Large Batch Sharing

Anonymous authors
Paper under double-blind review

Abstract

By reusing experiences collected from past different policies, experience replay significantly improves the training efficiency of reinforcement learning algorithms. Rapid convergence occurs when learning is based on pertinent experiences that offer valuable information. Nonetheless, how to effectively combine experience replay with multi-agent reinforcement learning is still an open challenge. We study how sharing collected experiences helps the training process and show that sharing a small amount of selected experiences between agents improves the learning process compared to the baseline where each agent is independent. The shared experiences are selected by each agent on internal statistics, ensuring their meaningfulness. Our first results on the multi-agent Pursuit environment highlight an improvement by a substantial margin and need to be consolidated by complementary experiments.

1 Introduction

Multi-Agent Systems (Van der Hoek & Wooldridge, 2008) have benefited from Reinforcement Learning (Sutton & Barto, 2018, RL) as it enabled to address many issues (Zhang et al., 2021). In particular, RL has made improvements in adaptive decision-making (Busoniu et al., 2008), handles partial observability (Omidshafiei et al., 2017), promotes emergent behavior and self-organization (Li et al., 2006), provides decentralized control tools (Panait & Luke, 2005) and allows agents to generalize over different tasks thanks to transfer learning (Foerster et al., 2016). However, when each agent learns independently, Multi-Agent Reinforcement Learning (MARL) experiences difficulties in the learning process due to the non-stationarity of the environment. Even though the convergence guarantees of MARL are an active field of research (Hernandez-Leal et al., 2019), MARL still succeeds in learning in complex environments (Tampuu et al., 2017).

In deep RL, neural network policies and value functions can be learnt thanks to stochastic gradient descent algorithms (Robbins & Monro, 1951, SGD) sampling an experience replay memory (Lin, 1992). This replay memory, or replay buffer, stores the transitions encountered along the interaction with the environment. SGD-based algorithms exploit such buffers to learn relevant functions, such as the Q-function in the case of Deep Q-Networks (Mnih et al., 2015), the return distribution for distributional approaches (Bellemare et al., 2017), or an actor and a critic (Lillicrap et al., 2016; Haarnoja et al., 2018) in the case of continuous state-action space problems. Most deep RL algorithms boildown to a sequence of SGD-based, supervised learning problems. In supervised learning, importance sampling can be used to speed up the convergence of SGD, by sampling non-uniformly the training set to reduce the variance of the stochastic gradient estimate. As there is a link between supervised learning and RL, accelerating the convergence thanks to non-uniform sampling has also been explored in the latter with Prioritized Experience Replay (Schaul et al., 2016, PER), drawing inspiration from Prioritized Sweeping (Moore & Atkeson, 1993). Extensions, modifications, and foundations of PER have been proposed, as in (Wang & Ross, 2019) and (Lahire et al., 2022). In this paper, we show that some of the techniques initially designed to accelerate convergence speed in the single-agent case can be used to help agents in their learning process.

In a partially observable environment which is "anonymous" (the environment behaves the same way for all agents) to homogeneous agents collecting at each time step their own reward, this

work proposes a sharing-experience scheme among agents. Each agent is initialized with its own neural network and replay buffer. At each learning step, each agent updates its neural network parameters thanks to an SGD step with a mini-batch composed of experiences from its own replay buffer, as well as experiences collected by other agents. The relevance of a given experience to a given agent is checked before being used for the SGD step thanks to statistics specific to the updated agent. Our experience-sharing method compares to independent learning, where agents are initialized with their own neural network and replay buffer and do not share anything. We evaluate the benefit of sharing a small amount of experiences collected by other agents in a mixed collaborative-competitive environment.

This article is a work-in-progress and must be considered a position paper. It is structured as follows. Section 2 clarifies the background and the goals of this work. Then Section 3 proposes an algorithm to improve learning in multi-agent settings by sampling efficiently useful experiences collected by the other agents. Section 4 empirically evaluates the proposed algorithm. We discuss each separate aspect of sampling, its explanations and perspectives. Section 5 discusses the limitations of this work and contrasts our findings with the existing literature. Section 6 summarizes and concludes.

2 Goal: Adaptive Multi-Agent Systems

We model our MARL problem as a Partially Observable Stochastic Game (POSG), a generalization of Stochastic (Markov) Games (Littman, 1994) to settings where agents are only able to observe parts of the state of the environment (Hansen et al., 2004). A POSG is a tuple $(M, \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, \mathcal{T}_e)$, where M is the number of agents, $\mathcal{S}$ is the set of all possible global states of the environment, $\mathcal{A} := A_1 \times A_2 \times ... \times A_M$ represents the set of actions, where A_i is the action space of agent i, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition probability function between global states, based on the joint action of the agents, $\mathcal{R} := R_1 \times R_2 \times ... \times R_M$ is the reward function, where $R_i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the individual reward function of agent i, $\mathcal{O} := O_1 \times O_2 \times ... \times O_M$ is the set of observations with O_i representing the individual observation set for each agent i, and $\mathcal{T}_e : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{O})$ is the observation function.

The behaviour of an agent is defined by its policy $\pi_i : O_i \times A_i \to [0, 1]$. The performance of a policy can be assessed through its Q-function $Q^{\pi_i} = \mathbb{E}[\sum_t \gamma^t r_{i,t} | \boldsymbol{\pi}]$, where $\boldsymbol{\pi} = (\pi_1, ..., \pi_M)$ is the joint policy of the agents acting in the environment, $\gamma \in [0, 1]$ is the discount factor, and $r_{i,t} = R_i(s_t, \boldsymbol{a}_t, s_{t+1})$ is the reward obtained by agent i at time step t for the joint action $\boldsymbol{a}_t \in \mathcal{A}$ at state $s_t \in \mathcal{S}$ and transitioning to the next state $s_{t+1} \in \mathcal{S}$. The goal of each agent is to find its policy π_i^* that has the largest possible Q-function.

As set previously, this work focuses on agents receiving their own reward in a partially observable environment that reacts to each agent the same way. If an action taken by an agent in a particular state yields reward, so does the environment for any other agent (the environment is said to be "anonymous"). We also make the assumption of homogeneous agents, sharing the same set of individual actions. All assumptions related to our multi-agent setting will be discussed in Section 5 along with extensions and limitations of our work.

In such a framework, two baseline algorithms can be used. First, one single agent, taking as input the observation and outputting the action for the agent receiving the observation can be trained thanks to the gathering of the agents' interactions with the environment. This method belongs to the Parameter Sharing (PS) algorithms, where neural network(s) and replay buffer (if any) are shared among all agents. The second baseline we consider are independent learners, where each agent trains its own neural network(s) thanks to its own interactions with the environment. Neural network(s) parameters (and replay buffers) remain private.

Our contribution, consisting in designing an efficient sharing of experiences between agents, can be applied to any underlying model-free deep RL algorithm using a replay buffer. For pedagogical purposes, we focus on the Deep Q-Networks (Mnih et al., 2015, DQN) algorithm, which is off-policy

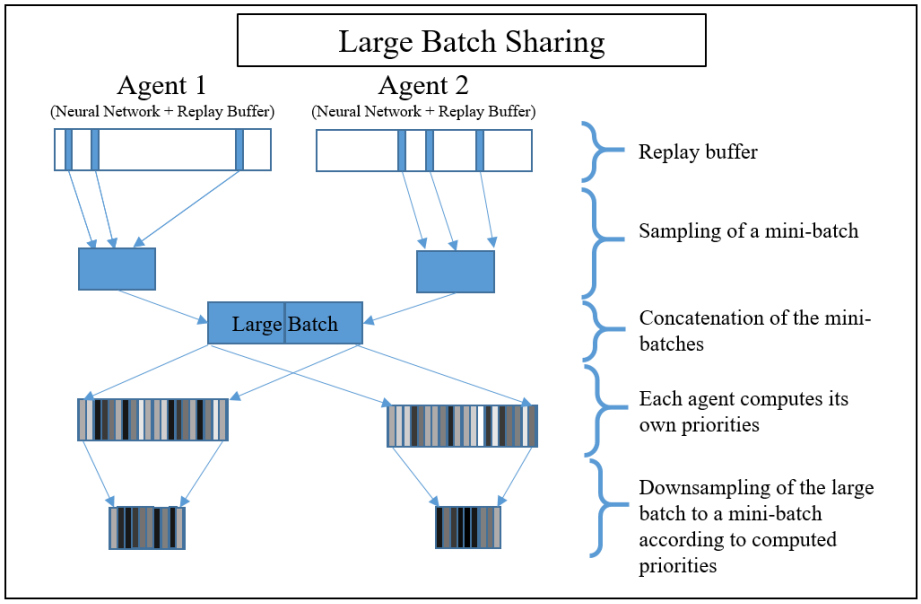

Figure 1: Illustration of the Large Batch Sharing algorithm in the case of two agents.

and designed for discrete actions, and we will explain in Section 5 how the proposed methods can be extended to other settings.

In single-agent RL with full observability, the optimal Q-function obeys equation $Q^*(s,a) = \mathbb{E}_{s',r}\left[r + \gamma \max_{a'} Q^*(s',a')\right]$, called the Bellman optimality equation. DQN is the approximate Value Iteration algorithm that uses a replay buffer of N samples (s,a,r,s'), a deep neural network Q_θ, and a few steps of gradient descent to minimize the Bellman optimality equation. Specifically, at each training step, DQN aims to take a gradient step on the empirical loss $\frac{1}{N}\sum_{i=1}^{N} \ell(Q_\theta(s_i,a_i),y_i)$, with $y_i = r_i + \gamma \max_{a'} Q_n(s_i',a')$ and ℓ a loss (for example the L2 loss). Minimization of this empirical loss by SGD implies drawing at each step a mini-batch of B transitions from the replay buffer and taking a descent step $\theta_{t+1} = \theta_t - \eta d$ in the direction of the gradient estimate $d = \frac{1}{B}\sum_{i=1}^{B} \nabla_\theta \ell(Q_\theta(s_i,a_i),y_i)$, with learning rate η.

The goal of this research is not to propose an algorithm outperforming the two baselines we previously mentioned, namely the PS and the IDQN (Independent DQN) algorithms, in the multi-agent setting we identified. In this work, we aim at studying multi-agent systems already deployed, but in a changing environment, forcing the agents to adapt to continue completing the task. In such an environment, the baseline to compare with is the IDQN, even though we will also test the PS in the comparisons. The IDQN algorithm allows each agent to continue learning by using its own interactions with the changing environment. If the communication between agents is allowed, all the replay buffers could be shared. The goal of this research is to study if sharing the whole replay buffer is necessary, or if a transfer of a small amount of experiences is sufficient to continue a proper learning. We also mention that, in cases where communication is limited or constrained, sharing the whole replay buffer is impossible as it requires a large bandwidth, whereas sharing a small amount of experiences remains feasible.

3 Large Batch Sharing

Our algorithm, named Large Batch Sharing (LBS) and illustrated in Figure 1, fills a large batch at each learning step with experiences collected by all agents which communicate. Then, the absolute Temporal Difference (TD) errors (which are L1 losses: $|Q_\theta(s_i,a_i) - y_i|$) are computed on this large batch by each agent and we down-sample the large batch to a mini-batch thanks to the distribution

induced by the absolute TD errors. Finally, an SGD step is taken thanks to this mini-batch to update the neural network parameters.

The rationale behind this choice is the following. The large batch, filled by experiences from all communicating agents, defines a training set on which the neural network Q_θ has to predict the optimal value function Q^*. In the context of deep neural networks, this optimality boils down to finding the optimal parameters θ^* such that: $\theta^* \in \arg\min_\theta \frac{1}{K}\sum_{k=1}^{K} \ell(Q_\theta(s_k, a_k), y_k)$, where K, the large batch size, is the product of B the mini-batch size and M the number of agents.

Writing u the uniform probability distribution over the K items of the training set, i.e. $\forall k \in [1; K], u_k = 1/K$, the empirical gradient of the loss function defined above can be written as an expectation:

$$\frac{1}{K}\sum_{k=1}^{K} \nabla_\theta \ell(Q_\theta(s_k, a_k), y_k) = \mathbb{E}_{k\sim u}[\nabla_\theta \ell(Q_\theta(s_k, a_k), y_k)].$$

Writing p any probability distribution over the training set, importance sampling can be used:

$$\mathbb{E}_{k\sim u}[\nabla_\theta \ell(Q_\theta(s_k, a_k), y_k)] = \mathbb{E}_{k\sim p}\left[\nabla_\theta \ell(Q_\theta(s_k, a_k), y_k)\frac{u_k}{p_k}\right] = \frac{1}{N}\mathbb{E}_{k\sim p}\left[\frac{1}{p_k}\nabla_\theta \ell(Q_\theta(s_k, a_k), y_k)\right].$$

All these expectations can be approximated with mean estimators. With B the mini-batch size, it yields:

$$\mathbb{E}_{k\sim p}\left[\frac{1}{p_k}\nabla_\theta \ell(Q_\theta(s_k, a_k), y_k)\right] \approx \frac{1}{B}\sum_{k=1}^{B} \frac{1}{p_k}\nabla_\theta \ell(Q_\theta(s_k, a_k), y_k) \ ,$$

$k \sim p$. We introduce $G_k^{(t)} = w_k \nabla_\theta \ell(Q_\theta(s_k, a_k), y_k)$ with $w_k = 1/(Kp_k)$ for any sampling scheme p such that $\forall k \in [1; K], \quad p_k > 0$. This quantity will be useful to study SGD based algorithms using sampling p, possibly non uniform. Note that, when $p = u$, $w_k = 1$.

Setting η as a constant learning rate, a standard (full) gradient descent update has the form: $\theta_{t+1} = \theta_t - \eta\mathbb{E}_{k\sim p}[G_k^{(t)}]$. This can be surprising at first sight since it seems to depend on the sampling scheme p used, whereas there is no sampling for the standard (full) gradient descent. It is indeed the case:

$$\mathbb{E}_{k\sim p}[G_k^{(t)}] = \sum_{k=1}^{K} p_k G_k^{(t)} = \frac{1}{K}\sum_{k=1}^{K} \nabla_\theta \ell(Q_\theta(s_k, a_k), y_k),$$

and this notation gives consistency when writing the stochastic gradient descent update under sampling scheme p:

$$\theta_{t+1} = \theta_t - \eta\frac{1}{B}\sum_{k=1}^{B} G_k^{(t)} = \theta_t - \eta\frac{1}{B}\sum_{k=1}^{B} \frac{1}{Kp_k}\nabla_\theta \ell(Q_\theta(s_k, a_k), y_k).$$

Following the notations of Katharopoulos & Fleuret (2018), let us define the convergence speed S of SGD under a sampling scheme p as $S(p) = -\mathbb{E}_{k\sim p}\left[\|\theta_{t+1} - \theta^*\|_2^2 - \|\theta_t - \theta^*\|_2^2\right]$. We recall that a stochastic gradient descent update with $B = 1$ has the form $\theta_{t+1} = \theta_t - \eta G_k^{(t)}$, where $G_k^{(t)}$ is the gradient estimate built from sampling element k with probability p. The following derivations from Wang et al. (2017) shed light on the relationship between variance of the stochastic gradient estimate and convergence speed:

$$S(p) = -\mathbb{E}_{k\sim p}\left[\theta_{t+1}^T\theta_{t+1} - 2\theta_{t+1}^T\theta^* - \theta_t^T\theta_t + 2\theta_t^T\theta^*\right] = 2\eta(\theta_t - \theta^*)^T\mathbb{E}_{k\sim p}[G_k^{(t)}] - \eta^2\mathbb{E}_{i\sim p}[{G_k^{(t)}}^T G_k^{(t)}].$$

Indeed, the term $\mathbb{E}_{k\sim p}[{G_k^{(t)}}^T G_k^{(t)}]$ can be called *variance of the stochastic gradient estimate*, since it is linked to the covariance matrix $\mathbb{V}\text{ar}_{k\sim p}[G_k^{(t)}]$ by $\mathbb{E}_{k\sim p}[{G_k^{(t)}}^T G_k^{(t)}] = \text{Tr}(\mathbb{V}\text{ar}_{k\sim p}[G_k^{(t)}]) +$

$\mathbb{E}_{k\sim p}[G_k^{(t)}]^T\mathbb{E}_{k\sim p}[G_k^{(t)}]$. Recall also from Eq. 3 that $\mathbb{E}_{k\sim p}[G_k^{(t)}]$ is a constant with respect to p. Hence, it is possible to gain a speed-up by sampling from the distribution that minimizes $\mathbb{E}_{k\sim p}[{G_k^{(t)}}^T G_k^{(t)}]$.

Since minimizing $\mathbb{E}_{k\sim p}[{G_k^{(t)}}^T G_k^{(t)}]$ is equivalent to minimizing $\mathrm{Tr}(\mathbb{V}\mathrm{ar}_{k\sim p}[G_k^{(t)}])$, the sampling scheme optimizing the convergence speed also minimizes the variance of the stochastic gradient steps performed. The higher the convergence speed, the lower the variance of the stochastic gradient estimate.

The optimal distribution is $p_k^* \propto \|\nabla_\theta \ell(Q_\theta(s_k, a_k), y_k)\|_2$, the per-sample gradient norm. This derivation is detailed in Appendix A.

Applying the chain rule to the per-sample gradient norm indicates that $\|\nabla_\theta \ell(Q_\theta(s_k, a_k), y_k)\|_2 = \|\partial\ell(Q_\theta(s_k, a_k), y_k)/\partial Q_\theta(s_k, a_k) \cdot \partial Q_\theta(s_k, a_k)/\partial\theta\|_2$. If ℓ corresponds to the $L2$-loss, then $\partial\ell(Q_\theta(s_k, a_k), y_k)/\partial Q_\theta(s_k, a_k)$ is the TD error. The per-sample gradient norm of the loss is the product of the absolute TD error and of the norm of the network output's gradient.

Consequently, the absolute TD error is a good proxy for the optimal sampling distribution, yielding the best variance reduction and the higher convergence speed. More importantly, sampling experiences with high absolute TD errors is equivalent to sampling experiences where the neural network has the most to learn. In our deployed multi-agent system where the environment can change, experiences with high absolute TD errors can be considered the most surprising, hence helping the neural networks to adapt rapidly.

We recall the Large Batch Sharing algorithm goes as follows. For each agent $i \in [1; M]$, 1/ draw a mini-batch of size B from the replay buffer with distribution p_1. Aggregate the $M - 1$ other mini-batches to obtain the large batch. 2/ On this large batch, compute probability distribution p_2 and down-sample the large batch to a mini-batch thanks to this distribution. 3/ Finally, perform the SGD step thanks to the samples in the mini-batch. We now detail the subtleties at each step previously described.

Step 1 of the algorithm could simply consist in drawing uniformly a mini-batch from the replay buffer (hence $p_1 = u$, where $u_i = 1/N,\ \forall i \in [1; N]$), but we will test different sampling, such as sampling high absolute TD errors. Step 2 computes the priority which determines which samples will yield the better SGD step thanks to distribution p_2. Note that, if p_2 and p_1 are both uniform distributions, then our algorithm is strictly equivalent to a parameter sharing baseline where replay buffers are fully shared, but neural networks are not. We will, of course, test this baseline but our aim is to explore different sampling schemes. In particular, we take p_2 as the distribution induced by the absolute TD errors. By doing this, we ensure that the samples coming from other agents really help in taking better SGD steps.

4 Experiments

Even though many experiments remain to be run, the first results obtain on the SISL (Stanford Intelligent Systems Laboratory) environment named Pursuit appear promising (Gupta et al., 2017). In the Pursuit scenario, a mixed collaborative-competitive environment is presented, involving a team of pursuers aiming to capture a group of evaders within a grid-world containing obstacles. The evaders, depicted in blue, move randomly, while the pursuers, represented in red, are under the control of RL agents. When a group of two or more agents successfully surrounds an evader, each agent receives a reward, and the evader is eliminated from the environment. The episode concludes either when all evaders are captured or after 500 steps, whichever comes first. Pursuers earn a small reward for being adjacent to an evader (even without complete surrounding) and incur a slight negative reward per timestep, encouraging them to complete episodes promptly. The setup involves 8 pursuers and 30 evaders. We perform the training for 800k timesteps and average our results over 10 seeds.

As explained earlier, the two baselines are 1/ the independent DQN agents (IDQN), and 2/ the parameter sharing agent (PS) owning one neural network and replay buffer collecting all experiments. Our agent uses p_1 as the uniform distribution, and p_2 as the distribution induced by the TD errors.

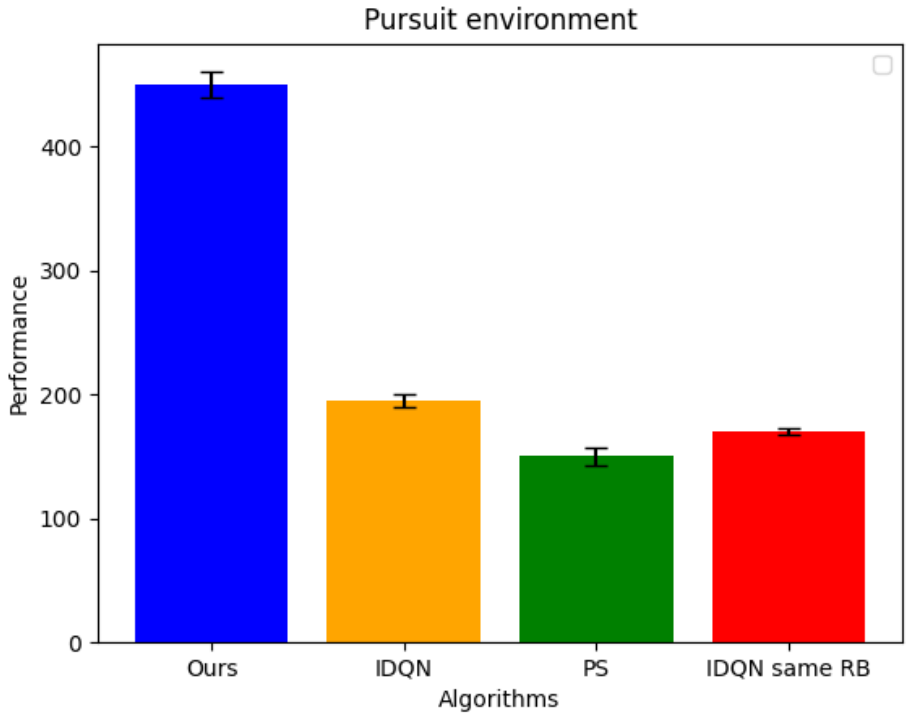

Figure 2: Performance of the four agents in Pursuit at 800k timesteps.

The fourth algorithm tested (IDQN same RB) consists in taking p_1 and p_2 as the uniform distribution. In this case, each agent has its own neural network but the replay buffer is shared by all agents. All hyperparameters for the DQN base agent are the same for all the tested configurations to ensure a fair comparison and are reported in Table 1.

Figure 2 shows the average sum of rewards per episode at the end of training for each tested algorithm. This result advocates for a high benefit of our sampling but remains to be consolidated by other experiments. In particular, a sensitivity analysis with respect to hyper-parameters specific to our method have to be tested. We plan to analyze the impact of: 1/ the amount of experiences to share between agents, and 2/ the sampling distribution used for p_1 and p_2.

Moreover, as already stated at the end of Section 2, we aim at studying multi-agent systems already deployed in a changing environment, forcing the agents to adapt to continue completing the task. The Pursuit environment, as well as the experiments we ran, do not illustrate this goal. Future works will focus on developing benchmarks of changing environments and training adaptive agents.

5 Discussions, limitations and related work

Our work restricts to homogeneous agents. An experience can be useful to another agent only if this agent can also take the action of the experience. Moreover, in the case of heterogeneous agents, where agents do not share the same action space, the baseline algorithms are different. In particular, the parameter sharing baseline is not applicable. Our work also restricts to "anonymous" environments and could not be extended to a different setting, where a certain action combined with a certain observation does not yield the same reward to all agents. Note that this kind of environment is rare in practice, and when encountered, it is often in the case of heterogeneous agents, as in (Calvo & Dusparic, 2018).

Our contribution is illustrated on collaborative systems, and it also applies to mixed and competitive settings, as long as agents remain homogeneous and the environment anonymous. We found more intuitive to highlight the benefits of sharing experiences on collaborative settings, as parameter sharing is rarely seen in practice on competitive environments. Indeed, in such settings, agents often explore the environment for their own profit, avoiding sharing information to maintain an advantage in the environment knowledge compared to the competitors.

In cooperative multi-agent systems, where all agents share the same reward at each step, our contribution can be used as well. However, note that cooperative tasks are harder to solve compared to collaborative ones, where each agent earns its own reward, due to the credit assignment problem. Even though it remains to be experimentally verified, we suspect our method to be less beneficial in this setting. Sharing experiments between agents is likely to make the credit assignment even more harder to solve, since knowing property of one experiment might help the credit assignment.

This work focuses on the DQN algorithm, as it is a common practice in the single-agent literature to study importance sampling and prioritization on this base agent. Our contribution extends directly to all off-policy algorithms using a replay buffer to learn a value function. This encompasses algorithms designed for continuous actions, such as DDPG (Lillicrap et al., 2016), TD3 (Fujimoto et al., 2018), SAC (Haarnoja et al., 2018), and also distributional RL agents, such as C51 (Bellemare et al., 2017), QR-DQN (Dabney et al., 2018b), and IQN (Dabney et al., 2018a). Sampling high absolute TD errors has also been proposed in the single-agent case with the Prioritized Experience Replay algorithm (Schaul et al., 2016, PER). Note that our work brings a theoretical justification for the sampling, whereas PER is presented as a heuristic. Sharing experiences can also be performed with on-policy algorithms such as PPO (Schulman et al., 2017). However, as prioritizing samples with policy-based algorithms remains a difficult problem, even in the single-agent setting, we keep this extension of our algorithm for future works.

This enumeration of limitations is necessary to understand the scope of our study and to place it in the literature. Perhaps the most similar work to ours is (Wang & Zhang, 2019). The authors use PER to select the shared experiences, but do not check the relevance of a particular sample for the agent which receives it, which is what we propose in our work. D'Eramo & Chalvatzaki (2022) couple multi-task RL algorithms with a task-sampling policy based on the intrinsic motivation paradigm. Nicholaus & Kang (2022) propose an exploration strategy which enables to fill the replay buffer with experiences collected from a distribution supposed to be beneficial for the learning process.

Prioritized sampling has also been studied in multi-agent RL in settings different to ours. In cooperative tasks, an extended literature has been produced in recent years studying the interplay between non-stationarity, credit assignment, and importance sampling. For instance, Mei et al. (2023) derive an importance sampling scheme allowing to correct the distribution of the replay buffers, hence enabling a better convergence. Bargiacchi et al. (2020) compare to (Mei et al., 2023) but frame their study in model-based MARL. Tian et al. (2023) study how PER can improve offline MARL thanks to a trajectory selection using Graph Attention Networks (Veličković et al., 2018).

6 Conclusion

This work focuses on an efficient experience sharing scheme between homogeneous agents learning in an anonymous environment. This scheme draws inspiration from the optimization literature, and our work remains an empirical study. This article remains a work in progress, and the first experimental results highlight the benefit of sharing a small amount of experiences between agents.

In the near future, complementary experiments have to be run. Different environments have to be tested, as well as different numbers of agents. In particular, we would like to confirm / infirm the following intuition: Our method brings a significant improvement on tasks where the independent learners baseline performs well, in environments necessitating specialization of agents. Complementary experiments will also study from an empirical point of view the amount of experiments to be shared between agents yielding the best learning improvement.

References

Eugenio Bargiacchi, Timothy Verstraeten, Diederik M Roijers, and Ann Nowé. Model-based multi-agent reinforcement learning with cooperative prioritized sweeping. *arXiv preprint arXiv:2001.07527*, 2020.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *International Conference on Machine Learning*, pp. 449–458. PMLR, 2017.

Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

Jeancarlo Arguello Calvo and Ivana Dusparic. Heterogeneous multi-agent deep reinforcement learning for traffic lights control. In *AICS*, pp. 2–13, 2018.

Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*, pp. 1096–1105. PMLR, 2018a.

Will Dabney, Mark Rowland, Marc Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018b.

Carlo D'Eramo and Georgia Chalvatzaki. Prioritized sampling with intrinsic motivation in multi-task reinforcement learning. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2022.

Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 29, 2016.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pp. 1587–1596, 2018.

Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *Autonomous Agents and Multiagent Systems: AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers 16*, pp. 66–83. Springer, 2017.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pp. 1861–1870, 2018.

Eric A Hansen, Daniel S Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of the 19th national conference on Artifical intelligence*, pp. 709–715, 2004.

Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33(6):750–797, 2019.

Angelos Katharopoulos and Francois Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *International Conference on Machine Learning*, pp. 2525–2534, 2018.

Thibault Lahire, Matthieu Geist, and Emmanuel Rachelson. Large batch experience replay. In *International Conference on Machine Learning*, pp. 11790–11813. PMLR, 2022.

Zhengping Li, Cheng Hwee Sim, and Malcolm Yoke Hean Low. A survey of emergent behavior and its impacts in agent-based systems. In *2006 4th IEEE international conference on industrial informatics*, pp. 1295–1300. IEEE, 2006.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR (Poster)*, 2016.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pp. 157–163. Elsevier, 1994.

Yongsheng Mei, Hanhan Zhou, Tian Lan, Guru Venkataramani, and Peng Wei. Mac-po: Multi-agent experience replay via collective priority optimization. *arXiv preprint arXiv:2302.10418*, 2023.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Andrew W Moore and Christopher G Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*, 13(1):103–130, 1993.

Isack Thomas Nicholaus and Dae-Ki Kang. Robust experience replay sampling for multi-agent reinforcement learning. *Pattern Recognition Letters*, 155:135–142, 2022.

Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *International Conference on Machine Learning*, pp. 2681–2690. PMLR, 2017.

Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11:387–434, 2005.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *ICLR (Poster)*, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

Qi Tian, Kun Kuang, Furui Liu, and Baoxiang Wang. Learning from good trajectories in offline multi-agent reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 11672–11680, 2023.

Wiebe Van der Hoek and Michael Wooldridge. Multi-agent systems. *Foundations of Artificial Intelligence*, 3:887–928, 2008.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

Che Wang and Keith Ross. Boosting soft actor-critic: Emphasizing recent experience without forgetting the past. *arXiv preprint arXiv:1906.04009*, 2019.

Linnan Wang, Yi Yang, Renqiang Min, and Srimat Chakradhar. Accelerating deep neural network training with inconsistent stochastic gradient descent. *Neural Networks*, 93:219–229, 2017.

Yishen Wang and Zongzhang Zhang. Experience selection in multi-agent deep reinforcement learning. In *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 864–870. IEEE, 2019.

Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control*, pp. 321–384, 2021.

A Convergence Speed and Optimal Sampling Distribution

From one sampling scheme to another, the variance of the gradient estimate is different, and we are looking for the optimal sampling scheme p^* over items of the large batch, the one with smallest variance. Indeed, for such a sampling scheme, the convergence speed of SGD is optimum. We define the convergence speed S for a sampling scheme p as $S(p) = -\mathbb{E}_{k\sim p}\left[\|\theta_{t+1} - \theta^*\|_2^2 - \|\theta_t - \theta^*\|_2^2\right]$. We recall that a stochastic gradient descent update has the form $\theta_{t+1} = \theta_t - \eta G_k^{(t)}$, where $G_k^{(t)}$ is the gradient estimate built from sampling element k with probability p. The following derivations from (Wang et al., 2017) shed light on the relationship between variance and convergence speed:

$$\begin{aligned}
S(p) &= -\mathbb{E}_{k\sim p}\left[\|\theta_{t+1} - \theta^*\|_2^2 - \|\theta_t - \theta^*\|_2^2\right] \\
&= -\mathbb{E}_{k\sim p}\left[\theta_{t+1}^T\theta_{t+1} - 2\theta_{t+1}^T\theta^* - \theta_t^T\theta_t + 2\theta_t^T\theta^*\right] \\
&= -\mathbb{E}_{k\sim p}\left[(\theta_t - \eta G_k^{(t)})^T(\theta_t - \eta G_k^{(t)}) + 2\eta {G_k^{(t)}}^T\theta^* - \theta_t^T\theta_t\right] \\
&= -\mathbb{E}_{k\sim p}\left[-2\eta(\theta_t - \theta^*)^T G_k^{(t)} + \eta^2 {G_k^{(t)}}^T G_k^{(t)}\right] \\
&= 2\eta(\theta_t - \theta^*)^T\mathbb{E}_{k\sim p}[G_k^{(t)}] - \eta^2\mathbb{E}_{k\sim p}[{G_k^{(t)}}^T G_k^{(t)}]
\end{aligned}$$

It is possible to gain a speed-up by sampling from the distribution that minimizes $\mathbb{E}_{k\sim p}[{G_k^{(t)}}^T G_k^{(t)}]$. This yields the constrained optimization problem:

$$\min_p \mathbb{E}_{k\sim p}[{G_k^{(t)}}^T G_k^{(t)}] = \min_p \sum_{k=1}^K p_k \|G_k^{(t)}\|_2^2$$
$$\text{such that } \sum_{k=1}^K p_k = 1 \text{ and } p_k \geq 0$$

Recall that $G_k = w_k \nabla_\theta \ell(Q_\theta(s_k, a_k), y_k)$ and $w_k = 1/(Kp_k)$. Let $g_k = \|\nabla_\theta \ell(Q_\theta(s_k, a_k), y_k)\|_2$. The problem boils down to:

$$\min_p \frac{1}{K^2}\sum_{k=1}^K \frac{1}{p_k} g_k^2,$$
$$\text{such that } \sum_{k=1}^K p_k = 1 \text{ and } p_k \geq 0.$$

Lemma A.1 (Optimal sampling distribution) *The optimal sampling distribution p^* verifies $p_k^* \propto \|\nabla_\theta \ell(Q_\theta(s_k, a_k), y_k)\|_2$, the per-sample gradient norm.*

Proof We note $\mu \in \mathbb{R}$ the Lagrange multiplier associated to the equality constraint, $\nu \in \mathbb{R}_+^N$ the Lagrange multipliers associated to the inequality constraints. Hence:

$$\text{Lag}(p, \mu, \nu) = \sum_{k=1}^K \frac{1}{p_k} g_k^2 + \mu\left(\sum_{k=1}^K p_k - 1\right) - \sum_{k=1}^K \nu_k p_k$$

Setting the derivatives of the Lagrangian with respect to the primal variables yields:

$$\forall\ k \in [1, K],\ -\frac{g_k^2}{p_k^2} + \mu - \nu_k = 0$$

Multiplying the above equation by p_k and using $\forall\ k,\ p_k\nu_k = 0$ (complementary slackness), we have: $p_k = g_k/\sqrt{\mu}$, which yields the result.

B Hyperparameters for the Pursuit environment

Table 1: Hyperparameters

Environment hyperparamters	
max cycles	500
shared reward	False
horizon	500
surrounded	True
tag reward	0.01
constrained window	1.0
obs range	7
x/y sizes	16/16
num evaders	30
n catch	2
n agents (pursuers)	8
urgency reward	-0.1
catch rewards	5
CNN network hyperparameters	
CNN layers	[32, 64, 64]
Stride	1
Kernel size	[2, 2]
DQN hyperparameters	
learning rate	0.00016
mini-batch size	32
buffer size	120000
initial exploration epsilon	0.1
final exploration epsilon	0.001
target network update freq	1000
factor K	8