MS&E 338 Final Project
Experience Replay in Reinforcement Learning: how to recycle (rare) data efficiently

Jamie Kang
jamiekang@stanford.edu

June 2018

1 Introduction

An online reinforcement learning agent observes a stream of transitions and learns from each experience $e_t = (s_t, a_t, r_t, s_t^{'})$ incrementally. Once used for update each experience is discarded which can be wasteful given that certain experiences may happen less frequently or posses relatively more information.

As a way to resolve this, experience replay (ER) has recently received much attention from the reinforcement learning community. In terms of application, it has been widely used to boost many reinforcement learning algorithms, including the Deep Q-Network (DQN) as done in [1]. There are mainly two motivations in this method: first, it allows an agent to recycle transitions in an efficient manner and second, it can reduce temporal correlations among updates. The latter has been of more interest given the recent advent of value function approximation algorithms. This is helpful when training approximation models namely by using machine learning schemes such as deep neural network. Despite its popularity and applications, however, ER has rarely been studied in terms of its data recycle aspect.

As of now, ER is mostly being used in its simplest form: an agent stores every recent transition into a replay buffer of fixed size and draws a sample from it uniformly to make an update. This approach enjoys a simple implementation and also a notion that it will "explore" more fairly since samples are uniformly distributed. However, as [3] and this project demonstrate, this is naive. [3] suggests Prioritized Experience Replay (PER), which quantifies the "unexpectedness" of each experience and prioritizes accordingly. Although this certainly mitigates some of the issues of ER, it still does not fully capture the presence of rare transitions. In this project, therefore, we focus on the data recycle aspect of replay algorithms by investigating how efficiently ER and PER reuse data, compare their effects through simulations, and develop new simple methods that can better take advantage of unusual and/or rare data. One of the new methods, Rare Prioritized Experience Replay (RPER) can be regarded as a simple add-on to the original PER algorithm. RPER demonstrates the best performance among various replay algorithms (including PER) in our simulation results.

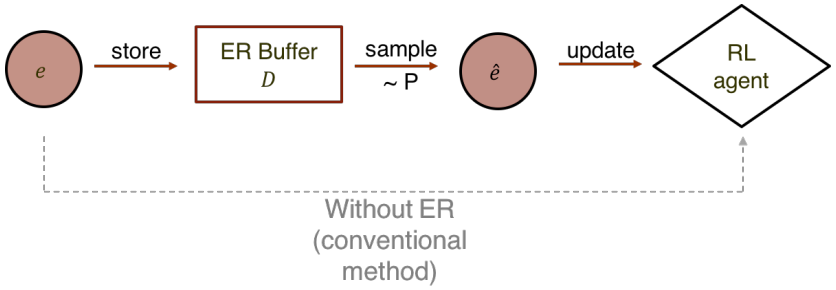

Figure 1: Mechanism of Experience Replay

2 Methods

This is done by studying the algorithms within tabular RL setting so that we can put less focus on training correlation reduction. In particular, we use Q-learning with ϵ-greedy exploration. However note that given its simplicity, RPER can be applied to different RL algorithms, such as PSRL or UCB. We expect a similar performance – if not better – in this setting, but will leave it as a possible direction for future work.

In the Q-update, we initially start with a large step size γ_0 and gradually reduce it over time for faster convergence. This is done by setting:

$$\gamma_l = \frac{M}{M + l + 1}$$

where l: number of episode, γ_l: discount factor at episode l and M is some large positive number.

For testing environment we use the Deep Sea problem[1]. We modified the problem to a simpler version where there is only a treasure chest with probability 1 instead of a random choice between treasure chest and bomb with some deterministic probability. There are two actions: 1 and 2. One action (unknown, but deterministic) will move the diver to the right and the other action will move her to the left. This is different across all cells, and thus the diver must learn which action takes her to the right in each cell. In order for the diver to reach the treasure chest, she always has to choose the actions that take her to the right. To enforce rigorous exploration and efficient use of data, each diagonal cell – that she must visit to get to the treasure chest – incurs a small cost of $\frac{-0.01}{N-2}$. The treasure is worth 1 and therefore the optimal path leads to a reward of 0.99. We use this to compute the average regret of each episode:

$$Regret_l = 0.99 - \sum_{i=1}^{I} Reward_{l,i}$$

where i: iteration count and I: total number of iteration.

For a more detailed explanation, see [2].

[1] We thank Maria Dimakopoulou for her python implementation of the environment.

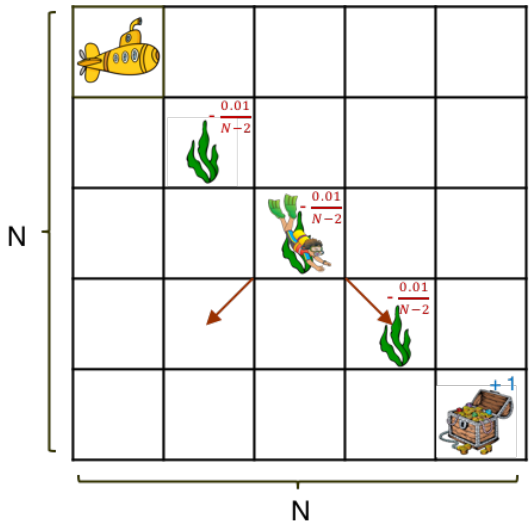

Figure 2: (Simplified) Deep Sea problem

3 Algorithms

Algorithm 1: Experience Replay

Input: Buffer capacity B, Total episode count L

Initialize an empty buffer D and value function Q
for *Episode* $l = 1, 2, ..., N$ **do**
  Sample initial state S
  while *not terminate* **do**
    Pick an action a according to ϵ-greedy using Q
    Observe reward r and next state $s^{'}$
    Store the experience $e = (s, a, r, s^{'})$ into D
    Sample an experience $\hat{e} = (\hat{s}, \hat{a}, \hat{r}, \hat{s}^{'})$ from D according to P
    Update Q using $\hat{e}$
    Update P Move to the next state $s^{'}$
  end
end

In the standard ER, sampling distribution P is uniform. This can be still data efficient since it recycles transitions rather than discarding them right after their one-time usage. Such a mechanism is especially useful in a setting where transitions are costly. However, its one critical issue is that all experiences are treated the same. Hence it fails to take advantage of less frequent transitions that may include more information that an agent can learn from.

One possible remedy suggested in [3] is PER, where experiences are priori-

tized according to their temporal difference (TD) errors, δ. In particular, since Q-learning already computes TD errors for update, this comes at a low computational cost. Intuition behind PER is that agent can learn more from unusual (which will be reflected in their large TD errors) transitions and thus should be recycled more frequently. Sampling distribution P can be computed by:

$$\begin{aligned} \delta &= r + \max_{a' \in A} Q(s', a') - Q(s, a) \\ p &= |\delta| + E \\ P(i) &= \frac{p_i^\alpha}{\sum_k^{|D|} p_k^\alpha} \end{aligned} \tag{1}$$

where E is some small positive constant so that transitions with zero TD error can still be sampled. Once recycled, the TD error is updated and so are the priorities.

One new replay algorithm – also mentioned in [3] – is Asymmetric PER (APER). Referring to the Anna Karenina principle, [3] suggests that positive TD errors may be more informative than negative errors. We apply this idea by adding a small incentive to transitions with positive errors:

$$p = |\delta|(1 + \beta \cdot I\{\delta > 0\}) \tag{2}$$

Our initial hypothesis is that this may not be helpful. Most importantly, in order to fully take advantage of this strategy the Q values must be initialized reasonably well, which is challenging without any prior knowledge of the problem. Our simulation results confirm this.

We develop another method that can recycle rare transitions more efficiently. We refer to this as Rare PER (RPER) and it prioritizes experiences by:

$$\tilde{p} = \frac{p}{F(s, a, r, s')} \tag{3}$$

where p is from original PER algorithm and $F(s, a, r, s')$ is the number of times the transition tuple has been observed. This can be regarded as a simple add-on to the original PER algorithm as it only needs to additionally keep track of number of observation using a basic data structure. If the transition has been observed only once, (i.e. very rare experience), then its priority remains the same, whereas transitions that have been observed more frequently (i.e. common experience) will have their priorities discounted by a large amount. Note that

$$\tilde{p} = \frac{p}{F(s, a, r, s')} \to 0 \text{ as } F(s, a, r, s') \to \infty,$$

which is quite intuitive. There will be almost nothing left to learn from a transition that has happened sufficiently many times, and thus there is no need to sample and learn from such experiences.

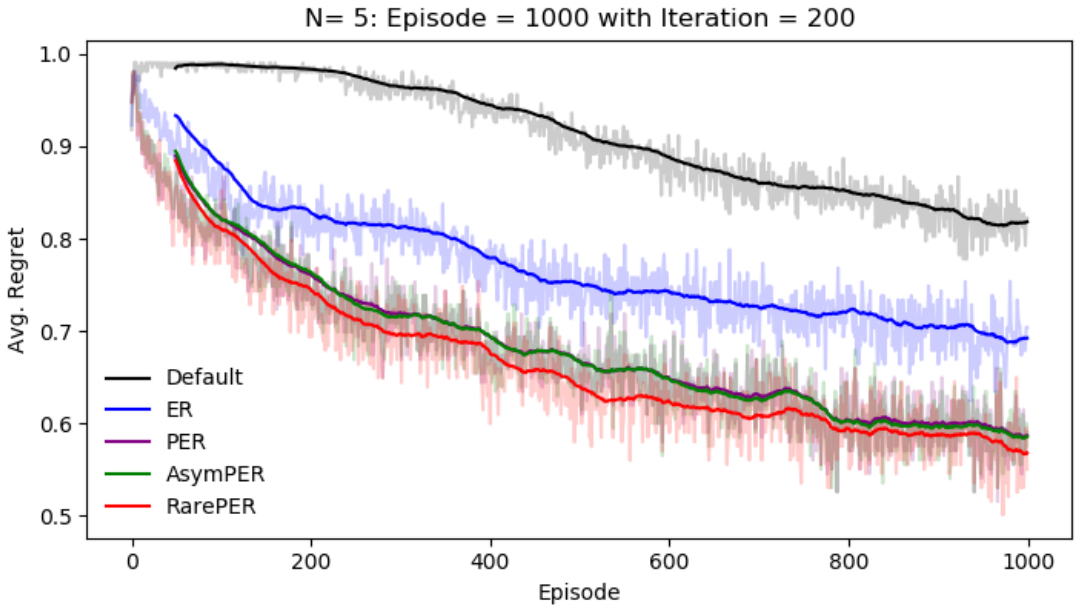

Figure 3: Regret per episode averaged over 200 iterations for various replay algorithms

4 Results

We examine the aforementioned replay algorithms on Deep Sea problem. For simplicity, we demonstrate results from using $N = 5$ world. However we have observed similar patterns for larger Ns. In this particular setting, there are $N^2 = 25$ states each with 2 actions. For this problem we use a replay buffer of size $B = 200$. As soon as the buffer is full and a new experience arrives, it removes the oldest to keep the buffer size fixed. For ϵ-greedy exploration, we use $\epsilon = 0.2$. Convergence speed and regrets may be optimized by a better hyperparameter tuning, but will not be discussed in this project. Larger replay buffer may be required for more complex problems. We evaluate the algorithms based on their regret per episode averaged over 200 iterations and is shown in Figure 3. Note that bold lines are obtained by taking moving average over 10 episodes and the lighter lines in the background are original values.

Clearly, any replay algorithm outperforms the default Q-learning. Just by using the uniform ER, final regret can be reduced by 15% after 1000 episodes. PER (in purple) further reduces this by another 15%. As our hypothesis, APER (in green) does not bring much improvement from PER. However, RPER (in red) further outperforms PER and reduces the final regret to below 57%. Given this can be achieved by a very low additional cost, such an improvement cannot be neglected. In short, in terms of regret performance, ER $<$ PER $\approx$ APER $<$ RPER.

To provide some insights into these results, we suggest a couple of ideas. First of all, the unsatisfactory performance of APER may be attributable to the structure of the problem as it only has negative rewards until it reaches the state with treasure chest. Moreover, due to the small size of the environment the number of rare transitions is also limited in this setting. Using a more sparse MDP structure may result in more rare transitions and thereby may benefit more from RPER.

5 Conclusion

In this project we have shown that experience replay and its variants are highly efficient in the tabular RL context as well as they are in value function approximation context (e.g. DQN). We also develop and examine two new algorithms: APER and RPER. Both can be implemented as add-ons to the original PER at a very small cost. In particular, our simulation result demonstrated that RPER recycles data most efficiently as RPER recycles experiences in a way that an agent can learn enough from rare transitions that may otherwise be less accessible. As suggested earlier, we conjecture that RPER may be even more beneficial on more sparse MDP structured problems, and we leave this as a possible direction for future research. Another future direction is to test RPER for value function approximation RL algorithms, such as DQN or Double-DQN for more complex problems. It has been shown in [3] that PER can significantly improve DQN for many game problems. Therefore, we expect even better improvements

by using RPER rather than PER. Combining RPER with a deep exploration algorithm, such as PSRL or UCRL, instead of ϵ-greedy Q-learning, might be another interesting study. We are also curious about better data structures to implement the replay algorithms. However, before all these, we are most curious about the mathematics behind RPER in terms of regret bound, convergence rate, algorithm complexity, and most importantly, why it works. We would be happy to discuss more about the aforementioned topics.

References

[1] 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, 2015.

[2] Ian Osband, Daniel Russo, Zheng Wen, and Benjamin Van Roy. Deep exploration via randomized value functions. *arXiv preprint arXiv:1703.07608*, 2017.

[3] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.