Speeding up Nash Equilibrium finding with population-based FoReL

Timothé Boulet
Aaron Marciano
Axel Nguyen–Kerbel
Clement Wang

CentraleSupélec × Google DeepMind

Nov 2023 - Apr 2024

University supervisor:
Hana Baili
hana.baili@centralesupelec.fr

Company supervisor:
Paul Muller
paul.fm.muller@gmail.com

Abstract

This project centers on the challenge of finding Nash equilibrium strategies within zero-sum games with two players. Robust theoretical techniques, like *Follow the regularized leader* (FoReL), have emerged to tackle this challenge, proving effective even in intricate games like Stratego, which has an immense state space of 10^{535}. Concurrently, Population-Based methods have shown promise in resolving complex Multi-Agent Reinforcement Learning issues. Our focus lies in leveraging population-based strategies to accelerate the convergence of FoReL. Specifically, we introduce a novel algorithm, Population Alternating Lyapunov FoReL (PAL-FoReL), which adheres to the dynamics of FoReL while taking advantage of its trajectory population to improve convergence speed. Our efforts were concentrated on Normal form games to design and test our algorithm. However, scalability was a central consideration, ensuring that PAL-FoReL could easily transition to more complex environments. This required meticulous design to mitigate the excessive computational burden and adaptability to complex scenarios, including imperfect information and extensive form games. PAL-FoReL exhibited significant performance superiority over preceding algorithms across a spectrum of tested Normal form games, including Kuhn Poker. Our code can be accessed in the GitHub repository: https://github.com/tboulet/Algorithms-for-Normal-Form-Games.

Acknowledgement

We would like to express our sincere gratitude to the following individuals and organizations for their invaluable support and contributions to our project:

- Paul Muller, research scientist at Google DeepMind, for serving as our supervisor throughout the project. His guidance, expertise, and willingness to explain complex theoretical concepts were instrumental in our progress. We are grateful for his continuous support, encouragement, and the freedom he provided us to explore various avenues.
- Hana Baili, our university supervisor, for her support throughout the duration of our project. Her feedback has been invaluable in shaping our research endeavors.
- Google DeepMind for generously supporting our project and providing resources that facilitated our research. We are grateful for the opportunity to collaborate and for hosting several of our meetings at their facilities.
- CentraleSupelec, our university, for offering us the opportunity to pursue this captivating project as part of our curriculum. We appreciate the academic environment that fostered our intellectual growth and provided the necessary resources for our work.
- Hub IA and Automatants, the AI student organization at CentraleSupélec, for providing us with access to a cluster where we could conduct our experiments. Their support was instrumental in the execution of our research activities.

Contents

1 Introduction

In the last decades, convergence in adversarial multi-agents games has been a colossal challenge in the field of Multi-Agent Reinforcement Learning (MARL). The Follow the Regularized Leader (FoReL) algorithm has been a popular choice to solve those games, because of its theoretically proven convergence of time-averaged policies to a Nash equilibrium [1]. However, for large games, where the policies are modelized by neural networks, averaging the policies is extremely costly or unfeasible, making FoReL impractical. In 2021, Perolat et al. [2] proposed an improvement of FoReL that theoretically guarantees convergence of the policy itself. This algorithm, which we will call Iterated Lyapunov FoReL (IL-FoReL), has been used as the basis for more complex algorithms to solve complex games such as Stratego [3].

In parallel, Population-Based methods have been used to solve very complex MARL problems such as Starcraft [4] or the first-person shooter game Quake III [5]. The idea is to maintain a population of agents and to update policies based on the performance of the whole population. This allows the algorithm to explore a larger space of policies, which is particularly useful in the case of adversarial games, where a policy is considered good when it is robust, i.e. it performs well against a wide range of opponents.

The objective of this project is to improve the convergence speed of FoReL algorithms thanks to population-based methods that have proven to be particularly useful for this.

Our code can be accessed in the GitHub repository:
https://github.com/tboulet/Algorithms-for-Normal-Form-Games.

2 Background Information

2.1 Game Theory

Multi-agent reinforcement learning studies reinforcement learning methods applied to cases where several agents interact, be it competitively or collaboratively, in order to maximize their own profit. This falls under the scheme of Game Theory [6][7]. We introduce here the main concepts of Game Theory, namely strategy, regret, Nash Equilibrium and Best Response, which will be at the heart of this project. We also focus on the notion of Normal Form Games which are the most simple form of games, and on which the project is focused.

2.1.1 Normal Form Games (NFG)

This project will give a lot of attention to normal-form games. A **normal-form game (NFG)** is a one-step game that is represented using a matrix. It can be defined as a tuple (Π, u, K) where $\Pi = (\Pi^1, \dots, \Pi^K)$ is the set of policies or strategies of the players K and $u : \Pi \to \mathbb{R}^K$ is a utility function. Let us denote

$$\{\mathcal{A}^1, \dots, \mathcal{A}^K\}$$

the action sets of the K players in the game. The payoff function

$$u : \prod_{k=1}^{K} \mathcal{A}^k \to \mathbb{R}^K$$

returns a numerical utility to a vector of K actions. Let us denote π^k the mixed strategy of the k-th player. π^k is a vector of a size corresponding to the number of actions available to the k-th player. We call a **strategy profile** the vector

$$(\pi^1, \dots, \pi^K)$$

Given a strategy profile, the expected utility for player k is

$$\overline{u}^k(\pi) = E_\pi[u^k(a) \mid a \sim \pi]$$

2.1.2 From NFG to EFG

The term **extensive form game (EFG)** extends the definition of NFG to sequential multi-step games. In practice, it makes more sense to model complex games such as highly sequential games as EFG, which introduce a notion of state contextualization, where the policy output also depends on the observation of the agent. However, it is always possible (although not adapted for large games) to model an EFG as an NFG.

In the NFG representation of an EFG, the set of actions becomes the entire set of possible plans decided in advance by the agent. For example, Kuhn Poker, which in its EFG form consists of around 5 different observations with 2 actions each, for each player, becomes an NFG game with 32 actions for player 1 and 64 actions for player 2 (but only 1 "state").

In this project, we chose to focus on the NFG framework, as this is a strong theoretical framework for studying algorithm behavior, but we kept in mind that the final application of our research would be on EFG games. Discussion about the extension of our methods to more general games can be found in Section 5.2.

2.1.3 Nash Equilibrium

A Nash equilibrium is a situation where every player is playing optimally with regard to the opponents' strategies. In a Nash Equilibrium, each player is playing the best response to opponents. A Nash Equilibrium corresponds to a profile of robust strategies and acted convergence players to a rational behavior. For those reasons, Nash Equilibrium is considered as the objective in Game Theory.

The **best response** is defined as such:

$$BR^k(\pi^{-k}) = \arg\max_{\pi^k} \overline{u}^k((\pi_k, \pi^{-k}))$$

where π^{-k} denotes the mixed strategies of the other players. A profile π_* is said to be a **Nash Equilibrium** if

$$\pi_*^k = BR^k(\pi_*^{-k})$$

for all players k.

To evaluate policies, a commonly used convergence metric is the **Nash convergence metric**. Intuitively, it measures the distance of a policy π from the Nash equilibrium. It is defined as:

$$\text{NashConv}(\pi) = \sum_k \overline{u}^k(\text{BR}^k(\pi^{-k}), \pi^{-k}) - \overline{u}^k(\pi)$$

For a given player k, we denote the set of mixed strategies as:

$$\chi_k = \Delta(A_k)$$

2.1.4 Regret

Let us now define the **regret**. Intuitively, the regret indicates the difference between a given utility or a given payoff, and the higher possible values of this utility if the considered player had played other policies instead of the one we currently consider. Unless otherwise stated, it is the average payoff difference between the player's

mixed strategy at a time t and the player's best strategy. At a given time, for a given sequence of mixed strategies $x(t)$, regret is defined as:

$$Reg_k(\tilde{t}) = \max_{p_k \in \chi_k} \left\{ \frac{1}{\tilde{t}} \int_0^{\tilde{t}} [u_k(p_k; x_{-k}(s)) - u_k(x(s))]\, ds \right\}$$

Thus, a player seeks to minimize his regret. A player k is said to have *no regret* under $x(t)$ if:

$$\limsup_{t \to \infty} Reg_k(t) \leq 0$$

2.2 Literature review

FoReL and Poincaré recurrent trajectories A well-known scheme that is often applied to zero-sum normal form games is the *Follow the Regularized Leader* algorithm (FoReL) [1]. It is an exploration-exploitation algorithm that seeks to maximize the player's cumulative pay-off (exploitation) minus a regularization term (exploration). FoReL is renowned for its characteristic of no regret.

The dynamics of the *Follow the Regularized Leader* algorithm (FoReL) are described by the following equations:

$$\mathbf{y}_k(t) = \int_0^t \overline{u}^k(\pi(s))\, ds$$

$$\pi^k(t) = \text{argmax}_{\pi^k \in \chi_k} \left(\langle \mathbf{y}_k(t), \pi^k \rangle - H(\pi^k) \right)$$

Here, χ_k represents the strategy space of the player k, $H(\pi^k)$ denotes the entropy function of the strategy vector π^k, and $\langle \cdot, \cdot \rangle$ denotes the operation of the inner product. Intuitively, the entropy function is minimized by the uniform distribution so the entropy term biases the argmax towards the uniform distribution to allow the exploration part.

In the case of the entropy regularizer, FoReL corresponds to the special case of Replicator Dynamics. Replicator Dynamics is described with this differential equation :

$$\frac{d}{dt}\pi^k(a) = \pi^k(a) \left[u(a, \pi) - \overline{u}^k(\pi) \right]$$

Throughout the rest of the report, we will present various algorithms based on FoReL. When clarity on the version is required, we will refer to the fundamental form as *vanilla FoReL*.

A trajectory is **Poincaré recurrent** if it returns arbitrarily close to itself infinitely many times in the future. In other words, every point of the trajectory is an accumulation point of the trajectory.

Mertikopoulos et al. [1] offered two powerful theorems on the trajectories of the policies taken by a FoReL algorithm:

Theorem 1. *Let Γ be a 2-player zero-sum game that admits an interior Nash equilibrium. Then, almost every solution trajectory of FoReL is Poincaré recurrent. Specifically, for (Lebesgue) almost every initial condition $\pi(0) \in \chi$, there exists an increasing sequence of times $t_n \to \infty$ such that $\pi(t_n) \to \pi(0)$.*

Proof. The proof involves several intricate steps. Refer to Mertikopoulos et al. [1] for a comprehensive understanding.

1. It establishes the incompressibility of the dynamics of the score sequence $y(t)$, ensuring that the volume of initial conditions remains constant over time. This property, coupled with Poincaré's recurrence theorem, guarantees recurrence if the solution orbit $y(t)$ remains within a compact set for all $t \geq 0$.

2. To prevent solutions from escaping to infinity, a transformed system based on score differences is introduced. Although the level sets of this system's coupling $G(y) = [h^*(y_i) - y_i, x_i^*]$ are unbounded, the score transformation described ensures compactness, ultimately leading to recurrence.

□

Theorem 2. *Let Γ be a two-player zero-sum game that does not admit an interior Nash equilibrium. Then, for every initial condition of FoReL, the induced trajectory of $\pi(t)$ converges to the boundary of χ. Specifically, if π^* is a Nash equilibrium of Γ with maximal support, $\pi(t)$ converges to the relative interior of the face of π spanned by supp(π^*).*

Proof. Similar to the preceding theorem, we only depict the basic understanding of the proof. For a comprehensive understanding, refer to Mertikopoulos et al. [1].

1. It can be established that the coupling $G(y) = \sum_{i \in N}[h_i^*(y_i) - y_i, x_i^*]$ strictly increases under (FoReL) for certain conditions, ensuring boundedness of the dynamics.

2. By analyzing the dynamics of the primal-dual coupling $L(t) = G(y(t))$, it is shown that it is Lipschitz continuous and exhibits a finite limit, indicating convergence.

3. By further analyzing the properties of $L(t)$, it is concluded that any limit point of the dynamics must lie on the boundary of the strategy space, as the game does not admit an interior equilibrium.

□

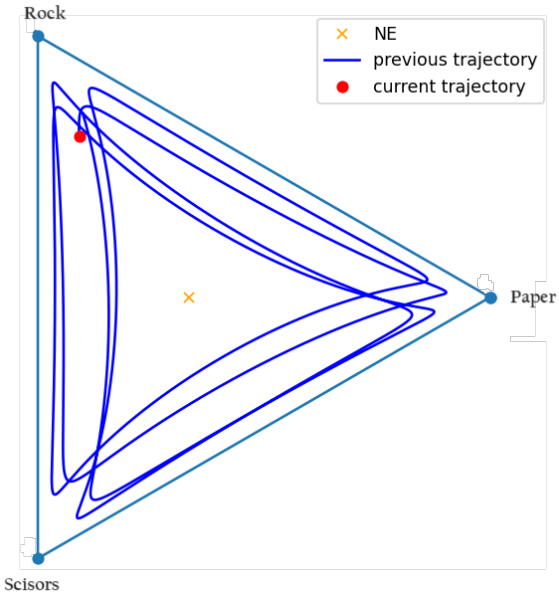


Figure 2: Example of Poincaré recurrent trajectory: FoReL on Rock-Paper-Scissors

Converging with FoReL. Perolat et al. [2] introduced a method to make the FoReL algorithm with entropy regularization converge with a simple reward transformation.

The intuition behind this approach stems from the previous theorems, which demonstrated that a reward independent of policies inevitably leads to a Poincaré recurrent trajectory. To induce convergence in the trajectory, the expected payoff has to incorporate a term that makes the Nash policy attractive. In simpler terms, we aim to introduce a "friction" component to the trajectory to prevent it from cycling endlessly. This is achieved through the following reward transformation:

$$R_\pi^k(a^k, a^{-k}) = r^k(a^k, a^{-k}) - \eta \log\left(\frac{\pi^k(a^k)}{\mu^k(a^k)}\right) + \eta \log\left(\frac{\pi^{-k}(a^{-k})}{\mu^{-k}(a^{-k})}\right)$$

where μ is a reference distribution over the actions and η is a scaling factor. Given specific assumptions outlined in Perolat et al. [2], which are validated in our games, the reward transformation guarantees exponential convergence rates. While we refrain from delving into the specifics here, all details are thoroughly provided in Perolat et al. [2].

However, since this method alters the reward function, it consequently modifies the game, thereby slightly changing the Nash policy as well. Perolat et al. [2] introduced two algorithms aimed at determining a Nash policy of the original game.

Decaying Lyapunov FoReL (DL-FoReL) (Alg.1). When $\eta = 0$ in the reward transformation, the game returns to its original state. Consequently, we could solve

the Replicator Dynamics equation with $\eta \to 0$ to identify the Nash policy of the original game. However, in practice, this algorithm exhibits limitations as it stops learning after a predefined number of iterations, making it challenging to apply to complex games.

Algorithm 1 Decaying Lyapunov FoReL (DL-FoReL)

Initialize $n_{iteration}$, the total number of iterations
Initialize η_{start}, the initial value of η
Initialize π
$\mu \leftarrow \pi$
$\eta \leftarrow \eta_{start}$
$\eta_{end} \leftarrow 0$
for iteration **in** 1 **to** $n_{iteration}$ **do**
  $Q \leftarrow estimate\_utility\_with\_reward\_transformation(\pi, \eta, \mu)$
  $\pi \leftarrow update\_pi(\pi, Q)$
  $\eta \leftarrow \eta_{start} + (\eta_{end} - \eta_{start}) \times \frac{\text{iteration}}{n_{iteration}}$
end for

Iterated Lyapunov FoReL (IL-FoReL) (Alg.2). Unlike DL-FoReL, this algorithm does not involve decaying η. Instead, it focuses on correcting the reward transformation by iteratively updating μ to match the current π after a certain number of steps. In a way, IL-FoReL can be viewed as a fixed-point iteration algorithm, where we sequentially update μ to π_k to obtain π_{k+1}. For a detailed proof of the algorithm's convergence, please refer to Perolat et al. [2]. In practice, IL-FoReL demonstrated superior convergence performance and exhibited greater robustness to variations in its hyperparameters.

Algorithm 2 Iterated Lyapunov FoReL (IL-FoReL)

Initialize $n_{iteration}$, the total number of iterations
Initialize n^L_{steps}, the number of Lyapunov steps between each update of μ
Initialize η
Initialize π
$\mu \leftarrow \pi$
for iteration **in** 1 **to** $n_{iteration}$ **do**
  for step **in** 1 **to** n^L_{steps} **do**
    $Q \leftarrow estimate\_utility\_with\_reward\_transformation(\pi, \eta, \mu)$
    $\pi \leftarrow update\_pi(\pi, Q)$
  end for
  $\mu \leftarrow \pi$
end for

Scaling up FoReL to more complex games Neural Replicator Dynamics (NeuRD) [8] is the application of Replicator Dynamics equations along with policy gradient. This association results in a powerful approach that benefits from both the theory behind FoReL and the function approximation capabilities of neural networks. It also unlocks the possibility of using FoReL in imperfect information games (IIG), such as the Kuhn Poker in [2]. Similar theorems to those found in Normal Form Games were also established for IIG in [2]. Particularly noteworthy is its application in mastering Stratego, as illustrated in [3], an IIG with an astronomical state space of 10^{535} states, surpassing even the complexity of Go, which stands at 10^{360} states.

Population based algorithms Population-based algorithms, such as Fictitious Play and Leagues, offer dynamic frameworks for learning and adapting strategies in multiagent environments. Fictitious Play, introduced by Brown in 1951 [9], involves players repeatedly engaging in a game and adjusting their strategies based on opponents' average strategies. This approach is guaranteed to converge towards a Nash equilibrium in two-player zero-sum games. The essence of Fictitious Play lies in its utilization of a growing population of agents to explore the policy space within a game. At each iteration, agents select actions based on their opponents' average strategies. This iterative process enables agents to gradually refine their strategies over time, with the ultimate goal of finding a Nash policy capable of maximizing its reward against all other policies on average. Leagues, on the other hand, involve organizing agents into groups or "leagues" where they compete and learn from each other's strategies.

In the context of Self-Play, these population-based algorithms can be instrumental in enhancing learning and convergence speed. In Fictitious Self-Play [10], by incorporating Fictitious Play to Self-Play, agents can explore a wider range of strategies by interacting with a growing population of opponents, leading to more diverse gameplay experiences and potentially faster convergence to optimal strategies.

Leagues for more complex games For more complex games like Starcraft II even fictitious self-play techniques can be insufficient to produce strong agents. To address this challenge, DeepMind developed a more sophisticated general-purpose solution, which they detailed in a recent Nature article [4]. Their approach called the League, extends the concept of self-play by introducing a collaborative dynamic among agents. Unlike conventional self-play, where agents solely aim to win against opponents, the League incorporates the idea of collaborative training akin to humans partnering with friends to improve their gameplay. In this framework, main agents strive to win against all opponents, while exploiter agents focus on exposing flaws in the main agent's strategy to facilitate improvement, rather than maximizing their win rate against all players.

This novel approach recognizes the importance of collaborative learning dynam-

ics beyond mere competition. By including exploiter agents dedicated to assisting the main agent's growth, the League creates a more comprehensive and effective training environment. Through this collaborative training process, the League learns complex strategies in StarCraft II autonomously and end-to-end.

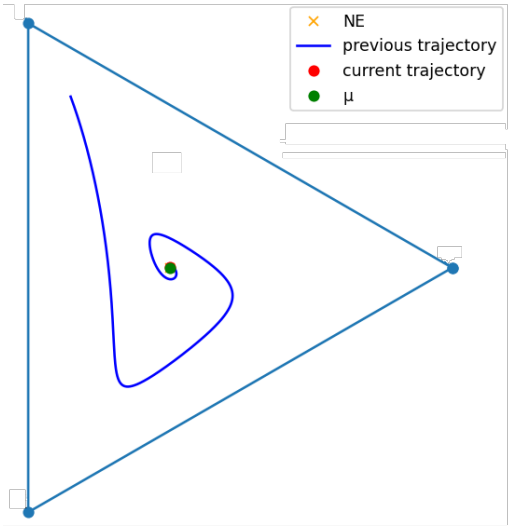
3 Extending FoReL

IL-FoReL has a strong theoretical foundation and we know that it works well. But how can we improve it? Lately, people have been interested in algorithms that use populations of policies, so we thought about adding this idea to FoReL. What would be in our population and how could we use it to improve FoReL? To keep things simple, we decided to stick to basic game types, namely, Normal Form Games, instead of getting tangled up in using fancy methods with neural networks, such as NeuRD [8] or DeepNash [3]. In this way, we can focus on improving the core of the algorithm with methods that converge quickly, without having to worry about training neural networks. We also made sure to think about how well our methods would work on larger scales, and we will talk more about that in Section 5.2. Therefore, most of our experiments will solve the replicator dynamics equation with the Euler method. We will focus our study on Matching Pennies, Rock-Paper Scissors, and Kuhn Poker. It is important to note that by default, the Q-values (or expected utility) are "model-based," implying that we possess precise knowledge of the utility matrices' values of all games.

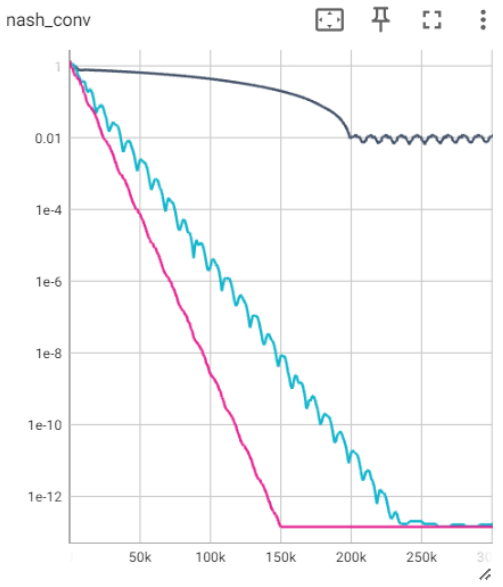
3.1 Intuitions and design of the algorithms

3.1.1 Premise of a better algorithm

Getting better convergence? Our primary focus was on preserving all policies generated by replicator dynamics to form our population. The subsequent consideration was how to utilize this population effectively, keeping in mind the eventual manipulation of neural networks for scalability. Hence, averaging the policies is not a straightforward task. The underlying idea behind FoReL with reward transformation is to introduce a friction term into replicator dynamics, preventing the joint policy from diverging significantly from the reference policy μ. But what happens if μ is π^* the Nash policy? Theoretically, even if the game is modified, the modified Nash is still the one from the original game and we would have $\pi(t) \to \pi^*$. We will call this purely theoretical and not practical algorithm variation L*-FoReL for convenience.



(a) L*-FoReL on Rock-paper-scissors



(b) NashConv plots: L*-FoReL (pink), IL-FoReL (blue), and DL-FoReL (black)

Figure 3: Comparison of L*-FoReL with IL-FoReL and DL-FoReL on rock-paper-scissors

L*-FoReL demonstrates faster convergence compared to IL-FoReL and DL-FoReL. This suggests that there's potential for enhancing the selection of μ, leading to the proposal of employing a population to approximate π^*, given that the time-average of the replicator dynamics trajectory yields π^*. Additionally, it's worth noting that L*-FoReL can serve as a pseudo lower bound for methods using populations, as setting $\mu = \pi^*$ is intuitively the most favorable constant μ choice available.

Adapting L*-FoReL with a population While averaging over time provides an approximation of the Nash policy, this isn't feasible when employing neural networks. Instead, our idea involves averaging policies over time to obtain a μ that closely resembles π^*. Subsequently, IL-FoReL can be applied using this μ. In this context, μ can be the mean of n neural networks. This mean will facilitate learning a policy through a single neural network trained using IL-FoReL.

3.1.2 Sampling and averaging

Taking the average of all the visited policies can be computationally very expensive, especially for neural networks. To limit the computation overhead, we opted for sampling n_{samples} policies and taking the average over these policies P_{sampled}.

Sampling How should we approach policy sampling? If FoReL tends to cycle around π^* [1], an intuitive strategy would involve maximizing the volume of the polyhedron outlined by the n_{samples} policies. However, maximizing volume poses an NP-hard challenge, leading us to explore simpler methods:

- **Random.** A straightforward approach involves randomly selecting n_{samples} policies from the trajectory.
- **Periodic.** Another approach involves selecting n_{samples} policies at regular intervals.
- **Greedy.** The greedy strategy approach involves initializing the sampled set P_{sampled} with the last policy and iteratively adding policies from the trajectory based on maximizing a certain criterion. We devised two versions of the greedy strategy. The first, referred to as "greedy-to-sum," entails selecting

$$\text{argmax}_{\pi \in T} \left(\sum_{\pi' \in P_{\text{sampled}}} d(\pi, \pi') \right)$$

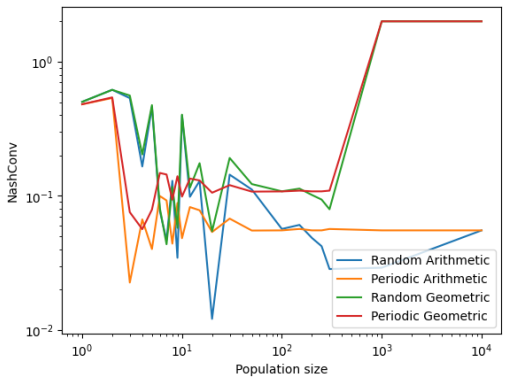
while the second, dubbed "greedy-to-average", involves selecting

$$\text{argmax}_{\pi \in T} d(\pi, \text{average}(P_{\text{sampled}}))$$

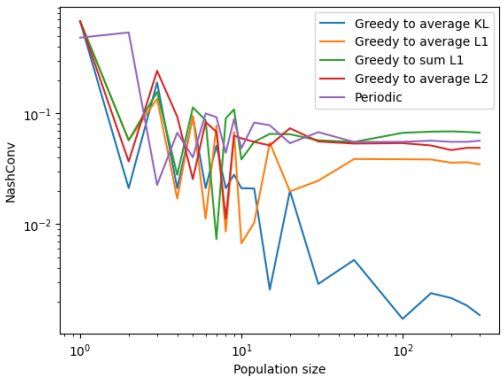
Here, T represents the Replicator Dynamics trajectory, average denotes either the arithmetic or geometric average, and d denotes a distance metric between distributions such as KL divergence, L1, or L2 distances. Interestingly, both methods are the same for the KL divergence and the L2 distance.

Averaging Regarding the averaging method, the geometric average is preferable for the reward transformation, as it resembles summing the reward transformations due to the logarithmic nature. However, the time average of the trajectory, resulting in the Nash policy, follows an average akin to a sum.

Experimenting sampling strategies Now, we will delve into an in-depth analysis of all these methods using the Matching Pennies Biased game. The experiment proceeded as follows: Initially, we executed FoReL without reward transformation on MP Biased to generate a Poincaré recurrent trajectory. Subsequently, our objective was to evaluate the proximity of each method to the Nash policy, quantified by the NashConv metric.



(a) Periodic vs Random & Arithmetic vs geometric average



(b) Greedy vs periodic sampling with arithmetic average

Figure 4: Comparison of NashConv with different sampling and averaging methods

Our experiments revealed that the geometric average yields significantly less relevant results compared to the arithmetic average (Fig. 4a). Regarding the comparison between periodic and geometric strategies, the former exhibits greater stochasticity. Since our primary interest lies in their performance with small populations, we opt to retain the periodic strategy. In Fig. 4b, we observe that the greedy-to-average approach with the KL divergence metric outperforms all others with larger population sizes. For smaller population sizes, the greedy strategy surpasses the periodic strategy, and the choice of metric is less critical. Based on these findings, we conclude that the greedy-to-average sampling with the KL divergence metric for policies is the most suitable approach for our purposes.

Analysis of Greedy Sampling Complexity It is noteworthy that greedy sampling can exhibit non-negligible time complexity. Specifically, greedy-to-average sampling operates at $\mathcal{O}(n_{samples} \times n_{population})$, while greedy-to-sum sampling shares the same complexity but with dynamic programming. Given that FoReL generates small steps while solving the Replicator Dynamics differential equation, the size of $n_{population}$ can grow significantly, leading to extended computational time. To mitigate this issue, a straightforward approach would involve periodic sub-sampling or clustering.

3.2 Algorithms

3.2.1 Iterated Lyapunov FoReL (IL-FoReL)

IL-FoReL was developed in [2]. More details are given in Section 2.2. This algorithm will serve as a comparison to our algorithms.

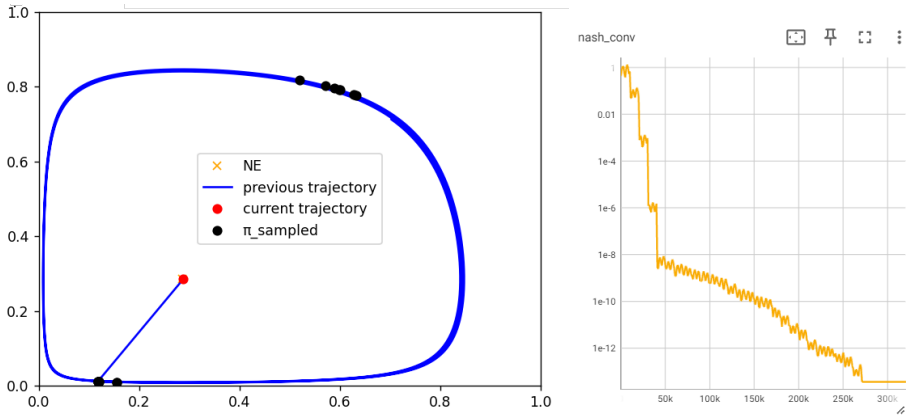
3.2.2 Population FoReL (Pop FoReL)

This algorithm takes its inspiration directly from FoReL and the work on sampling and averaging (Section 3.1.2). We start with a Poincaré recurrent cycle during which we accumulate a population. After a fixed number of steps, we sample the population and take the average of the sampled population. And we restart from there each time (See Alg. 3).

Algorithm 3 Population FoReL (pop FoReL)

Initialize $n_{iteration}$, the number of iterations
Initialize n_{steps}^{PC}, the number of steps in each recurrent phase
Initialize the sampling and averaging method
Initialize empty population P
Initialize π
for iteration **in** 1 **to** $n_{iteration}$ **do**
  for step **in** 1 **to** n_{steps}^{PC} **do**
    $\pi \leftarrow FoReL\_iteration(\pi)$
    Add π to the population P
  end for
  $\pi \leftarrow sample\_and\_average\_population(P)$
  Reset population P
end for

It is worth noting that Population FoReL is purely here to test our hypothesis that we can reach the Nash policy solely with the vanilla Poincaré recurrent FoReL and averaging after subsampling the trajectory. Indeed, population FoReL is the most basic algorithm with a population derived from FoReL but it can hardly be scaled up as it is impossible to take the average of neural networks.



(a) Trajectory visualization (b) Nashconv metric

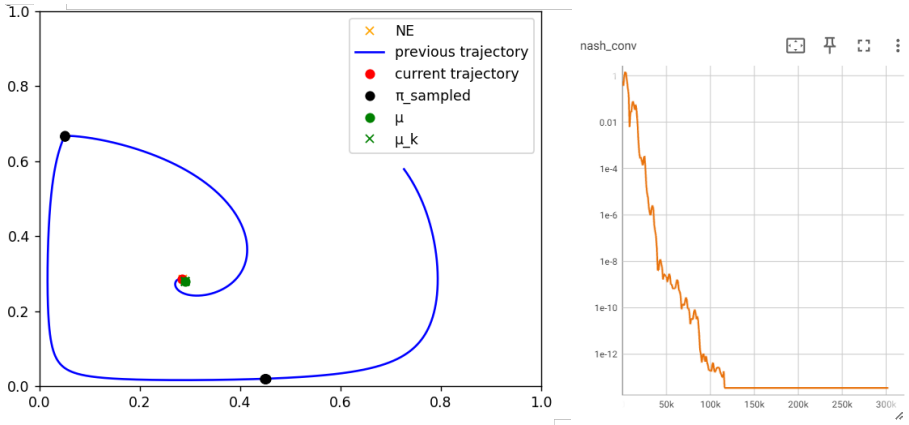
Figure 5: Population FoReL on MP biased

3.2.3 Population Alternating Lyapunov FoReL (PAL-FoReL)

The concept underlying this algorithm involves alternating between Poincaré Recurrence phases and phases comprising Lyapunov iterations. During the Poincaré Recurrence phase, the algorithm cyclically explores around the Nash Equilibrium, facilitating a more accurate estimation of μ through sampling and averaging. This way, we expect μ to be close to π^*. Then, in the Lyapunov FoReL phase, the algorithm uses the updated μ to converge towards the modified Nash Equilibrium.

Algorithm 4 Population Alternating Lyapunov FoReL (PAL-FoReL)

Initialize $n_{iteration}$ the number of iterations
Initialize n_{steps}^{PC}, the number of steps in the recurrent phase
Initialize n_{steps}^{L} the number of steps in Lyapunov phase
Initialize the sampling & averaging method
Initialize empty population P
Initialize π
$\mu \leftarrow \pi$
for iteration **in** 1 **to** $n_{iteration}$ **do**
  for pc_step **in** 1 **to** n_{steps}^{PC} **do**
    $\pi \leftarrow FoReL\_iteration(\pi)$
    Add π to the population P
  end for
  $\mu \leftarrow sample\_and\_average\_population(P)$
  Reset population P
  for l_step **in** 1 **to** n_{steps}^{L} **do**
    $\pi \leftarrow Lypunov\_FoReL\_iteration(\pi, \eta, \mu)$
  end for
end for



(a) Trajectory visualization (b) Nashconv metric

Figure 6: PAL-FoReL on MP biased

4 Experiments and results

Here, we outline our experiments and provide our analysis of the results.

4.1 Experimental Protocol

We trained our algorithms using two modes of Q-value (or utility u) estimation :

- Model-based, where the true Q-values are directly computed from the game so that we have the exact values needed for Replicator Dynamics.
- Monte-Carlo estimation, where the Q-values are estimated using a Monte-Carlo (model-free) method. This method is more general and can be used for more complex games, even games where we do not directly know the rewards.

The model-based method is considerably faster than the Monte-Carlo method because of its minimum variance but requires the knowledge of the game's tree. We chose to focus on the model-based method during our project, as the problem of convergence with exact Q-values is already a challenging one, and because the Monte Carlo method converges to the true Q-values if the number of Monte Carlo iterations becomes large enough.

We trained our algorithms on Kuhn's Poker for 500k games for the model-based method, and 1M games for the Monte Carlo method.

We performed a hyperparameter sweep for PAL-FoReL and IL-FoReL using the WandB platform to pick our hyperparameters for the final benchmark. The sweep results are accessible in the appendix (Section B.1).

4.2 Model-Based benchmark

The Nash Conv metric in log scale is represented in Figure 7 for the different algorithms. We observe that PAL-FoReL (green) outperforms IL-FoReL (orange) in terms of convergence. This result is consistent with our expectations, as the population-based approach of PAL-FoReL allows for a more accurate approximation of the Nash policy.

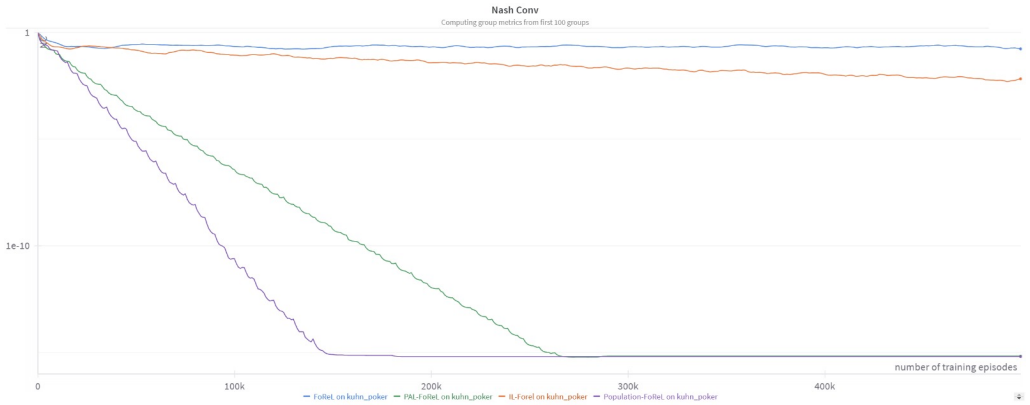


Figure 7: NashConv for different algorithms on Kuhn's Poker

These results hint at the potential of population-based algorithms in enhancing the convergence of FoReL-based methods. The superior performance of PAL-FoReL compared to IL-FoReL underscores the importance of leveraging populations to approximate the Nash policy more accurately. This approach enables the algorithm to explore a broader range of strategies, leading to faster convergence. It's noteworthy that Population FoReL demonstrates even quicker convergence than PAL-FoReL, emphasizing that population sampling and averaging are at the heart of the success of PAL-FoReL.

4.3 The influence of the sampled population size

The ability of our algorithm to perform well with a small sampled population size is crucial for scalability.

Here, we present the influence of the size of the sampled population on the convergence of Population FoReL (Fig. 8) and PAL-FoReL (Fig. 9). In both plots, for reference, we included PAL-FoReL (green) and Population FoReL (purple) with a sampled population size of 10, as well as IL-FoReL (orange). For population-based algorithms, as stated in the previous section we only used the greedy-to-average strategy with the KL-divergence and arithmetic average.

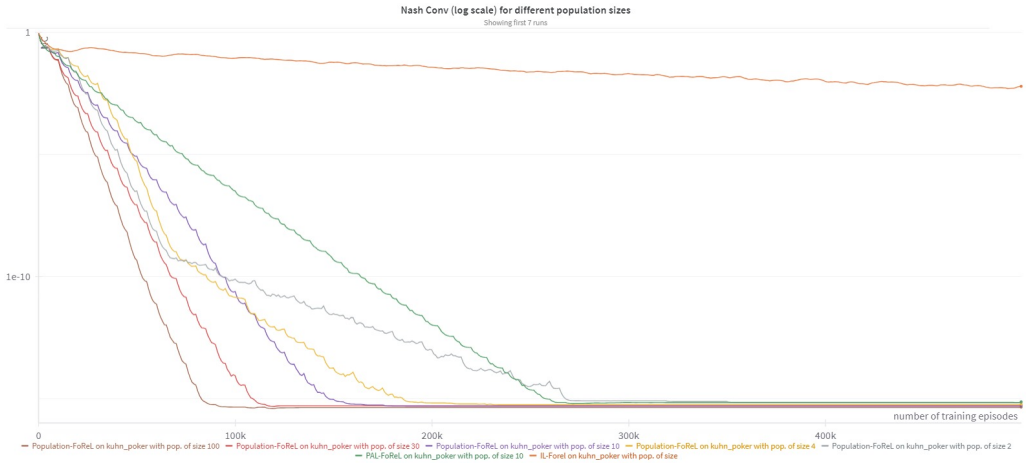


Figure 8: Population FoReL performance for different population sizes

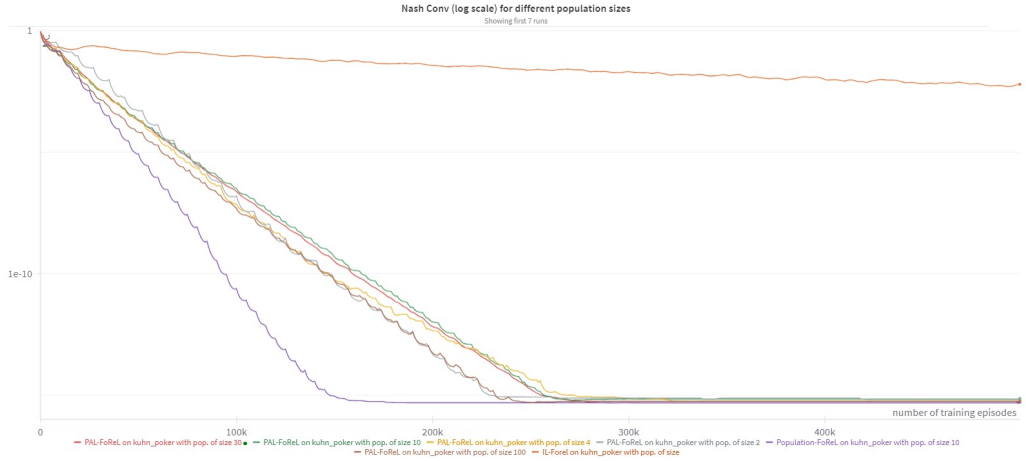


Figure 9: PAL-FoReL performance for different population sizes

Our experiments revealed two distinct patterns for Population FoReL. With large sampled population sizes ($n_{\text{sample}} > 50$), increasing n_{sample} enhances the convergence rate. However, for smaller population sizes ($n_{\text{sample}} < 20$), the impact is less evident. This trend extends to PAL-FoReL, where the size of the sampled population appears to have no discernible effect on the convergence rate.

4.4 Sampling strategies comparison

We verified the conclusions drawn in section 3.1.2 on the different sampling strategies for our algorithms. The results are presented in Fig.10 for Population FoReL and in Fig.11 for PAL FoReL. We note significant enhancements in algorithm convergence through the adoption of the greedy strategy compared to random or periodic policy

sampling methods. This outcome is readily understandable because both periodic and random sampling strategies depend on the Poincaré recurrent trajectory. However, given the complexity of Kuhn Poker, it's probable that vanilla FoReL doesn't have ample time to complete a full loop during the recurrent phase. In such scenarios, the greedy sampling strategy remains the only approach to effectively leveraging the population.

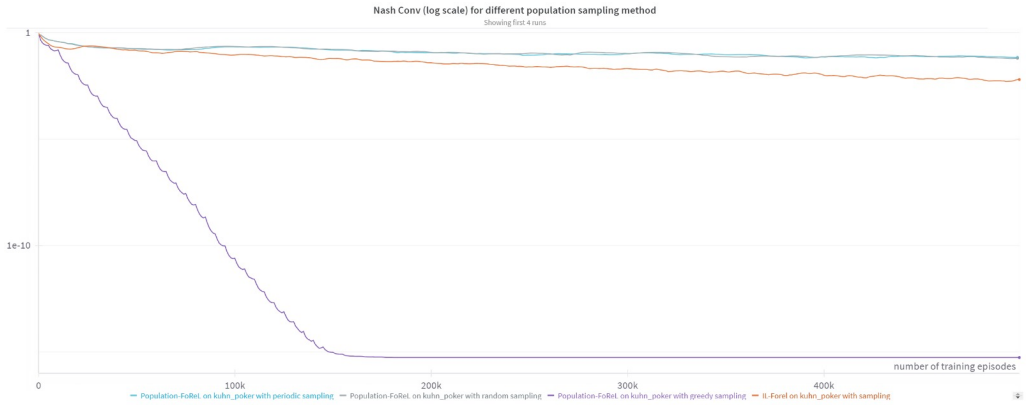


Figure 10: Population FoReL performance for different sampling method

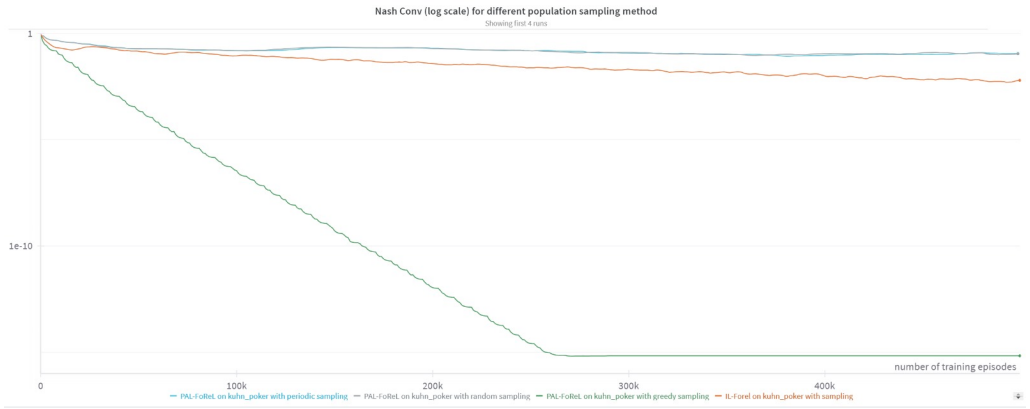


Figure 11: PAL-FoReL performance for different sampling method

4.5 Monte Carlo estimation of Q-values

We also tested our algorithms using the Monte Carlo estimation of the Q-values. The results are presented in Fig.12.

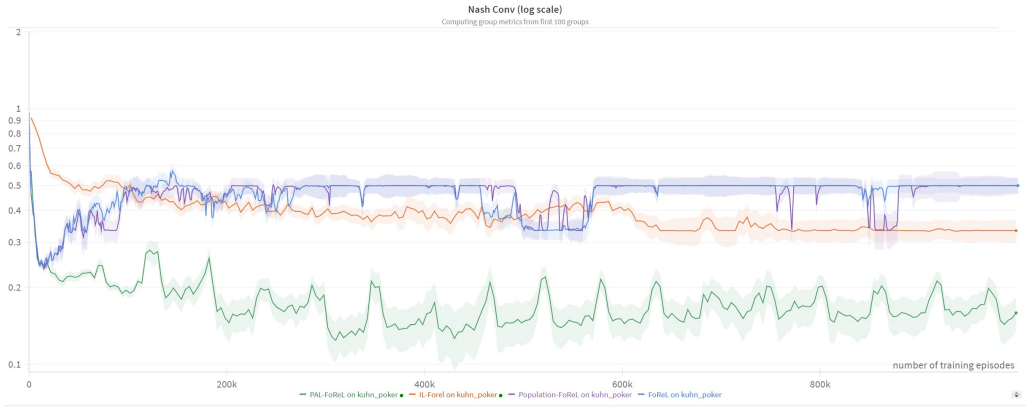
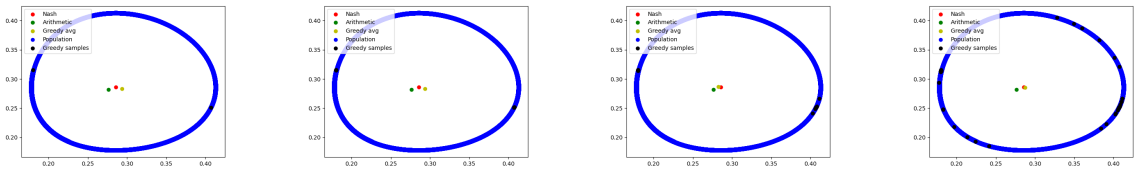


Figure 12: NashConv for different algorithms on Kuhn's Poker using MC method for Q-value estimation

An initial observation is that all algorithms encounter considerable difficulty in achieving accurate results in the Model-free setting. This outcome was somewhat anticipated since all updates involve noise due to the evaluation of the Q-value. Population FoReL encounters significant challenges in convergence despite exhibiting the best convergence rate in the model-based setting. However, PAL-FoReL demonstrates resilience to the noise and continues to converge more effectively than IL-FoReL.

5 Future works

5.1 Improving PAL-FoReL



(a) Size: 2 (b) Size: 10 (c) Size: 20 (d) Size: 50

Figure 13: Greedy sampling with KL divergence on MP Biased with different population sampling sizes

While the greedy-to-average sampling method with KL-divergence shows improved performance, it still falls short of being optimal. Even with $n_{\text{sample}} = 10$, the sampling strategy behaves similarly to having $n_{\text{sample}} = 2$. Therefore, although not tested, a potentially more effective approach could involve a combination of greedy and random sampling during the initial steps.

5.2 Back to more general cases

When designing our algorithms, scalability was always a key consideration. As a result, our method, PAL-FoReL, can be effectively scaled up to tackle more complex games using the NeuRD framework [8]. This involves leveraging policy gradient techniques alongside Replicator dynamics. Notably, by maintaining a low population size, PAL-FoReL can be deployed with minimal computational overhead, as the algorithm's performance isn't significantly impacted by small population size. To avoid retaining all states of the neural networks, one could consider periodic saving steps in addition to methods described in Section 3.1.2. Moreover, within the NeuRD framework, PAL-FoReL gains the capability to address sequential games and imperfect information games by integrating observations as inputs to the policy network. While the formulas remain consistent, they are now conditioned on the input state. When employing greedy sampling, obtaining a distribution of the observations becomes necessary. The distance metric across distributions is substituted by an average of π conditioned by the action, weighted by the distribution of observations.

Moving forward, the next phase of this project involves implementing PAL-FoReL with NeuRD on the Kuhn Poker game to directly assess its scalability on imperfect information games with neural networks. [2] could inspire this upcoming work since IL-FoReL was implemented on Kuhn Poker.

5.3 Other population-based ideas

Population-based approximation of Q instead of μ The concept behind PAL-FoReL involves aligning μ closely with π^* using population-based concepts. An intriguing proposition would be to directly assess the feasibility of this approach for u within the Replicator Dynamics equation:

$$\frac{d}{dt}\pi^k(a) = \pi^k(a)\left[u(a,\pi) - \overline{u}^k(\pi)\right]$$

Here, u relies on the current policy. Similar to μ, it would be insightful to examine the potential impact on $\pi(t)$ if u were replaced by u^*, the utility for π^*. However, substituting both u terms in the equation renders it trivial. Hence, the focus should be on studying the behavior when one term is replaced while the other remains unchanged. If the outcomes prove noteworthy, a similar population-based strategy could be employed to substitute u with a population-based estimation.

Leagues training Originally, the project considered employing leagues to enhance FoReL, but this avenue was not explored further. Given FoReL's tendency for chaotic behavior when dealing with more than two players, we decided to pursue PAL-FoReL instead. PAL-FoReL appeared to offer greater stability and promise. Leagues training involves organizing agents into groups or "leagues" and having them compete against each other, to improve their performance through competition and collaboration within their respective leagues [4].

6 Conclusion

This research project introduced two innovative algorithms for identifying Nash Equilibrium strategies in two-player zero-sum games: Population FoReL and PAL-FoReL. These algorithms leverage the robust theoretical underpinnings of FoReL in conjunction with population-based concepts. Through a series of experiments, we explored various iterations of our algorithms. Both approaches demonstrated faster convergence compared to IL-FoReL. The efficacy of Population FoReL and PAL-FoReL can largely be attributed to the efficient greedy sampling technique. The success of Population FoReL and PAL-FoReL algorithms can be largely attributed to the effectiveness of the greedy sampling method.

Furthermore, our research focused on designing an algorithm that could be easily adapted for the neural networks to more complex games, such as imperfect information games and extensive form games. We also discovered that PAL-FoReL exhibits consistent behavior regardless of the population size, achieving equivalent NashConv values. This scalability aspect is a significant finding, as it enables PAL-FoReL to operate with minimal computational overhead even when utilizing a small population. Consequently, PAL-FoReL exhibits promise for scalability to more intricate games and environments. We also discovered that PAL-FoReL exhibits consistent behavior regardless of the population size, achieving equivalent NashConv values.

References

[1] Panayotis Mertikopoulos, Christos Papadimitriou, and Georgios Piliouras. Cycles in adversarial regularized learning. In *Proceedings of the twenty-ninth annual ACM-SIAM symposium on discrete algorithms*, pages 2703–2717. SIAM, 2018.

[2] Julien Perolat, Remi Munos, Jean-Baptiste Lespiau, Shayegan Omidshafiei, Mark Rowland, Pedro Ortega, Neil Burch, Thomas Anthony, David Balduzzi, Bart De Vylder, et al. From poincaré recurrence to convergence in imperfect information games: Finding equilibrium via regularization. In *International Conference on Machine Learning*, pages 8525–8535. PMLR, 2021.

[3] Julien Perolat, Bart De Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T Connor, Neil Burch, Thomas Anthony, et al. Mastering the game of stratego with model-free multiagent reinforcement learning. *Science*, 378(6623):990–996, 2022.

[4] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, November 2019. ISSN 1476-4687. doi: 10.1038/s41586-019-1724-z. URL `https://www.nature.com/articles/s41586-019-1724-z`. Number: 7782 Publisher: Nature Publishing Group.

[5] Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castañeda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, Nicolas Sonnerat, Tim Green, Louise Deason, Joel Z. Leibo, David Silver, Demis Hassabis, Koray Kavukcuoglu, and Thore Graepel. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, May 2019. doi: 10.1126/science.aau6249. URL `https://www.science.org/doi/10.1126/science.aau6249`. Publisher: American Association for the Advancement of Science.

[6] John Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, 1951. ISSN 0003486X. URL `http://www.jstor.org/stable/1969529`.

[7] C. A. B. Smith, John Von Neumann, and Oskar Morgenstern. Theory of games and economic behavior. *Journal of the American Statistical Association*, 40: 263, 1945. URL `https://api.semanticscholar.org/CorpusID:124430577`.

[8] Daniel Hennes, Dustin Morrill, Shayegan Omidshafiei, Remi Munos, Julien Perolat, Marc Lanctot, Audrunas Gruslys, Jean-Baptiste Lespiau, Paavo Parmas, Edgar Duenez-Guzman, et al. Neural replicator dynamics. *arXiv preprint arXiv:1906.00190*, 2019.

[9] G.W. Brown. Iterative solutions of games by fictitious play. *Activity Analysis of Production and Allocation*, 1951.

[10] Lanctot Heinrich and Silver. Fictitious self-play in extensive-form games. *Proceedings of Machine Learning Research*, 2015.

A Implemented games

For our experiments, we picked some simple yet non-trivial games. Those games are not highly complex because the purpose of the project was to perform a proof of concepts on relatively simple games (which are already a challenge for state-of-the-art MARL algorithms) in order to open extension ideas on bigger games.

A.1 Matching Pennies

Matching Pennies is a common two-player zero-sum game studied in game theory. As a matrix can represent it, this game is a **normal-form game**. The usual way to define this game is through its payoff matrix (Tab.1).

Each player can choose two actions: head or tail. If both players choose the same action, the "even" player wins a reward of 1 and the odd player loses 1. The opposite happens in the case where the players choose opposite actions. The mixed strategy that leads to a Nash Equilibrium is $(\frac{1}{2}, \frac{1}{2})$ for both players. There is no pure strategy leading to a Nash Equilibrium in this game.

Player 1 \Player 2	Head: H	Tail: T
Head: H	(1, -1)	(-1, 1)
Tail: T	(-1, 1)	(1, -1)

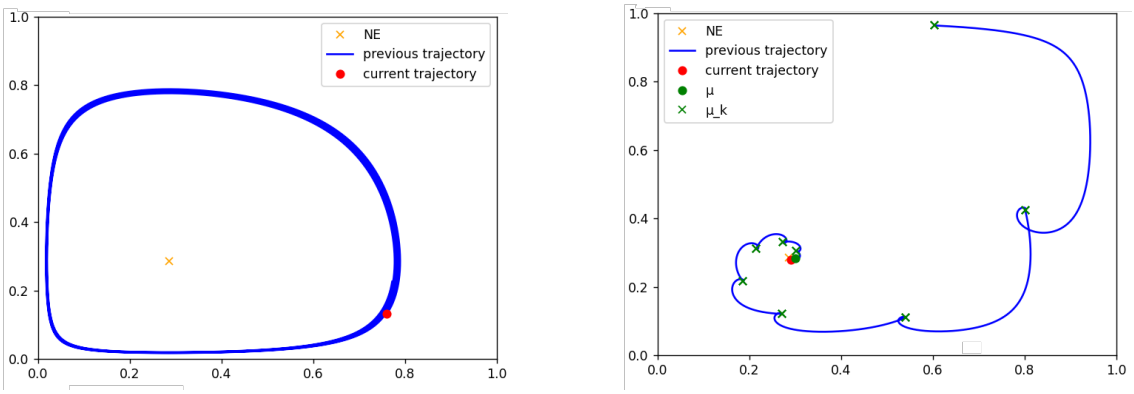
Table 1: Payoff matrix of MP

This game is one of the most basic non-trivial 2-player 0-sum games. It was widely used in our project as a toy game to test our implementation. In addition, the low number of actions (2 for each player) allows easy visualization (Fig. 14) of the player's policies through the training.

Matching Pennies Biased Because the Nash equilibrium was at $(\frac{1}{2}, \frac{1}{2})$, this could hide some kind of bias in training. For example, an algorithm that only tries to maximize its entropy would converge to the Nash Equilibrium, without understanding the game. To have a relevant game, we introduced a bias in the rewards while keeping the game 0-sum, so that the Nash Equilibrium is not at any particular point. Biased Matching Pennies is a version of the previous game with a slightly different payoff table (Tab.2) while preserving the general dynamic and purpose of the game. The Nash Equilibrium of this biased game is at $(\frac{2}{7}, \frac{5}{7})$ for both players.

Player 1 \Player 2	Head: H	Tail: T
Head: H	(4, -4)	(-1, 1)
Tail: T	(-1, 1)	(1, -1)

Table 2: Payoff matrix of MP Biased



(a) FoReL Poincaré cycling

(b) IL-FoReL converging to Nash

Figure 14: FoReL and IL-FoReL visualization on MP Biased

A.2 Rock Paper Scissors

Rock Paper Scissors is a two-player zero-sum normal-form game in which each player can choose among three items: rock, paper, and scissors. There is an inherent order to this game: rock wins against scissors, scissors wins paper, and paper wins against rock.

Player 1 \Player 2	Rock	Paper	Scissors
Rock	(0, 0)	(-1, 1)	(1, -1)
Paper	(1, -1)	(0, 0)	(-1, 1)
Scissors	(-1 , 1)	(1, -1)	(0, 0)

Table 3: Payoff matrix of Rock-paper-scissors

The mixed strategy leading to a Nash Equilibrium is $(\frac{1}{3},\frac{1}{3},\frac{1}{3})$

Rock Paper Scissors Biased is a version of the previous game in which the rewards are modified with the pay-off table 4 so that the Nash policy is not a uniform distribution over the actions.

Player 1 \Player 2	Rock	Paper	Scissors
Rock	(0, 0)	(-4, 4)	(1, -1)
Paper	(4, -4)	(0, 0)	(-1, 1)
Scissors	(-1 , 1)	(1, -1)	(0, 0)

Table 4: Payoff matrix of biased Rock-paper-scissors

A.3 Kuhn Poker

Kuhn Poker is a simplified version of poker. In this version, the card deck contains only three cards: Jack, Queen, and King. Here is the process that describes this game:

- Each player antes 1
- Each player is dealt a card, the third one is put aside unseen
- Player 1 can check or bet 1
  - If player one checks, then player two can check or bet 1
    - If player two checks, there is a showdown: the higher card wins 1 from the other player.
    - If player two bets, then player one folds or calls.
      - If player one folds, then player two takes the pot of 3
      - If player one calls there is a showdown for the pot of 4:the higher wins two from the other player
  - If player one bets, then player two can fold or call
    - If the player two folds, then player one takes the pot of 3.
    - If player two calls, there is a showdown for the pot of 4.

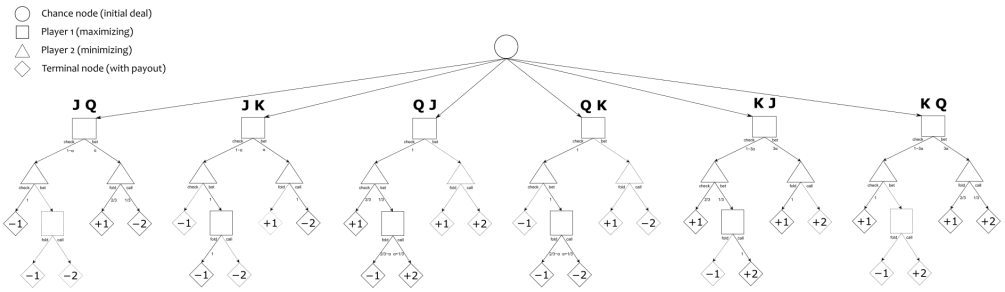


Figure 15: Illustration of the Game Tree of Kuhn Poker

Kuhn's Poker is our main objective for this project, since it is already far from a trivial game for a human being. We accomplished our final benchmark in this game.

The choice for implementing the algorithms in this game results from the greater complexity of this zero-sum normal-form game. It allows us to test our algorithms not only on elementary games like the ones above but also on a still simple but slightly, more complex game.

B Experiments logs

B.1 WandB sweep results

- Sweep results for PAL-FoReL with model-based Q value extraction, on Kuhn's Poker: here.
- Sweep results for IL-FoReL with model-based Q value extraction, on Kuhn's Poker: here.
- Sweep results for PAL-FoReL with Monte Carlo Q value estimation, on Kuhn's Poker: here.
- Sweep results for IL-FoReL with Monte Carlo Q value estimation, on Kuhn's Poker: here.

B.2 WandB repository for all runs

- All our model-based runs are available here.
- All our Monte Carlo runs are available here