Scalable Deep Reinforcement Learning Algorithms for Mean Field Games

Mathieu Laurière[⋆] lauriere@google.com
Google Research

Sarah Perrin[⋆] sarah.perrin@inria.fr
Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL

Sertan Girgin[⋆] sertan@google.com
Google Research

Paul Muller pmuller@google.com
DeepMind

Ayush Jain ayush.jain@berkeley.edu
UC Berkeley

Théophile Cabannes cabannes@google.com
Google Research

Georgios Piliouras georgios.piliouras@gmail.com
Singapore University of Technology and Design

Julien Pérolat perolat@google.com
DeepMind

Romuald Élie relie@google.com
DeepMind

Olivier Pietquin[†] pietquin@google.com
Google Research

Matthieu Geist[†] mfgeist@google.com
Google Research

Abstract

Mean Field Games (MFGs) have been introduced to efficiently approximate games with very large populations of strategic agents. Recently, the question of learning equilibria in MFGs has gained momentum, particularly using model-free reinforcement learning (RL) methods. One limiting factor to further scale up using RL is that existing algorithms to solve MFGs require the mixing of approximated quantities such as strategies or q-values. This is non-trivial in the case of non-linear function approximation that enjoy good generalization properties, *e.g.* neural networks. We propose two methods to address this shortcoming. The first one learns a mixed strategy from distillation of historical data into a neural network and is applied to the Fictitious Play algorithm. The second one is an online mixing method based on regularization that does not require memorizing historical data or previous estimates. It is used to extend Online Mirror Descent. We demonstrate numerically that these methods efficiently enable the use of Deep RL algorithms to solve various MFGs. In addition, we show that these methods outperform SotA baselines from the literature.

1. Introduction

Despite outstanding success of machine learning in numerous applications (Goodfellow et al., 2016, 2020), learning in games remains difficult because two or more agents require the consideration of non-stationary environments. In

particular, it is hard to scale in terms of number of agents because one has to keep track of all agents' behaviors, leading to a prohibitive combinatorial complexity (Daskalakis et al., 2006; Tuyls and Weiss, 2012). Recently, Mean Field Games (MFGs) have brought a new perspective by replacing the atomic agents by their distribution. Introduced concurrently by Lasry and Lions (2007) and Huang et al. (2006), the mean field approximation relies on symmetry and homogeneity assumptions, and enables to scale to an infinite number of agents. Solving MFGs is usually done by solving a forward-backward system of partial (or stochastic) differential equations, which represent the dynamics of both the state distribution and the value function. However, traditional numerical methods (see *e.g.* (Achdou and Laurière, 2020)) do not scale well in terms of the complexity of the state and action spaces. Furthermore, they require the model (dynamics and rewards) to be completely known.

On the other hand, Reinforcement Learning (RL) has proved very efficient to solve Markov Decision Processes (MDPs) and games with complex structures, from *e.g.* chess Campbell et al. (2002) to multi-agent systems (Lanctot et al., 2017). To tackle problems with highly complex environments, RL algorithms can be combined with efficient function approximation. Deep neural networks with suitable architectures are arguably the most popular choice, thanks to their ease of use and their generalization capabilities. Deep RL has brought several recent breakthrough results, *e.g.*, Go (Silver et al., 2016), Atari games (ALE Mnih et al. (2013)), poker (Brown and Sandholm, 2018; Moravčík et al., 2017) or even complex video games such as Starcraft (Vinyals et al., 2019).

Many recent works have combined MFGs with RL to leverage their mutual potential – albeit mostly without deep neural nets thus far. The intertwinement of MFGs and RL happens through an *optimization* (or learning) procedure. The simplest algorithm of this type is the (Banach-Picard) fixed-point approach, consisting in alternating a best response computation against a given population distribution with an update of this distribution (Huang et al., 2006). However, this method fails in many cases by lack of contractivity as proved by Cui and Koeppl (2021). Several other procedures have thus been introduced, often inspired by game theory or optimization algorithms. Fictitious Play (FP) and its variants average either distributions or policies (or both) to stabilize convergence (Cardaliaguet and Hadikhanloo, 2017; Elie et al., 2020; Perrin et al., 2020; Xie et al., 2021; Perrin et al., 2021a,b), whereas Online Mirror Descent (OMD) (Hadikhanloo, 2017; Perolat et al., 2021b) relies on policy evaluation. Other works have leveraged regularization (Anahtarci et al., 2020; Cui and Koeppl, 2021; Guo et al., 2020b) to ensure convergence, at the cost of biasing the Nash equilibrium. These methods require to sum or average some key quantities: FP needs to average the distributions, while OMD needs to sum Q-functions. This is a key component of most (if not all) smoothing methods and needs to be tackled efficiently. These operations are simple when the state space is finite and small, and the underlying objects can be represented with tables or linear functions. However, there is no easy and efficient way to sum non-linear approximations such as neural networks, which raises a major challenge when trying to combine learning methods (such as FP or OMD) with deep RL.

The main contribution of the paper is to solve this important question in dynamic MFGs. We propose two algorithms. The first one, that we name Deep Average-network Fictitious Play (D-AFP), builds on FP and uses the Neural Fictitious Self Play (NFSP) approach (Heinrich and Silver, 2016) to compute a neural network approximating an average over past policies. The second one is Deep Munchausen Online Mirror Descent (D-MOMD), inspired by the Munchausen reparameterization of Vieillard et al. (2020). We prove that in the exact case, Munchausen OMD is equivalent to OMD. Finally, we conduct numerical experiments and compare D-AFP and D-MOMD with SotA baselines adapted to dynamic MFGs. We find that D-MOMD converges faster than D-AFP on all tested games from the literature, which extends to Deep RL the results obtained for exact algorithms in (Perolat et al., 2021b; Geist et al., 2022).

2. Background

2.1 Mean Field Games

A Mean Field Game (MFG) is a strategic decision making problem with a continuum of identical and anonymous players. In MFGs, it is possible to select a *representative player* and to focus on its policy instead of considering all players individually, which simplifies tremendously the computation of an equilibrium. We place ourselves in the context of a game and are interested in computing Nash equilibria. We stress that we consider a *finite horizon* setting as it encompasses a broader class of games, which needs time-dependant policies and distributions. We denote by N_T the finite time horizon and by n a generic time step. We focus on finite state X and action A spaces. We denote by Δ_X the set of probability distributions on X. Each distribution can be viewed as a vector of length $|X|$. Then, the game is characterized by one-step reward functions $r_n : X \times A \times \Delta_X \to \mathbb{R}$, and transition probability functions

$p_n : X \times A \times \Delta_X \to \Delta_X$, for $n = 0, 1, \dots, N_T$. The third argument corresponds to the current distribution of the population. An initial distribution m_0 of the population is set and will remain fixed over the paper. The two main quantities of interest are the policy of the representative player $\pi = (\pi_n)_n \in (\Delta_A^X)^{N_T+1}$ and the distribution flow (*i.e.* sequence) of agents $\mu = (\mu_n)_n \in \Delta_X^{N_T+1}$. Let us stress that we denote the time of the game n and that π and μ are thus time-dependant objects. Given a population mean field flow μ, the goal for a representative agent is to maximize over π the total reward:

$$J(\pi, \mu) = \mathbb{E}_\pi\Big[\sum_{n=0}^{N_T} r_n(x_n, a_n, \mu_n)\Big| x_0 \sim m_0\Big]$$
$$\text{s.t.: } a_n \sim \pi_n(\cdot|x_n),\ x_{n+1} \sim p_n(\cdot|x_n, a_n, \mu_n),\ n \geq 0.$$

Note that the reward r_n together with the transition function p_n are functions of the population distribution at the current time step μ_n, which encompasses the mean field interactions. A policy π is called a best response (BR) against a mean field flow μ if it is a maximizer of $J(\cdot, \mu)$. We denote by $BR(\mu)$ the set of best responses to μ.

Given a policy π, a meam field flow μ is said to be induced by π if: $\mu_0 = m_0$ and for $n = 0, \dots, N_T - 1$, $\mu_{n+1}(x) = \sum_{x',a'} \mu_n(x')\pi_n(a'|x')p_n(x|x', a', \mu_n)$, which we can simply write $\mu_{n+1} = P_n^{\mu_n,\pi_n}\mu_n$, where $P_n^{\mu_n,\pi_n}$ is the transition matrix of x_n. We denote by μ^π or $\Phi(\pi) \in \Delta_X^{N_T+1}$ the mean-field flow induced by π.

Definition 1. A pair $(\hat{\pi}, \hat{\mu})$ is a (finite horizon) Mean Field Nash Equilibrium (MFNE) if (1) $\hat{\pi}$ is a BR against $\hat{\mu}$, and (2) $\hat{\mu}$ is induced by $\hat{\pi}$.

Equivalently, $\hat{\pi}$ is a fixed point of the map $BR \circ \Phi$. Given a mean field flow μ, a representative player faces a traditional MDP, which can be studied using classical tools. The value of a policy can be characterized through the Q-function defined as: $Q_{N_T+1}^{\pi,\mu}(x, a) = 0$ and for $n \leq N_T$, $Q_n^{\pi,\mu}(x, a) = \mathbb{E}\Big[\sum_{n' \geq n} r_{n'}(x_{n'}, a_{n'}, \mu_{n'})\Big|(x_n, a_n) = (x, a)\Big]$. It satisfies the Bellman equation: for $n \leq N_T$,

$$Q_n^{\pi,\mu}(x, a) = r_n(x, a, \mu_n) + \mathbb{E}_{x',a'}[Q_{n+1}^{\pi,\mu}(x', a')], \tag{1}$$

where $x' \sim p(\cdot|x, a, \mu_n)$ and $a' \sim \pi_n(\cdot|x, a, \mu_n)$, with the convention $Q_{N_T+1}^{\pi,\mu}(\cdot, \cdot) = 0$. The optimal Q-function $Q^{*,\mu}$ is the value function of any best response π^* against μ. It is defined as $Q_n^{*,\mu}(x, a) = \max_\pi Q_n^{\pi,\mu}(x, a)$ for every n, x, a, and it satisfies the optimal Bellman equation: for $n \leq N_T$,

$$Q_n^{*,\mu}(x, a) = r_n(x, a, \mu_n) + \mathbb{E}_{x',a'}[\max_{a'} Q_{n+1}^{*,\mu}(x', a')], \tag{2}$$

where $Q_{N_T+1}^{*,\mu}(\cdot, \cdot) = 0$.

2.2 Fictitious Play

The most straightforward method to compute a MFNE is to iteratively update in turn the policy π and the distribution μ, by respectively computing a BR and the induced mean field flow. The BR can be computed with the backward induction of (2), if the model is completely known. We refer to this method as *Banach-Picard (BP) fixed point iterations*. See Alg. 4 in appendix for completeness. The convergence is ensured as soon as the composition $BR \circ \Phi$ is a strict contraction (Huang et al., 2006). However, this condition holds only for a restricted class of games and, beyond that, simple fixed point iterations typically fail to converge and oscillations appear (Cui and Koeppl, 2021).

To address this issue, a memory of past plays can be added. The *Fictitious Play (FP) algorithm*, introduced by Brown (1951) computes the new distribution at each iteration by taking the average over all past distributions instead of the latest one. This stabilizes the learning process so that convergence can be proved for a broader class of games under suitable assumptions on the structure of the game such as potential structure (Cardaliaguet and Hadikhanloo, 2017; Geist et al., 2022) or monotonicity (Perrin et al., 2020). The method can be summarized as follows: after initializing Q_n^0 and π_n^0 for $n = 0, \dots, N_T$, repeat at each iteration k:

$$\begin{cases} 1.\ \text{Distribution update: } \mu^k = \mu^{\pi^k}, \bar{\mu}^k = \frac{1}{k-1}\sum_{i=1}^{k-1} \mu^i \\ 2.\ Q\text{-function update: } Q^k = Q^{*,\bar{\mu}^k} \\ 3.\ \text{Policy update: } \pi_n^{k+1}(.|x) = \text{argmax}_a\, Q_n^k(x, a). \end{cases}$$

In the distribution update, $\bar{\mu}^k$ corresponds to the population mean field flow obtained if, for each $i = 1, \dots, k-1$, a fraction $1/(k-1)$ of the population follows the policy π^i obtained as a BR at iteration i. At the end, the algorithm returns the latest mean field flow μ^k as well as a policy that generates this mean field flow. This can be achieved either through a single policy or by returning the vector of all past BR, $(\pi^i)_{i=1,\dots,k-1}$, from which $\bar{\mu}^k$ can be recovered. See Alg. 5 in appendix for completeness.

2.3 Online Mirror Descent

The aforementioned methods are based on computing a BR at each iteration. Alternatively, we can follow a policy iteration based approach and simply *evaluate* a policy at each iteration. In finite horizon, this operation is less computationally expensive than computing a BR because it avoids a loop over the actions to find the optimal one.

The *Policy Iteration (PI) algorithm* for MFG (Cacace, Simone et al., 2021) consists in repeating, from an initial guess π^0, μ^0, the update: at iteration k, first evaluate the current policy π^k by computing $Q^{k+1} = Q^{\pi^k, \mu^k}$, then let π^{k+1} be the greedy policy such that $\pi^{k+1}(\cdot|x)$ is a maximizer of $Q^k(x, \cdot)$. The evaluation step can be done with the backward induction (1), provided the model is known. See Alg. 6 in appendix for completeness.

Here again, to stabilize the learning process, one can rely on information from past iterations. Using a weighted sum over past Q-functions yields the so-called *Online Mirror Descent (OMD) algorithm* for MFG, which can be summarized as follows: after initializing q_n^0 and π_n^0 for $n = 0, \dots, N_T$, repeat at each iteration k:

$$\begin{cases} 1.\ \text{Distribution update: } \mu^k = \mu^{\pi^k} \\ 2.\ Q\text{-function update: } Q^k = Q^{\pi^k, \mu^k} \\ 3.\ \text{Regularized } Q\text{-function update: } \bar{q}^{k+1} = \bar{q}^k + \frac{1}{\tau} Q^k \\ 4.\ \text{Policy update: } \pi_n^{k+1}(\cdot|x) = \mathrm{softmax}(\bar{q}_n^{k+1}(x, \cdot)). \end{cases}$$

For more details, see Alg. 7 in appendix. Although we focus on softmax policies in the sequel, other conjugate functions of steep regularizers could be used in OMD, see Perolat et al. (2021b). This method is known to be empirically faster than FP, as illustrated by Perolat et al. (2021b). Intuitively, this can be explained by the fact that the learning rate in FP is of the order $1/k$ so this algorithm is slower and slower as the number of iterations increases.

2.4 Deep Reinforcement Learning

Reinforcement learning aims to solve optimal control problems when the agent does not know the model (*i.e.*, p and r) and must learn through trial and error by interacting with an environment. In a finite horizon MFG setting, we assume that a representative agent is encoded by a policy π, either explicitly or implicitly (through a Q-function) and can realize an episode, in the following sense: for $n = 0, \dots, N_T$, the agent observes x_n (with $x_0 \sim m_0$), chooses action $a_n \sim \pi_n(\cdot|x_n)$, and the environment returns a realization of $x_{n+1} \sim p_n(\cdot|x_n, a_n, \mu_n)$ and $r_n(x_n, a_n, \mu_n)$. Note that the agent does not need to observe directly the mean field flow μ_n, which simply enters as a parameter of the transition and cost functions p_n and r_n.

Based on such samples, the agent can approximately compute the Q-functions $Q^{\pi,\mu}$ and $Q^{*,\mu}$ following (1) and (2) respectively where the expectation is replaced by Monte-Carlo samples. In practice, we often use trajectories starting from time 0 and state $x_0 \sim m_0$ instead of starting from any pair (x, a). Vanilla RL considers infinite horizon, discounted problems and looks for a stationary policy, whereas we consider a finite-horizon setting with non-stationary policies. To simplify the implementation, we treat time as part of the state by considering (n, x_n) as the state. We can then use standard Q-learning. However, it is important to keep in mind that the Bellman equations are not fixed-point equation for some stationary Bellman operators.

When the state space is large, it becomes impossible to evaluate precisely every pair (x, a). Motivated by both memory efficiency and generalization, we can approximate the Q-functions by non linear functions such as neural networks, say $Q_\theta^{\pi,\mu}$ and $Q_\theta^{*,\mu}$, parameterized by θ. Then, the quantities in (1) and (2) are replaced by the minimization of a loss to train the neural network parameters θ. Namely, treating time as an input, one minimizes over θ the quantities $\hat{\mathbb{E}}\left[\left|Q_{\theta,n}^{\pi,\mu}(x,a) - r_n(x,a,\mu_n) - Q_{\theta_t,n+1}^{\pi,\mu}(x',a')\right|^2\right]$ and $\hat{\mathbb{E}}\left[\left|Q_{\theta,n}^{*,\mu}(x,a) - r_n(x,a,\mu_n) - \max_{a'} Q_{\theta_t,n+1}^{*,\mu}(x',a')\right|^2\right]$, where $\hat{\mathbb{E}}$ is an empirical expectation based on Monte Carlo samples and θ_t is the parameter of a target network.

3. Deep Reinforcement Learning for MFGs

To develop scalable methods for solving MFGs, a natural idea consists in combining the above optimization methods (FP and OMD) with deep RL. This requires summing or averaging policies or distributions, and induces hereby a major challenge as they are approximated by non linear operators, such as neural networks. In this section, we develop innovative and scalable solutions to cope with this. In the sequel, we denote $Q_\theta((n,x),a)$ with the time in the input when we refer to the neural network Q-function.

3.1 Deep Average-network Fictitious Play

To develop a model-free version of FP, one first needs to compute a BR at each iteration, which can be done using standard deep RL methods, such as DQN (Mnih et al., 2013). A policy that generates the average distribution over past iterations can be obtained by simply keeping in memory all the BRs from past iterations. This approach has already been used successfully *e.g.* by Perrin et al. (2020, 2021b). However, it requires a memory that is linear in the number of iterations and each element is potentially large (*e.g.*, a deep neural network), which does not scale well. Indeed, as the complexity of the environment grows, we expect FP to need more and more iterations to converge, and hence the need to keep track of a larger and larger number of policies.

An alternative approach is to learn along the way the policy generating the average distribution. We propose to do so by keeping a buffer of state-action pairs generated by past BRs and learning the average policy by minimizing a categorical loss. To tackle complex environments, we rely on a neural network representation of the policy. This approach is inspired by the Neural Fictitious Self Play (NFSP) method (Heinrich and Silver, 2016), developed initially for imperfect information games with a finite number of players, and adapted here to the MFG setting. The proposed algorithm, that we call D-AFP because it learns an average policy, is summarized in Alg. 8. Details are in Appx. C.

This allows us to learn an approximation of the MFNE policy with a single neural network instead of having it indirectly through a collection of neural networks for past BRs. After training, we can use this neural average policy in a straightforward way. Although the buffer is not needed after training, a drawback of this method is that during the training it requires to keep a buffer whose size increases linearly with the number of iterations. This motivates us to investigate a modification of OMD which is not only empirically faster, but also less memory consuming.

3.2 Deep Munchausen Online Mirror Descent

We now turn our attention to the combination of OMD and deep RL. One could simply use RL for the policy evaluation step by estimating the Q-function using equation (1). However, it is not straightforward to train a neural network to approximate the cumulative Q-function. To that end, we propose a reparameterization allowing us to compute the cumulative Q-function in an implicit way, building on the Munchausen trick from Vieillard et al. (2020) for classical RL (with a single agent and no mean-field interactions).

Reparameterization in the exact case. OMD requires summing up Q-functions to compute the regularized Q-function $\bar{q}$. However, this quantity $\bar{q}$ is hard to approximate as there exists no straightforward way to sum up neural networks. We note that this summation is done by Perolat et al. (2021a) via the use of the NeuRD loss. However, this approach relies on two types of approximations, as one must learn the correct Q-function, but also the correct sum. This is why we instead transform the OMD formulation into Munchausen OMD, which only relies on one type of approximation. We start by describing this reparameterization in the exact case, *i.e.*, without function approximation. We show that, in this case, the two formulations are equivalent. We consider the following modified Bellman equation:

$$\begin{cases} \tilde{Q}^{k+1}_{N_T+1}(x,a) = 0 \\ \tilde{Q}^{k+1}_{n-1}(x,a) = r(x,a,\mu^k_{n-1}) + \tau \ln \pi^k_{n-1}(a|x) \\ \quad + \mathbb{E}_{x',a'}\left[\tilde{Q}^{k+1}_n(x',a') - \tau \ln \pi^k_n(a'|x')\right], \end{cases} \tag{3}$$

where $x' \sim p_n(\cdot|x,a,\mu^k_{n-1})$ and $a' \sim \pi^k_n(\cdot|x')$. The red term penalizes the policy for deviating from the one in the previous iteration, π^k_{n-1}, while the blue term compensates for this change in the backward induction, as we will explain in the proof of Thm 1 below.

The Munchausen OMD (MOMD) algorithm for MFG is as follows: after initializing π^0, repeat for $k \geq 0$:

$$\begin{cases} \text{Distribution update: } \mu^k = \mu^{\tilde{\pi}^k} \\ \text{Regularized } Q\text{-function update: } \tilde{Q}_n^{k+1} \text{ as in (3)} \\ \text{Policy update: } \tilde{\pi}_n^{k+1}(\cdot|x) = \mathrm{softmax}(\frac{1}{\tau}\tilde{Q}_n^{k+1}(x,\cdot)). \end{cases}$$

Theorem 1. *MOMD is equivalent to OMD in the sense that $\tilde{\pi}^k = \pi^k$ for every k.*

As a consequence, despite seemingly artificial log terms, this method does not bias the Nash equilibrium (in contrast with, *e.g.*, Cui and Koeppl (2021) and Xie et al. (2020)). Thanks to this result, MOMD enjoys the same convergence guarantees as OMD, see (Hadikhanloo, 2017; Perolat et al., 2021b).

Proof. **Step 1: Softmax transform.** We first replace this projection by an equivalent viewpoint based on the Kullback-Leibler (KL) divergence, denoted by $KL(\cdot\|\cdot)$. We will write $Q_n^{k+1} = Q_n^{\pi^k,\mu^{\pi^k}}$ for short and $Q_n^0 = \bar{q}_n^0$. We have: $\bar{q}_n^{k+1} = \frac{1}{\tau}\sum_{\ell=0}^{k+1} Q_n^\ell$. We take π_n^0 as the uniform policy over actions, in order to have a precise equivalence with the following for $\bar{q}_n^0 = 0$. We could consider any π_n^0 with full support, up to a change of initialization $\bar{q}_n^0$. We have:

$$\pi_n^{k+1}(\cdot|x) = \mathrm{softmax}\left(\frac{1}{\tau}\sum_{\ell=0}^{k+1} Q_n^\ell(\cdot|x)\right) = \underset{\pi\in\Delta_A}{\mathrm{argmax}}\Big(\langle\pi, Q_n^{k+1}(x,\cdot)\rangle - \tau KL(\pi\|\pi_n^k(\cdot|x))\Big),$$

where $\langle\cdot,\cdot\rangle$ denotes the dot product.

Indeed, this can be checked by induction, using the Legendre-Fenchel transform: omitting n and x for brevity,

$$\pi^{k+1} \propto \pi^k e^{\frac{1}{\tau}q^{k+1}} \propto \pi^{k-1}e^{\frac{1}{\tau}Q^k}e^{\frac{1}{\tau}Q^{k+1}} = \pi^{k-1}e^{\frac{1}{\tau}(Q^k+Q^{k+1})} \propto \cdots \propto e^{\bar{q}^{k+1}}.$$

Step 2: Munchausen trick. Simplifying a bit notations,

$$\pi^{k+1} = \mathrm{argmax}(\langle\pi, Q^{k+1}\rangle - \tau KL(\pi\|\pi^k)) = \mathrm{argmax}(\langle\pi, \underbrace{Q^{k+1} + \tau\ln\pi^k}_{\tilde{Q}^{k+1}}\rangle \underbrace{-\tau\langle\pi,\ln\pi\rangle}_{+\tau\mathcal{H}(\pi)}) = \mathrm{softmax}\Big(\frac{1}{\tau}\tilde{Q}^{k+1}\Big)$$

where $\mathcal{H}$ denotes the entropy and we defined $\tilde{Q}_n^{k+1} = Q_n^{k+1} + \tau\ln\pi_n^k$. Since Q_n^{k+1} satisfies the Bellman equation (1) with π replaced by π^k and μ replaced by μ^k, we deduce that $\tilde{Q}_n^{k+1}$ satisfies (3). □

Remark: In OMD, $\frac{1}{\tau}$ (denoted α in the original paper (Perolat et al., 2021b)), is homogeneous to a learning rate. In the MOMD formulation, τ can be seen as a temperature.

Stabilizing trick. We have shown that MOMD is equivalent to OMD. However, the above version of Munchausen sometimes exhibits numerical instabilities. This is because, if an action a is suboptimal in a state x, then $\pi_n^k(a|x) \to 0$ as $k \to +\infty$, so $\tilde{Q}^k(x,a)$ diverges to $-\infty$ due to the relation $\tilde{Q}_n^{k+1} = Q_n^{k+1} + \tau\ln\pi_n^k$. This causes issues even in the tabular setting when we get close to a Nash equilibrium, due to numerical errors on very large numbers. To avoid this issue, we introduce another parameter $\alpha \in [0,1]$ and we consider the following modified Munchausen equation:

$$\check{Q}_{n-1}^{k+1}(x,a) = r_{n-1}(x,a,\mu_{n-1}^k) + \alpha\tau\log(\pi_{n-1}^{k-1}(a|x)) + \mathbb{E}_{x',a'}\Big[\tilde{Q}_n^{k+1}(x',a') - \tau\ln\pi_n^k(a'|x')\Big], \tag{4}$$

where $x' \sim p_n(\cdot|x,a,\mu_{n-1}^k)$ and $a' \sim \pi_n^k(\cdot|x')$. In fact, such iterations have a natural interpretation as they can be obtained by applying OMD to a regularized problem in which, when using policy π, a penalty $-(1-\alpha)\tau\log(\pi_n(\cdot|x_n))$ is added to the reward $r_n(x_n,a_n,\mu_n)$. Details are provided in Appx. D.

Deep RL version. Motivated by problems with large spaces, we then replace the Munchausen Q-function at iteration k, namely $\check{Q}^k$, by a neural network whose parameters θ^k are trained to minimize a loss function representing (4).

Since we want to learn a function of time, we consider (n, x) to be the state. To be specific, given samples of transitions $\left\{\Big((n_i, x_i), a_i, r_{n_i}(x_i, a_i, \mu^k_{n_i}), (n_i+1, x'_i)\Big)\right\}_{i=1}^{N_B}$, with $x'_i \sim p_{n_i}(x_i, a_i, \mu^k_{n_i})$, the parameter θ^k is trained using stochastic gradient descent to minimize the empirical loss: $\frac{1}{N_B}\sum_i \left|\check{Q}_\theta((n_i, x_i), a_i) - T_i\right|^2$, where the target T_i is:

$$T_i = -r_{n_i}(x_i, a_i, \mu^k_{n_i}) - \alpha\tau \log(\pi^{k-1}(a_i|(n_i, x_i))) - \sum_{a'} \pi^{k-1}(a'|(n_i+1, x'_i))\Big[\check{Q}_{\theta^{k-1}}((n_i+1, x'_i), a') - \tau \log(\pi^{k-1}(a'|(n_i+1, x'_i)))\Big]. \tag{5}$$

Here the time n is passed as an input to the Q-network along with x, hence our change of notation. This way of learning the Munchausen Q-function is similar to DQN, except for two changes in the target: (1) it incorporates the penalization for deviating from the previous policy, and (2) we do not take the argmax over the next action but an average according to the previous policy. See Alg. 9 for a simplified version and Appx. C for more details.

4. Experiments

In this section, we first discuss the metric used to assess quality of learning, detail baselines to which we compare our algorithms, and finally present numerical results on diverse and numerous environments.

4.1 Exploitability

To assess the quality of a learned equilibrium, we check whether, in response to the reward generated by the population MF flow, a typical player can improve their reward by deviating from the policy used by the rest of the population. This is formalized through the notion of exploitability. The exploitability of a policy π is defined as: $\mathcal{E}(\pi) = \max_{\pi'} J(\pi'; \mu^\pi) - J(\pi; \mu^\pi)$, where μ^π is the mean field flow generated from m_0 when using policy π. Intuitively a large exploitability means that, when the population plays π, any individual player can be much better off by deviating and choosing a different strategy, so π is far from being a Nash equilibrium policy. Conversely, an exploitability of 0 means that π is an MFNE policy. Similar notions are widely used in computational game theory (Zinkevich et al., 2007; Lanctot et al., 2009). In the sequel, we consider problems for which a BR can be computed exactly given a mean-field flow. Otherwise an approximate exploitability could be used as a good proxy to assess convergence, see *e.g.* Perrin et al. (2021b).

4.2 Baselines

To assess the quality of the proposed algorithms, we consider three baselines from the literature: Banach-Picard (BP) fixed point iterations, policy iterations (PI), and Boltzmann iterations (BI). FP can be viewed as a modification of the first one, while OMD as a modification of the second one. They have been discussed at the beginning of Sec. 2.2 and Sec. 2.3 respectively, in the exact case. Adapting them to the model-free setting with deep networks can be done in a similar way as discussed above for D-AFP and D-MOMD. See Appx. C for more details. The third baseline has been introduced recently by Cui and Koeppl (2021). It consists in updating in turn the population distribution and the policy, but here the policy is computed as a weighted softmax of the optimal Q-values (and hence requires the resolution of an MDP at each iteration). More precisely, given a reference policy π_B, a parameter $\eta > 0$, and the Q-function Q^k computed at iteration k, the new policy is defined as: $\pi^k_n(a|x) = \frac{\pi_{B,n}(a|x)\exp\left(Q^k_n(x,a)/\eta\right)}{\sum_{a'}\pi_{B,n}(a'|x)\exp\left(Q^k_n(x,a')/\eta\right)}$. In the plots, D-BP, D-AFP, D-PI, D-BI and D-MOMD refer respectively to Deep Banach-Picard iterations, Deep Average-network Fictitious Play, Deep Policy Iteration, Deep Boltzmann Iteration, and Deep Munchausen OMD.

4.3 Numerical results

Epidemics model. We first consider the SIS model of Cui and Koeppl (2021), which is a toy model for epidemics. There are two states: susceptible (S) and infected (I). Two actions can be used: social distancing (D) or going out (U). The probability of getting infected increases if more people are infected, and is smaller when using D instead of U. The transitions are: $p(S|I, D, \mu) = p(S|I, U, \mu) = 0.3$, $p(I|S, U, \mu) = 0.9^2 \cdot \mu(I)$, $p(I|S, D, \mu) = 0$, the reward is: $r(s, a, \mu) = -1_I(s) - 0.5 \cdot 1_D(s)$, and the horizon is $N_T = 50$. Note that, although this model has only two states, the

state dynamics is impacted by the distribution, which is generally challenging when computing MFG solutions. As shown in Fig. 1, both D-MOMD and D-AFP generate an exploitability diminishing with the learning steps, whereas the other baselines are not able to do so. Besides, D-MOMD generates smooth state trajectories, as opposed to the one observed in Cui and Koeppl (2021), that contained many oscillations. For this example and the following ones, we display the best exploitability curves obtained for each method after running sweeps over hyperparameters. See Appx. E for some instances of sweeps for D-MOMD.

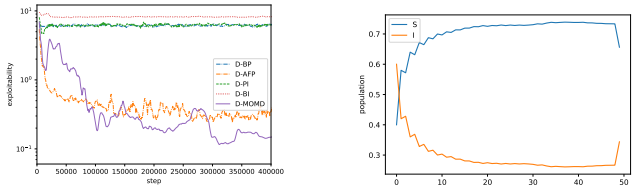

Figure 1: Left: exploitability. Right: evolution of the distribution obtained by the policy learned with D-MOMD.

Linear-Quadratic MFG. We then consider an example with more states, in 1D: the classical linear-quadratic environment of Carmona et al. (2015), whose main advantage is to benefit from an explicit closed form solution in a continuous space-time domain. We focus on a discretized approximation (Perrin et al., 2020) of the time grid $\{0, \dots, N_T\}$, where the dynamics of the underlying state process controlled by action (a_n) is given by $x_{n+1} = x_n + a_n \Delta_n + \sigma \epsilon_n \sqrt{\Delta_n}$, with $(\bar{m}_n)_n$ the average of the population states, Δ_n the time step and $(\epsilon_n)_n$ i.i.d. noises on $\{-3, \dots, 3\}$, approximations of $\mathcal{N}(0,1)$ random variables. Given a set of actions $(a_n)_n$ in state trajectory $(x_n)_n$ and a mean field flow $(\mu_n)_n$, the reward $r(x_n, a_n, \mu_n)$ of a representative player is given for $n < N_T$ by $\Big[-\frac{1}{2}|a_n|^2 + q a_n(\bar{m}_n - x_n) - \frac{\kappa}{2}|\bar{m}_n - x_n|^2 \Big] \Delta_n$, together with the terminal reward $-\frac{c_{term}}{2}|\bar{m}_{N_T} - x_{N_T}|^2$. The reward penalizes high actions, while providing incentives to remain close to the average state despite the noise $(\epsilon_n)_n$. For the experiments, we used $N_T = 10$, $\sigma = 1$, $\Delta_n = 1$, $q = 0.01$, $\kappa = 0.5$, $c_{term} = 1$ and $|X| = 100$.

In Figure 2, we see that the distribution estimated by D-MOMD concentrates, as is expected from the reward encouraging a mean-reverting behavior: the population gathers as expected into a bell-shaped distribution. The analytical solution (Appx. E in (Perrin et al., 2020)) is exact in a continuous setting (*i.e.*, when the step sizes in time, state and action go to 0) but only approximate in the discrete one considered here. Hence, we choose instead to use the distribution estimated by exact tabular OMD as a benchmark, as it reaches an exploitability of 10^{-12} in our experiments. Figure 2 (bottom right) shows that the Wasserstein distance between the learnt distribution by D-MOMD and the benchmark decreases as the learning occurs. In Figure 2 (bottom left), we see that D-MOMD and D-AFP outperform other methods in minimizing exploitability.

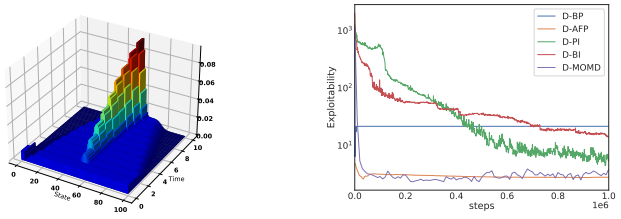

Figure 2: Left: evolution of the distribution obtained by the policy learned with D-MOMD. Right: moving average (50 iterations) of the L^1 distance between the exact solution and the one learned with D-MOMD.

Exploration. We now increase the state dimension and turn our attention to a 2-dimensional grid world example. The state is the position. An action is a move, and valid moves are: left, right, up, down, or stay, as long as the agent does not hit a wall. The reward is: $r(x, a, \mu) = r_{\text{pop}}(\mu(x))$, where $r_{\text{pop}}(\mu(x)) = -\log(\mu(x))$ discourages being in a crowded area – which is referred to as crowd aversion. Note that $\mathbb{E}_{x \sim \mu}(r_{\text{pop}}(\mu(x))) = \mathcal{H}(\mu)$, *i.e.*, the last term of the reward provides, in expectation, the entropy of the distribution. This setting is inspired by the one considered by Geist et al. (2022). The results are shown in Fig. 3. D-MOMD and D-AFP outperform all the baselines. The induced distribution matches our intuition: it spreads symmetrically until filling almost uniformly the four rooms.

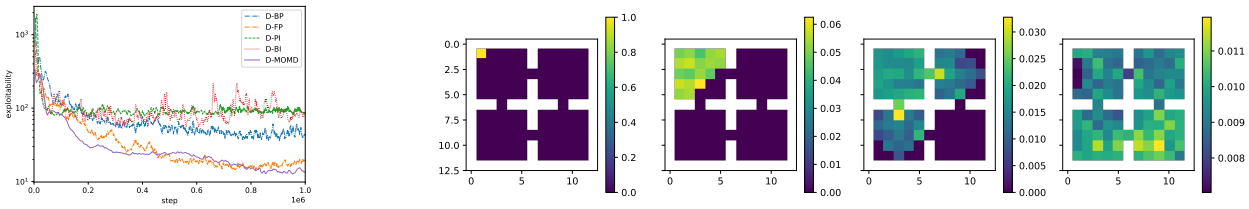

Figure 3: Left: exploitability. Right: evolution of the distribution obtained by the policy learned with D-MOMD.

Crowd modeling with congestion. We consider the same environment but with a maze, and complicate the reward: $r(x, a, \mu) = r_{\text{pos}}(x) + r_{\text{move}}(a, \mu(x)) + r_{\text{pop}}(\mu(x))$, where $r_{\text{pos}}(x) = -\text{dist}(x, x_{\text{ref}})$ is the distance to a target position x_{ref}, $r_{\text{move}}(a, \mu(x)) = -\mu(x)\|a\|$ is a penalty for moving ($\|a\| = 1$) which increases with the density $\mu(x)$ at x – which is called congestion effect in the literature. We see in Fig. 4 that D-MOMD outperforms the other methods.

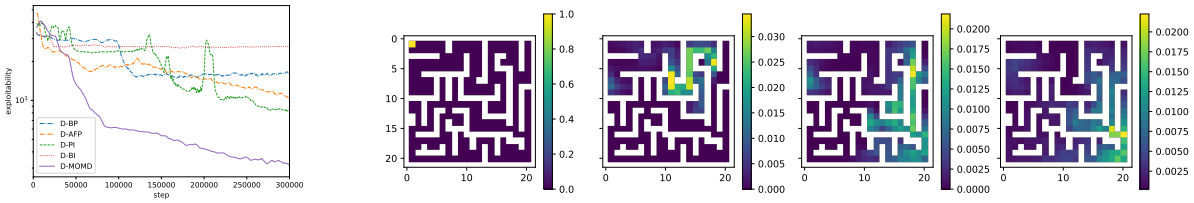

Figure 4: Maze. Left: exploitability. Right: evolution of the distribution with the policy learned with D-MOMD.

5. Conclusion

In this work, we proposed two scalable algorithms that can compute Nash equilibria in various MFGs in the finite horizon setting. The first one, D-AFP, is the first implementation of Fictitious Play for MFGs that does not need to keep all previous best responses in memory and that learns an average policy with a single neural network. The second one, D-MOMD, takes advantage of a subtle reparameterization to learn implicitly a sum of Q-functions usually required in the Online Mirror Descent algorithm. We demonstrated numerically that they both perform well on five benchmark problems and that D-MOMD consistently performs better than D-AFP and three baselines from the literature.

Related Work. Only a few works try to learn Nash equilibria in non-stationary MFGs. Guo et al. (2020a) establish the existence and uniqueness of dynamic Nash equilibrium under strict regularity assumptions in Section 9, but their RL and numerical experiments are restricted to the stationary setting. Perolat et al. (2021b) prove that continuous-time OMD converges to the Nash equilibrium in monotone MFGs, with numerical experiments involving trillions of states; however, their approach is purely based on the model and not RL. Mishra et al. (2020) propose an RL-based sequential approach to dynamic MFG, but this method relies on functions of the distribution, which is not scalable when the state space is large. Perrin et al. (2020) prove the convergence of FP for monotone MFGs and provide experimental results with model-free methods (Q-learning); however, they avoid the difficulty of mixing policies by relying on tabular representations for small environments. Cui and Koeppl (2021) propose a deep RL method with regularization to learn Nash equilibria in finite horizon MFGs; however, the regularization biases the equilibrium and they were not able to make it converge in complex examples, whereas D-MOMD outperforms the baselines in all the games we considered. Perrin et al. (2021a) and Perrin et al. (2021b) use another version of FP with neural networks, but their approach needs to keep in memory all the BRs learned during the iterations, which cannot scale to complex MFGs.

Future work. We would like to include more complex examples, with larger state spaces or even continuous spaces. Continuous state spaces should be relatively easy to address, as neural networks can handle continuous inputs, while continuous actions would require some adjustments particularly to compute argmax or softmax of Q-functions. Furthermore continuous spaces require manipulating continuous population distributions, raising additional questions related to how to represent and estimate them efficiently.

Acknowledgements

This research/project is supported in part by the National Research Foundation, Singapore under its AI Singapore Program (AISG Award No: AISG2-RP-2020-016), NRF 2018 Fellowship NRF-NRFF2018-07, NRF2019-NRF-ANR095 ALIAS grant, grant PIE-SGP-AI-2020-01, AME Programmatic Fund (Grant No. A20H6b0151) from the Agency for Science, Technology and Research (A*STAR) and Provost's Chair Professorship grant RGEPPV2101

References

Yves Achdou and Mathieu Laurière. Mean field games and applications: Numerical aspects. *Mean Field Games*, pages 249–307, 2020.

Berkay Anahtarci, Can Deha Kariksiz, and Naci Saldi. Q-learning in regularized mean-field games. *arXiv preprint arXiv:2003.12151*, 2020.

George W Brown. Iterative solution of games by fictitious play. *Activity analysis of production and allocation*, 13(1): 374–376, 1951.

Noam Brown and Tuomas Sandholm. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

Cacace, Simone, Camilli, Fabio, and Goffi, Alessandro. A policy iteration method for mean field games. *ESAIM: COCV*, 27:85, 2021. doi: 10.1051/cocv/2021081.

Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

Pierre Cardaliaguet and Saeed Hadikhanloo. Learning in mean field games: the fictitious play. *ESAIM Cont. Optim. Calc. Var.*, 2017. ISSN 1292-8119. doi: 10.1051/cocv/2016004.

René Carmona, Jean-Pierre Fouque, and Li-Hsien Sun. Mean field games and systemic risk. *Communications in Mathematical Sciences*, 13(4):911–933, 2015.

Kai Cui and Heinz Koeppl. Approximately solving mean field games via entropy-regularized deep reinforcement learning. In *proc. of AISTATS*, 2021.

Constantinos Daskalakis, Paul W. Goldberg, and Christos H. Papadimitriou. The complexity of computing a nash equilibrium. In *proc. of ACM Symposium on Theory of Computing*, 2006.

Romuald Elie, Julien Perolat, Mathieu Laurière, Matthieu Geist, and Olivier Pietquin. On the convergence of model free learning in mean field games. In *proc. of AAAI*, 2020.

Matthieu Geist, Julien Pérolat, Mathieu Laurière, Romuald Elie, Sarah Perrin, Olivier Bachem, Rémi Munos, and Olivier Pietquin. Concave utility reinforcement learning: the mean-field game viewpoint. In *proc. of AAMAS*, 2022.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 2020.

Xin Guo, Anran Hu, Renyuan Xu, and Junzi Zhang. A general framework for learning mean-field games. *arXiv preprint arXiv:2003.06069*, 2020a.

Xin Guo, Renyuan Xu, and Thaleia Zariphopoulou. Entropy regularization for mean field games with learning. *arXiv preprint arXiv:2010.00145*, 2020b.

Saeed Hadikhanloo. Learning in anonymous nonatomic games with applications to first-order mean field games. *arXiv preprint arXiv:1704.00378*, 2017.

Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016.

Minyi Huang, Roland P. Malhamé, and Peter E. Caines. Large population stochastic dynamic games: closed-loop McKean-Vlasov systems and the Nash certainty equivalence principle. *Communications in Information & Systems*, 6, 2006.

Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. Monte carlo sampling for regret minimization in extensive games. In *proc. of NeurIPS*, volume 22, pages 1078–1086, 2009.

Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. *arXiv preprint arXiv:1711.00832*, 2017.

Jean-Michel Lasry and Pierre-Louis Lions. Mean field games. *Jpn. J. Math.*, 2007. ISSN 0289-2316. doi: 10.1007/s11537-007-0657-8.

Rajesh K Mishra, Deepanshu Vasal, and Sriram Vishwanath. Model-free reinforcement learning for non-stationary mean field games. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 1032–1037. IEEE, 2020.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

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 *proc. of ICML*, 2021a.

Julien Perolat, Sarah Perrin, Romuald Elie, Mathieu Laurière, Georgios Piliouras, Matthieu Geist, Karl Tuyls, and Olivier Pietquin. Scaling up mean field games with online mirror descent. *arXiv preprint arXiv:2103.00623*, 2021b.

Sarah Perrin, Julien Pérolat, Mathieu Laurière, Matthieu Geist, Romuald Elie, and Olivier Pietquin. Fictitious play for mean field games: Continuous time analysis and applications. In *proc. of NeurIPS*, 2020.

Sarah Perrin, Mathieu Laurière, Julien Pérolat, Romuald Élie, Matthieu Geist, and Olivier Pietquin. Generalization in mean field games by learning master policies. *arXiv preprint arXiv:2109.09717*, 2021a.

Sarah Perrin, Mathieu Laurière, Julien Pérolat, Matthieu Geist, Romuald Élie, and Olivier Pietquin. Mean field games flock! the reinforcement learning way. In *proc. of IJCAI*, 2021b.

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

Karl Tuyls and Gerhard Weiss. Multiagent learning: Basics, challenges, and prospects. *AI Magazine*, 2012.

Nino Vieillard, Olivier Pietquin, and Matthieu Geist. Munchausen reinforcement learning. In *proc. of NeurIPS*, 2020.

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

Qiaomin Xie, Zhuoran Yang, Zhaoran Wang, and Andreea Minca. Provable fictitious play for general mean-field games. *CoRR*, abs/2010.04211, 2020.

Qiaomin Xie, Zhuoran Yang, Zhaoran Wang, and Andreea Minca. Learning while playing in mean-field games: Convergence and optimality. In *proc of ICML*, 2021.

Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *proc. of NeurIPS*, 2007.

Appendix A. More Numerical Examples

A.1 Multi-population chasing.

We finally turn to an extension of the MFG framework, where agents are heterogeneous: each type of agent has its own dynamics and reward function. The environment can be extended to model multiple populations by simply extending the state space to include the population index on top of the agent's position. Following Perolat et al. (2021b), we consider three populations and rewards of the form: for population $i = 1, 2, 3$, $r^i(x, a, \mu^1, \mu^2, \mu^3) = -\log(\mu^i(x)) + \sum_{j \neq i} \mu^j(x)\bar{r}^{i,j}(x)$. where $\bar{r}^{i,j} = -\bar{r}^{j,i}$, with $\bar{r}^{1,2} = -1$, $\bar{r}^{1,3} = 1$, $\bar{r}^{2,3} = -1$. The initial distributions are in the corners, the number of agents of each population is fixed, and the reward encourages the agent to chase the population it dominates and flee the dominating one. We see in Fig. 5 that D-AFP outperform the baselines and D-MOMD performs even better.

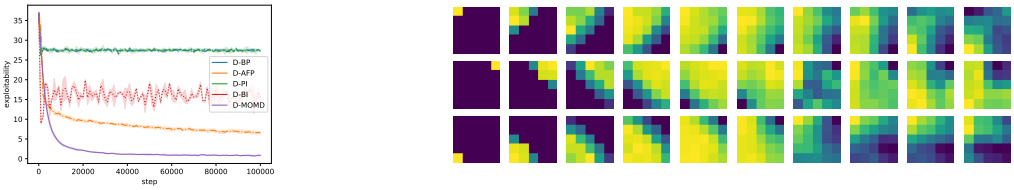

Figure 5: Multi-population chasing. Left: Exploitability. Right: evolution of the distributions for the populations.

Appendix B. Algorithms in the exact case

In this section we present algorithms to compute MFG equilibria when the model is fully known.

B.1 Subroutines

The distribution computation for a given policy π is described in Alg. 1 using forward (in time) iterations. The evaluation of the state-action value function for a *given policy* π and mean field flow μ is described in Alg. 2. The computation of the optimal value function $Q^{*,\mu}$ for a given μ is described in Alg. 3. A best response against μ can be obtained by running this algorithm and then taking an optimizer of $Q_n^{*,\mu}(x, \cdot)$ for each n, x.

Algorithm 1 Forward update for the distribution

Parameter: policy $\pi = (\pi_n)_{n=0,\dots,N_T}$
Output: mean field flow $\mu = (\mu_n)_{n=0,\dots,N_T} = \mu^\pi = (\mu_n^\pi)_{n=0,\dots,N_T}$
Let $\mu_0 = m_0$
for $n = 1 \dots, N_T$ **do**
    Let $\mu_n^\pi = P_{n-1}^{\mu_{n-1}^\pi, \pi_{n-1}} \mu_{n-1}^\pi$
end for
Return $\mu = (\mu_n)_{n=0,\dots,N_T}$

B.2 Main algorithms with fully known model

Banach-Picard Fixed point iterations are presented in Alg. 4. Fictitious Play is described in Alg. 5. Policy iteration (for MFG) is presented in Alg. 6. Online Mirror Descent is described in Alg. 7.

Algorithm 2 Backward induction for the value function evaluation

Parameters: policy $\pi = (\pi_n)_{n=0,\dots,N_T}$, mean field flow $\mu = (\mu_n)_{n=0,\dots,N_T}$
Output: state-action value function $Q^{\pi,\mu}$
Let $Q_{N_T}(x,a) = r_{N_T}(x,a,\mu_{N_T})$
for $n = N_T - 1, \dots, 0$ **do**
    Compute

$$Q_n(x,a) = r_n(x,a,\mu_n) + \mathbb{E}_{x'\sim p_n(\cdot|x,a,\mu_n), a'\sim\pi_n(\cdot|x')}[Q_{n+1}(x',a')]$$

    where the expectation is computed in an exact way using the knowledge of the transition:

$$\mathbb{E}[Q_{n+1}(x',a')] = \sum_{x'} p_n(x'|x,a,\mu_n) \sum_{a'} \pi_{n+1}(a'|x') Q_{n+1}(x',a')$$

end for
Return $Q = (Q_n)_{n=0,\dots,N_T}$

Algorithm 3 Backward induction for the optimal value function

Parameters: mean field flow $\mu = (\mu_n)_{n=0,\dots,N_T}$
Output: optimal state-action value function $Q^{*,\mu}$
Let $Q_{N_T}(x,a) = r_{N_T}(x,a,\mu_{N_T})$
for $n = N_T - 1, \dots, 0$ **do**
    Compute

$$Q_n(x,a) = r_n(x,a,\mu_n) + \underbrace{\mathbb{E}_{x'\sim p_n(\cdot|x,a,\mu_n)}[\max_{a'\in A} Q_{n+1}(x',a')]}_{=\sum_{x'} p_n(x'|x,a,\mu_n) \max_{a'} Q_{n+1}(x',a')}$$

    where the expectation is computed in an exact way using the knowledge of the transition p_n
end for
Output: $Q = (Q_n)_{n=0,\dots,N_T}$

Algorithm 4 Banach-Picard (BP) fixed point

Input: Number of iterations K; optional softmax temperature η
Initialize π^0
for $k = 0, \dots, K$ **do**
    Forward update: Compute $\mu^k = \mu^{\pi^{k-1}}$, *e.g.* using Alg. 1 with $\pi = \pi^{k-1}$
    Best response computation: Compute a BR π^k against μ^k, *e.g.* by computing Q^{*,μ^k} using Alg. 3 with $\mu = \mu^k$ and then taking $\pi_n^k(\cdot|x)$ as a(ny) distribution over $\operatorname{argmax} Q_n^{*,\mu^k}(x,\cdot)$ for every n, x; alternatively, compute a soft version: $\pi_n^k(\cdot|x) = \operatorname{softmax}(Q_n^{*,\mu^k}(x,\cdot)/\eta)$
end for
Output: $\mu^K = (\mu_n^K)_{n=0,\dots,N_T}$ and policy $\pi^K = (\pi_n^K)_{n=0,\dots,N_T}$

Algorithm 5 Fictitious Play (FP)

Input: Number of iterations K
Initialize π^0
for $k = 0, \ldots, K$ **do**
  Forward update: Compute $\mu^k = \mu^{\pi^{k-1}}$, *e.g.* using Alg. 1 with $\pi = \pi^{k-1}$
  Average distribution update: Compute $\bar{\mu}^k$ as the average of $(\mu^0, \ldots, \mu^{\pi^k})$:

$$\bar{\mu}_n^k(x) = \frac{1}{k}\sum_{i=1}^{k} \mu_n^i(x) = \frac{k-1}{k}\bar{\mu}_n^{k-1}(x) + \frac{1}{k}\mu_n^k(x)$$

  Best response computation: Compute a BR π^k against $\bar{\mu}^k$, *e.g.* by computing $Q^{*,\bar{\mu}^k}$ using Alg. 3 and then taking $\pi_n^k(\cdot|x)$ as a(ny) distribution over $\operatorname{argmax} Q_n^{*,\bar{\mu}^k}(x, \cdot)$ for every n, x
end for
Output: $\bar{\mu}^K = (\bar{\mu}_n^K)_{n=0,\ldots,N_T}$ and policy $\bar{\pi}^K = (\bar{\pi}_n^K)_{n=0,\ldots,N_T}$ generating this mean field flow

Algorithm 6 Policy Iteration (PI)

Parameters: softmax temperature η; number of iterations K
Initialize the sequence of tables $(\bar{q}_n^0)_{n=0,\ldots,N_T}$, *e.g.* with $\bar{q}_n^0(x, a) = 0$ for all n, x, a
Let the projected policy be: $\pi_n^0(a|x) = \operatorname{softmax}(\bar{q}_n^0(x, \cdot)/\eta)(a)$ for all n, x, a
for $k = 1, \ldots, K$ **do**
  Forward Update: Compute $\mu^k = \mu^{\pi^{k-1}}$, *e.g.* using Alg. 1 with $\pi = \pi^{k-1}$
  Backward Update: Compute $Q^k = Q^{\pi^{k-1},\mu^k}$, *e.g.* using backward induction Alg. 2 with $\pi = \pi^{k-1}$ and $\mu = \mu^k$ and then let $\pi_n^k(\cdot|x)$ be a(ny) distribution over $\operatorname{argmax} Q_n^{*,\bar{\mu}^k}(x, \cdot)$ for every n, x; alternatively, compute a soft version: $\pi_n^k(\cdot|x) = \operatorname{softmax}(Q_n^k(x, \cdot)/\eta)$
end for
Output: Q^K, π^K

Algorithm 7 Online Mirror Descent (OMD)

Parameters: inverse learning rate parameter τ; number of iterations K
Initialize the sequence of tables $(\bar{q}_n^0)_{n=0,\ldots,N_T}$, *e.g.* with $\bar{q}_n^0(x, a) = 0$ for all n, x, a
Let the projected policy be: $\pi_n^0(a|x) = \operatorname{softmax}(\bar{q}_n^0(x, \cdot))(a)$ for all n, x, a
for $k = 1, \ldots, K$ **do**
  Forward Update: Compute $\mu^k = \mu^{\pi^{k-1}}$, *e.g.* using Alg. 1 with $\pi = \pi^{k-1}$
  Backward Update: Compute $Q^k = Q^{\pi^{k-1},\mu^k}$, *e.g.* using backward induction Alg. 2 with $\pi = \pi^{k-1}$ and $\mu = \mu^k$
  Update the regularized Q-function and the projected policy: for all n, x, a,

$$\bar{q}_n^k(x, a) = \bar{q}_n^{k-1}(x, a) + \frac{1}{\tau}Q_n^k(x, a)$$
$$\pi_n^k(a|x) = \operatorname{softmax}(\bar{q}_n^k(x, \cdot))(a)$$

end for
Return $\bar{q}^K, \pi^K$

Appendix C. Deep RL Algorithms

We now present details on the deep RL algorithms used or developed in this paper. In this work, we focus on the use of deep RL for policy computation from the point of view of a representative agent. We assume that this agent has access to an oracle that can return $r_n(x_n, a_n, \mu_n)$ and a sample of $p_n(\cdot|x_n, a_n, \mu_n)$ when the agent follows is in state x_n and uses action a_n. In fact, the collection of samples is split into episodes. At each episode, the agent start from some x_0 sampled from m_0. Then it evolves by following the current policy, and the transitions are added to a replay buffer.

In order to focus on the errors due to the deep RL algorithm, we assume that the distribution is updated in an exact way following Alg. 1.

For the Deep RL part, the approaches can be summarized as follows:

- Alg. 2: Intuitively, the updates are replaced by stochastic gradient steps so as to minimize the following loss with respect to θ:

$$\left|Q_\theta((n, x_n), a_n) - r_n(x_n, a_n, \mu_n) - \hat{\mathbb{E}}_{x'_{n+1}\sim p_n(\cdot|x_n,a_n,\mu_n), a'_{n+1}\sim\pi_{n+1}(\cdot|x'_{n+1})}[q((n+1, x'_{n+1}), a'_{n+1})]\right|^2, \tag{6}$$

  where $\hat{\mathbb{E}}$ denotes an empirical expectation over a finite number of samples picked from the replay buffer and q is replaced by a target network $Q_{\theta'}$ whose parameters are frozen while training θ and that are updated less frequently than θ. Combined with Policy Iteration (Alg. 6), this leads to the algorithm referred to as **Deep Policy Iteration (D-PI)**.

- Alg. 3: We can proceed similarly, except that the target becomes:

$$r_n(x_n, a_n, \mu_n) + \hat{\mathbb{E}}_{x'_{n+1}\sim p_n(\cdot|x_n,a_n,\mu_n)}[\max_{a'} q((n+1, x'_{n+1}), a')].$$

  In fact, in our implementation we use DQN (Mnih et al., 2013) as a subroutine for the BR computation. Combined with Banach-Picard iterations (Alg. 4), this leads directly to the algorithm referred to as **Deep Banach-Picard (D-BP)**.

- To obtain **Deep Average-network Fictitious Play (D-AFP)** (Algo. 8), at each iteration, the best response against the current distribution is learnt using DQN (Mnih et al., 2013). This policy is used to generate trajectories, whose state-action samples are added to a reservoir buffer $\mathcal{M}_{SL}$. This buffer stores state-actions generated using past policies from previous iterations. Then, an auxiliary neural network for the logits representing the average policy is trained using supervised learning using $\mathcal{M}_{SL}$: stochastic gradient is used to find $\bar{\theta}$ minimizing approximately the loss:

$$\mathcal{L}(\bar{\theta}) = \mathbb{E}_{(s,a)\sim\mathcal{M}_{SL}}\left[-\log\left(\bar{\pi}_{\bar{\theta}}(a|s)\right)\right]$$

  In our implementation, for the representation of the average policy, we use a neural network $\bar{\ell}_\omega$ with parameteres ω for the logits, and then we compute the policy as: $\bar{\pi} = \mathrm{softmax}(\bar{\ell}_\omega)$. This step is reminiscent of Neural Fictitious Self Play introduced in Heinrich and Silver (2016), although the overall algorithm is different.

- To obtain **Deep Munchausen Online Mirror Descent (D-MOMD)** (Algo. 9), we can simply modify the target in (6) as follows:

$$\Big|Q_\theta((n, x_n), a_n) - r_n(x_n, a_n, \mu_n) - \tau \ln \pi_{n-1}(a_n|x_n) \\ - \hat{\mathbb{E}}_{x'_{n+1}\sim p_n(\cdot|x_n,a_n,\mu_n),} \sum_{a'} \pi_n(a'|x_n)[q((n+1, x'_{n+1}), a'_{n+1}) - \tau \ln \pi_n(a'|x'_{n+1})]\Big|^2,$$

  where $q = Q_{\theta'}$ is a target network whose parameters θ' are frozen while training θ.

C.1 Pseudocode

Algorithm 8 D-AFP

Initialize an empty reservoir buffer $\mathcal{M}_{SL}$ for supervised learning of average policy
Initialize mean field distribution flow $\bar{\mu}^0 = m_0$
for $k = 1, \dots, K$ **do**
  1. BR: Train $\hat{\pi}_{\theta^k}$ against $\bar{\mu}^{k-1}$, *e.g.* using DQN
  Collect $N_{samples}$ state-action using $\hat{\pi}_{\theta^k}$ and add them to $\mathcal{M}_{SL}$
  2. Average policy: Update $\bar{\pi}_{\bar{\theta}^k}$ by adjusting $\bar{\theta}^k$ (through gradient descent) to minimize:

$$\mathcal{L}(\bar{\theta}) = \mathbb{E}_{(s,a)\sim\mathcal{M}_{SL}}\left[-\log\left(\bar{\pi}_{\bar{\theta}}(a|s)\right)\right],$$

  where $\bar{\pi}_{\bar{\theta}}$ is the neural net policy with parameters $\bar{\theta}$
  3. Distribution: generate $\bar{\mu}^k$ with $\bar{\pi}_{\bar{\theta}^k}$
end for
Return $\bar{\mu}^K, \bar{\pi}_{\bar{\theta}^K}$

Algorithm 9 D-MOMD

Input: Munchausen parameters τ and α; numbers of OMD iterations K and DQN estimation iterations L
Output: cumulated Q value function, policy π
Initialize the parameters θ^0
Set $\pi^0(a|(n,x)) = \mathrm{softmax}\Big(\frac{1}{\tau}\check{Q}_{\theta^0}((n,x),\cdot)\Big)(a)$
for $k = 1, \dots, K$ **do**
  1. Distribution: Generate μ^k with π^{k-1}
  2. Value function: Initialize θ^k
  for $\ell = 1, \dots, L$ **do**
    Sample a minibatch of N_B transitions: $\left\{\Big((n_i,x_i),a_i,r_{n_i}(x_i,a_i,\mu^k_{n_i}),(n_i+1,x'_i)\Big)\right\}_{i=1}^{N_B}$ with $n_i \leq N_T$, $x'_i \sim p_{n_i}(\cdot|x_i,a_i,\mu^k_{n_i})$ and a_i is chosen by an $\epsilon-$greedy policy based on $\check{Q}_{\theta^k}$
    Update θ^k with one gradient step of:

$$\theta \mapsto \tfrac{1}{N_B}\textstyle\sum_{i=1}^{N_B}\left|\check{Q}_\theta((n_i,x_i),a_i) - T_i\right|^2$$

    where T_i is defined in (5)
  end for
  3. Policy: for all n, x, a, let

$$\pi^k(a|(n,x)) = \mathrm{softmax}\Big(\tfrac{1}{\tau}\check{Q}_{\theta^k}((n,x),\cdot)\Big)(a)$$

end for
Return $\check{Q}_{\theta^K}, \pi^K$

Appendix D. Details on the link between MOMD and regularized MDPs

Consider regularizing the MFG with only entropy, that is

$$J(\pi, \mu) = \mathbb{E}_\pi \left[\sum_{n=0}^{N_T} (r_n(s_n, a_n, \mu_n) - (1-\alpha)\tau \ln \pi_n(a_n|s_n)) \right]. \tag{7}$$

Notice that we choose $(1-\alpha)\tau$ here because it will simplify later, but it would work with any temperature (or learning rate from the OMD perspective).

Now, let's solve this MFG with OMD with learning rate $(\alpha\tau)^{-1}$, adopting the KL perspective. The corresponding algorithm is:

$$\pi_n^{k+1} \in \operatorname{argmax} \langle \pi_n, q_n^k \rangle - \alpha\tau \mathrm{KL}(\pi_n || \pi_n^k) + (1-\alpha)\tau \mathcal{H}(\pi_n) \tag{8}$$

$$\begin{cases} q_{N_T}^{k+1} = r_{N_T}^{k+1} \\ q_n^{k+1} = r_n^{k+1} + \gamma P \langle \pi_{n+1}^{k+1}, q_{n+1}^{k+1} - (1-\alpha)\tau \ln \pi_{n+1}^{k+1} \rangle \end{cases} \tag{9}$$

Next, using $Q_n^k = q_n^k + \alpha\tau \ln \pi_n^k$, we can rewrite the evaluation part as:

$$\pi_n^{k+1} = \operatorname{softmax}\left(\frac{Q_n^k}{\tau} \right) \tag{10}$$

$$\begin{cases} Q_{N_T}^{k+1} = r_{N_T}^{k+1} + \alpha\tau \ln \pi_{N_T}^{k+1} \\ Q_n^{k+1} = r_n^{k+1} + \alpha\tau \ln \pi_n^{k+1} + \gamma P \langle \pi_{n+1}^{k+1}, Q_{n+1}^{k+1} - \tau \ln \pi_{n+1}^{k+1} \rangle \end{cases} \tag{11}$$

We remark that it corresponds to the "scaled" version of Munchausen OMD, meaning that it amounts to solving the MFG regularized with $(1-\alpha)\tau\mathcal{H}(\pi)$ with OMD. We retrieve with $\alpha = 1$ the unscaled version of Munchausen OMD, addressing the unregularized MFG. It also makes a connection with the Boltzmannn iteration method of Cui and Koeppl (2021), in which a similar penalization except that the penalty involves a fixed policy instead of using the current policy. Their prior descent method, in which the reference policy is updated from time to time, can thus be viewed as a first step towards Munchausen OMD.

Appendix E. Hyperparameters sweeps

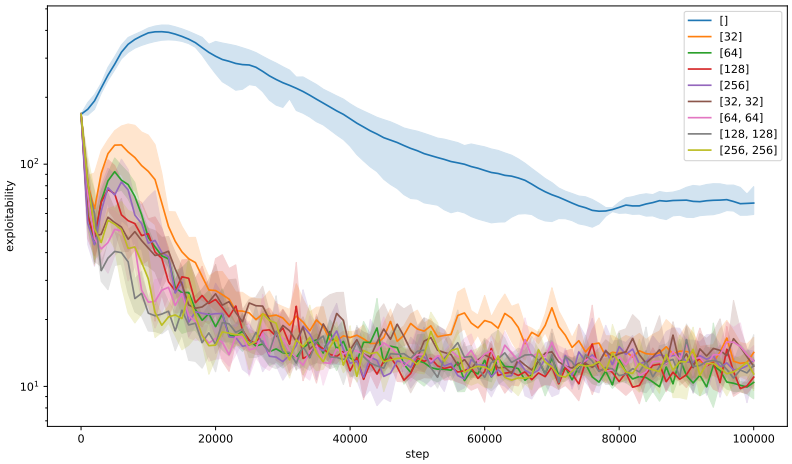

Figure 6: D-MOMD, Exploration game with four rooms: Sweep over the network size. The neural network architecture is feedforward fully connected with one or two hidden layers, except for the curve with label `[]`, which refers to a linear function. This illustrates in particular that the policy can not be well approximated using only linear functions, hence the need for non-linear approximations, which raises the difficulty of averaging or summing such approximations (here neural networks).

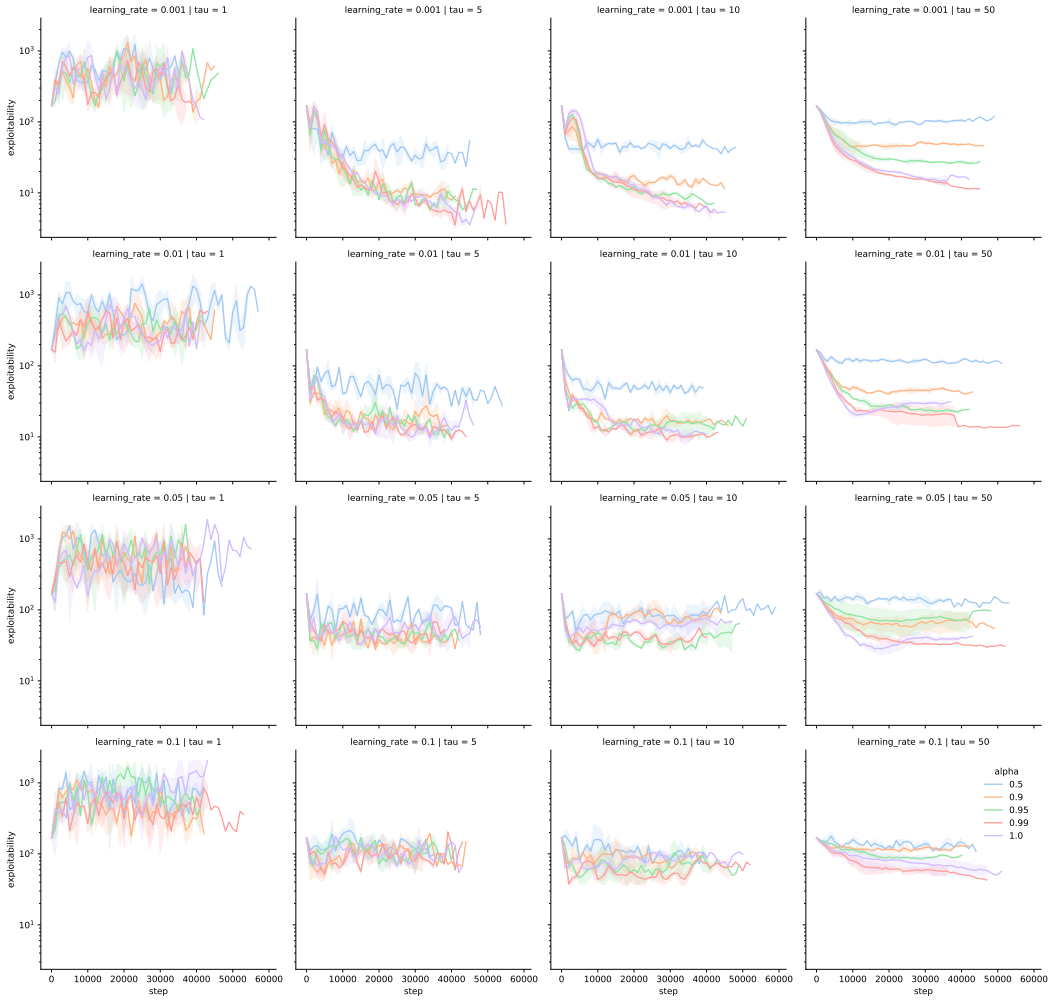

Figure 7: D-MOMD, Exploration game with four rooms: Sweep over τ, α and learning rate.