Combining Experience Replay with Exploration by Random Network Distillation

Francesco Sovrano

***Abstract*—Our work is a simple extension of the paper "Exploration by Random Network Distillation"[1]. More in detail, we show how to efficiently combine Intrinsic Rewards with Experience Replay in order to achieve more efficient and robust exploration (with respect to PPO/RND) and consequently better results in terms of agent performances and sample efficiency. We are able to do it by using a new technique named Prioritized Oversampled Experience Replay (POER), that has been built upon the definition of *what is the important experience useful to replay*. Finally, we evaluate our technique on the famous Atari game *Montezuma's Revenge* and some other hard exploration Atari games.**

***Index Terms*—Deep Reinforcement Learning, Actor-Critic, Prioritized Experience Replay, PPO, Intrinsic Rewards, Montezuma's Revenge**

I. INTRODUCTION

A REINFORCEMENT LEARNING (RL) problem is typically formalized as a Markov Decision Process (MDP), in which an agent interacts with an environment, observing the effects of its actions (in the environment) while trying to maximize a cumulative return/reward. In other words, a RL agent learns how to optimally interact with the environment, by receiving some environmental feedbacks called rewards. The more an action is good, the higher should be the reward. But in many scenarios, rewards are very rare and difficult to get, thus making Reinforcement Learning very hard to achieve.

A good RL agent has to be able to efficiently explore the environment while optimally exploiting the information it already found. Balancing exploration with exploitation is a well-known problem in RL, very hard to solve when the environment is too big and the rewards are too sparse and difficult to find by taking random sequences of actions (eg. in the Atari game *Montezuma's Revenge*).

In literature, several techniques exist to improve the sample efficiency (the ability to exploit past information) or the exploratory skills of existing RL algorithms.

For example, many techniques used to improve sample efficiency are based on Experience Replay (ER). ER consists in storing, into a buffer, information (experience) about the agent actions in the environment and then replaying this experience during training. The key idea behind ER is to store memory of important and meaningful states in order to exploit it, but replaying such memory can introduce some bias. In other words, the agent might tend not to explore new states, because it is too focused on exploiting the old information.

The authors are at the University of Bologna, Department of Computer Science and Engineering (DISI), Mura Anteo Zamboni 7, 40127, Bologna, Italy

978-1-7281-1884-0/19/$31.00

On the other side, many techniques for improving exploration exist in the RL literature. Among them we cite the "exploration by random network distillation"[1] (PPO/RND). PPO/RND is a recent breakthrough in Actor-Critic based RL, because it has given a significant progress on several hard exploration problems, such as the famous Atari game *Montezuma's Revenge*. PPO/RND uses Proximal Policy Optimization (PPO) [2] in conjunction with Random Network Distillation (RND). RND is a recent technique for Intrinsic Motivation based on prediction errors, in which the environmental feedback (the reward) is augmented with an extra intrinsic reward that is proportional to the error of a neural network predicting features of the observations given by a fixed randomly initialized neural network [1].

Our work is a simple extension of PPO/RND. We show how to efficiently combine Intrinsic Rewards with Experience Replay in order to achieve more efficient and robust exploration than PPO/RND and consequently better results in terms of agent performances and sample efficiency. We are able to do it by using a new technique named Prioritized Oversampled Experience Replay (POER), that has been built upon the definition of *what is the important experience useful to replay*. In POER we mix oversampling [3] with experience prioritization [4], trying to achieve *the goal of an optimal balance between exploration and exploitation*. In order to do this, we:

- give a definition of *important* information
- find a way to know when information is *uncommon*
- understand how to use important uncommon information to improve the exploratory skills of the agent

More in detail, with our experiments we show how POER affects the average cumulative return of a baseline PPO/RND agent in the following hard exploration [5] Atari games: *Montezuma's Revenge, Solaris, Venture*.

Interestingly we find that our technique seems to properly balance exploration and exploitation especially in *Montezuma's Revenge*, while in some other games it does not [1].

A. Structure of the article

In section II we introduce some related works, providing at section III the necessary background information about Reinforcement Learning. In section IV we show how to combine Experience Replay and Intrinsic Rewards in PPO. While in section V we describe the results of our experiments through an ablative analysis, trying to highlight the complexity behind combining experience replay and intrinsic rewards, by showing among other things:

[1] maybe due to the insufficient amount of training time, or due to the adopted replay frequency

- How a too frequent experience replay can negatively affect the agent performances.
- How an accurate choice of what kind of experience to replay can impact on the agent performances.

II. Related work

Our work is an extension of PPO/RND[1]. PPO/RND is a recent breakthrough in Actor-Critic based Reinforcement Learning, because it has given a significant progress on several hard exploration problems, such as the famous Atari game *Montezuma's Revenge*. PPO/RND uses Proximal Policy Optimization (PPO) [2] in conjunction with Random Network Distillation (RND). RND is a recent technique for Intrinsic Motivation based on prediction errors, in which the intrinsic reward is the error of a neural network predicting features of the observations given by a fixed randomly initialized neural network [1]. Furthermore, PPO/RND introduces a simple method to flexibly combine intrinsic and extrinsic rewards.
Our work extends PPO/RND[1] with a new Prioritized Oversampled Experience Replay technique. The goal of our work is to combine the sample efficiency provided by experience replay with the exploratory properties of RND, in order to achieve more efficient and robust exploration and consequently better results in terms of agent performances and sample efficiency.
There have been several (successful) attempts to combine Actor-Critic algorithms (eg. PPO) with Experience Replay. Probably some of the most interesting are:

- "ACER"[6], a sophisticated technique based on trust region policy optimization
- "Self-imitation learning"[7], a simple prioritized experience replay technique applied to PPO

The main differences between our work and [7] are that the latter:

- does not make any use of intrinsic rewards
- does not use any oversampling technique integrated in the experience replay mechanism

Furthermore "Self-imitation learning"[7] prioritizes experience using the *Advantage* function while our technique prioritizes experience using *Intrinsic Rewards*. In section V we show a comparison of the aforementioned prioritization approaches.
Anyway, trying to combine exploration with exploitation is historically a challenging problem in modern RL. Among all the works related to this problem we cite [8]: a new and interesting approach that tries to balance between exploration and exploitation by employing optical flow estimation errors to examine the novelty of new observations and deliver permanent performance without encountering catastrophic forgetting problems.

III. Reinforcement Learning Background

This section contains a short introduction to Reinforcement Learning techniques, mostly with the aim to fix notation. The content is quite standard, and we largely borrowed it from our previous works [9], [10], [11].
A Reinforcement Learning problem is typically formalized as

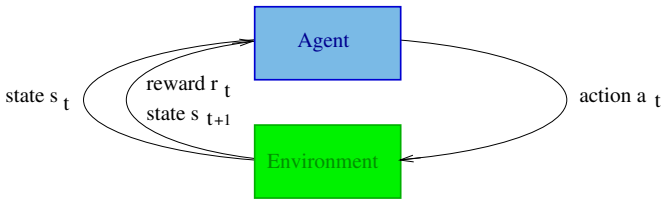

Fig. 1: Basic operations of a Markov Decision Process

a Markov Decision Process (MDP). In this setting, an agent interacts at discrete time steps with an external environment. At each time step t, the agent observes a state s_t and chooses an action a_t according to some policy π, that is a mapping (a probability distribution) from states to actions. As a result of its action, the agent obtains a reward r_t (see Fig. 1), and the environment passes to a new state $s' = s_{t+1}$. The process is then iterated until a terminal state is reached. The future cumulative reward $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is the total cumulated reward from time starting at t. $\gamma \in [0, 1]$ is the so called *discount factor*: it represents the difference in importance between present and future rewards.
The goal of the agent is to maximize the expected cumulative return starting from an initial state $s = s_t$.
The *action value* $Q^\pi(s, a) = \mathbb{E}^\pi[R_t | s = s_t, a = a_t]$ is the expected return for selecting action a in state s_t and prosecuting with strategy π.
Given a state s and an action a, the optimal *action value* function $Q^*(s, a) = \max_\pi Q^\pi(s, a)$ is the best possible action value achievable by any policy.
Similarly, the *value* of state s given a policy π is $V^\pi(s) = \mathbb{E}^\pi[R_t | s = s_t]$ and the optimal value function is $V^*(s) = \max_\pi V^\pi(s)$.
The starting points for the RL methodology are two fundamental dynamic programming algorithms: value iteration and policy iteration. In the value-based approach, we define the parameters of a value function that quantifies the maximum cumulative reward obtainable from a state belonging to the state space. While, in the policy-based approach, the policy parameters are tuned in a direction of improvement.
In figure 2 a simple overview of the RL families is shown. Value-based approaches (eg. DQN[12]) try to find a policy that

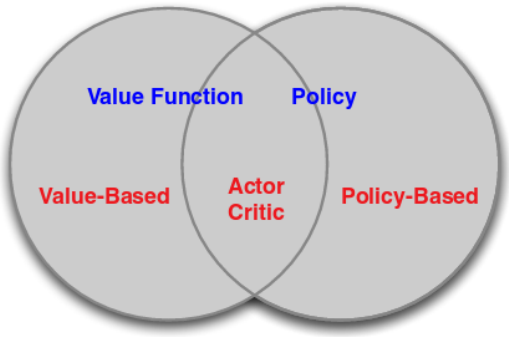

Fig. 2: **Reinforcement Learning**: Algorithms families

maximizes the cumulative return by keeping a set of expected returns estimates for some policy π. Usually π is either the "current" policy or the optimal one. An algorithm that uses the current policy is called *on-policy* algorithm, while an algorithm using the optimal one is called *off-policy*.
In policy-based RL, instead of parametrizing the value function and doing ϵ-greedy policy improvements to the value function parameters, we parametrize the policy $\pi_\theta(a|s)$ and update the parameters θ descending gradient into a direction that improves π, because sometimes the policy is easier to approximate than the value function.
Actor-Critic methods are in the middle between policy-based and N-step value-based RL. In Actor-Critic we have two function approximators: one for the policy (the Actor) and one for the value function (the Critic).

A. Advantage Actor-Critic and PPO

The idea behind Actor-Critic is that it is possible to reduce the variance of the policy while keeping it unbiased by subtracting a learned state-value function $V(s)$ known as baseline. $R - V(s)$ is called the advantage function $A(s)$. The advantage is used to measure how much better than "average" it is to take an action given a state.
Thus, the Actor updates its parameters θ_1 in the direction of:

$$\mathbb{E}[A(s) \cdot \nabla_{\theta_1} \log \pi_{\theta_1}(a|s)] \tag{1}$$

While the Critic updates its parameters θ_2 in the direction of:

$$c_1 \nabla_{\theta_2} \Omega \left(R - V^{\theta_2}(s)\right) \tag{2}$$

Where Ω usually is the L2 loss [13], and c_1 is a regularization constant used to keep the Critic learning rate lower than the Actor. Commonly $c_1 < 1$ (eg: $c_1 = \frac{1}{2}$). But it is not uncommon to have $c_1 = 1$.
The Actor-Critic family can count many different algorithms including: A3C[14], A2C[2], PPO[2].

*1) **A3C**:* The Asynchronous Advantage Actor-Critic (A3C) algorithm [14] is an on-policy technique based on Actor-Critic. A3C, instead of using an experience replay buffer like DQN[12], uses multiple agents on different threads to explore the state spaces, and it makes decorrelated updates to the Actor and the Critic. It is important to mention that A3C uses a different version of the policy gradient formula in order to better tackle the exploration problem. Thus, in A3C, the Actor objective function is changed to:

$$J^{\text{ON}}(\theta_1) = \mathbb{E}\left[A(s) \cdot \log \pi_{\theta_1}(a|s) - \beta S(\pi_{\theta_1}|s)\right] \tag{3}$$

Where $S(\pi_\theta|s)$ is the entropy of policy π_θ given state s, and β is an entropy regularization constant.

*2) **A2C**:* In A3C each agent talks to the global parameters independently, so it is possible sometimes the parallel agents would be playing with different policies and therefore the aggregated update would not be optimal. The aim of A2C is to resolve this inconsistency. A2C uses a coordinator that waits for all the parallel actors to finish before updating the global parameters, for this reason A2C is said to be the synchronous version of A3C. [15]

*3) **PPO**:* Proximal Policy Optimization (PPO) [2] is an Actor-Critic algorithm based on A2C.
The idea behind PPO is that, in order to improve training stability, we should avoid parameter updates that change the policy too much at one step. PPO is a simpler variation of Trust Region Policy Optimization (TRPO) [16] that prevents (too) big changes to the policy parameters by clipping them in a predefined range.
PPO is much simpler to implement than TRPO, more general, and according to [2] it has empirically better sample complexity.
In order to understand PPO, first of all lets denote the *probability ratio* between the old policy and the new one as:

$$r(\theta_1) = \frac{\pi_{\theta_1}(a|s)}{\pi_{\theta_1{}^{\text{old}}}(a|s)} \tag{4}$$

Then, the new Actor's objective function for PPO is:

$$\mathbb{E}\left[A(s) \cdot r(\theta_1) - \beta S(\pi_{\theta_1}|s)\right] \tag{5}$$

Let $\hat{r}(\theta_1) = \text{clip}\left(r(\theta_1), 1-\epsilon, 1+\epsilon\right)$, then the Actor objective function of PPO is:

$$J^{\text{PPO}}(\theta_1) = \mathbb{E}\left[\min\left(r(\theta_1) \cdot A(s), \hat{r}(\theta_1) \cdot A(s)\right) - \beta S(\pi_{\theta_1}|s)\right] \tag{6}$$

where ϵ is the clipping range hyper-parameter.
In PPO the Critic uses the same clipping technique used by the Actor, but instead of keeping the minimum between the clipped and the non-clipped objective, it keeps the maximum. Let $\hat{V}^{\theta_2}(s) = V^{\theta_2{}^{\text{old}}}(s) + \text{clip}\left(V^{\theta_2}(s) - V^{\theta_2{}^{\text{old}}}(s), -\epsilon, \epsilon\right)$, the objective function of the Critic is:

$$J^{\text{PVO}}(\theta_2) = c_1 \max\left(\Omega(R - V^{\theta_2}(s)), \Omega(R - \hat{V}^{\theta_2}(s))\right) \tag{7}$$

with $c_1 = 0.5$.

B. Experience Replay

Experience Replay [17] is actually a valuable and common tool for RL that has gained popularity thanks to Deep Q-learning [18].
The benefits coming from experience replay are:

- More efficient use of previous experience
- Less sample correlation, giving better convergence behaviour when training a function approximator

Importance sampling is probably one of the most used techniques to implement efficient experience replay mechanisms in Actor-Critic algorithms (eg. ACER [6]). The idea behind *importance sampling* is to weight the action gain of an old policy according to its relevance with respect to the current policy.
Furthermore, to improve learning performances it is possible to prioritize experience (instead of sampling it uniformly) in order to replay important experience more frequently [4].
It is interesting to note that the *probability ratio* adopted in the PPO loss (eq. 6) is exactly the same importance weight used in importance sampling. This probably makes PPO already suitable for efficient experience replay.

C. Intrinsic Rewards

Improving exploration may be a complex task and it can be achieved in several ways. Some of them requires changing the gradient formula in order to maximize policy entropy (as in A3C), but usually entropy regularization is not enough. Another interesting approach consists in giving *intrinsic rewards*, in order to motivate exploration/curiosity. Intrinsic rewards are rewards that do not involve receiving feedback from the environment (the "outside"), in fact they are a feedback from the agent itself (the "inside") and for this reason they completely differ from usual rewards in RL (the extrinsic rewards, from "outside").
Most formulations of intrinsic rewards for exploration can be grouped into two broad classes [5], [19]:

- Count-Based [20], [21]: the intrinsic reward is inversely proportional to the number of times a new state has been seen.
- Intrinsic Motivation [1], [22], [23], [24], [25]: the intrinsic reward is proportional to how much the new state contains new information.

Count-Based methods in practice tends to fail with huge state spaces, while Intrinsic Motivation is usually more difficult to achieve and generalize. An interesting attempt to unify Count-Based methods and Intrinsic Motivation has been shown in [5]. Among the Intrinsic Motivation techniques for exploration we cite Random Network Distillation [1] (RND): a recent technique based on prediction errors, in which the intrinsic reward is the error of a neural network (the Predictor) predicting features of the observations given by another fixed randomly initialized neural network (the Target). The Predictor is trained to minimize its predictions error.

IV. Combining Experience Replay and Intrinsic Rewards, in PPO

PPO[2] is probably one of the best state-of-the-art actor-critic algorithms, also because of its simplicity and versatility. As briefly shown in section III-A, PPO already uses importance weights in its loss and this makes PPO already suitable for efficient experience replay without any need to re-integrate importance sampling. [2]
The goal of our paper is to show how to combine "Experience Replay" and "Intrinsic Rewards", two different and conflicting techniques, within PPO. More in detail: in this section we show how to combine PPO/RND[1] with a new Experience Replay (ER) mechanism inspired by [3] and [4]. PPO/RND is a recent breakthrough in Actor-Critic RL, because it has given a significant progress on several hard exploration Atari games. PPO/RND uses PPO in conjunction with RND. PPO/RND introduces a simple method to flexibly combine intrinsic and extrinsic rewards by using two separate Critics for intrinsic and extrinsic *values*, and then mixing these *values* together in the final *advantage* by a weighted sum that gives more importance to the extrinsic advantage.

[2]this is more probably the reason behind the results of "Self-imitation learning"[7]

A. Exploration vs Exploitation

Exploration and exploitation are usually seen as two opposite sides of the same coin. It is hard to optimally balance them.
Exploration means:

- explore the state space
- gather more information

Exploitation means:

- exploit the already seen state space
- make the best decision given current information

Balancing exploration with exploitation may be a very important task to accomplish, because in many realistic scenarios the best long-term strategy may involve short-term sacrifices, and gathering enough information to make the best overall decisions is usually not a naive process.
Lets try to understand it with an example. Lets assume that:

- we are playing a very complex turn-based strategy game with a partially observable environment
- we can perform only one move per turn

For simplicity, lets say that in this game there are two possible strategies we can exclusively follow:

- We can play the move we believe is best: this is called Exploitation
- We can play an experimental move: this is called Exploration

As you may intuitively imagine, optimally choosing between these two strategies might not be trivial at all. In RL we can say the same about combining Experience Replay (for exploitation) with Intrinsic Rewards (for exploration). For example, in section V we will show how a too frequent experience replay can negatively affect the agent performances.

B. Experience Replay Prioritization and Oversampling

ER consists in storing, into a *buffer*, information (experience) about the agent actions in the environment and then replaying this experience during training. The experience is stored into mini-batches containing information about performed actions, seen states, rewards, etc.. Every mini-batch contains information about B_s consecutive steps of an episode, where B_s is called batch size. The key idea behind ER is to store memory of **important and uncommon batches** in order to exploit it. But replaying this memory can introduce some bias, worsening the exploratory skills of the agent.
In order to properly use ER within PPO we need to:

- give a definition of *important* batch
- find a way to know when a batch is *uncommon*
- understand how to use important uncommon batches to improve the exploratory skills of the agent
- understand how to efficiently implement ER

1) Definition of important batch: We say that a batch is important if it belongs to one of the following **importance classes**:

- The class of batches that contain positive extrinsic rewards.
- The class of batches that lead to positive extrinsic rewards (without containing such rewards).

- The class of batches that may probably lead to unseen states and consequently to new positive extrinsic rewards.

Usually the first and the second importance classes are underrepresented in hard exploration games/problems (eg. some famous Atari games: *Montezuma's Revenge*, *Freeway*, etc.. [5]). Thus, taking inspiration from [3], we decided to **oversample** the important batches, by building 3 different experience buffers (one for each class) and by uniformly sampling from them (during replay).

2) How to identify and use uncommon batches: Until now, we have defined what important batches are, but we still have not defined how to know when a batch is uncommon. We say that a batch is **uncommon** when it has a **high cumulative intrinsic reward**. Thus, we can detect uncommon batches through RND. In fact, when the cumulative intrinsic reward (generated by RND) of a batch is high, then we can say that the states contained in the batch are likely to be uncommon. We can exploit this information by **prioritizing** experience replay with the cumulative intrinsic rewards of the batches, in order to mainly replay important and uncommon batches. The result is a new Prioritized Oversampled Experience Replay technique meant to work in conjunction with intrinsic rewards and able to improve the efficiency of RND in terms of state-space exploration.

3) ER implementation: We know what important batches are and how to prioritize them in order to improve exploration efficiency, but we still have not properly defined how we can concretely implement an efficient Experience Replay (ER) mechanism. A (too) naive Prioritized ER algorithm may add too much complexity to the learning algorithm making experience replay impracticable. For this reason, we decided to use a simple variation of the efficient algorithm proposed in [4]. More in detail, our ER algorithm uses a circular buffer called Experience Buffer. This buffer has a fixed size B and it is fed with new batches until it is completely full. When the buffer is full we have to **drop an old batch** in order to insert a new one, thus we compute a random number p_d in $[0, 1]$ and if p_d is lower than the drop probability P_d, then the batch having the lowest priority is replaced, otherwise the batch at position $i \mod B$ is replaced and i is incremented by 1.

The replay frequency is defined by a Poisson distribution with mean μ (the replay ratio constant). In other words, if $\mu = \frac{1}{2}$ then we replay 1 old batch every 2 new batches. This means that at the end of every new batch, k batches are randomly sampled from the experience buffer, where k follows the aforementioned Poisson distribution. During the sampling operation, we take a random number z lower than the sum of all the priorities in the buffer, then the batch with the highest prefix sum lower than or equal to z is sampled.

Every time a batch is replayed, its priority is updated according to the new intrinsic rewards given by RND.

Please, remember that intrinsic rewards are a measure of the novelty of a state, the more the agent explores the more some states may lose novelty. In other words, for the same state s the intrinsic reward at time t may significantly differ from the intrinsic reward at time $t + c$ with $c > 0$.

Furthermore, similarly to [26], all the aforementioned experience buffers are shared among all the workers of the A3C network.

C. Intrinsic Reward Replay

Intrinsic Rewards are given by the RND. The RND network is completely separated by the Actor and the Critic. The RND is continuously trained in order to properly evaluate the states in terms of their novelty: the less a state is seen by the agent, the more it is novel. This implies that high intrinsic rewards are associated to very uncommon states. For these reasons the Experience Replay with Intrinsic Rewards requires some precautions:

1) the RND can not be trained during the replay phase
2) the intrinsic cumulative return of the replayed batches must be updated in order to properly compute the advantage and the critic loss
3) the value used to get the advantage has always to be up to date

Training the RND during replay would cause wrong intrinsic rewards, because the RND would start giving lower intrinsic rewards to the uncommon states, mostly because with ER we usually replay the uncommon states.

V. Experiments

We conducted some experiments in order to prove the efficacy of PPO/RND extended with our Prioritized Oversampled Experience Replay technique.

Mainly because our work is an extension of [1] [3], we decided to focus our experiments primarily on the Atari game *Montezuma's Revenge* and on a few other hard exploration games: *Solaris* and *Venture*.

For PPO/RND[1] we used the implementation publicly available at [27], while for the prioritized experience replay mechanism based on [4] we used the code publicly available at [28] and then we quickly adapted it to our purposes. The changes we made to the default PPO/RND implementation are:

- We disabled the default intrinsic rewards scaling. In [1] the intrinsic rewards are scaled by their running standard deviation.
- In the RND loss, we changed the dropout rate to 0.5.
- We used A3C instead of A2C: we removed any synchronization barrier between the workers, thus updating the gradient using the Hogwild[29] approach.
- We used Proximal Value Optimization as in the default PPO[2] implementation.
- We optimized the vanilla A3C implementation in order to train more efficiently with GPUs. We did it through a technique called "delayed training" that is heavily inspired by [30]. This technique stores batches (even the replayed ones) into a buffer until a big-enough *super-batch* is ready for training.
- We resized the games screen to a 42×42 (instead of 84×84 as in [1]) grey-scaled image.
- We set the maximum number of allowed steps per episode to 3000.

[3] and also because we do not have the hardware of OpenAI for testing on many more environments in less than a life-time

- We disabled sticky actions.

We think it is important to mention that, among the other things, we kept unchanged the following aspects of the default PPO/RND implementation:

- The CNN-based[4] neural network.
- The extrinsic rewards clipping, in $[-1, 1]$.
- The state representation, made of 4 consecutive grey-scaled screens.

The source of the code we used for our experiments is publicly available at [31]. The aforementioned code is a new release of the code we published with [9], and it is meant to be easily extendible, readable and reusable.

A. Statistics for evaluation metrics

We have collected several statistics during training:

- "extrinsic_reward": average cumulative extrinsic reward per episode
- "extrinsic_reward_per_step": average cumulative extrinsic reward per step in an episode
- "extrinsic_value_per_step": average Critic extrinsic value per step in an episode

For every experiment we show how the *mean* and *standard deviation* of these statistics change during training time. Time is measured in *parameters updates*, but in the last experiment (Section V-C) time is measured in *steps* in order to focus more on sample efficiency.

B. Ablative Analysis on Montezuma's Revenge

With our experiments we try to answer the following questions:

1) Is ER useful when combined with PPO/RND?
2) Is experience prioritization useful in this context?
3) Is prioritized drop useful in this context?

We are going to show, through our ablative analysis, that extending PPO/RND with Prioritized Oversampled Experience Replay (POER) can improve sample efficiency without reducing exploration.
In order to perform an ablative analysis we need to define a *default* set of hyper-parameters, this way we are able to understand the impact of changing these parameters.
In all the experiments, our *default* set of hyper-parameters is the same described in [1], but due to the fact that we extended [1] with POER, we have also the following extra hyper-parameters (see section IV-B for more details):

- The replay frequency $\mu = 0.5$.
- The prioritized drop probability $P_d = 1$.
- The experience buffer size $B = 2^7$.

Furthermore, the super-batch size we adopted is 2^6. This means that every super-batch is made of 2^6 batches.
In all the experiments, we used PPO trained on $t = 128$ different Actor-Critics (ACs).

[4]with no RNNs

1) Is POER useful when combined with PPO/RND?: With this experiment we try to understand the effects of PPO/RND combined with POER. We do it by changing the replay frequency μ. We compare the *default* experiment (having $\mu = 0.5$) with 3 different experiments having respectively $\mu = 0$, $\mu = 1$ and $\mu = 2$. The experiment having $\mu = 0$ is said to be the *baseline*, because it is equivalent to PPO/RND without any ER mechanism.
In figure 3 we show the training statistics. As you can see, during training the *default* implementation of PPO/RND/POER (blue line) produces better results (in terms of mean extrinsic reward) than the *baseline* (orange line). As expected, our implementation of PPO/RND combined with POER seems to be more sample efficient than the baseline, this is more evident when plotting the statistics against the *steps* instead of the *parameters updates* as shown in figure 6 .
It is interesting to notice that the replay frequency μ is a very important parameter to tune. In fact we can see that when μ is too big (eg. $\mu = 2$), the exploratory skills of the agent tends to be worse. Intuitively this means that if we replay too much, then we also exploit too much the past experience without producing enough new information and thus losing in terms of exploration.

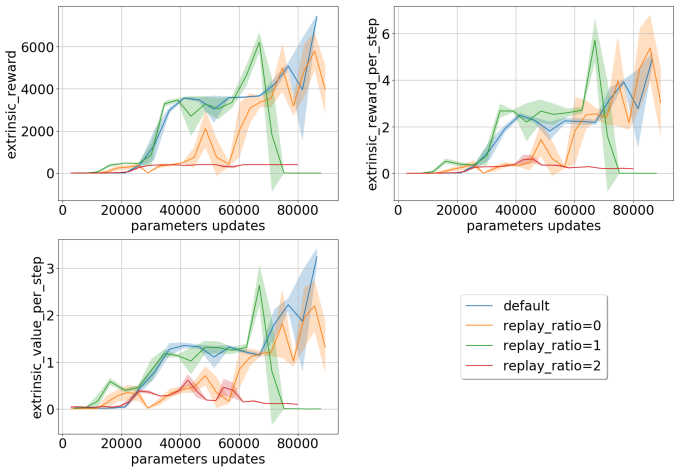

Fig. 3: **Experiment 1 - Replay Frequency**

2) Is experience prioritization useful in this context?: With this experiment we try to understand the effects of experience prioritization in PPO/RND combined with POER. We do it by disabling prioritization and by changing the priority function. We compare the *default* (that uses *ER prioritized with intrinsic rewards*) with three different experiments having respectively *no prioritization*, *ER prioritized with extrinsic rewards* and *ER prioritized with advantages*.
In figure 4 we show the training statistics. As you can see, during training the *default* (blue line) produces the best results, while the other experiments perform significantly worse.
We believe that this fact supports the theory that we should mainly replay only *uncommon batches* (identified by high intrinsic rewards).

3) Is prioritized drop useful in this context?: With this experiment we try to understand the effects of changing the prioritized drop probability P_d in PPO/RND combined with

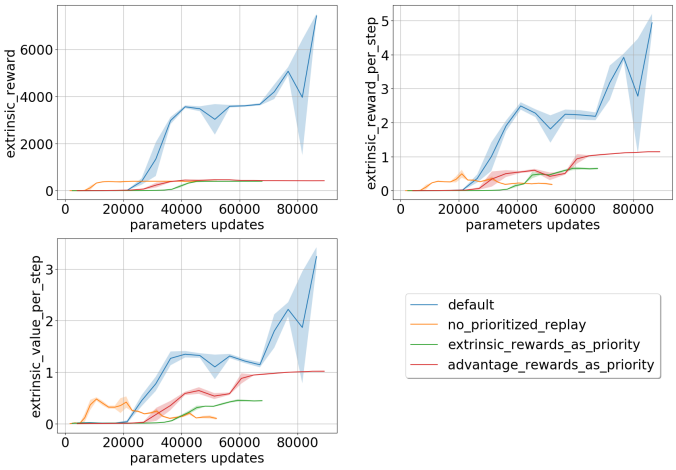

Fig. 4: **Experiment 2 - Replay Prioritization**

POER. We compare the *default* (that has $P_d = 1$) with two different experiments having respectively $P_d = 0.5$ and $P_d = 0$.
In figure 5 we show the training statistics. As you can see, during training the *default* (blue line) produces the best results, while $P_d = 0$ (green line) produces the worst results only initially. Thus, the prioritized drop seems an extremely important feature in POER. In other words, the strategy of replaying only the most uncommon important batches seems to be much more effective than randomly replaying both common and uncommon important batches.

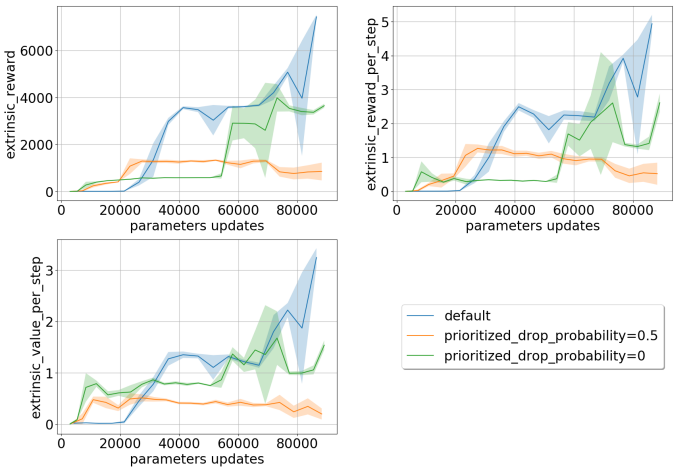

Fig. 5: **Experiment 3 - Prioritized Drop Probability**

C. Ablative Analysis on other hard exploration games

In the previous ablative analysis we have shown that our combination of PPO/RND with POER, as expected, performs better in terms of sample efficiency than the baseline PPO/RND on the Atari game *Montezuma's Revenge*.
Technique-combination papers are somewhat rare in RL research because their results tend to have low significance: *somehow we already expect independently good ideas for different aspects of an algorithm to combine somewhat well.* Despite the aforementioned expectation, we will show with the following ablative analysis that our technique-combination does not always lead to performance improvements on hard exploration games, supporting the counter-intuitive fact that combining different ideas on different aspects of an algorithm does not always lead to better results. This is true especially when trying to combine techniques for exploration with techniques for exploitation.
With this experiments we compare the performance of three different algorithms:

- PPO/RND/POER (using the *default* set of hyperparameters defined in section V-B).
- PPO/RND (the *default* but with $\mu = 0$).
- PPO/POER (the *default* but with no intrinsic rewards and thus with ER prioritized by extrinsic rewards)

in 3 different hard exploration [5] Atari games:

- *Montezuma's Revenge*: characterized by very sparse rewards, also difficult to get by random actions because the agent can die very easily.
- *Solaris*: compared to *Montezuma* it is characterized by more frequent rewards that are also easier to get by random actions.
- *Venture*: characterized by sparse rewards but apparently easier to get by random actions than *Montezuma*.

In order to focus more on sample efficiency, in figure 6 we show the training statistics plotted against the *steps* instead of the *parameters updates*. As we can see: in *Montezuma's Revenge* the best algorithm is PPO/RND/POER (blue line), in *Solaris* the best one is PPO/POER (orange line), while in *Venture* only those algorithms using RND perform decently enough and, as expected, PPO/RND/POER is the most sample efficient. In all the aforementioned games we can see that the statistics of the algorithms using POER seem to have a lower *standard deviation* and a more stable *mean* than PPO/RND, and we think this is an indicator of the better sample efficiency of PPO/RND/POER with respect to PPO/RND.
We believe that tuning further the replay frequency μ might help to improve the performance of PPO/RND/POER in both *Solaris* and *Venture*.

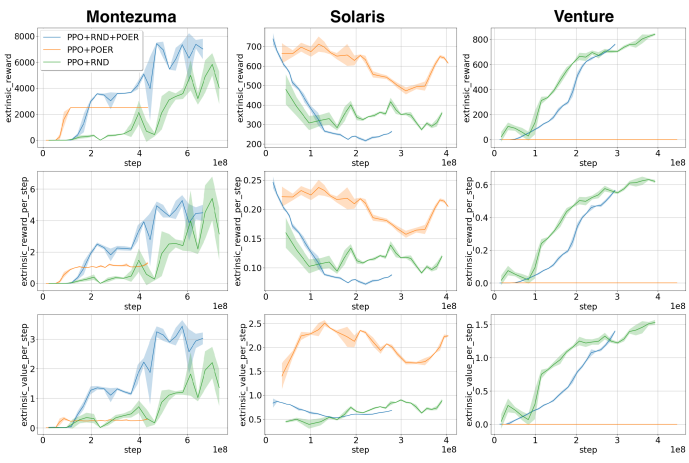

Fig. 6: **Experiment 4 - Different Hard Exploration Games**

VI. CONCLUSIONS

In this paper we have shown a simple way to combine Experience Replay with Intrinsic Rewards, or in other words a simple way to balance exploration with exploitation.
In order to do that, we:

- give a definition of *important* batch
- find a way to know when a batch is *uncommon*
- understand how to use important uncommon batches to improve the exploratory skills of the agent

Furthermore, through our experiments we try to answer the following questions:

1) Is ER useful when combined with PPO/RND?
2) Is experience prioritization useful in this context?
3) Is prioritized drop useful in this context?

The answers we got are:

1) ER seems very useful, but a too high replay frequency might unbalance the agent toward the exploitation of old information, losing in terms of exploratory skills, especially in environments characterized by more frequent extrinsic rewards than *Montezuma*.
2) Experience prioritization based on intrinsic rewards is definitively important in order to replay the uncommon batches.
3) A fully prioritized drop seems to give the best performances. The strategy of replaying only the most uncommon important batches seems to be much more effective than randomly replaying both common and uncommon important batches.

REFERENCES

[1] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," *arXiv preprint arXiv:1810.12894*, 2018.

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[3] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *arXiv preprint arXiv:1611.05397*, 2016.

[4] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[5] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, "Unifying count-based exploration and intrinsic motivation," in *Advances in Neural Information Processing Systems*, 2016, pp. 1471–1479.

[6] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas, "Sample efficient actor-critic with experience replay," *arXiv preprint arXiv:1611.01224*, 2016.

[7] J. Oh, Y. Guo, S. Singh, and H. Lee, "Self-imitation learning," *arXiv preprint arXiv:1806.05635*, 2018.

[8] H.-K. Yang, P.-H. Chiang, K.-W. Ho, M.-F. Hong, and C.-Y. Lee, "Never forget: Balancing exploration and exploitation via learning optical flow," *arXiv preprint arXiv:1901.08486*, 2019.

[9] F. Sovrano, "Deep reinforcement learning and sub-problem decomposition using hierarchical architectures in partially observable environments," Master's thesis, Università di Bologna, 2018. [Online]. Available: http://amslaurea.unibo.it/16718/

[10] A. Asperti, D. Cortesi, C. De Pieri, G. Pedrini, and F. Sovrano, "Crawling in rogue's dungeons with deep reinforcement techniques," *IEEE Transactions on Games*, 2019.

[11] A. Asperti, D. Cortesi, and F. Sovrano, "Crawling in rogue's dungeons with (partitioned) a3c," *arXiv preprint arXiv:1804.08685*, 2018.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[13] Tensorflow, "Tensorflow's l2 loss," https://www.tensorflow.org/api_docs/python/tf/nn/l2_loss.

[14] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[15] A. Stooke and P. Abbeel, "Accelerated methods for deep reinforcement learning," *arXiv preprint arXiv:1803.02811*, 2018.

[16] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning*, 2015, pp. 1889–1897.

[17] L.-J. Lin and T. M. Mitchell, *Memory approaches to reinforcement learning in non-Markovian domains*. Citeseer, 1992.

[18] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[19] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *International Conference on Machine Learning (ICML)*, vol. 2017, 2017.

[20] G. Ostrovski, M. G. Bellemare, A. van den Oord, and R. Munos, "Count-based exploration with neural density models," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2721–2730.

[21] H. Tang, R. Houthooft, D. Foote, A. Stooke, O. X. Chen, Y. Duan, J. Schulman, F. DeTurck, and P. Abbeel, "# exploration: A study of count-based exploration for deep reinforcement learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 2753–2762.

[22] N. Chentanez, A. G. Barto, and S. P. Singh, "Intrinsically motivated reinforcement learning," in *Advances in neural information processing systems*, 2005, pp. 1281–1288.

[23] J. Schmidhuber, "A possibility for implementing curiosity and boredom in model-building neural controllers," in *Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, 1991, pp. 222–227.

[24] S. Still and D. Precup, "An information-theoretic approach to curiosity-driven reinforcement learning," *Theory in Biosciences*, vol. 131, no. 3, pp. 139–148, 2012.

[25] Ö. Şimşek and A. G. Barto, "An intrinsic reward mechanism for efficient exploration," in *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006, pp. 833–840.

[26] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver, "Distributed prioritized experience replay," *arXiv preprint arXiv:1803.00933*, 2018.

[27] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by random network distillation," https://github.com/openai/random-network-distillation, 2018.

[28] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," https://github.com/openai/baselines, 2017.

[29] B. Recht, C. Re, S. Wright, and F. Niu, "Hogwild: A lock-free approach to parallelizing stochastic gradient descent," in *Advances in neural information processing systems*, 2011, pp. 693–701.

[30] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," *arXiv preprint arXiv:1802.01561*, 2018.

[31] F. Sovrano, "Combining 'experience replay' with 'exploration by random network distillation'," https://github.com/Francesco-Sovrano/Combining--experience-replay--with--exploration-by-random-network-distillation-, 2019.