomised 0-1 loss for the host is to set $P(y_1 \mid x_2) = P(y_2 \mid x_2)$, i.e. play uniformly on the messages.

Example E This game is structurally equivalent to Example D, however it has a different marginal outcome distribution p_x. The game is structured as follows:

$$\begin{aligned} \mathcal{X} &= \{x_1 \quad x_2 \quad x_3 \quad x_4\} \\ p_x &= (0.45 \quad 0.05 \quad 0.25 \quad 0.25) \\ y_1 &= \{x_1 \quad x_2 \quad\quad\quad\quad\} \\ y_2 &= \{\quad\quad x_2 \quad x_3 \quad x_4\} \end{aligned}$$

Example F A notably symmetrical game. It has three outcomes and three messages. We have:

$$\begin{aligned} \mathcal{X} &= \{x_1 \quad x_2 \quad x_3\} \\ p_x &= (1/3 \quad 1/3 \quad 1/3) \\ y_1 &= \{x_1 \quad x_2 \quad\quad\} \\ y_2 &= \{\quad\quad x_2 \quad x_3\} \\ y_3 &= \{x_1 \quad\quad x_3\} \end{aligned}$$

It is both graph and matroid. For randomised 0-1 loss, the worst-case optimal strategy for the host is to play uniformly. In a fully cooperative setting ($L_C = L_H$), it should be possible for the host and contestant to reach zero loss: the host can remove any uncertainty for the contestant by playing in a particular way. We show that this happens in 5.2.

Example G Another very symmetrical game. Structurally, it looks like an extension of Example F, adding one message and one outcome. It is also graph as well as matroid. The game is structured as:

$$\begin{aligned} \mathcal{X} &= \{x_1 \quad x_2 \quad x_3 \quad x_4\} \\ p_x &= (1/4 \quad 1/4 \quad 1/4 \quad 1/4) \\ y_1 &= \{x_1 \quad x_2 \quad\quad\quad\} \\ y_2 &= \{\quad\quad x_2 \quad x_3 \quad\quad\} \\ y_3 &= \{\quad\quad\quad\quad x_3 \quad x_4\} \\ y_4 &= \{x_1 \quad\quad\quad\quad x_4\} \end{aligned}$$

For randomised 0-1 loss, the worst-case optimal strategy for the host is to play in a completely uniform manner. While for the contestant, any strategy that puts probability a on either x_1 or x_3 (whichever is in the given message) while putting probability $1 - a$ on either x_2 or x_4 is worst-case optimal.

Example H A structurally equivalent game to example F, however, with a non-uniform marginal outcome distribution. For the structure of the game, we have:

$$\begin{aligned} \mathcal{X} &= \{x_1 \quad x_2 \quad x_3\} \\ p_x &= (1/5 \quad 3/5 \quad 1/5) \\ y_1 &= \{x_1 \quad x_2 \quad\quad\} \\ y_2 &= \{\quad\quad x_2 \quad x_3\} \\ y_3 &= \{x_1 \quad\quad x_3\} \end{aligned}$$

4.5.1 Metrics

In analysing the strategy landscape, we need to acquire a clear overview of the learned data that we see. Therefore, it may be necessary to design and observe additional metrics that may give us more understanding of our current location within that strategy landscape.

Ray RLlib and Tune allow us to perceive a large variety of insightful metrics that we can track during the learning process. Many of these metrics regard algorithm-specific variables that e.g. measure the current loss or learning rate of the method. Additionally, we can observe the current agent rewards. Note that throughout all of game theory and reinforcement learning, agent performance is usually determined by the reward, or utility, which implies that a larger value is better. In our implementation, we take this into account by returning the negative of the losses L_H and L_C.

But besides the built-in metrics, we designed and implemented some supplementary metrics to support us in obtaining knowledge on the strategic landscape. We describe these metrics below.

Current P and Q In Ray Tune, there is no standard way to measure the agent's actions over time. Hence, we added all probabilities from P and Q to the custom metrics on each time step. This way, we can observe what the strategies look like during the learning process. This gives us far greater insight into the probability updating game mechanics than merely observing the expected losses of the players. For instance, we can watch whether the host is getting close to RCAR, whether the players play some unique optimal strategy or possibly a range of non-unique optimal strategies that all form equilibria.

Expected generalised entropy We continually track the expected generalised entropy $\mathbb{E}(H_L) = \sum_{y \in \mathcal{Y}} P(y) H_L(P(\cdot \mid y))$. As mentioned before, Van Ommen et al. (2016) showed that a Nash equilibrium is guaranteed when the maximum expected generalised entropy equals the minimum worst-case expected loss for zero-sum games. Despite that this theorem does not hold for general-sum games, it might still be interesting to observe this value if the game has very competitive - or very cooperative - aspects. Additionally, it still holds that if the current expected loss of the contestant equals the current expected generalised entropy, the contestant has reached a minimum expected loss for the given coarsening mechanism. This information may give us an indication of whether the players converged towards some equilibrium.

4.5.2 MARL algorithm optimisation

During the development of our MARL setup, we invested a considerable amount of time in optimising the performance of all of the algorithms and setting up the right configurations to achieve the best results. We emphasised learning performance, as policy gradient methods lack any theoretical guarantees for converging to Nash equilibria. These learning methods are essentially single-agent learning methods, extrapolated to a multi-agent environment. Reiterating from section 4.1.5, the learning performance of RL methods for such environments may significantly be reduced due to the environment becoming non-stationary. In section 4.1.8 we elaborate on how we apply a fully decentralised MARL approach. This, however, does not solve the problem. Hence, we deliberately optimise and test our empirical performance; to make our strategy analysis more reliable.

We used the built-in grid search of the Ray Tune research platform to optimise the set of hyperparameters for all of the algorithms that were utilised. It should be noted that during

experimentation with the various algorithms and action distribution approaches, it became quite clear early on that PPO outperformed - and would most probably stay outperforming - its competitors. Afterwards, we invested additional time in optimising the hyperparameters of PPO, to increase its performance even further. In appendix A, we demonstrate all of the hyperparameters that we tuned, for each algorithm.

5 Results

This chapter comprises the findings from our experimental runs. These results are split into the performance evaluation runs and our main analysis. As for the performance evaluation, we structure these findings to emphasise the differences and similarities between the actor-critic algorithms as well as the varying action distribution approaches.

5.1 Algorithm comparison

Intuitively, we wish to tackle the learning of probability updating games with multiple techniques, so that we can learn which kinds of algorithms show better results than others. Indeed, applying an adequately large set of algorithms that learn our games is desirable if we want to perform a rich comparison and ultimately obtain a superior learning performance. Ray RLlib has many RL algorithms available. However, we had to dismiss the majority of these algorithms for compatibility reasons; some are not suitable for continuous action spaces (e.g. DQN), while others cannot be combined with multi-agent environments (e.g. evolutionary strategies). Primarily, the remaining algorithms that did not cause any major problems during runtime were the actor-critic methods PPO, A2C, DDPG, TD3 and SAC.

We optimised the algorithms by tuning their hyperparameters (appendix A), and subsequently ran them for the following probability updating game configurations:

- Monty Hall and Fair Die;
- the randomised 0-1, Brier and logarithmic loss functions;
- a zero-sum and fully cooperative setting.

This sums to a total of $2 \times 3 \times 2$ games. Of these games, we present the most defining results. We let the learning methods train for 60 seconds in each game.

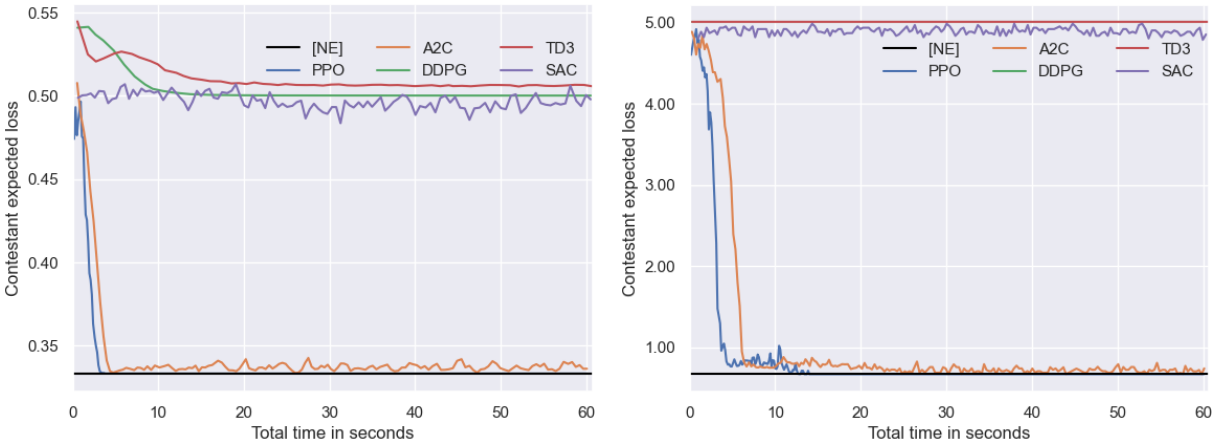
As for evaluating the learning performance, we measure only the contestant's loss L_C. This is satisfactory, as these games are either zero-sum or fully cooperative. For measuring learning stability and converging to Nash equilibria, we also measure the RCAR distance as formulated in section 4.5.1.

5.1.1 Independent&Punish - algorithm comparison

In terms of implementation, **Independent&Punish** is the simplest action distribution approach. Intuitively, it can be said that we do not consider the shape of the output values from the learning method, and instead just punish the agent if a given output value is not valid. A disadvantage of this approach is that now, the agent has to learn the rules of the valid action space as well as find an optimal action. Then again, it could be that deep actor-critic methods can overcome this disadvantage with ease.

Figure 5.1 displays the learning performance of PPO, A2C, DDPG, TD3 and SAC during training time, for randomised 0-1 loss in Monty Hall and Fair Die respectively. At first sight, it seems clear that PPO and A2C are dominant in both games. Only these two algorithms manage to converge to a Nash equilibrium. They are very fast in reaching the global optimum; PPO takes

less than four seconds for Monty Hall and around 14 seconds for Fair Die. Also, we observe that while A2C seems to struggle to stay at the equilibrium consistently, PPO excels at staying at the global optimum.



(a) Monty Hall. A Nash equilibrium is found at $\mathbb{E}[L_C] = 1/3$.

(b) Fair Die. A Nash equilibrium is found at $\mathbb{E}[L_C] = 2/3$.

Figure 5.1: Learning performance during training using **Independent&Punish** for zero-sum games with randomised 0-1 loss. The x-axis displays the time in seconds. The y-axis displays the contestant's expected loss $\mathbb{E}[L_C]$. The expected loss corresponding to a Nash equilibrium is plotted in black.

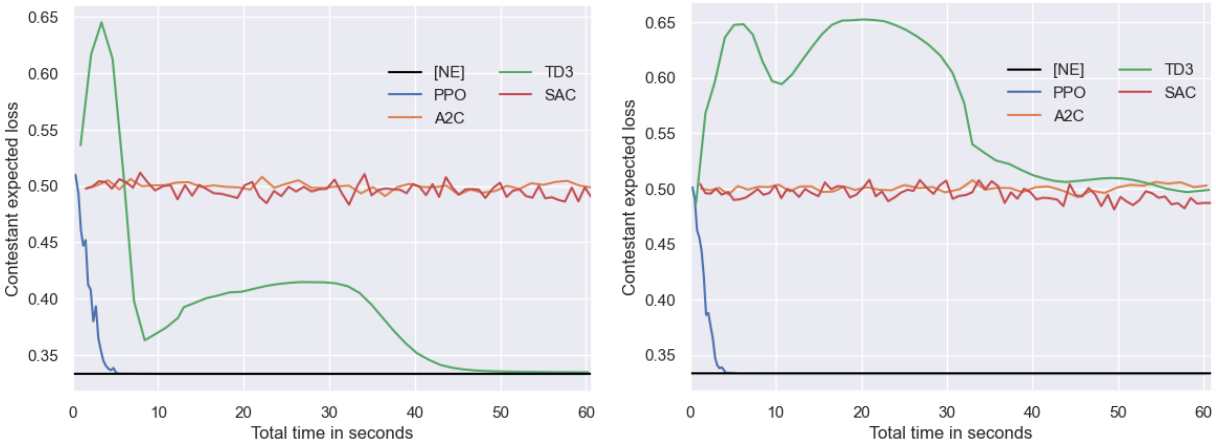
Also, note that for Fair Die, the initially incurred losses are very large. Indeed, $\mathbb{E}[L_C] = 5$, which represents infinity (see section 4.4.4). The fact that the contestant incurs infinite loss is caused by Fair Die having more than 1 degree of freedom due to its message structure (see section 4.5). As mentioned in 4.4.2, **Independent&Punish** will consequently produce invalid actions, as the independent variables do not sum to one. Despite this, PPO and A2C manage to learn that such actions are not desirable very quickly, and continue to converge to a Nash equilibrium.

We also ran these games with Brier and logarithmic loss. They tell the same story; PPO and A2C remain far superior and reach Nash equilibria, while the others do not seem to learn the game.

5.1.2 Independent&Softmax - algorithm comparison

The following approach, **Independent&Softmax**, solves the issue of producing invalid strategies on higher degrees of freedom. The model computes its outputs like it usually would (either deterministic or with e.g. a normal distribution), after which the values are normalised using softmax. Based on the algorithm, exploration is achieved in varying ways, and is not necessarily associated with the distribution from the softmax calculation, but applied on all of the variables independently. It could be that this affects the learning performance of the algorithm.

During the evaluation of **Independent&Softmax**, we excluded DDPG. Essentially, TD3 is a successor to DDPG; it is the same algorithm, but with a few additions. Along with the fact that DDPG and TD3 functioned very similarly in the previous experiments, we decided that DDPG is redundant for our subsequent experiments.



(a) Zero-sum. A Nash equilibrium is found at $\mathbb{E}[L_C] = 1/3$.

(b) Cooperative. A Nash equilibrium is found at $\mathbb{E}[L_C] = 1/3$.

Figure 5.2: Learning performance during training using **Independent&Softmax** in **Monty Hall** with randomised 0-1 loss.

Figure 5.2 displays how **Independent&Softmax** performs over time for zero-sum and cooperative **Monty Hall** with randomised 0-1 loss. Figure 5.3 shows the method for zero-sum **Monty Hall** with logarithmic loss and 5.4 displays it in **Fair Die** with zero-sum and cooperative Brier loss. We observe and confirm in every figure that A2C and SAC show inferior learning capacity. This is unexpected for A2C since it performed not far worse than PPO in the previous version. TD3's learning performance is quite puzzling. It is very inconsistent, although it sometimes does reach a Nash equilibrium (figure 5.2). In figure 5.3, TD3 interestingly surpasses the Nash equilibrium. However, this is not a good thing; this game is zero-sum, i.e. the host strategy should maximise the expected entropy $\mathbb{E}[H_L]$, by playing the unique RCAR strategy for Monty Hall. This, unfortunately, does not happen, hence the contestant exploits this and reaches a lower loss than should be possible.

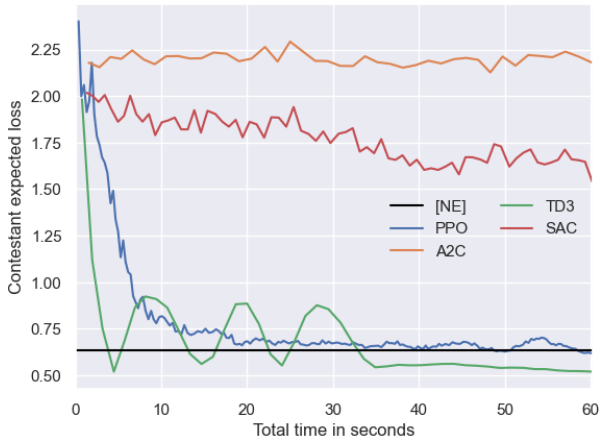
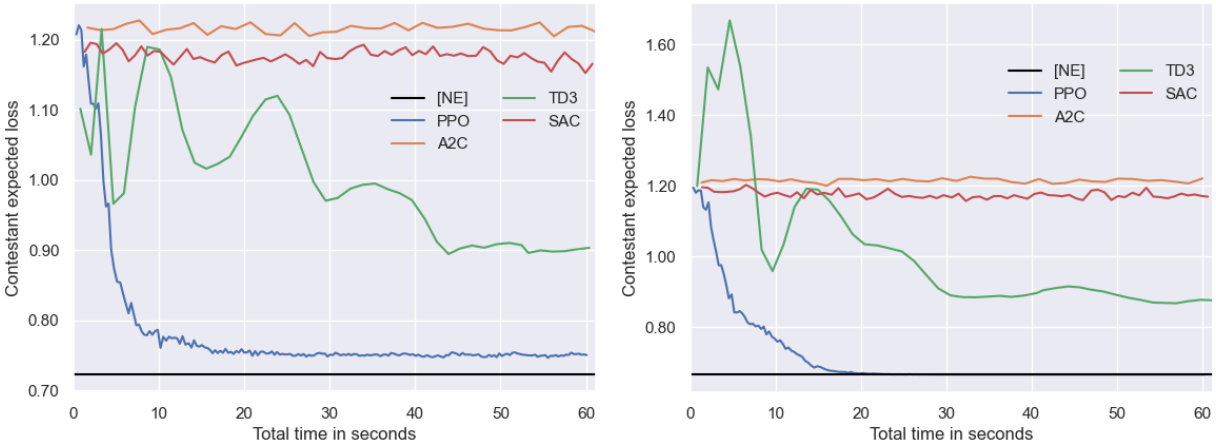


Figure 5.3: Learning performance during training using **Independent&Softmax** in zero-sum **Monty Hall** with logarithmic loss. A Nash equilibrium is found at $\mathbb{E}[L_C] \approx 0.636$.

PPO performs excellently, except for one particular game setting: zero-sum Fair Die with Brier loss (figure 5.4). It seems to be stuck in some local optimum. For the cooperative setting, however, PPO seemed to converge much more easily. Possibly, PPO would have reached the

global optimum if we ran it for longer. Also, these runs were not averaged, so this may also be due to bad luck.



(a) Zero-sum. A Nash equilibrium is found at $\mathbb{E}[L_C] \approx 0.722$.

(b) Cooperative. A Nash equilibrium is found at $\mathbb{E}[L_C] = 2/3$.

Figure 5.4: Learning performance during training using **Independent&Softmax** in **Fair Die** with Brier loss.

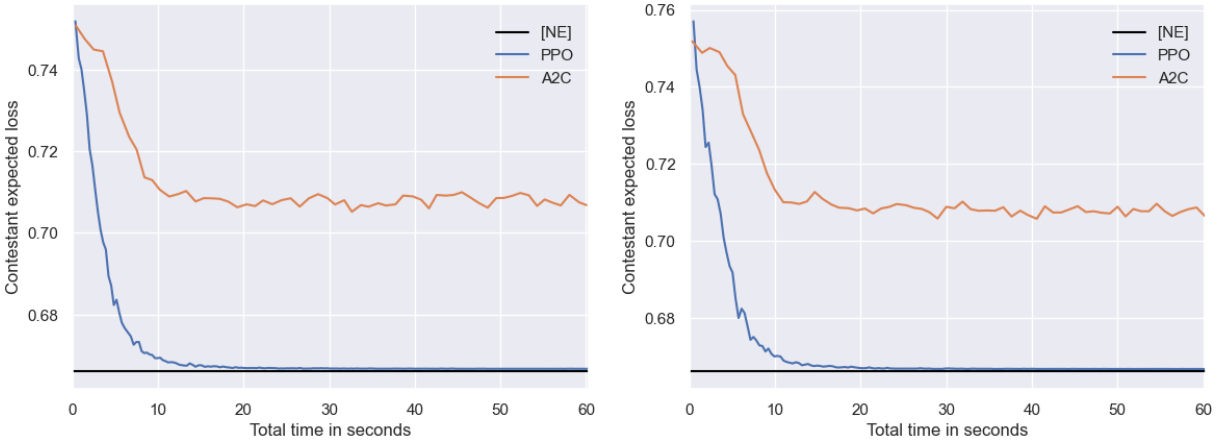
We found that the other configurations agree with the above results.

5.1.3 Dirichlet - algorithm comparison

This approach is an elegant solution to learning strategies in probability updating games. By learning the categorical distributions directly, **Dirichlet** allows for efficient learning. We expect that this will improve the convergence rates towards Nash equilibria even further.

It should be noted that for this approach, we could only get PPO and A2C to perform properly. DDPG and TD3 are not suitable because of their deterministic nature. SAC, however, might have worked, but the current implementation in Ray did not accept the action space that we implemented corresponding with **Dirichlet**. SAC's empirical performance left much to be desired during the previous experiments. Hence, we concluded that investing in getting SAC to work would most probably not have additional benefits.

As can be observed in figure 5.5, PPO shows clear dominance in achieving and retaining a Nash equilibrium strategy pair. A2C never seems to converge (quickly, i.e. within 60 seconds) towards a Nash equilibrium.



(a) Zero-sum. A Nash equilibrium is found at $\mathbb{E}[L_C] = 2/3$.

(b) Cooperative. A Nash equilibrium is found at $\mathbb{E}[L_C] = 2/3$.

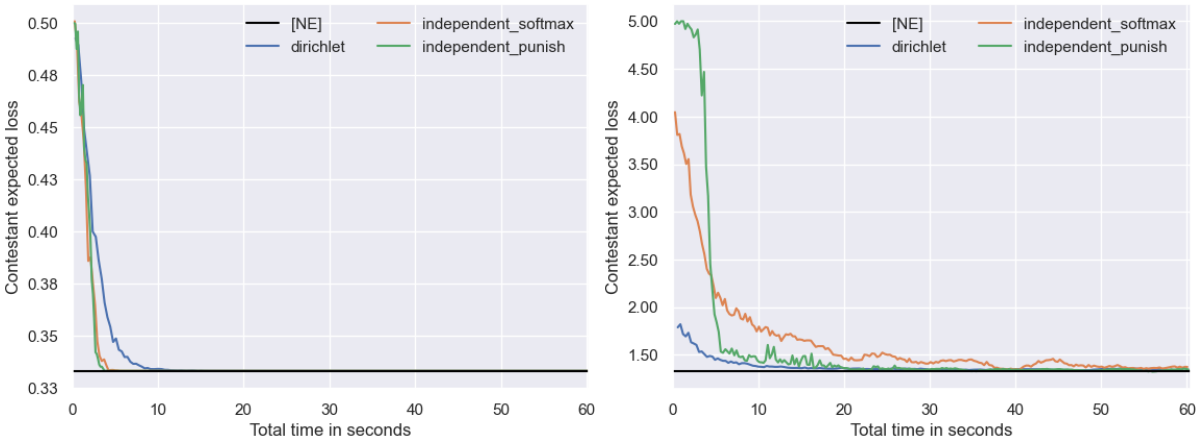
Figure 5.5: Learning performance during training using **Dirichlet** in **Fair Die** with randomised 0-1 loss.

5.1.4 PPO - action distribution comparison

Throughout the previous evaluations, PPO proved to be the best algorithm for learning probability updating games and finding Nash equilibria.

We initially proposed three unique approaches towards implementing an action distribution. We tested **Independent&Punish**, **Independent&Softmax** and **Dirichlet** side by side with PPO as the main algorithm. We executed the experiment with the same configurations laid out at the start of section 5.1.

Per our findings, we conclude that **Dirichlet** is the most robust learner. This is most distinctly exemplified in figure 5.6. Other game configurations are in line with our conclusions.



(a) Cooperative **Monty Hall** with randomised 0-1 loss. A Nash equilibrium is found at $\mathbb{E}[L_C] = 1/3$.

(b) Zero-sum **Fair Die** with logarithmic loss. A Nash equilibrium is found at $\mathbb{E}[L_C] \approx 1.3293$.

Figure 5.6: Learning performance for PPO. For the easy game (left), there is not a large distinction. However, for the harder game (right), it becomes clear that **Dirichlet** is the most scalable for probability updating games.

On the left, we see a game that we know is a relatively simple game to learn; fully cooperative

Monty Hall with randomised 0-1 loss. On the right, we see a game that we know is slightly harder to learn; zero-sum Fair Die with logarithmic loss. It is harder because 1) the zero-sum + logarithmic loss combination causes frail Nash equilibria, and 2) the contestant has more degrees of freedom due to a more complex $\mathcal{Y}$. This figure illustrates the general trend: as games become harder, **Dirichlet** preserves the highest learning stability and performance. As mentioned before, **Independent&Punish** does not scale well with more complex message structures. As for the distinction between **Dirichlet** and **Independent&Softmax**, one could argue that **Dirichlet** benefits from directly optimising a distribution. As a consequence, the amount of exploration follows the knowledge of the model, which could make it more robust for harder games.

One interesting finding is visualised in figure 5.7. Here, we see that **Independent&Punish** manages to outperform the other two approaches. The interesting part is that this only happens for Monty Hall with Brier loss. We do not have a specific explanation for this occurrence. Despite this, we conclude that **Dirichlet** is still superior overall, mainly for reasons mentioned earlier: the approach is more scalable for larger and harder probability updating games.

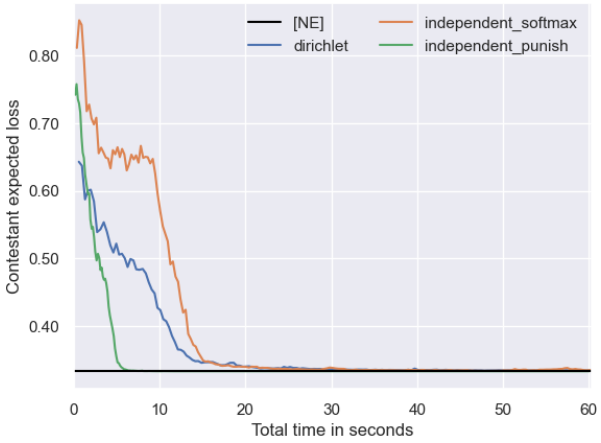


Figure 5.7: Learning performance for PPO on cooperative **Monty Hall** with Brier loss.

5.2 Strategy analysis

In this section, we report the results from the strategy analysis. The goal of this study is to investigate Nash equilibria for relaxed probability updating games. We run PPO with the Dirichlet action distribution on the games listed in section 4.5. We apply randomised matrix loss, allowing us to define specific loss values for any outcome pair $(x, x') \in \mathcal{X}$. In most configurations, the contestant and the host are assigned dissimilar matrices, therefore relaxing the zero-sum constraint.

Some game samples (section 4.5) are graph or matroid games, while others are not. Recall that for zero-sum graph/matroid games with L symmetric with respect to exchanges in $\mathcal{Y}$, RCAR characterises Nash equilibria. For our graph/matroid game samples, we deliberately selected matrices that either do or do not satisfy this symmetry property. Conversely, for zero-sum games that are neither graph nor matroid, no matrix will satisfy the conditions on L that allow RCAR to characterise Nash equilibria. Nonetheless, we run them with the same matrices. As a result, we can observe whether equilibria that emerge for general-sum graph/matroid games may indeed be unique to graph/matroid games.

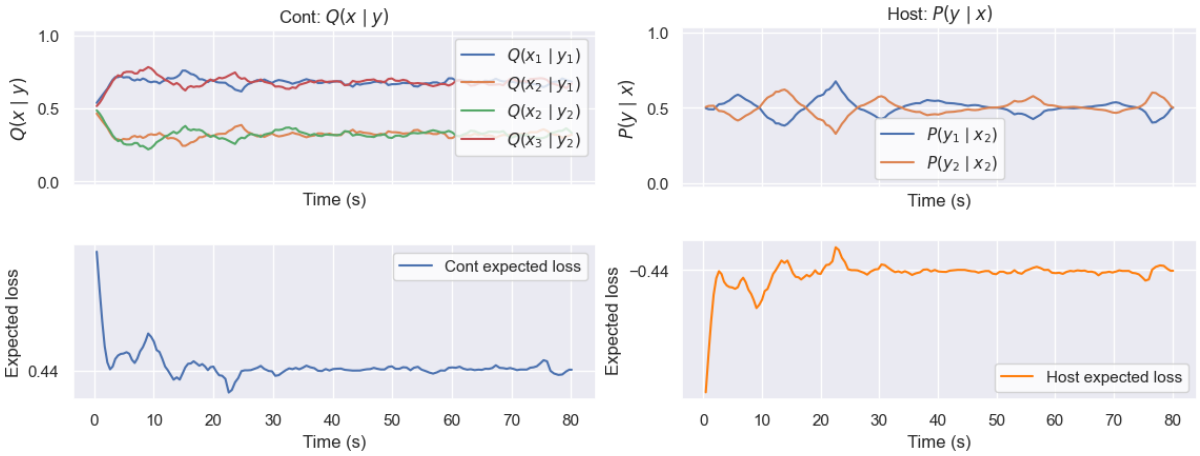
Our MARL method is approximate and there is no theoretical guarantee for converging to Nash

equilibria. Therefore, if the agents converge to a strategy pair, it is not guaranteed to be Nash. It is also important to note that some games have multiple Nash equilibria. In fact, we observe games that have infinite Nash equilibria. It is, therefore, possible that MARL methods return different optimal strategies when run more than once. To confirm our approximate (stochastic) results, we run each experiment multiple times, in parallel. We utilised the distributed nature of Ray to run 5 separate "trials" side by side, each trial using 2 CPU cores. We found that running configurations for 80 seconds is adequate for convergence for most configurations. Otherwise, we ran them for 300 seconds, after which most did converge. Though, some games never seem to converge, as we will see in the course of this section.

5.2.1 Base case: emerging strategies zero-sum constrained games

Before running any general-sum game, we must confirm that PPO indeed converges to worst-case optimal strategy pairs for the verifiable case: regular zero-sum games.

Figure 5.8 shows P and Q evolving over time for zero-sum **Monty Hall** with Brier loss. As expected, the agents approximate the unique Nash equilibrium; The host approximates the RCAR strategy, playing the uniform distribution, and the contestant plays $Q(\cdot \mid y) \approx P(\cdot \mid y)$. Or speaking in terms of the original Monty Hall problem, the contestant will switch to the other door 2/3 of the time. However, during the experiment, we observe slight deviations that seem to occur randomly. Interestingly, we also see this "deviant" behaviour for logarithmic loss.

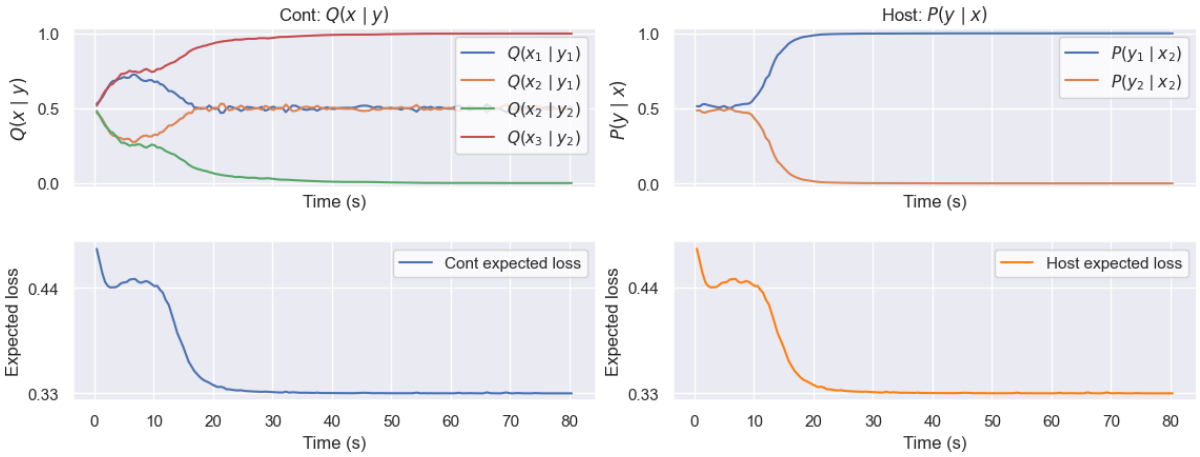


(a) The **contestant**. A Nash equilibrium is found at $\mathbb{E}[L_C] = 4/9$.

(b) The **host**. A Nash equilibrium is found at $\mathbb{E}[L_H] = -4/9$.

Figure 5.8: Q and P over time in **Monty Hall** with zero-sum Brier loss. The upper graphs show the probabilities of Q/P. The lower graph shows the expected loss. The x-axis represents runtime in seconds.

However, if we look at the fully cooperative **Monty Hall** with Brier loss (figure 5.9), this behaviour is absent. Instead, the agents remain much more stably at their optimal strategies.



(a) The **contestant**. A Nash equilibrium is found at $\mathbb{E}[L_C] = 1/3$.

(b) The **host**. A Nash equilibrium is found at $\mathbb{E}[L_H] = -1/3$.

Figure 5.9: Q and P over time in **Monty Hall** with fully cooperative Brier loss.

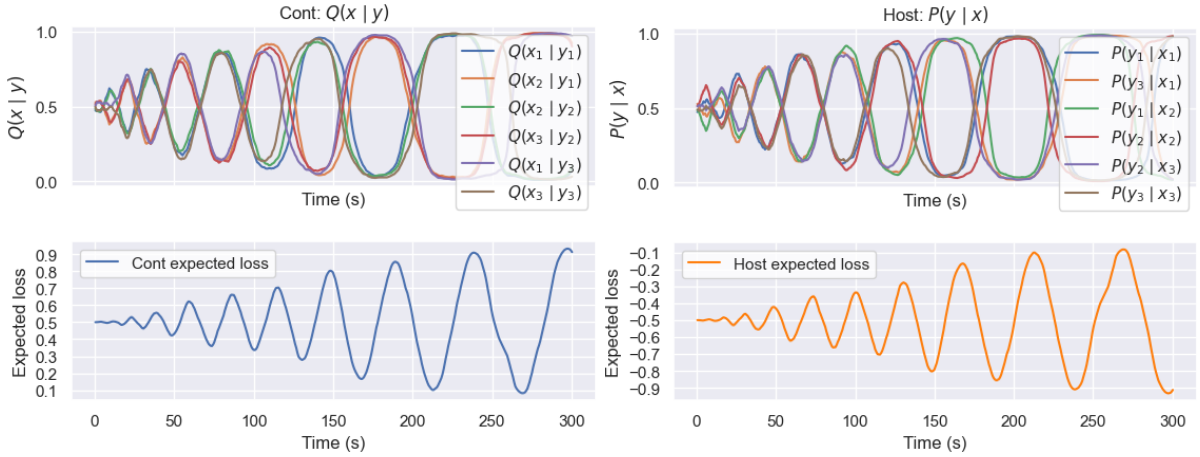
The "deviant" behaviour from figure 5.8 turns into oscillatory behaviour for randomised 0-1 loss games with unique Nash equilibria. Take for example the symmetrical graph and matroid **Example F**. This game is different from **Monty Hall** in the sense that here, the host has one unique worst-case optimal strategy for randomised 0-1 loss. The strategies are depicted in table 5.1. Observe how the players act uniformly given each message/outcome.

$P(y \mid x)$	x_1	x_2	x_3
y_1	**1/2**	**1/2**	0
y_2	0	**1/2**	**1/2**
y_3	**1/2**	0	**1/2**

$Q(x \mid y)$	y_1	y_2	y_3
x_1	**1/2**	0	**1/2**
x_2	**1/2**	**1/2**	0
x_3	0	**1/2**	**1/2**

Table 5.1: The unique worst-case optimal strategies for zero-sum **Example F** with randomised 0-1 loss. The variables are indicated in bold. The constants are forced to be either 0 or 1.

Figure 5.10 shows P and Q over time for **Example F**. The agents oscillate back and forth without converging to the unique Nash equilibrium. We observe the same in the zero-sum **Example G** (not depicted). **Example G** is another very symmetrical graph and matroid game, with a uniform outcome distribution (just like **Example F**). Yet, note that these oscillations seem symmetrical and consistent. The agents appear to oscillate around the unique Nash equilibrium. Therefore, even if this behaviour should occur for non-zero-sum games, it may still give us a hint at what a Nash equilibrium could look like.

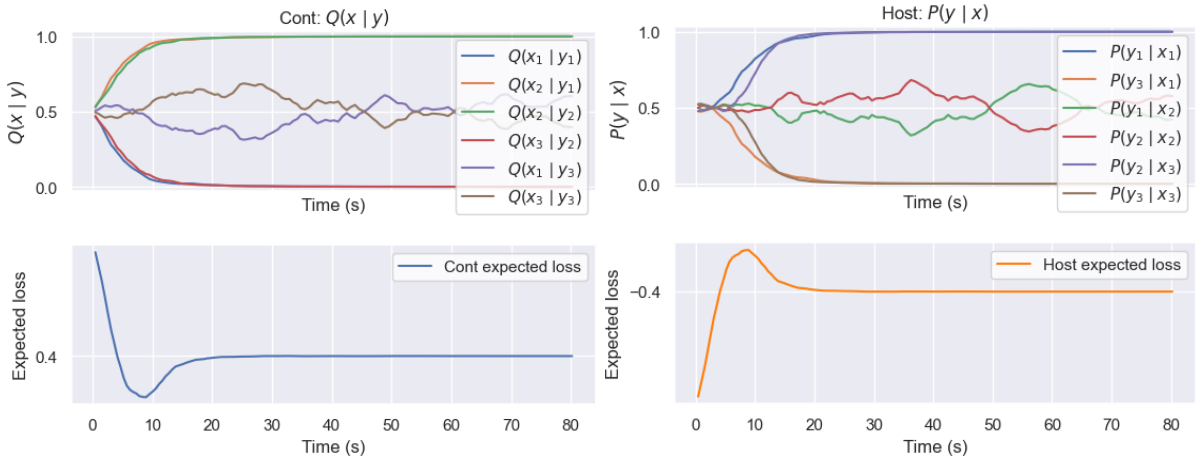


(a) The **contestant**. The agent fluctuates around the Nash equilibrium $\mathbb{E}[L_C] = 0.5$.

(b) The **host**. The agent fluctuates around the Nash equilibrium $\mathbb{E}[L_H] = -0.5$.

Figure 5.10: Q and P over time in **Example F** with zero-sum randomised 0-1 loss.

This behaviour is not due to the zero-sum constraint; 5.11 depicts the same loss configuration (zero-sum randomised 0-1 loss) in **Example H**. Recall that this game is almost identical to **Example F** but with a non-uniform outcome p_x. Here, we do not find oscillations or even deviations. The strategies do walk randomly due to their indifference towards their respective outcomes/messages, however, this does not affect their expected losses.

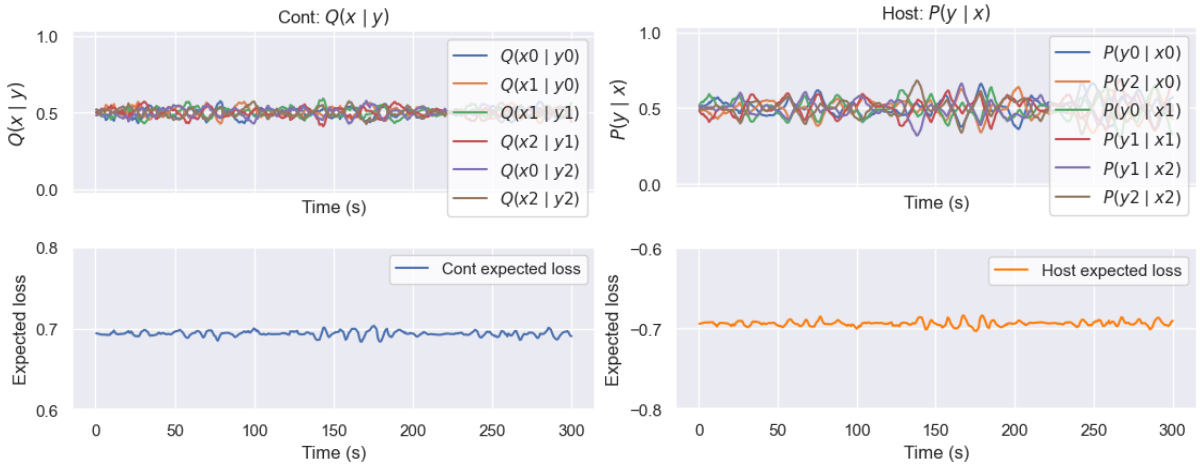


(a) The **contestant**. A Nash equilibrium is found at $\mathbb{E}[L_C] = 0.4$.

(b) The **host**. A Nash equilibrium is found at $\mathbb{E}[L_H] = -0.4$.

Figure 5.11: Q and P over time in **Example H** with zero-sum randomised 0-1 loss. Both players exhibit something we call the "random walk" behaviour; the host shows indifference towards y given x_2. Likewise, the contestant shows indifference towards x given y_3. The expected losses of both players are not dependent on these probabilities.

In addition, figure 5.12 exemplifies the strategies in zero-sum **Example F** with a proper loss function, showing the lightly deviant behaviour like in figure 5.8, but not the large oscillations like figure 5.10.



(a) The **contestant**. A Nash equilibrium is found at $\mathbb{E}[L_C] = 0.693$.

(b) The **host**. A Nash equilibrium is found at $\mathbb{E}[L_H] = -0.693$.

Figure 5.12: Q and P over time in zero-sum **Example F** with logarithmic loss. Players keep very close to the unique Nash equilibrium, and do not show large oscillations.

Other games with zero-sum configurations we did not mention either converged smoothly or suffered the same problems. We explain the deviant and the oscillatory behaviour in the discussion, section 6. Based on that explanation, we argue that exploring strategies for non-zero-sum games can still be explored this way.

5.2.2 Non-zero-sum configurations

For each game, we selected multiple matrices. Let A_0 be the matrix that represents randomised 0-1 loss. From now, we refer to the zero-sum randomised 0-1 loss game as $[A_C = A_0, A_H = -A_0]$. We illustrate notable custom matrices in accordance to our results in tables 5.2, 5.3, 5.4 and 5.5. For each custom matrix A_i, we initially ran the following configurations:

Somewhat competitive to zero-sum

- $[A_C = A_i, A_H = -A_0]$: assign A_i to the contestant and $-A_0$ to the host.
- $[A_C = A_0, A_H = -A_i]$: assign A_0 to the contestant and $-A_i$ to the host.
- $[A_C = A_i, A_H = -A_i]$: assign A_i to the contestant and $-A_i$ to the host. this is a zero-sum game.

Somewhat cooperative to fully cooperative

- $[A_C = A_i, A_H = A_0]$: assign A_i to the contestant and A_0 to the host.
- $[A_C = A_0, A_H = A_i]$: assign A_0 to the contestant and A_i to the host.
- $[A_C = A_i, A_H = A_i]$: assign A_i to both the contestant and the host. This is a fully cooperative game.

This way, we get a general overview of how A_i affects P and Q. Depending on these results, we decide whether to run more configurations for further investigation. Note that the contestant is never assigned a negative loss function. We argue that the contestant, as a decision maker, ought to maximise the number of correct predictions. If the contestant incurs an even smaller (e.g. negative) loss for predicting wrong outcomes, the contestant would lose this primary goal.

Before analysing general-sum configurations, Let us first redefine some useful terms. Let $\mathcal{P}$ be the set of possible host strategies. Consider **Monty Hall** again. Let $\mathcal{P}^*_{A_0,-A_0} \subseteq \mathcal{P}$ define the set of optimal host strategies for $[A_C = A_0, A_H = -A_0]$. Recall that $\mathcal{P}^*_{A_0,-A_0}$ is the infinite set of strategies that are indifferent to y given x_2. The unique RCAR strategy is contained in $\mathcal{P}^*_{A_0,-A_0}$.

As for **Monty Hall**, among numerous matrices that we ran, only the two matrices that exhibit the symmetry property produced some interesting results. They are illustrated by table 5.2. Notably, only matrices in these forms (and their negatives) are symmetric with respect to exchanges in $\mathcal{Y}$.

A_1	x'_1	x'_2	x'_3
x_1	0	1	1
x_2	**2**	0	**2**
x_3	1	1	0

(a) A_1: "Incorrect predictions are especially undesirable if x_2 is the true outcome."

A_2	x'_1	x'_2	x'_3
x_1	0	**2**	1
x_2	1	0	1
x_3	1	**2**	0

(b) A_2: "Incorrectly predicting x_2 is especially undesirable."

Table 5.2: Loss matrices for **Monty Hall**. Both are symmetric with respect to exchanges in $\mathcal{Y}$.

We found that for the non-zero-sum $[A_C = A_0, A_H = -A_i]$, $-A_0$, $-A_1$ and $-A_2$ all appear to produce the same set of host strategies $\mathcal{P}^*_{A_0,-A_0}$, which includes the RCAR strategy. In contrast: for asymmetric matrices, the host picks either y_1 or y_2. Moreover, in the more cooperative case $[A_C = A_0, A_H = A_i]$, A_0, A_1 and A_2 produce the same set of strategies as in the former setting. Again, for asymmetric matrices, the host prefers either y_1 or y_2. For both cases mentioned, the contestant chooses the strategy we recognise as the worst-case optimal strategy for zero-sum **Monty Hall**: never choosing x_2.

Yet, we found that A_1 and A_2 do not produce the same set of strategies for all non-zero-sum cases. For the somewhat cooperative $[A_C = A_i, A_H = A_0]$ and fully cooperative $[A_C = A_i, A_H = A_i]$, A_1 generates P that puts all mass on either y_1 or y_2, therefore excluding P_{RCAR}. Conversely, A_2 still produces $\mathcal{P}^*_{A_0,-A_0}$, including P_{RCAR}.

Interestingly, A_1 starts oscillating for the somewhat competitive $[A_C = A_i, A_H = -A_0]$ and the zero-sum $[A_C = A_i, A_H = -A_i]$. Learning from our observations on oscillatory behaviour in subsection 5.2.1, this hints at a unique Nash equilibrium. Matrix A_2, on the other hand, does not generate oscillatory behaviour and instead produces $\mathcal{P}^*_{A_0,-A_0}$ again.

Note that for A_i symmetric with respect to exchanges in $\mathcal{Y}$, the zero-sum case $[A_C = A_i, A_H = -A_i]$ ought to produce P_{RCAR} due to the theorems by Van Ommen et al. (2016). Contrary, these theoretical guarantees do not cover the other cases that we mentioned. Yet, our empirical observations suggest that RCAR may describe Nash equilibria for a subset of non-zero-sum games, at least for L_C and L_H both symmetric w.r.t. exchanges in $\mathcal{Y}$.

For the graph and matroid games with three outcomes, **Example F** and **Example H**, it holds that except for the uniform matrix, there exists no matrix A_i that is symmetric w.r.t. exchanges in $\mathcal{Y}$. Despite that, we selected and ran numerous other matrices for these games. As mentioned in subsection 5.2.1, **Example F** showed oscillatory behaviour for many of the matrix configurations. Specifically, the agents appear to oscillate around the (potential) equilibrium for competitive matrix configurations. **Example H** mostly converges smoothly, although not always to RCAR strategies. However, there are two distinct matrices that for both games consistently converge

to either P_{RCAR} (in **Example H**) or appear to oscillate around P_{RCAR} (in **Example F**). Table 5.3 illustrates these (asymmetric) matrices.

A_3	x'_1	x'_2	x'_3
x_1	0	**5**	1
x_2	1	0	1
x_3	1	1	0

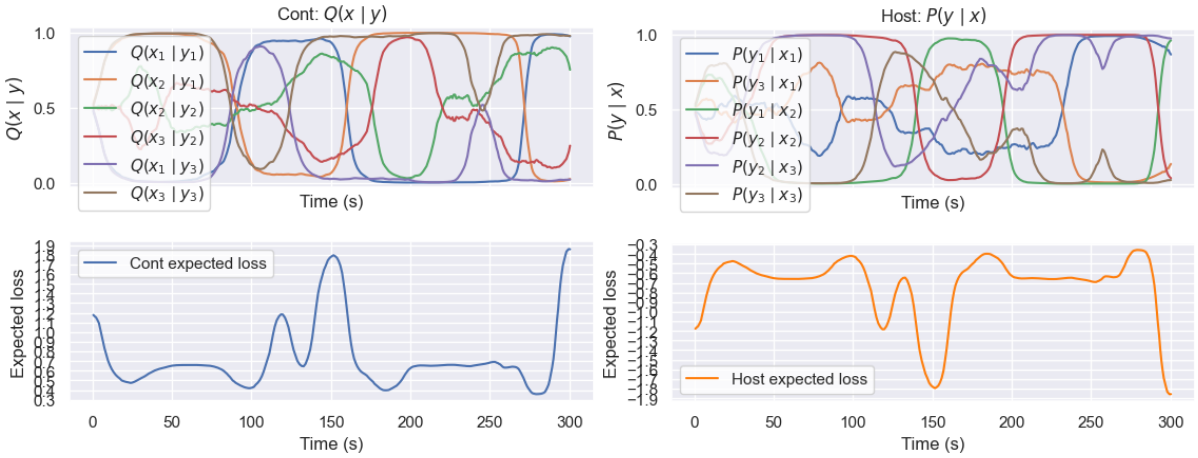
(a) A_3: "Incorrectly predicting x_2 is especially undesirable if x_1 is the true outcome."

A_4	x'_1	x'_2	x'_3
x_1	0	1	1
x_2	**5**	0	1
x_3	**5**	1	0

(b) A_4: "Incorrectly predicting x_1 is especially undesirable."

Table 5.3: Asymmetric loss matrices for **Example F** and **Example H**.

For **Example H**, both matrices generate oscillations for all competitive cases ($[A_C = A_i, A_H = -A_0]$, $[A_C = A_0, A_H = -A_i]$ and the zero-sum $[A_C = A_i, A_H = -A_i]$). But, looking more closely at the behaviour in figure 5.13, we find that no consistent oscillatory pattern is present. This is quite different from the oscillations from the zero-sum case with symmetric loss: $[A_C = A_0, A_H = -A_0]$ (figure 5.10).



(a) The **contestant**.

(b) The **host**.

Figure 5.13: Q and P over time in **Example F** for zero-sum $[A_C = A_4, A_H = -A_4]$. The agents do not converge and the oscillatory behaviour does not seem to be periodically consistent.

From this figure, it is not clear what a Nash equilibrium would look like. A similar image is produced from the zero-sum $[A_C = A_3, A_H = -A_3]$. The non-zero-sum competitive cases also show similar graphs. A further analytical investigation could conclude whether P_{RCAR} characterises worst-case optimality for these specific settings or not. However, the fact that A_3 and A_4 are not symmetric w.r.t. exchanges in $\mathcal{Y}$, along with our currently unconvincing experimental findings, lead us to believe that other configurations might be of more interest to us.

We now outline our results on graph and matroid games **Example C** and **Example G**. Unlike **Example F Example H**, there *do* exist matrices symmetric w.r.t. exchanges in $\mathcal{Y}$ for these games. We selected a set of matrices, some of which symmetric, to strengthen (or perhaps refute) our beliefs that for symmetrical matrices, RCAR characterises optimal strategies for some

general-sum graph and matroid games.

A_5	x'_1	x'_2	x'_3	x'_4
x_1	0	1	**5**	1
x_2	**5**	0	**5**	1
x_3	**5**	1	0	1
x_4	**5**	1	**5**	0

(a) A_5: "Incorrectly predicting either x_1 or x_3 is especially undesirable."

A_6	x'_1	x'_2	x'_3	x'_4
x_1	0	**5**	**5**	**5**
x_2	1	0	1	1
x_3	**5**	**5**	0	**5**
x_4	1	1	1	0

(b) A_6: "Incorrect predictions are especially undesirable if the true outcome is either x_1 or x_3."

A_7	x'_1	x'_2	x'_3	x'_4
x_1	0	1	1	1
x_2	1	0	1	**5**
x_3	1	1	0	1
x_4	1	**5**	1	0

(c) A_7: "Incorrectly predicting x_2 is especially undesirable if x_4 is the true outcome, and vice versa."

A_8	x'_1	x'_2	x'_3	x'_4
x_1	0	**5**	**5**	**5**
x_2	1	0	1	**5**
x_3	**5**	**5**	0	**5**
x_4	1	**5**	1	0

(d) A_8: "Incorrect predictions are especially undesirable if either x_1 or x_3 is the true outcome. Also, incorrectly predicting either either x_2 or x_4 is especially undesirable."

Table 5.4: Loss matrices for **Example C** and **Example G**. These are all symmetric w.r.t. exchanges in $\mathcal{Y}$.

Table 5.4 presents four matrices, A_5, A_6, A_7 and A_8. These matrices are symmetric w.r.t. exchanges in $\mathcal{Y}$ for **Example C** and **Example G**. We found that for both games in the zero-sum case $[A_C = A_i, A_H = -A_i]$, only these four matrices generated either RCAR strategies for the host or oscillated around RCAR. Subsequently, we discovered that they also produced either RCAR strategies or oscillated around RCAR for the somewhat competitive non-zero-sum cases $[A_C = A_i, A_H = -A_0]$ and $[A_C = A_0, A_H = -A_i]$. Other, asymmetrical matrices do not generate these RCAR strategies consistently. For instance, take asymmetrical matrix A_9 (table 5.5) and consider **Example C**. Table 5.6 shows the RCAR strategy P_{RCAR} for **Example C** and optimal strategy $P^*_{A_9}$ for the non-zero-sum competitive $[A_C = A_0, A_H = -A_9]$. $P^*_{A_9}$ appears the unique optimal strategy. If true, then $P_{\text{RCAR}} \notin \mathcal{P}^*_{A_0,-A_9}$. Further inspection finds that indeed, no Nash equilibrium seems to exist for $[A_C = A_0, A_H = -A_9]$ that includes P_{RCAR} in **Example C**.

A_9	x'_1	x'_2	x'_3	x'_4
x_1	0	1	1	1
x_2	1	0	**5**	1
x_3	1	**5**	0	1
x_4	1	1	1	0

(a) A_9: "Incorrectly predicting x_2 is especially undesirable if x_3 is the true outcome, and vice versa."

Table 5.5: An instance of an asymmetrical matrix for **Example C** and/or **Example G**. Note that this matrix actually looks similar to the symmetrical A_7.

$P_{\text{RCAR}}(y \mid x)$	x_1	x_2	x_3	x_4
y_1	1	**1**	0	0
y_2	0	**0**	**0**	0
y_3	0	0	**1**	1

$P^*_{A_9}(y \mid x)$	x_1	x_2	x_3	x_4
y_1	1	**0**	0	0
y_2	0	**1**	**1**	0
y_3	0	0	**0**	1

Table 5.6: Example strategies in **Example C** for the host. The RCAR strategy P_{RCAR} appears optimal for any symmetrical matrix in table 5.4. Yet for asymmetrical matrix $\boldsymbol{A}_9$, $P^*_{A_9}$ is optimal but not P_{RCAR}.

Example D and **Example E** are also games with 4 outcomes, though they are non-graph and non-matroid. The agents seem to reach equilibria easily for cooperative matrix configurations. However just like for the previous examples, RCAR strategies do not appear (consistently) for moderately cooperative games. For competitive configurations, we found just some oscillatory behaviour that hints at RCAR strategies. Though, these observations are again not consistent: an RCAR strategy may appear for $[\boldsymbol{A}_C = \boldsymbol{A}_0, \boldsymbol{A}_H = -\boldsymbol{A}_i]$, but not for $[\boldsymbol{A}_C = \boldsymbol{A}_i, \boldsymbol{A}_H = -\boldsymbol{A}_0]$ and vice versa.

6 Discussion

In this chapter, we discuss and explain the results of our strategy analysis. In interpreting these findings, we attempt to answer our remaining research questions.

6.1 Evolutionarily stable and unstable strategies

Throughout our experiments, we found that PPO converges to Nash equilibria smoothly for some games and losses, while failing for other configurations. Additionally, there were also cases where the agents started walking randomly through the space of probabilities. It appears we encountered evolutionarily stable strategies as well as evolutionarily *unstable* strategies. Maynard Smith (1974) first coined the term evolutionarily stable strategy (ESS). He used it to analyse the evolution of phenotypes in populations. It is a refinement over the regular Nash equilibria; hence it is more strict. Below, we explain our observations in terms of evolutionarily stable strategies.

Consider a traditional bimatrix game between host H and contestant C. Recall from section 3.3 that $u_i(s_i, s_{-i})$ denotes the utility (or equivalently, reward) of player i. Let (s^*_H, s^*_C) be a Nash equilibrium, that is for all players $i \in \{H, C\}$ we have

$$u_i(s^*_i, s^*_{-i}) \geq u_i(s_i, s^*_{-i}) \text{ for all } s_i \neq s^*_i$$

For ease of explanation, let us take best-response host strategy s^*_H. s^*_H is evolutionarily stable if for any other strategy $s_H \in S_H$, *both* these conditions are satisfied:

1. $u_H(s^*_H, s^*_C) > u_H(s_H, s^*_C)$
2. $u_C(s_H, s^*_C) \geq u_C(s_H, s_C)$ for all $s_C \in S_C$

When both (s^*_H, s^*_C) are evolutionarily stable, they are in an evolutionarily stable Nash equilibrium. The first condition implies a *strict* Nash equilibrium; any other strategy is strictly worse than the best response strategy s^*_i. The second condition implies that when player i plays a non-best-response strategy, the other player $-i$ can not receive a better reward by playing some other strategy s_{-i} in the new circumstance.

Figure 5.9 illustrates how PPO learns such equilibria in probability updating games smoothly and can retain them. We found that evolutionarily stable equilibria occur more frequently when P and Q put most probabilities at either 1 or 0. We also found equilibria to be more often evolutionarily stable when the outcome distribution,p_x, is not uniform.

We also found Nash equilibria that do not satisfy the first condition but do satisfy the second. In other words, the equilibrium is not a strict or a non-unique Nash equilibrium. In terms of probability updating games, an infinite number of $P \in \mathcal{P}$ likely produce equivalent equilibria. The result of this is the "random walk" behaviour that we observed in figure 5.11. We see how the contestant and the host seem indifferent given a certain outcome and message, respectively, randomly walking through the space of probabilities. This, without affecting the expected losses of either player. Each one of those strategy pairs is a Nash equilibrium. This is more frequently the case for randomised 0-1 loss and certain randomised matrix loss functions than for proper loss functions, where equilibria generally are more sparse.

Finally, for some probability updating games, the second condition is not satisfied. In other words, the contestant can exploit the new circumstance once the host does not play s_H^*. In this evolutionarily unstable equilibrium, a small change in P causes Q to be sub-optimal given the new situation. While the host tries to correct its mistake by changing back to P, the contestant has probably already exploited the situation, changing Q to any Q' that would have lowered its expected loss. Thus, for practical applications (like RL), unstable equilibria are unlikely to converge. This is exactly what the results prove for probability updating games. We found that for graph and matroid games with uniform p_x (e.g. **Monty Hall**, **Example F**, **Example G**), when RCAR characterises a Nash equilibrium, the equilibrium is often not stable. Figures 5.3, 5.8 and 5.12 exemplify this for Brier and logarithmic loss. We find that for *proper* loss functions in general, insignificant errors in the model can suddenly remove the agents from the equilibrium they previously converged on. But, they can converge back to the optimal strategy pair (P, Q) relatively quickly and afterwards even retain their equilibrium for a while. In contrast, for randomised 0-1 loss (and generally, randomised matrix loss), convergence appears very difficult for unstable strategies. Figures 5.10 and 5.13 depict the learning behaviour for unstable equilibria with such loss functions. The agents oscillate back and forth around the unstable equilibrium, without any sign of convergence.

There is a comfortable explanation as to why convergence is so hard for randomised matrix loss with unstable equilibria. We will explain it through an example. Recall that if a loss function is improper, the expected loss is not minimised for the contestant when it plays $Q = P$. Instead, randomised matrix loss tends to be minimised when Q puts all probability mass on a specific x given y. (For randomised 0-1 loss, this x happens to be the one that maximises $P(x \mid y)$). Now consider zero-sum **Example F** with randomised 0-1 loss again, figure 5.10. In the Nash equilibrium, we have $P(y_1 \mid x_1) = 1/2$. As a result, we get $P(x_1 \mid y_1) = P(x_2 \mid y_1) = 1/2$. For randomised 0-1 loss, the contestant wants to put all probability mass on $\arg\max_{x \in y_1} P(x \mid y_1)$. However, the probabilities are all equal, so we get $Q(x_1 \mid y_1) = Q(x_2 \mid y_1) = 1/2$. Now suppose the host changes to $P(y_1 \mid x_1) = 0.501$. Consequently, we get $P(x_1 \mid y_1) > P(x_2 \mid y_1)$ and therefore $L_C(x, Q)$ is now minimised when the contestant puts all its probability mass on x_1: $Q(x_1 \mid y_1) = 1, Q(x_2 \mid y_1) = 0$. When Q changes that way, P can in turn exploit this by minimising the probability of x_1 given y_1, turning the the tables again. This process then repeats indefinitely, crossing the Nash equilibrium but never converging to it.

A similar example shows why the Nash equilibrium in **Example H** with randomised 0-1 loss *is* evolutionarily stable. Take figure 5.11 again. Given the worst-case optimal strategy P, we have $P(x_1 \mid y_1) = 0.4 < P(x_2 \mid y_1) = 0.6$. Therefore, the contestant (already) puts all his mass on x_2 given y_1. Now suppose the host changes its strategy infinitesimally, e.g. to $P(x_1 \mid y_1) = 0.99$. This does not affect $P(x_1 \mid y_1)$ and $P(x_2 \mid y_1)$ in any drastic way, as $P(x_1 \mid y_1) < P(x_2 \mid y_1)$ will still hold. Thus, the contestant still minimises its expected loss by keeping all its probability mass on x_1.

One interesting observation is that the oscillations seem quite symmetrical. The probabilities move in a wave-like pattern, but the midpoint of that wave, or the average of those probabilities, tend to be the Nash equilibrium. Although, we acknowledge that this observation might be a coincidence, as this may not hold for other game structures and/or loss functions. Nonetheless, the oscillatory movement did give us hints as to where the unstable equilibrium lies, and what the corresponding expected losses may be.

6.2 Nash equilibria in non-zero-sum games

One of our main aims was to learn how Nash equilibria would emerge and look like for non-zero-sum games. In our experiments, we observed that (P and Q) vary greatly depending on the game structure and specific losses L_C and L_H. We divide our findings for non-zero-sum games in two groups, namely

- the somewhat cooperative games $[\boldsymbol{A}_C = \boldsymbol{A}_i, \boldsymbol{A}_H = \boldsymbol{A}_0]$ and $[\boldsymbol{A}_C = \boldsymbol{A}_0, \boldsymbol{A}_H = \boldsymbol{A}_i]$, and
- the somewhat competitive games $[\boldsymbol{A}_C = \boldsymbol{A}_i, \boldsymbol{A}_H = -\boldsymbol{A}_0]$ and $[\boldsymbol{A}_C = \boldsymbol{A}_0, \boldsymbol{A}_H = -\boldsymbol{A}_i]$.

We begin by describing the first case. Traditionally, Nash equilibria tend to be more socially cohesive as games are increasingly cooperative. For probability updating games, we find that the host and the contestant will guide each other towards a state that is more desirable for themselves, however, is often also desirable for the other. Some games like **Example F** and **Example G** sometimes have Nash equilibria where both players incur *zero* expected loss. Without the game being fully-cooperative, the host truly minimises the expected entropy to zero, removing any uncertainty for the contestant. More generally, we observe that moderately cooperative games tend to have Nash equilibria where the host heavily concentrates its probabilities on specific y given x, and vice versa for the contestant. We remark that such strategies can also be RCAR sometimes. In **Example C**, the RCAR strategy is to put all mass on y_1 given x_2 and on y_3 given x_3. However when observing cooperative configurations, the host tends to do the opposite of that; putting zero probability on y_1 given x_2 and on y_3 given x_3. This strategy is not RCAR. There were some exceptions though. In **Monty Hall**, both symmetric matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ (table 5.2) produce optimal strategies that include the RCAR strategy, which is the uniform distribution. Altogether, our results suggest that **Monty Hall** is an exception to the rule, while **Example C** is more representative for graph and matroid games. The general rule seems to be that in Nash equilibria that emerge for moderately cooperative games, the host does not play the RCAR strategy. This leads us to believe that RCAR does *not* necessarily characterise Nash equilibria for somewhat cooperative non-zero-sum games, regardless of the loss function or game structure.

Our experiments indicate something quite different for more competitive games. Naturally, Nash equilibria tend to not be socially optimal for more competitive games. Unless we consider repeated games where players are adequately far-sighted to prefer long-term benefits over short-term rewards. Nonetheless, we designed the experimental setup in such a way that one-shot rewards (losses) are optimised, instead of the aggregate of repeated games. Hence we expect equilibria to arise that correspond to the one-shot game. In section 5.2, we observed that for non-zero-sum graph and/or matroid games with symmetric matrices RCAR strategies can emerge in the Nash equilibria. Expanding upon this, it appeared from our experiments that for the somewhat competitive cases, if the game is graph and/or matroid and if the loss function is symmetric with respect to exchanges in $\mathcal{Y}$, then P_{RCAR} emerges as an optimal strategy for the host. Specifically, if the Nash equilibrium for the given game configuration is a stable one, P_{RCAR} indeed emerges and the agents converge to the Nash equilibrium. Though if the Nash equilibrium is non-unique, some optimal $P^* \in \mathcal{P}^*$ would arise but would be different in each trial. The variance in the numerous trials hint at a set $\mathcal{P}^*$ in which, indeed, $P_{\text{RCAR}} \in \mathcal{P}^*$. But if the Nash equilibrium is an unstable one, the agents do not converge. Instead, the probabilities in P and Q oscillate symmetrically around a midpoint. Interestingly, this midpoint is P_{RCAR} for the host, which might seem like a coincidence at first sight. However, using the theory from 6.1, it seems reasonable that P_{RCAR} indeed characterises such unstable Nash equilibria.

Summarising, we suggest suggests that RCAR may characterise Nash equilibria in non-zero-sum

graph and matroid games with randomised matrix loss, setting $[\boldsymbol{A}_C = \boldsymbol{A}_i, \boldsymbol{A}_H = -\boldsymbol{A}_0]$ and $[\boldsymbol{A}_C = \boldsymbol{A}_0, \boldsymbol{A}_H = -\boldsymbol{A}_i]$, where $\boldsymbol{A}_0$ denotes the matrix for randomised 0-1 loss and $\boldsymbol{A}_i$ is symmetric w.r.t. exchanges in $\mathcal{Y}$.

6.3 Nash equilibria in repeated games

Following the literature review on repeated games in section 3.4, we concluded that *if* RCAR characterises worst-case optimality for a one-shot probability updating game, it will also characterise a Nash equilibrium in the finitely repeated version of that game. Or more specifically, if (P_{RCAR}, Q^*) is a Nash equilibrium in the one-shot game, then the strategy pair where players play (P_{RCAR}, Q^*) for T times is a Nash equilibrium in the T-period repeated game. Here, Q^* denotes a best-response strategy for the contestant given P_{RCAR}.

Conversely, for infinitely repeated games, the folk theorems hint at a large set of Nash equilibria but do not necessarily characterise them. It is to our knowledge very difficult to experimentally investigate optimal strategies for infinite games. In contrast to regular reinforcement learning, one would have to either ensure some infinitely far-sighted behaviour for both players or run the algorithm for an infinite amount of time; both approaches seem impossible. However, we *can* conclude from the folk theorems that the payoff profile (or, analogously for probability updating games: the expected losses of the players) that correspond to RCAR, are also payoff profiles in a Nash equilibrium for the repeated game. This does hint at a very high probability that RCAR also characterises Nash equilibria for infinitely repeated zero-sum probability updating games.

6.4 Limitations and open questions

On observing the experimental data, we recognised that our learning approach cannot converge to every Nash equilibrium; some game configurations have evolutionarily unstable Nash equilibria. Our learning method, like many practical applications, was not able to reach and retain such strategy pairs. For our investigation and our particular cases, this turned out to not be a major problem. In future research, it is important to understand that there are indeed many unstable Nash equilibria in probability updating games. To our knowledge, there exists no solution for this problem.

Furthermore, we acknowledge that in our experimental investigation we use a limited amount of matrices to confirm our conclusions. Hence, there is a possibility that certain configurations were missed that would have made us draw different conclusions. Also, whether RCAR characterises optimality when just *one* matrix is symmetric w.r.t. exchanges in $\mathcal{Y}$ remains unknown still. In addition, we concluded that RCAR characterises optimality for "more competitive" cases, but how are these cases defined? And when is a game competitive enough for this rule to hold? What about if the game has both cooperative *and* competitive aspects? Such games become entirely possible when using randomised matrix loss. Could P then be *partially* RCAR for it to characterise optimality? These are all questions that could have been answered if more different matrix and game configurations had been explored.

On the other hand, running each matrix, for each matrix configuration, for an adequate time is very time and energy-consuming. This investigation was already at the limit of its time constraint. Hence, examining more configurations could in reality have diminished the overall quality of this research. In line with that, one limiting factor was the limited CPU. If we had access to more CPUs and many more CPU cores, we could have parallelised learning such that

many configurations could have been learned side by side.

We investigated non-zero-sum graph and matroid games with symmetric loss. However, graph and matroid games are not the only games for which RCAR characterises worst-case optimality. Section 3.2.3 describes how for any traditional probability updating game with logarithmic L, RCAR also characterises Nash equilibria. One may ask, how are Nash equilibria described in non-zero-sum games with logarithmic loss? This research question did not fit within the scope of this study. A future study could look at making logarithmic games non-zero-sum by applying affine transformations on L to create a different loss function while preserving the required local and proper properties.

Additionally, another matter that fell outside of our scope was the relaxation of both the zero-sum and the one-shot constraint together. Namely, to investigate whether RCAR characterises Nash equilibria for repeated, non-zero-sum probability updating games. Such strategies can probably be investigated experimentally for finitely repeated games. The idea would then be to use the same MARL setup, but instead of giving the model feedback after each game turn (the one-shot rewards), we let the game run for T periods, where T is adequately large. After T turns, the players receive the total rewards as defined in section 3.4. This would stimulate the model to look for strategies with long-term benefits and may produce strategies that are more socially optimal. Here, it is important to incorporate a sense of time for the agents. This could be done by adding the current turn number to the observation space. All things considered, we do believe that Nash equilibria in these repeated games are much more difficult to learn than one-shot games. This is because the game will get significantly more non-stationary and complex as a consequence of letting players determine a strategy for each round.

7 Conclusion

In this study, we investigated Nash equilibria in relaxed probability updating games. A probability updating game is a simultaneous, zero-sum game between a host and a contestant. Here, the host sees a true outcome and sends a message (i.e. a subset of the possible outcomes) to the contestant, according to its strategy P. The contestant predicts, according to its strategy Q, which outcome in the subset is the actual true outcome. After P and Q are chosen, player rewards are determined by calculating the expected loss over all possible outcomes and messages. We relaxed either the zero-sum constraint or the one-shot constraint from this game. Our research questions regarded whether the RCAR condition, which characterises worst-case optimality (and thus Nash equilibria) for traditional probability updating games, still characterises Nash equilibria for relaxed probability updating games.

To investigate Nash equilibria for non-zero-sum games, we applied Proximal Policy Optimisation (PPO) to learn optimal strategies empirically. A probability updating game has an unusual property that not many games show; the action sets of the players contain categorical distributions. Hence, we set PPO to learn parameterised Dirichlet distributions to learn player strategies efficiently. We implemented PPO in a fully decentralised manner, allowing no communication between the two agents. To make the game non-zero-sum, we added an independent loss function for the host. We applied randomised matrix loss, in which we assigned dissimilar matrices to the players.

Our findings suggest that RCAR very likely characterises Nash equilibria for non-zero-sum graph or matroid games, where both matrices are symmetric w.r.t. exchanges in the message structure and the game is moderately competitive. However, if instead, we assign matrices to the players such that the game becomes more cooperative, RCAR does not characterise Nash equilibria.

We also reviewed the Nash folk theorems on repeated games and consequently deduced that if RCAR characterises optimality for a certain one-shot game, then RCAR also characterises a Nash equilibrium for the finitely repeated game. In addition, we deduced that RCAR then also very likely characterises a Nash equilibrium for the infinitely repeated game: because the expected loss that the contestant incurs, corresponding to a Nash equilibrium for the one-shot game, matches with the expected loss that corresponds to a Nash equilibrium of the infinitely repeated game.

A future investigation could investigate for which specific type of non-zero-sum competitive games RCAR characterises optimality and for which it does not. Another subject would be to examine if this finding also holds for non-zero-sum logarithmic games, which can be created by utilising affine transformations. A potential major advance would be to develop theoretical guarantees for RCAR characterising Nash equilibria in specific non-zero-sum games. It also remains an open question how Nash equilibria are characterised for infinitely repeated games. Finally, it would be interesting to investigate optimal strategies in general-sum repeated probability updating games, combining the two relaxations.

A Hyperparameters

General hyperparameters and settings		
Variable	**Description**	**Value**
`train_batch_size`	SGD training batch size	64
`fc_net`	Fully connected network shape	256×256
PPO-specific		
Variable	**Description**	**Value**
`sgd_minibatch_size`	SGD training minibatch size	8
`num_sgd_iter`	Number of epochs per training batch	1
`lr`	Learning rate or stepsize of SGD	8×10^{-5}
`clip_param`	PPO clip parameter	0.1
`vf_clip_param`	PPO value function clip parameter	0.1
A2C-specific		
Variable	**Description**	**Value**
`min_iter_time`	Minimal time (s) per iteration	1
DDPG- & TD3-specific		
Variable	**Description**	**Value**
`eval_num_episodes`	Number of episodes for evaluation	1
`random_timesteps`	Random timesteps from start	40
`timesteps_per_iter`	Timesteps per iteration	40
`timesteps_per_iter`	Timesteps per iteration	40
`learning_starts`	Model steps before learning starts	40
SAC-specific		
Variable	**Description**	**Value**
`timesteps_per_iter`	Amount of timesteps per iteration	16
`learning_starts`	Model steps before learning starts	500
`min_iter_time_s`	Minimal time (s) per iteration	1

Table A.1: List of hyperparameters. Variable names are according to the Ray RLlib framework. We only included parameters that we modified. We kept the others at their default configuration from RLlib's implementation: https://docs.ray.io/en/latest/rllib-algorithms.html.

B Code repository

We have open-sourced all of our code for research and reproduction purposes. The code repository can be found here: https://github.com/caldaibis/probability-updating-games-marl. It comprises several Python modules: one containing all logic for the general-sum probability updating game, one for learning on such games and one that visualises the resulting data. The main module, however, is the user endpoint. Note that it needs to be called with a certain set of parameters. For instructions on how to install and run the application, see README.md.

Bibliography

Aumann, R. J., & Shapley, L. S. (2013). Long-term competition - A game-theoretic analysis. *Annals of Economics and Finance*, *14*(2 B). https://doi.org/10.1007/978-1-4612-2648-2{_}1

Benoit, J. P., & Krishna, V. (1987). Nash equilibria of finitely repeated games. *International Journal of Game Theory*, *16*(3). https://doi.org/10.1007/BF01756291

Bernardo, J. M. (1979). *Expected Information as Expected Utility* (tech. rep. No. 3).

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym. *arXiv*. http://arxiv.org/abs/1606.01540

Chen, X., Deng, X., & Teng, S. H. (2009). Settling the complexity of computing two-player Nash equilibria. *Journal of the ACM*, *56*(3). https://doi.org/10.1145/1516512.1516516

Friedman, J. W. (1971). A non-cooperative equilibrium for supergames. *Review of Economic Studies*, *38*(1), 1–12. https://doi.org/10.2307/2296617

Fudenberg, D., & Maskin, E. (1986). The Folk Theorem in Repeated Games with Discounting or with Incomplete Information. *Econometrica*, *54*(3). https://doi.org/10.2307/1911307

Gill, R. D., van der Laan, M. J., & Robins, J. M. (1997). Coarsening at Random: Characterizations, Conjectures, Counter-Examples. Springer, New York, NY. https://doi.org/10.1007/978-1-4684-6316-3{_}14

Grünwald, P. D., & Dawid, A. P. (2004). Game theory, maximum entropy, minimum discrepancy and robust Bayesian decision theory. *The Annals of Statistics*, *32*(4), 1367–1433. https://doi.org/10.1214/009053604000000553

Grünwald, P. D., & Halpern, J. Y. (2003). Updating probabilities. *Journal of Artificial Intelligence Research*, *19*, 243–278. https://doi.org/10.1613/jair.1164

Gupta, J. K., Egorov, M., & Kochenderfer, M. (2017). Cooperative Multi-agent Control Using Deep Reinforcement Learning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *10642 LNAI*, 66–83. https://doi.org/10.1007/978-3-319-71682-4{_}5

Heitjan, D. F., & Rubin, D. B. (1991). *Ignorability and Coarse Data* (tech. rep. No. 4). https://www.jstor.org/stable/2241929

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2018). Deep Reinforcement Learning That Matters. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1). https://ojs.aaai.org/index.php/AAAI/article/view/11694

Hu, J., & Wellman, M. P. (2003). Nash Q-Learning for General-Sum Stochastic Games. *Journal of Machine Learning Research*, *4*, 1039–1069.

Johnson, D. S. (2007). The NP-completeness column: Finding needles in haystacks. *ACM Transactions on Algorithms*, *3*(2). https://doi.org/10.1145/1240233.1240247

Larsen, T. N., Teigen, H. Ø., Laache, T., Varagnolo, D., & Rasheed, A. (2021). Comparing Deep Reinforcement Learning Algorithms' Ability to Safely Navigate Challenging Waters. *Frontiers in Robotics and AI*, *8*. https://doi.org/10.3389/FROBT.2021.738113

Liang, E., Liaw, R., Moritz, P., Nishihara, R., Fox, R., Goldberg, K., Gonzalez, J. E., Jordan, M. I., & Stoica, I. (2017). RLlib: Abstractions for Distributed Reinforcement Learning. *35th International Conference on Machine Learning, ICML 2018*, *7*, 4768–4780. https://arxiv.org/abs/1712.09381v4

Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., & Stoica, I. (2018). Tune: A Research Platform for Distributed Model Selection and Training. https://arxiv.org/abs/1807.05118v1

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., & Mordatch, I. (2017). Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in Neural Information Processing Systems*, *2017-December*.

Maynard Smith, J. (1974). The theory of games and the evolution of animal conflicts. *Journal of Theoretical Biology*, *47*(1). https://doi.org/10.1016/0022-5193(74)90110-6

Moore, E. F. (2016). Gedanken-Experiments on Sequential Machines. *Automata studies. (am-34)* (pp. 129–154). Princeton University Press. https://doi.org/10.1515/9781400882618-006

Mordatch, I., & Abbeel, P. (2018). Emergence of grounded compositional language in multi-agent populations. *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*.

Muthoo, A., Osborne, M. J., & Rubinstein, A. (1996). A Course in Game Theory. *Economica*, *63*(249). https://doi.org/10.2307/2554642

Nash, J. (1951). Non-Cooperative Games. *The Annals of Mathematics*, *54*(2), 286. https://doi.org/10.2307/1969529

OpenAI. (2017). Proximal Policy Optimization. https://openai.com/blog/openai-baselines-ppo/

Rubin, D. B. (1976). Inference and missing data. *Biometrika*, *63*(3), 581–592. https://doi.org/10.1093/biomet/63.3.581

Savant, M. V. (1990). Game Show Problem. *PARADE*. https://web.archive.org/web/20130121183432/http://marilynvossavant.com/game-show-problem/

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust Region Policy Optimization. https://proceedings.mlr.press/v37/schulman15.html

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Openai, O. K. (2017). Proximal Policy Optimization Algorithms. https://arxiv.org/abs/1707.06347v2

Selvin, S. (1975). Letters to the Editor. *The American Statistician*, *29*(1), 67–71. https://doi.org/10.1080/00031305.1975.10479121

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Van Den Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature*, *550*(7676). https://doi.org/10.1038/nature24270

Singh, S., Kearns, M., & Mansour, Y. (2000). Nash Convergence of Gradient Dynamics in General-Sum Games. *UNCERTAINTY IN ARTIFICIAL INTELLIGENCE PROCEEDINGS*, 541.

Sutton, R., & Barto, A. (2018). *Reinforcement learning: An introduction* (Second edition). The MIT Press.

Terry, J. K., Black, B., Grammel, N., Jayakumar, M., Hari, A., Sullivan, R., Santos, L., Perez, R., Horsch, C., Dieffendahl, C., Williams, N. L., Lokesh, Y., & Ravi, P. (2020). PettingZoo: Gym for Multi-Agent Reinforcement Learning. https://arxiv.org/abs/2009.14471v7

Terry, J. K., Black, B., & Hari, A. (2020). SuperSuit: Simple Microwrappers for Reinforcement Learning Environments. https://arxiv.org/abs/2008.08932v1

Terry, J. K., Hari, A., Black, B., Grammel, N., Santos, L., & Manocha, D. (2020). Parameter Sharing is Surprisingly Useful for Multi-Agent Deep Reinforcement Learning. *arXiv*.

The Ray Team. (2021). Ray RLlib Available Algorithms. https://docs.ray.io/en/master/rllib-algorithms.html#maddpg

Van Ommen, T. (2015). *Better predictions when models are wrong or underspecified* (Doctoral dissertation). Mathematical Institute, Faculty of Science, Leiden.

Van Ommen, T., Koolen, W. M., Feenstra, T. E., & Grünwald, P. D. (2016). Robust probability updating. *International Journal of Approximate Reasoning*, *74*, 30–57. https://doi.org/10.1016/j.ijar.2016.03.001

v. Neumann, J. (1928). Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, *100*(1), 295–320. https://doi.org/10.1007/BF01448847

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, *8*(3-4), 229–256. https://doi.org/10.1007/BF00992696