Prioritized Sequence Experience Replay

Marc Brittain *
Iowa State University
mwb@iastate.edu

Josh Bertram *
Iowa State University
bertram1@iastate.edu

Xuxi Yang *
Iowa State University
xuxiyang@iastate.edu

Peng Wei
Iowa State University
pwei@iastate.edu

Abstract

Experience replay is widely used in deep reinforcement learning algorithms and allows agents to remember and learn from experiences from the past. In an effort to learn more efficiently, researchers proposed prioritized experience replay (PER) which samples important transitions more frequently. In this paper, we propose Prioritized Sequence Experience Replay (PSER) a framework for prioritizing sequences of experience in an attempt to both learn more efficiently and to obtain better performance. We compare performance of uniform, PER and PSER sampling techniques in DQN on the Atari 2600 benchmark and show DQN with PSER substantially outperforms PER and uniform sampling.

1 Introduction

Reinforcement learning is a powerful technique to solve sequential decision making problems. Advances in deep learning applied to reinforcement learning resulted in the DQN algorithm [16] which uses a neural network to represent the state-action value. With experience replay and a target network, DQN achieved state-of-the-art performance in the Atari 2600 benchmark and other domains at the time.

While the performance of deep reinforcement learning algorithms can be above human-level in certain applications, the amount of effort required to train these models is staggering both in terms of data samples required and wall-clock time needed to perform the training. This is because reinforcement learning algorithms learn control tasks via trial and error, much like a child learning to ride a bicycle [19]. In gaming environments, experience is reasonably inexpensive to acquire, but trials of real world control tasks often involve time and resources we wish not to waste. Alternatively, the number of trials might be limited due to wear and tear of the system, making data-efficiency critical [7]. In these cases where simulations are not available or where acquiring samples requires significant effort or expense, it becomes more necessary to utilize the acquired data more efficiently – learn more with less effort.

As an important component in deep reinforcement learning algorithms, experience replay has been shown to both provide uncorrelated data to train a neural network and to significantly improve the data efficiency [14, 21, 23]. In general, experience replay can reduce the amount of experience required to learn at the expense of more computation and more memory [18].

There are various sampling strategies to sample transitions from the experience replay memory. The original primary purpose of experience replay memory was to decorrelate the input passed into the neural net, and therefore the original sampling strategy was uniform sampling. Prioritized experience

*Equal contribution

Preprint. Under review.

replay (PER) [18] demonstrated that the agent can learn more effectively from some transitions than from others. By sampling important transitions within the replay memory more often at each training step, PER makes experience replay more efficient and effective than uniform sampling.

In this paper we propose an extension to PER that we term Prioritized Sequence Experience Replay (PSER) that not only assigns high priority to important transitions, but also increases the priorities of previous transitions leading to the important transitions. To evaluate our results, we use the standard DQN algorithm [16] as the benchmark to provide a more apples-to-apples comparison of sampling techniques on the final performance of the algorithm. Experimental results on the Atari 2600 benchmark show using PSER substantially improves the performance of DQN, outperforming PER and uniform sampling.

2 Related Work

2.1 DQN and its extensions

With the DQN algorithm in [16], deep learning and reinforcement learning were successfully combined by using a deep neural network to approximate the state-action values, where the input of the neural network is the current state s in the form of pixels and the output is the state-action values corresponding to different actions (i.e. Q-values). It is known that neural networks may be unstable and diverge when applying non-linear approximators in reinforcement learning (RL) algorithms [19]. DQN uses experience replay and target networks to address the instability issues. At each time step, based on the current state, the agent selects an action based on some policy (i.e. ϵ-greedily) with respect to the action values, and adds a transition (s_t, a_t, r_t, s_{t+1}) to a replay memory buffer. The neural network is then optimized using stochastic gradient descent to minimize the squared TD error of the transitions sampled from replay memory buffer. The gradient of the loss is back-propagated only into the parameters of the online network and a target network is updated from the online network periodically.

Many extensions to DQN have been proposed to improve its performance. Double Q-learning [20] was proposed to address the overestimation due to the action selection using the online network. Prioritized experience replay (PER) [18] was proposed to replay important experience transitions more frequently, enabling the agent to learn more efficiently. Dueling networks [22] is a neural network architecture which can learn state and advantage value, which is shown to stabilize learning. Using multi-step targets [19] instead of a single reward is also shown to lead to faster learning. Distributional RL [2] was proposed to learn the distribution of the returns instead of the expected return to more effectively capture the information contained in the value function. Noisy DQN [6] proposed another exploration technique by adding parametric noise to the network weights.

Rainbow [8] combined the above mentioned 6 variants together into one agent, achieving better data efficiency and performance on the Atari 2600 benchmark, leading to a new state-of-the-art at the time. Through the ablation procedure described in the paper, the contribution of each component was isolated. Distributed prioritized replay [9] utilized a massively parallel approach to show that with enough scaling a new state-of-the-art score can be achieved, but at a cost of orders of magnitude more data. (Typical amounts of frames used for Atari 2600 benchmark games are 200 million frames. The distributed prioritized replay paper has a faster wall-clock time execution, but orders of magnitude more frames were required.) A comprehensive survey of deep RL algorithms including other extensions of DQN can be found at [12].

2.2 Experience replay

Since the use of deep neural networks as function approximators in reinforcement learning [15], experience replay has played an important role in providing uncorrelated data for the online neural network training of deep RL algorithms [16, 13]. There are also studies into how experience replay can influence the performance of deep RL algorithms [5, 23].

In the experience replay of DQN, the observation sequences are stored in the replay memory and sampled uniformly for the training of the neural network in order to remove the correlations in the data. However, this uniform sampling strategy ignores the importance of each transition and is shown to be inefficient for learning [18].

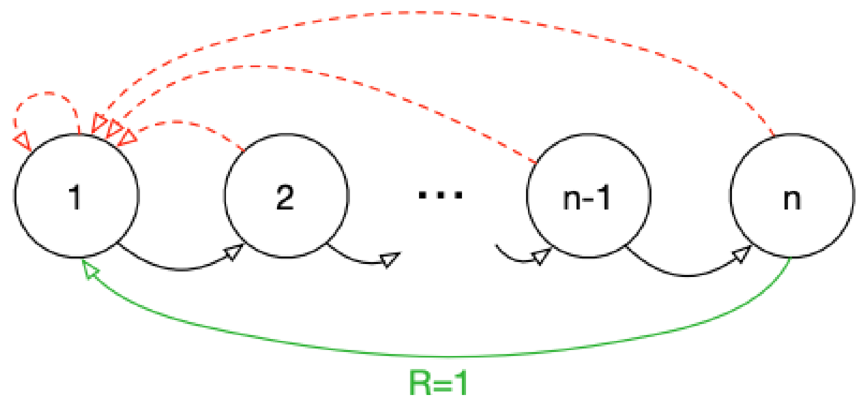


Figure 1: Blind Cliff Walk problem. The agent must guess the correct action to make it to the reward state at the end of the chain. Any other action results in no reward and returns the agent to the beginning of the chain. In this problem, it is very difficult for the agent to obtain reward in the first place, but also very difficult to learn how to return to the reward.

It is well-known that model-based planning algorithms such as value iteration can be made more efficient by prioritizing updates in an appropriate order. Based on this idea, prioritized sweeping [17, 1] was proposed to update the states with the largest Bellman error, which can speed up the learning for state-based and compact (using function approximator) representations of the model and the value function. Similar to prioritized sweeping, prioritized replay [18] assigns priorities to each transition in the experience replay memory based on the TD error [19] in model-free Deep RL algorithms, which is shown to improve the learning efficiency tremendously compared with uniform sampling from the experience replay memory. There are also several other proposed methods trying to improve the sample efficiency of DRL algorithms. [11] proposed a sampling technique which updates the transitions backward from a whole episode. [10] proposed an approach to select appropriate transition sequences to accelerate the learning. [24] proposed a new experience replay method which gives the reward transition a high priority and then propagate the priority backward to its previous transition once it has been sampled.

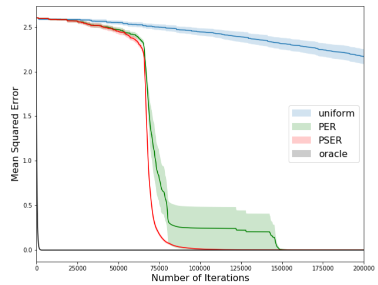
The approach proposed in this paper is an extension of prioritized replay. While we assign a priority to a transition in the replay memory, we also propagate this priority information to previous transitions, and experimental results on the Atari 2600 benchmark show that our algorithm substantially improves upon both PER and uniform sampling.

3 Prioritized Sequence Experience Replay

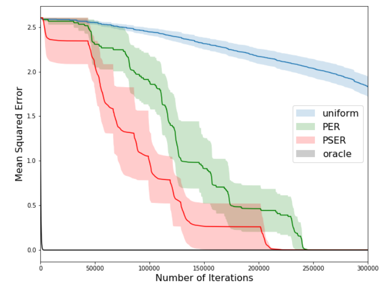
Using a replay memory is shown to be able to stabilize the neural network [16], but the uniform sampling technique is shown to be inefficient [18]. To improve the sampling efficiency, [18] proposed prioritized experience replay (PER) to use the last observed TD error to make more effective use of the replay memory for learning. In this paper we propose an extension of PER named Prioritized Sequence Experience Replay (PSER), which can also take advantage of information about the trajectory by propagating the priorities back through the sequence of transitions.

3.1 A motivating example

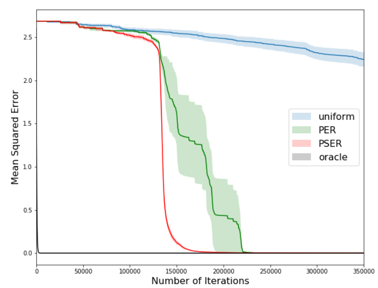
To motivate and understand the potential gains of PSER, we implemented four different agents in the artificial 'Blind Cliffwalk' environment introduced in [18], shown in Figure 1. With only n states, the environment requires an exponential number of random steps until the first non-zero reward; to be precise, the chance that a random sequence of actions will lead to the reward is 2^{-n}. The first agent replays transitions uniformly from the experience at random, while the second agent invokes an oracle to prioritize transitions, which greedily selects the transition that maximally reduces the global loss in its current state (in hindsight, after the parameter update). The last two agents use PER



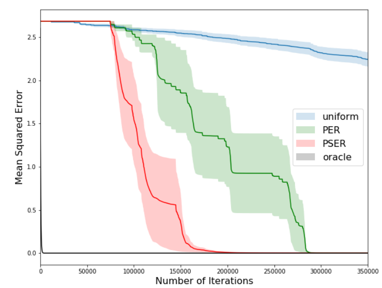
(a) 15 states with all transitions initialized with max priority.



(b) 15 states with all transitions initialized with ϵ priority.



(c) 16 states with all transitions initialized with max priority.



(d) 16 states with all transitions initialized with ϵ priority.

Figure 2: Comparison of convergence speed for a PSER, PER, uniform, and oracle agent in the Blind Cliffwalk environment with 15 and 16 states. PSER shows improved performance with faster convergence speed as compared to PER and uniform in all cases.

and PSER sampling techniques respectively. In this chain environment with sparse reward, there is only one non-zero reward that is located at the end of chain marked in green. The agent must learn the correct sequence of actions in order to reach the goal state and collect the reward. Any incorrect action results in a terminal state and the agent starts back at the beginning of the chain. For the details of the experiment setup, see the appendix.

To provide some early intuition of the benefits of PSER, we compare performance on the Blind Cliffwalk in Figure 2. We track the mean squared error between the ground truth Q-value and Q-learning result every 100 iterations. Better performance in this experiment means that the loss curve more closely matches the oracle. The PER paper demonstrated that by prioritizing transitions based off of the TD error, improvements in performance were obtained over uniform sampling, reducing the difference between the oracle and uniform sampling strategies. Our results show that by prioritizing the transition with TD error and decaying a portion of this priority to previous transitions, further improvements are obtained with much faster and earlier convergence as compared to PER. We show results of this Blind Cliffwalk environment with 15 and 16 states and also show how the initialization of the transition's priority (max priority or small non-zero priority, ϵ) in the replay memory affects convergence speed. We find that PSER consistently outperforms PER in this problem.

Examining the curves in Figure 2 more closely, there is an initial period where PSER is comparable to both uniform and PER. This is due to all samples initially having the same priority in the replay memory which results in uniform sampling. This uniform sampling continues until the goal transition is sampled from the replay memory and a non-zero TD error is encountered.

At this point, how the agent updates the priority of this transition in the replay memory is what causes the divergence in performance between the algorithms. Uniform sampling continues to sample transitions with equal probability from the replay memory. PER updates the priority of the one transition in the memory, but all other transitions in the memory are still chosen at uniform which still results in inefficient sampling as we need to wait until the transition preceding the goal state is sampled. PSER capitalizes on the high TD error that was received and decays a portion of the new priority of the goal state to the preceding states to encourage sampling of the states that led to the goal state. It is clear from Figure 2 that by decaying the priority of the high TD error states, we can encourage faster convergence to the true Q-value in an intuitive and effective way.

3.2 Prioritized sequence decay

In this subsection we formally define the concept of the prioritized sequence and decaying priority backward in time within the replay memory. Formally, the priority of transitions will be decayed as follows:

Suppose in one episode, we have a trajectory of transitions T_0 to T_{n-1} ($T_i = (s_i, a_i, r_i, s_{i+1})$) stored in the experience replay memory with priorities $p = (p_0, p_1, \cdots, p_{n-1})$. If the agent observes a new transition $T_n = (s_n, a_n, r_n, s_{n+1})$, we first calculate its priority p_n based on its TD error similar to the PER algorithm:

$$\delta = r_n + \gamma Q_{\text{target}}(s_{n+1}, \arg\max_a Q(s_{n+1}, a)) - Q(s_n, a_n) \tag{1}$$

$$p_n = |\delta| + \epsilon, \tag{2}$$

where ϵ is a small positive constant to allow transitions with zero TD-error a small probability to be resampled.

As in [18], according to the calculated priority, the probability of sampling transition i is:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \tag{3}$$

where the exponent α determines how much prioritization is used. We then decay the priority exponentially (with decay coefficient ρ) to the previous transitions stored in the replay memory for that episode and apply a $\max$ operator in an effort to preserve any previous priority assigned to the decayed transitions:

$$\begin{aligned} p_{n-1} &= \max\{p_n \cdot \rho^1, p_{n-1}\} \\ p_{n-2} &= \max\{p_n \cdot \rho^2, p_{n-2}\} \\ &\cdots \end{aligned} \tag{4}$$

Here we note that as the priority is decayed, we expect that after some number of updates the decayed priority p_{n-k} is negligible and is therefore wasted computation. We therefore define a window of size W over which we will allow the priority p_n to be decayed, after which we will stop. We arbitrarily selected a threshold of 1% of p_n as a cutoff for when the decayed priority becomes negligible. We compute the window size W, then, based off the value of the hyperparameter ρ as follows:

$$p_n \cdot \rho^W \leq 0.01 p_n \tag{5}$$

$$W \leq \frac{\ln 0.01}{\ln \rho}. \tag{6}$$

Through the above formulation for PSER, we identified an issue which we termed "priority collapse" during the decay process. Suppose for a given environment, PSER has already decayed the priority backward for the "surprising" transition, which we will call T_i. Let's assume that currently all of the Q-values are 0 and we sampled a transition in the replay memory, T_{i-2}, that led to T_i. From Equation 1 and Equation 2 the priority for transition T_{i-2} would drop to ϵ. The result is that a priority sequence that was recently decayed has almost no effect as it is almost guaranteed to be eliminated at the next sampling. When this happens to multiple states we term this "priority collapse" and the potential benefits of PSER are eliminated making it nearly equivalent to traditional PER.

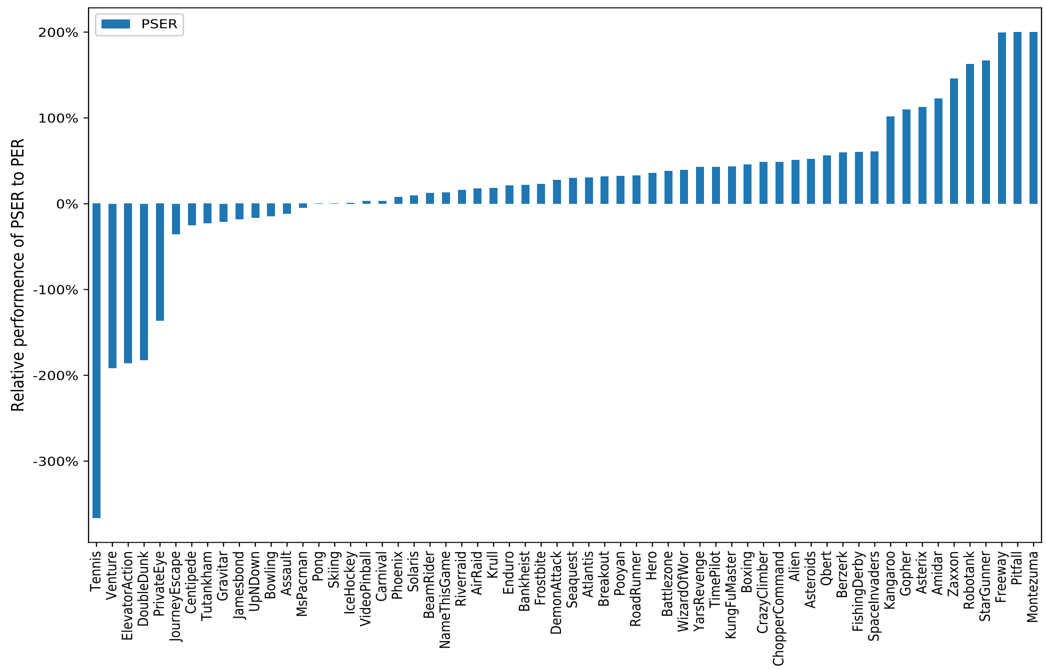


Figure 3: Relative performance of prioritized sequence experience replay (PSER) to prioritized experience replay (PER) in all 60 Atari 2600 benchmark games. 0% on the vertical axis implies equivalent performance; positive numbers mean PSER performed better; negative numbers mean PSER performed worse.

In order to prevent this catastrophic "priority collapse" we design a parameter, η which forces the priority to decrease slowly. When updating the priority of a sampled transition in the replay memory, we want to maintain a portion of its previous priority to prevent it from decreasing too quickly:

$$p_i \leftarrow \max(|\delta| + \epsilon, \eta \cdot p_i). \tag{7}$$

where i here refers to the index of the sampled transition within the replay memory.

This provides time for the Bellman update process to propagate information about the TD error through the sequence and for the neural network to learn an appropriate Q-value approximation. The full algorithm is presented in the appendix.

4 Experimental Methods

We now describe the methods and setup used for configuring and evaluating the learning agents.

4.1 Evaluation methodology

We used the Arcade Learning Environment [3] to evaluate the performance of our proposed algorithm. We follow the same training and evaluation procedures of [8, 16, 20]. We calculate the average score during training every 1M frames in the environment. After every 1M frames, we then stop training and evaluate the agent's performance for 500K frames. We also truncate the episode lengths to 108K frames (or 30 minutes of simulated play) as in [20, 8]. In the results section, we report the mean and median human normalized scores of PSER, PER, and uniform DQN in the Atari 2600 benchmark and in the appendix we provide full learning curves for all games in the *no-op starts* testing regime.

4.2 Hyperparameter tuning

DQN has a number of different hyperparameters that can be tuned. To provide a comparison with our baseline DQN agent, we used the hyperparameters that are provided in [16] for the DQN agent formulation (see appendix for more details).

Our PSER implementation also has hyperparameters that require tuning. Due to the large amount of time it takes to run the full 200M frames for the DQN tests (multiple days), we used a coordinate descent approach to tune the PSER parameters for a subset of the Atari 2600 benchmark.

In the coordinate descent approach, we define a set of different values to test for each parameter. Then, holding all other parameters constant, we tune one parameter until the best result is obtained. We then fix this tuned parameter and move to the next parameter and repeat this process until all parameters have been tuned. While this does not test every combination of parameter value, it greatly reduces the hyperparameter search space and proved to provide good results.

The hyperparameters obtained during the hyperparameter search were used for all Atari 2600 benchmark results reported in this paper. Hyperparameters were not tuned for each game so as to better measure how the algorithm generalizes over the whole suite of the Atari 2600 benchmark. We found the best results were obtained with W = 10, ρ = 0.65, and η = 0.7.

5 Analysis

In this section, we analyze the main experimental results using the Atari 2600 benchmark available within the OpenAI gym environment [4]. We show that by adding PSER to the DQN agent we can achieve substantial improvement to performance as compared to the PER and uniform sampling.

5.1 Baselines

We compared our algorithm DQN with PSER with two baselines: (1) DQN with uniform experience replay and (2) DQN with prioritized replay (PER). The PSER implementation used an identical DQN agent, except that the replay memory was altered to implement our prioritized sequence experience replay scheme. All DQN agents used identical DQN hyperparameters to allow for direct comparison of prioritized sequence experience replay and prioritized replay.

5.2 Comparison with baselines

Figure 3 shows the relative performance of prioritized sequence experience replay and prioritized replay for all 60 Atari 2600 benchmark games. We can see from this figure that PSER leads to substantial improvements over PER. Out of 60 games, PSER outperformed PER in 45 games, 10 of which resulted in a relative difference of over 100%. In the games where PER outperformed PSER, we notice that the relative difference is less than 40% in all but 5 of the Atari 2600 benchmark games, which shows that even when PSER did not perform well, the performance is still comparable to that of PER.

In Table 1 we compare the final evaluation performance of PSER, PER, and uniform sampling on the Atari 2600 benchmark by calculating the median and mean human normalized scores (See the appendix for the learning curves of all Atari games). PSER achieves a median score of 88% and a mean score of 536% in the no-ops regime, significantly improving upon both PER and uniform sampling.

Table 1: Median and Mean human normalized scores of the best agent snapshot across 55 Atari games where human scores were available. For methods marked with an asterisk, the scores come from the corresponding publication. Uniform's results come from [18] to ensure the same hyperparameters were used. Uniform's results did not include all 55 Atari games.

Sampling Strategy	Median	Mean
PSER	88%	536%
PER	55%	444%
Uniform(*)	48%	122%

5.3 Learning Speed

Each agent is run on a single GPU and the learning speed for each variant varies depending on the game. For a full 200 million frames of training, this corresponded to approximately 5-10 days of computation time depending on the hardware used. We found that the learning speed of prioritized sequence experience replay is comparable to the prioritized replay when a small decay coefficient value is used. As this value increases, there is an increase in the computation time due to the larger decay window.

6 Discussion

We have demonstrated that prioritizing sequences of transitions can achieve more reward in less training time in many Atari 2600 benchmark games.

While performing this analysis, we tested different approaches to PSER and discovered phenomena that we did not expect. The inclusion of the previous priority parameter was debatable at first since we are assigning an artificial priority to transitions where the neural network may have already learned a good representation. The benefit to this parameter is that transitions we decayed priority to would no longer drop priority before being sampled several more times. This helped to more efficiently propagate value back through the transitions that led to reward.

When running on the Atari 2600 benchmark games, we needed to choose a fixed hyperparameter set for a fair comparison between PSER and PER. However, the Atari 2600 benchmark games vary in how to obtain reward and how long the delay is between action and reward. Therefore, by choosing a fixed decay window we are potentially losing performance on games where there is a longer sequence of actions before reward. We speculate that by introducing an adaptive decay window, better performance can be achieved in the Atari 2600 benchmark.

In our analysis, we followed the same approach in [8] to add new transitions to the replay memory with the current maximum priority to ensure that recent transitions would be sampled frequently. In the approach by [9], the authors added transitions to the replay memory with priority selected by the actors in an online approach. We found that for a single agent, performance was better when adding transitions with the max priority as compared to adding with the agent's current policy. We hypothesize that this is due to an instability in the agent's learning. By only sampling transitions based on the agent's current policy, we reduce the number of recent transitions sampled and focus on transitions where the agent has a poor representation. This has the potential to cause learning instability since we are not sampling transitions where the agent has a good representation.

From our analysis in the Blind Cliff-walk environment, adding with the max priority delayed the convergence to the true Q value as compared to adding with a small priority, ϵ. Intuitively, adding transitions to the replay memory with the current TD error makes sense to encourage the agent to initially sample these high priority transitions sooner. Given that this is a greedy approach (adding transitions with the current TD error), it would be interesting to try different approaches to the initial prioritization of transitions in the replay memory and the effect that this has on the convergence speed of different algorithms.

7 Conclusion

In this paper we introduced Prioritized Sequence Experience Replay (PSER), a method for prioritizing sequences of transitions to both learn more efficiently and effectively. This method shows substantial performance improvements over prioritized replay in the Atari 2600 benchmark with PSER outperforming PER in 45 out of 60 Atari games. We show that improved ability for information to flow during the training process can lead to faster convergence, as well as, increased performance, potentially leading to increased data efficiency for deep reinforcement learning problems.

References

[1] David Andre, Nir Friedman, and Ronald Parr. Generalized prioritized sweeping. In *Advances in Neural Information Processing Systems*, pages 1001–1007, 1998.

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

[3] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[5] Tim de Bruin, Jens Kober, Karl Tuyls, and Robert Babuška. The importance of experience replay database composition in deep reinforcement learning. In *Deep Reinforcement Learning Workshop, NIPS*, 2015.

[6] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *International Conference on Learning Representations*, 2018.

[7] Yarin Gal. Uncertainty in deep learning. *University of Cambridge*, 2016.

[8] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *Association for the Advancement of Artificial Intelligence*, 2018.

[9] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *International Conference on Learning Representations*, 2018.

[10] Thommen Karimpanal George and Roland Bouffanais. Experience replay using transition sequences. *Frontiers in neurorobotics*, 12:32, 2018.

[11] Su Young Lee, Sungik Choi, and Sae-Young Chung. Sample-efficient deep reinforcement learning via episodic backward update. *arXiv preprint arXiv:1805.12375*, 2018.

[12] Yuxi Li. Deep reinforcement learning. *CoRR*, abs/1810.06339, 2018.

[13] 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. *arXiv preprint arXiv:1509.02971*, 2015.

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

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

[16] 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.

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

[18] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *International Conference on Learning Representations*, 2016.

[19] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[20] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, volume 2, page 5. Phoenix, AZ, 2016.

[21] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

[22] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1995–2003, 2016.

[23] Shangtong Zhang and Richard S Sutton. A deeper look at experience replay. *arXiv preprint arXiv:1712.01275*, 2017.

[24] Yangxin Zhong, Borui Wang, and Yuanfang Wang. Reward backpropagation prioritized experience replay, 2017.

Appendix

1 Blind Cliffwalk

For the Blind Cliffwalk experiments, we use a tabular Q-learning setup with four different experience replay scheme, where the Q-values are represented using a tabular look-up table.

For the tabular Q-learning algorithm, the replay memory of the agent is first filled by exhaustively executing all 2^n possible sequences of actions until termination (in random order). This guarantees that exactly one sequence will succeed and hit the final reward, and all others will fail with zero reward. The replay memory contains all the relevant experience (the total number of transitions is $2^{n+1} - 2$, at the frequency that it would be encountered when acting online with a random behavior policy.

After generating all the transitions in the replay memory, the agent will next select a transition from the replay memory to learn at each time step. For each transition, the agent first computes its TD-error using:

$$\delta_t := R_t + \gamma_t \max_a Q\left(S_t, a\right) - Q\left(S_{t-1}, A_{t-1}\right) \tag{8}$$

and updates the parameters using stochastic gradient ascent:

$$\theta \quad \leftarrow \quad \theta + \eta \cdot \delta_t \cdot \nabla_\theta \left. Q\right|_{S_{t-1}, A_{t-1}} = \theta + \eta \cdot \delta_t \cdot \phi\left(S_{t-1}, A_{t-1}\right) \tag{9}$$

The four different replaying scheme we will be using here is uniform, oracle, PER, and PSER. For uniform replaying scheme, the agent will randomly select the transition from the replay memory uniformly. For the oracle replaying scheme, the agent will greedily select the transition that maximally reduces the global loss (in hindsight, after the parameter update). For the PER replaying scheme, the agent will first set the priorities of all transitions to either 0 or 1. Then after each update, the agent will assign new priority to the sampled transition using:

$$p = |\delta| + \epsilon \tag{10}$$

and the probability of sampling transition i is

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{11}$$

where δ is the TD error for the sampled transition which can be calculated from equation (1), $\alpha = 0.5$ and $\epsilon = 0.0001$. For the PSER replaying scheme, we first calculate the priority as in equation (3) and propagate back the priority 5 steps before:

$$\begin{aligned} p_{n-1} &= \max\{\rho^1 p_n, p_{n-1}\} \\ p_{n-2} &= \max\{\rho^2 p_n, p_{n-2}\} \\ p_{n-3} &= \max\{\rho^3 p_n, p_{n-3}\} \\ &\dots \end{aligned} \tag{12}$$

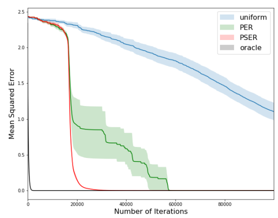
Then the agent will select transition with probability based on equation (4).

For this experiment, we vary the size of the problem (number of states n) from 13 to 16. The discount factor is set to $\gamma = 1 - \frac{1}{n}$ which keeps values on approximately the same scale independently of n. This allows us to use a fixed step-size of $\eta = \frac{1}{4}$ in all experiments.

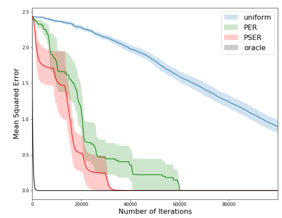
We track the MSE between the ground truth Q value and Q-learning result every 100 iterations. Better performance in this experiment means that the loss curve more closely matches the oracle. The PER paper demonstrated that PER improves performance over uniform sampling. Our results show that PSER further improves the results with much faster and earlier convergence as compared to PER. We show results varying the state-space size from 13 to 16 and and that PSER consistently outperforms PER in this problem as shown in Figure 4.

2 Hyperparameters

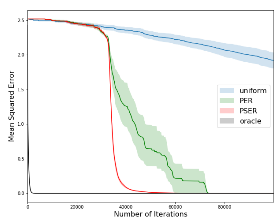
Throughout this paper, our primary baseline for comparison was Prioritized Experience Replay (PER) and uniform sampling. For each implementation we used the standard DQN algorithm without



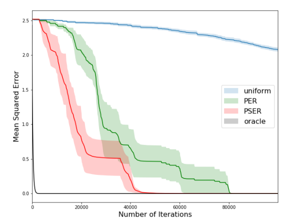
(a) 13 states with all transitions initialized with max priority.



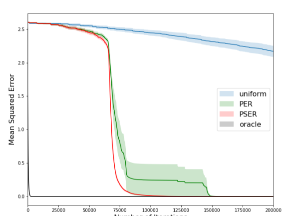
(b) 13 states with all transitions initialized with ϵ priority.



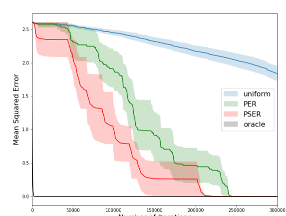
(c) 14 states with all transitions initialized with max priority.



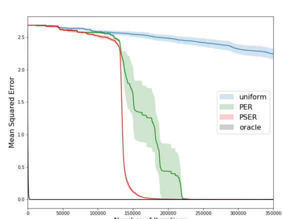
(d) 14 states with all transitions initialized with ϵ priority.



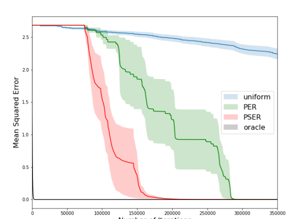
(e) 15 states with all transitions initialized with max priority.



(f) 15 states with all transitions initialized with ϵ priority.



(g) 16 states with all transitions initialized with max priority.



(h) 16 states with all transitions initialized with ϵ priority.

Figure 4: Results of the Blind Bliffwalk environment comparing the number of iterations until convergence of the true Q value between a PSER, PER, uniform, and oracle agent. We can see that in each case, PSER further improves upon the performance of PER and leads to faster convergence to the true Q value.

any additional modifications to provide a more apples-to-apples comparisons between the different sampling techniques. All of the hyperparameters for DQN were the same between the PSER, PER and uniform implementations and were obtained from [16].

In selecting our final set of hyperparameters for PSER, we tested a range of different values over a subset of Atari games. Table 1 lists the range of values that were tried for each parameter and Table 2 lists the chosen parameters. To obtain the final set of parameters, two parameters were held constant while we tuned one, then we fixed the tuned parameter with best performance and tuned the next. This greatly reduced the search space of parameters and led to a set of parameters that performed well.

Table 2: Hyperparameters tested in experiments.

HYPERPARAMETER	RANGE OF VALUES
DECAY WINDOW W	5, 10, 20, 50, 100
DECAY COEFFICIENT ρ	0.3, 0.5, 0.85, 0.95, 0.99
PREVIOUS PRIORITY η	0, 0.3, 0.5, 0.7

Table 3: Prioritized Sequence Experience Replay hyper-parameters.

PARAMETER	VALUE
DECAY WINDOW W	10
DECAY COEFFICIENT ρ	0.65
PREVIOUS PRIORITY η	0.7

3 Psuedocode

Algorithm 1 lists the pseudocode for the PSER algorithm.

Algorithm 1 Prioritized Sequence Experience Replay

Input: minibatch k, step-size ξ, replay period K and size N, exponents α and β, budget T, decay window W, decay coefficient ρ, previous priority η.
Initialize replay memory $\mathcal{H} = \emptyset$, $\Delta = 0$, $p_1 = 1$
Observe S_0 and choose $A_0 \sim \pi_\theta(S_0)$
for $t = 1$ **to** T **do**
  Observe S_t, R_t, γ_t
  Store transition $(S_{t-1}, A_{t-1}, R_{t-1}, \gamma_t, S_t)$ in $\mathcal{H}$ with maximal priority $p_t = \max_{i<t} p_i$
  if $t \equiv 0 \bmod K$ **then**
    for $j = 1$ **to** k **do**
      Sample transition $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$
      Compute importance-sampling weight $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$
      Compute TD-error
      $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
      Update transition priority $p_j \leftarrow \max\{|\delta_j| + \epsilon, \eta \cdot p_j\}$
      Accumulate weight-change $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
      for $l = 1$ **to** W **do**
        Update transition priority $p_{j-l} \leftarrow \max\{(|\delta_j| + \epsilon) \cdot \rho^l, p_{j-l}\}$ to transition l steps backward
      end for
    end for
    Update weights $\theta \leftarrow \theta + \xi \cdot \Delta$, reset $\Delta = 0$
    From time to time copy weights into target network $\theta_{\text{target}} \leftarrow \theta$
  end if
  Choose action $A_t \sim \pi_\theta(S_t)$
end for

Table 4: **no-op** Starts evaluation regime: Here we report the raw scores across all games, averaged over 200 evaluation episodes, from the agent snapshot that obtained the highest score during training.

GAME	PER	PSER
AIRRAID	7,727.5	**9,257.0**
ALIEN	1,624.2	**2,737.4**
AMIDAR	215.2	**898.7**
ASSAULT	**2,886.2**	2,565.2
ASTERIX	2,888.8	**10,369.2**
ASTEROIDS	850.5	**1,449.3**
ATLANTIS	392,782.5	**533,401.5**
BANKHEIST	537.9	**670.2**
BATTLEZONE	21,620.0	**31,745.0**
BEAMRIDER	9,375.9	**10,641.6**
BERZERK	457.3	**846.2**
BOWLING	**46.2**	39.9
BOXING	60.2	**95.7**
BREAKOUT	281.4	**388.6**
CARNIVAL	5,118.4	**5,289.5**
CENTIPEDE	**5,142.6**	3,995.4
CHOPPERCOMMAND	3,247.5	**5,324.0**
CRAZYCLIMBER	73,211.5	**120,004.0**
DEMONATTACK	12,942.4	**17,106.0**
DOUBLEDUNK	-15.4	**-0.7**
ELEVATORACTION	**39,205.5**	1,436.5
ENDURO	502.2	**621.7**
FISHINGDERBY	6.8	**12.8**
FREEWAY	0.0	**30.5**
FROSTBITE	720.5	**909.9**
GOPHER	2,972.1	**10,191.0**
GRAVITAR	**633.2**	514.5
HERO	13,232.4	**19,056.8**
ICEHOCKEY	**-3.1**	**-3.1**
JAMESBOND	**638.0**	532.2
JOURNEYESCAPE	-1,698.0	**-1,188.0**
KANGAROO	2,621.0	**8,015.5**
KRULL	6,109.0	**7,355.6**
KUNGFUMASTER	14,546.5	**22,616.5**
MONTEZUMA	0.0	**0.5**
MSPACMAN	**2,698.3**	2,574.6
NAMETHISGAME	6,125.0	**6,994.0**
PHOENIX	9,640.6	**10,421.5**
PITFALL	**0.0**	-3.6
PONG	**20.2**	**20.2**
POOYAN	2,253.6	**3,113.7**
PRIVATEEYE	**910.7**	172.0
QBERT	7,896.8	**14,052.6**
RIVERRAID	6,202.9	**7,275.1**
ROADRUNNER	24,733.5	**34,408.0**
ROBOTANK	5.4	**52.5**
SEAQUEST	4,170.5	**5,637.8**
SKIING	**-12,665.9**	-12,706.0
SOLARIS	1,714.0	**1,885.6**
SPACEINVADERS	1,064.4	**1,995.2**
STARGUNNER	2,653.0	**29,154.0**
TENNIS	**20.6**	-6.0
TIMEPILOT	4,370.0	**6,747.0**
TUTANKHAM	**201.3**	160.2
UPNDOWN	**12,215.2**	10,356.5
VENTURE	**873.0**	19.0
VIDEOPINBALL	238,321.3	**246,270.5**
WIZARDOFWOR	863.5	**1,284.0**
YARSREVENGE	14,933.2	**23,039.2**
ZAXXON	633.0	**4,047.5**

4 Full results

See Figure 5 for the detailed learning curves of every Atari game using the no-op starts testing regime. These learning curves are smoothed with a moving average of 10 to improve readability. In each Atari game, DQN with PER and DQN with PSER are presented. See Table 4 for a breakdown of the best score achieved by each algorithm.

Bolded entries within each row highlight the result with the highest performance within the row.

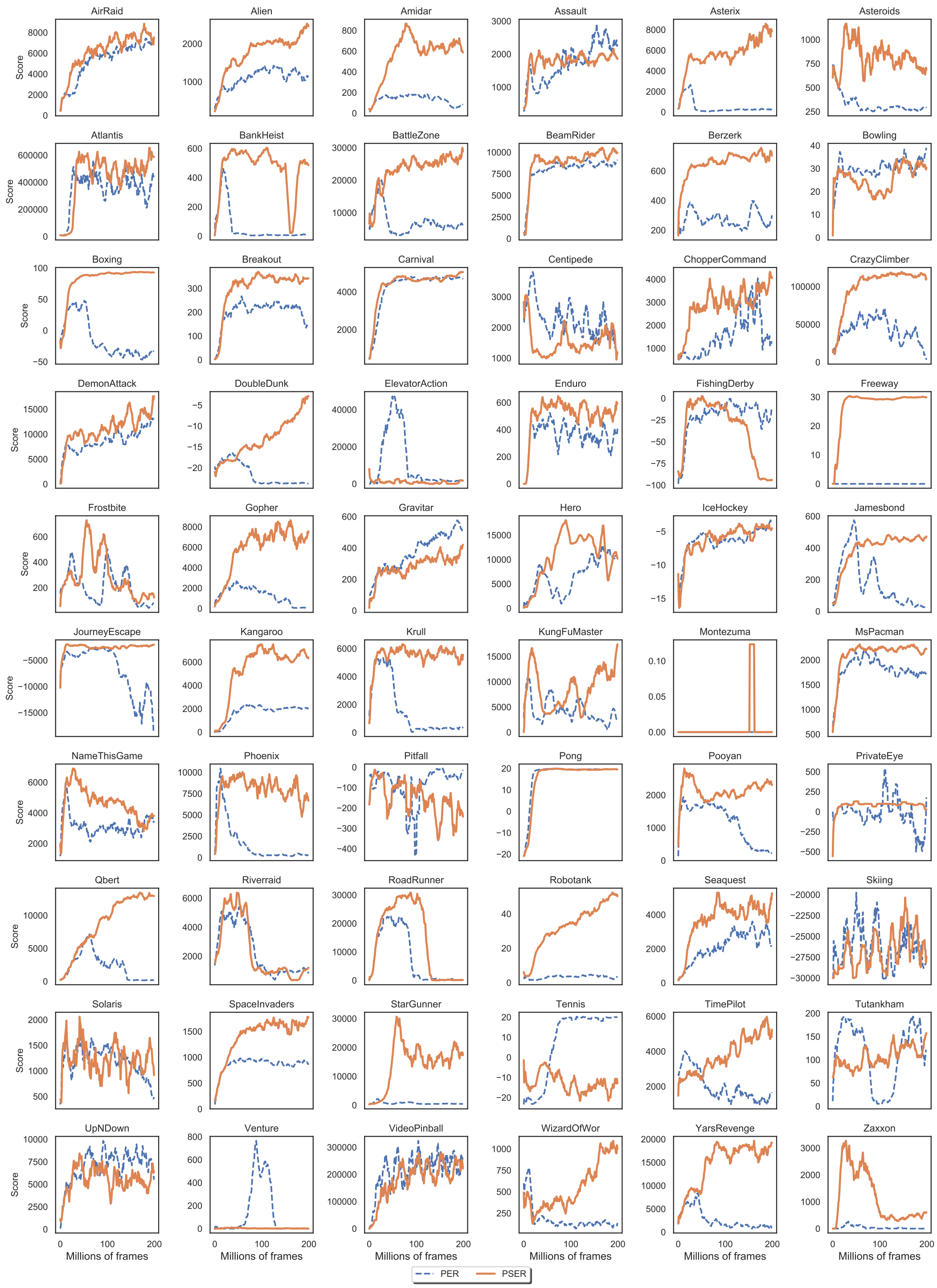


Figure 5: Results of DQN with PER and DQN with PSER.