Investigating the Interplay of Prioritized Replay and Generalization

Parham Mohammad Panahi[1], **Andrew Patterson**[1], **Martha White**[1,2,3], **Adam White**[1,2,3]
[1] University of Alberta [2] Alberta Machine Intelligence Institute (Amii) [3] CIFAR AI Chair
{parham1, ap3, whitem, amw8}@ualberta.ca

Abstract

Experience replay, the reuse of past data to improve sample efficiency, is ubiquitous in reinforcement learning. Though a variety of smart sampling schemes have been introduced to improve performance, uniform sampling by far remains the most common approach. One exception is Prioritized Experience Replay (PER), where sampling is done proportionally to TD errors, inspired by the success of prioritized sweeping in dynamic programming. The original work on PER showed improvements in Atari, but follow-up results were mixed. In this paper, we investigate several variations on PER, to attempt to understand where and when PER may be useful. Our findings in prediction tasks reveal that while PER can improve value propagation in tabular settings, behavior is significantly different when combined with neural networks. Certain mitigations—like delaying target network updates to control generalization and using estimates of expected TD errors in PER to avoid chasing stochasticity—can avoid large spikes in error with PER and neural networks but generally do not outperform uniform replay. In control tasks, none of the prioritized variants consistently outperform uniform replay. We present new insight into the interaction between prioritization, bootstrapping, and neural networks and propose several improvements for PER in tabular settings and noisy domains.

1 Introduction

Experience Replay (ER) is widely used in deep reinforcement learning (RL) and appears critical for good performance. The core idea of ER is to record transitions (experiences) in a memory, called a buffer, replay them by sub-sampling mini-batches to update the agent's value function and policy. ER allows great flexibility in agent design. ER can be used to learn from human demonstrations (pre-filling the replay buffer with human data) allowing off-line pre-training and fine-tuning. ER has been used to learn many value functions in parallel, as in Hindsight ER (Andrychowicz et al., 2018), Universal Value Function Approximators (Schaul et al., 2015), and Auxiliary Task Learning (Jaderberg et al., 2016; Wang et al., 2024). ER can be seen as a form of model-based RL where the replay buffer acts as a non-parametric model of the world (Pan et al., 2018; Van Hasselt et al., 2019), or ER can be used to directly improve model-based RL systems (Lu et al., 2024). In addition, ER can be used to mitigate forgetting in continual learning systems (Anand & Precup, 2024). ER has proven crucial for mitigating the sample efficiency challenges of online RL, as well as mitigating instability due to off-policy updates and non-stationary bootstrap targets. The most popular alternative, asynchronous training, requires multiple copies of the environment, which is not feasible in all domains and typically makes use of a buffer anyway (e.g., Horgan et al. (2018)).

There are many different ways ER can be implemented. The most widely used variant, i.i.d or uniform replay, samples experiences from the buffer with equal probability. As discussed in the original paper (Lin, 1991), ER can be combined with lambda-returns and various sampling methods. Experience can be sampled in reverse order it occurred, starting at terminal states. Transitions can be sampled from a priority queue ordered by TD errors—the idea being transitions that caused large updates are more important and should be resampled. Samples can be drawn with or without replacement—avoiding saturating the mini-batch with high priority transitions. The priorities can be periodically updated. We could use importance sampling to re-weight the distribution in the queue, and generally we could dynamically change the distribution during the course of learning. Despite the multitude of possible variants (Igata et al., 2021; Sun et al., 2020; Kumar & Nagaraj, 2023; Lee et al., 2019; Hong et al., 2023; Kobayashi, 2024; Li et al., 2021; 2022) simple i.i.d replay remains the most widely used approach.[1]

The exception to this is Prioritized Experience Replay (PER) (Schaul et al., 2016), where experience is sampled from the buffer based on TD errors. Like prioritized sweeping that inspired it (Moore & Atkeson, 1993), PER in principle should be more efficient than i.i.d sampling. Imagine, a sparse reward task where non-zero reward is only observed at the end of long trajectories; sampling based on TD errors should focus value updates near the terminal state efficiently propagating reward information across the state space. This approach was shown to improve over i.i.d sampling in Atari when combined with Double DQN (Schaul et al., 2016). The results in follow up studies, however, were mixed and did not show a clear benefit for using PER generally (Fedus et al., 2020; Hessel et al., 2018; Li et al., 2021; Ma et al., 2023; Horgan et al., 2018; Fu et al., 2022). Compared with i.i.d sampling, PER introduces six hyper-parameters controlling importance sampling and how additional experiences are mixed with the prioritized distribution.

In this paper, we explore several different variations of PER in carefully designed experiments in the hopes of better understanding where and when PER is useful. Canonical PER (Schaul et al., 2016) uses several additional components that make the impacts of prioritization harder to analyze. We compare PER with several simplified variants using simple chain tasks where value propagation and prioritization should be critical for performance. We find that only in tabular prediction, do all prioritized variants outperform i.i.d replay. Combining basic prioritization with sampling without replacement and updating the priorities in the buffer (things not done in public implementations), further improves performance in the tabular case. Our results show that prioritization, bootstrapping, and function approximation cause problematic over-generalization, possibly motivating the design choices of PER which ultimately causes the method to function more like i.i.d sampling under function approximation. Our results in chain domains with neural network function approximation and across several classic control domains, perhaps unsurprisingly, shows no clear benefit for any prioritized method.

We also introduce and investigate a natural extension to PER based on ideas from Gradient TD methods (Sutton et al., 2009; Patterson et al., 2022). These methods stabilize off-policy TD updates by learning an estimate of the expected TD error. This estimate can be used to compute priorities and is less noisy than using instantaneous TD errors. This expected PER algorithm works well in tabular prediction tasks and noisy counter-examples where PER fails, but is generally worse than i.i.d sampling under function approximation and ties i.i.d in classic control problems—though it appears more stable. Although somewhat of a negative result, expected PERs performance suggests noise is not the explanation for i.i.d sampling's superiority over PER and more research is needed to find generally useful prioritization mechanisms.

2 Background, Problem Formulation, and Notation

In this paper, we investigate problems formulated as discrete-time, finite Markov Decision Processes (MDP). On time step, t, the agent selects an action $A_t \in \mathcal{A}$ in part based on the current state, $S_t \in \mathcal{S}$. The MDP transitions to a new state S_{t+1} and emits a reward signal $R_{t+1} \in \mathcal{R}$. The agent's action choices are determined by it's policy $A_t \sim \pi(\cdot|S_t)$, and the goal of learning is to adjust π to maximize the future expected return $\mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots |S_t = s, A_t = a]$, where $\gamma \in [0, 1]$. The expectation is dependent on future actions determined by π and future states and rewards according to the MDP.

[1] See Wittkuhn et al. (2021) for a nice review.

We focus on action-value methods for learning π. In particular, Q-learning estimates the state-action value function $q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \;\; \forall\, s, a \in \mathcal{S} \times \mathcal{A}$ via temporal difference updates from sample interactions: $\hat{q}(S_t, A_t) = \hat{q}(S_t, A_t) + \alpha\delta_t$, where $\delta_t \doteq R_{t+1} + \gamma \max_a \hat{q}(S_{t+1}, \cdot) - \hat{q}(S_t, A_t)$ is called the TD-error with learning-rate parameter $\alpha \in \mathbb{R}^+$. Actions are selected according to an ϵ-greedy policy: selecting $A_t = \arg\max \hat{q}(S_t, \cdot)$ $1 - \epsilon$ percentage of the time and a random action otherwise. In many tasks, it is not feasible to learn an action-value for every state. In these cases, we use a non-linear parametric approximation of the value, $\hat{q}_{\mathbf{w}}(s,a) \approx q_\pi(s,a)$, where $\mathbf{w}$ are the parameters of a neural network (NN), which are adjusted via semi-gradient Q-learning rule.

Semi-gradient Q-learning when combined with NNs is often unstable, and so DQN is often preferred. The DQN algorithm combines: target networks, ER, and an optimizer (Mnih et al., 2015). Target Networks replace $\max_a \hat{q}_{\mathbf{w}}(S_{t+1}, \cdot)$ in the TD-error with an older copy of the network. In this paper we use the Adam optimizer (Kingma & Ba, 2015). Experience Replay is used to perform mini-batch updates to $\hat{q}_{\mathbf{w}}$ from a finite, first-in-first-out buffer. Sampling from the buffer is uniform or i.i.d meaning the value estimate on the current step is updated based experiences observed in the recent past, not necessarily the most recent transition. The appendix contains the pseudo code for DQN and its key hyperparameters.

In control tasks we learn $\hat{q}$ and π, however, in prediction tasks π is given and fixed and we are interested in learning the state-value function: $\hat{v}(s) \approx v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s]$. This can be done using the Temporal Difference learning algorithm (Sutton, 1988), the state-value analog of TD update above, which has a semi-gradient variant for learning $\hat{v}_{\mathbf{w}} : \mathcal{S} \to \mathbb{R}$. See Sutton & Barto (2018) for an extensive overview of all these topics.

3 Variants of Prioritized Replay

In this section, we define two variants of PER that we use to better understand the role of prioritization. We use the name DM-PER for Schaul et al. (2016)'s prioritized replay algorithm and *Uniform* to refer to classic i.i.d replay.

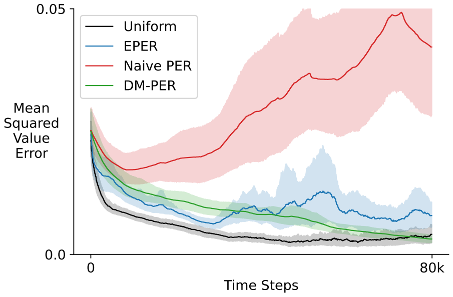

Figure 1: Prioritization can be problematic in noisy prediction with NNs. Results averaged over 30 trials; shaded region are 95% bootstrap Confidence Intervals (CI).

We start by describing a simplified prioritized replay algorithm, which we will call *Naive PER*. Starting from a uniform replay, the Naive PER algorithm modifies only the sampling strategy from uniform to proportional to TD error. We record the TD-error as soon as a sample is added to the buffer, then update that TD-error when a transition is sampled. We do not mix in uniform sampling, we do not squash the priorities with an exponential hyperparameter, and we do not use importance weights. In this way, the Naive PER algorithm closely resembles tabular prioritized sweeping, except we sample probabilistic according to the priorities rather than use a priority queue. The full pseudocode for Naive PER can be found in the appendix.

In order to study the effects of noise on the prioritization strategy, we introduce a new prioritization variant *expected* PER (EPER). Instead of using the sample TD-error, δ_t, which can be noisy when the reward or the transition dynamics are stochastic, EPER uses an estimate of the expected TD-error $\mathbb{E}\left[\delta_t \mid S_t = s\right]$. This expectation averages out random effects from action selection, transition dynamics, and the reward signal. In control, the expectation is conditioned on both state and action.

Learning this expectation can be formulated as a simple least-squares regression problem with samples δ_t as the target, yielding the following online update rule: $\theta_{t+1} \leftarrow \theta_t + \alpha(\delta_t - h_\theta(S_t))\nabla_\theta h_\theta(S_t)$, where h_θ is a parametric approximation of δ_t with parameters θ. This secondary estimator forms the basis of the gradient TD family of methods (Sutton et al., 2009; Patterson et al., 2022) making it natural to combine with recent gradient TD algorithms such as EQRC (Patterson et al., 2022).

In other words, if we use EQRC instead of DQN, we can use EPER to attain a less noisy signal for computing priorities with no extra work because EQRC is estimating h_θ anyway.

Figure 1 demonstrates the potential benefits of EPER over Naive PER. The task is to estimate the state-value function of a random policy in a Markov chain with the only reward on the terminal transition (described in more detail in the next section). The terminal reward is polluted by zero mean non-symmetric noise. The hyperparameters of all methods are systematically tuned, and still we see Naive PER is negatively impacted. The DM-PER algorithm is robust in this case, which is not surprising given its use of importance sampling and mixing in i.i.d samples. EPER is not as robust, but achieves this with a much simpler approach.

4 An Empirical Investigation of Prioritization in Replay

The idea of prioritized replay is based on the tabular notion of value propagation and the interplay between neural network generalization and prioritized replay remains an open question. This section explores the combined effect of prioritized replay and neural network generalization in RL agents.

4.1 Comparing Sample Efficiency in Prediction

In this section we ask several questions in a sparse reward task where rapid value propagation should require careful sampling from the replay buffer. Does naive prioritization improve performance over uniform replay? Do the additional tricks in DM-PER reduce the efficiency of value propagation when they are not really required? Finally, does robustness to noisy TD errors, as in EPER, matter in practice? We investigate these questions with tabular and neural network representations.

We consider both policy evaluation and control problems in a 50-state Markov chain environment visualized in Figure 2. This is an episodic environment with $\gamma = 0.99$ chosen to present a difficult value propagation problem. In every episode of interaction, the agent starts at the leftmost state and at each step takes the `left` or `right` action which moves it the corresponding neighbour state. The only reward in this environment is +1 when reaching the rightmost state at which point the episode terminates.

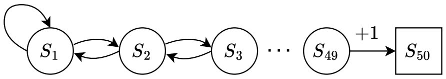

Figure 2: The 50-state Markov chain environment.

In the policy evaluation experiments, the objective is to estimate the state value function of the random policy. The data for the replay buffer is generated by running the random policy, making this an on-policy prediction task. The performance measure is the Mean Squared Value Error (MSVE) between estimated value function and true value function: $\text{MSVE}(\mathbf{w}) = \sum_s d(s)(v_\pi(s) - \hat{v}_{\mathbf{w}}(s))^2$ where $d(s)$ is the state visitation distribution under the uniform policy. In this experiment we have two settings, one where the value function is tabular and one where it is approximated by a two layer neural network with 32 hidden units in each layer and rectified linear unit (ReLU). We systematically tested a broad set of learning rates, buffer size, and batch sizes—over 50 combinations with 30 seeds each. The sensitivity to learning rate can be found in the Appendix. In Figure 3 we shown a representative result with batch size 8, buffer size 8000, and learning rate 8^{-4} in the tabular setting and 8^{-5} in the neural network setting. The remaining results are in the Appendix.

Figure 3 shows the learning curves of different replay methods for policy evaluation in the 50-state Markov chain over time. All three prioritized replay variants perform similarly and they are more sample efficient than uniform replay. The heatmaps show estimated values across states over time. Comparing the heatmap of tabular uniform replay with tabular Naive PER shows an increase in value propagation through the chain when using prioritization.

In the neural network setting, the error of Naive PER increases during early learning and then drops to the level of other prioritized replay methods. The gap between other prioritized methods (DM-PER and EPER) and uniform replay is smaller in the neural network setting compared with

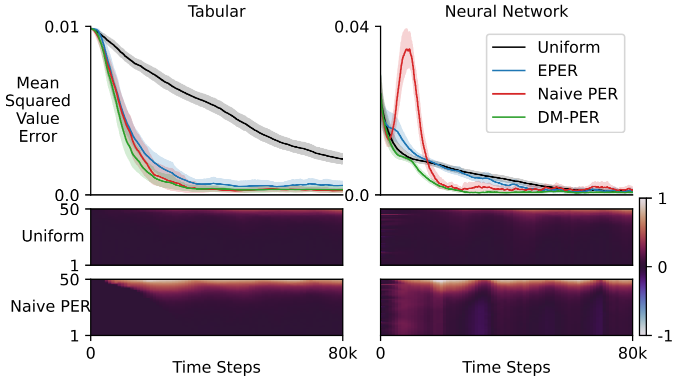

Figure 3: Prioritized methods can improve sample efficiency in prediction on the 50-state chain in tabular (left) and NN prediction (right). With NN function approximation Naive PER exhibits an increase in MSVE during early learning. The heatmaps show estimated values of the states, 1 to 50, over time. Results are averaged over 30 seeds; shaded regions are 95% bootstrap CI.

the tabular setting. Additionally, these two prioritized methods do not exhibit an increase in the MSVE like Naive PER.

Perhaps Naive PER over-samples a few transitions which causes the network to spend a lot of its capacity minimizing the error of those transitions at the cost of a worse prediction in other states. It is possible that EPER can mitigate the over-sampling issue because the initial estimates are randomized which helps avoid over-sampling certain transitions. DM-PER reduces the negative effect of over-sampling by using importance sampling weights to reduce the magnitude of updates with high priority transitions.

To better understand what is happening, we visualize the probability of updating a state over time in Figure 4. We calculate this probability by summing over the probabilities of sampling a transition starting from a given state at a given time based on transitions in the replay buffer. The heatmap presents these probabilities, computed every 1000 states over one run. We use the same hyperparameter settings as in Figure 3.

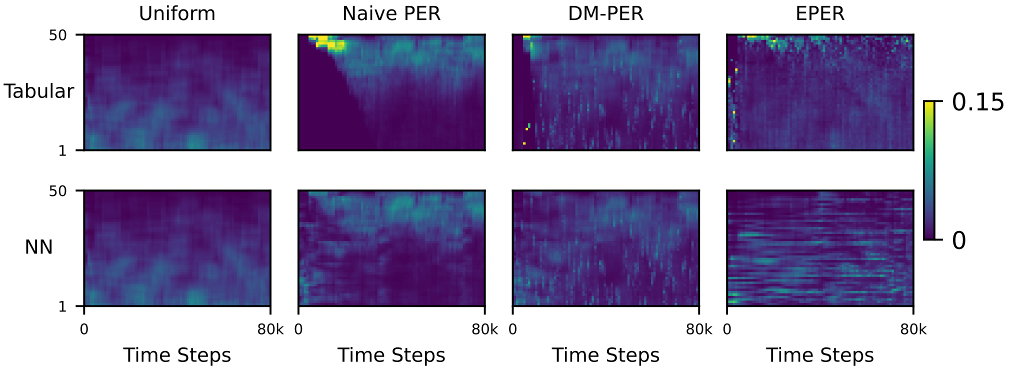

Figure 4: Probability of sampling a transition starting from each state (1 to 50) from the buffer at each time point, in the 50-state Markov chain for one run.

The sampling probabilities under uniform replay reflect the state visitation of the random policy, putting a higher probability mass on earlier states. The sampling distribution of tabular Naive PER follows the intuition from prioritized sweeping by putting most of the probability mass on the rewarding transition at the end of chain, then, increasing the probability of nearby states in a backward fashion to help value propagation. Under neural network function approximation the pattern is similar but more uniform. This is caused by the random initialization of network parameters which generates non-zero TD errors across the state space. The sampling distribution of both DM-PER and EPER are, on the other hand, more structured. Both feature non-terminal transitions with high probability (bright spots) and striping. It is hard to speculate why this occurs, nevertheless, these patterns provide evidence that combining prioritization with NNs can result distributions very different from the tabular case.

4.2 Overestimation due to Prioritization, Generalization, and Bootstrapping

In the previous section we saw that Naive PER exhibited a spike in early learning, but why? One possible explanation is that Naive PER is oversampling the terminal transition which causes the NN to inappropriately over-estimate nearby states, causing more oversampling, and so on, spreading across all states. The heatmap in Figure 4 provides some evidence of this. One way to prevent over-generalization is to employ target networks to reduce the effect of bootstrapping. Note that the task is on-policy prediction, and we do not need a target network to stabilize learning. We use the same setup as in figure 3 and only consider Naive PER with neural nets.

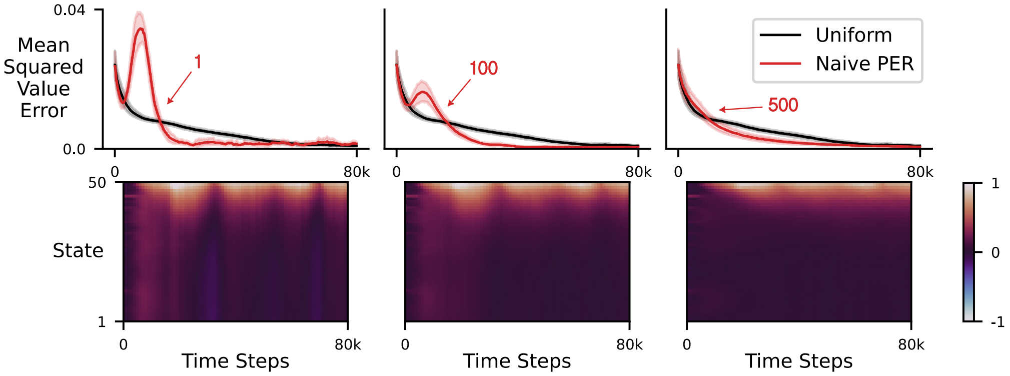

Figure 5: Target Networks can mitigate Naive PER's poor performance in the 50-state Markov chain prediction task with NNs. Red numbers above curves indicate Target Network update rate.

The results in Figure 5 show the performance of Naive PER with three different target network update rates (1, 100, 500). An update rate of 1 is identical to not using target networks at all. As we update the target network less frequently we see the spike in the learning curves is reduced. Notice that heatmap for the value function with an update rate of 500 is very similar to Naive PER in the tabular case (see Figure 3). We only see a minor performance improvement over uniform replay in Figure 5, but this is expected because updating the target network infrequently is reducing the update rule's ability to propagate value backwards via bootstrapping.

4.3 Comparing Sample Efficiency in Control

In this section we turn our attention to a simple control task, again designed in such a way that value propagation via smart sampling should be key. Here our main question is: do the insights about the benefits of prioritization persist when the policy changes and exploration is required.

In the tabular setting, we use Q-learning (without target networks) and in neural network setting we explore two setups: (1) DQN (with target refresh rate of 100) and (2) EQRC (as an alternative method without target network). We report steps to goal as the performance metric for the 50-state Markov chain problem. Buffer size is fixed to 10000, batch size 64, and we pick the best learning rate for each method (see the Appendix for details). Each control agent is run for 100000 steps with an ϵ-greedy policy with $\epsilon = 0.1$.

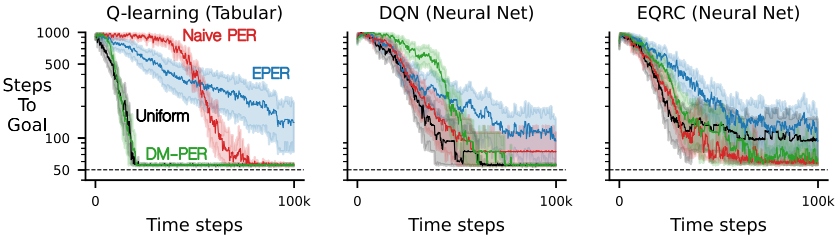

Figure 6: Prioritization is not more sample efficient than uniform for control in the 50-state Markov chain environment. Results averaged over 50 seeds; shaded regions are 95% bootstrap CI.

The results in Figure 6 are somewhat unexpected. The dotted line depicts the performance of the optimal policy. Even in the tabular case, Q-learning with uniform replay is a better than all the three prioritized methods. DM-PER performs just as well as uniform replay, but this could be explained by the fact that DM-PER's sampling is closer to uniform compared with the other prioritization schemes as shown previous in Figure 4. Naive PER eventually reaches the near optimal policy and EPER performs poorly. Under neural network approximation, all tested algorithms have a wide overlapping confidence regions, even with 50 seeds, making differences between methods non conclusive. It seems good performance in prediction does not necessarily translate into improvement in control, even in the same MDP.

4.4 Sampling Without Replacement & Updating Priorities

In this section we explore two simple but natural improvements to replay that could improve performance. There are many possible refinements, and many have been explored in the literature already. Here we select two that have not been deeply explored before, specifically (1) sampling transitions with or without replacement, and (2) recomputing priorities of samples in the buffer.

When sampling a mini-batch from the replay buffer, one has the option to sample transitions with or without replacement. This decision is important in PER because sampling with replacement can cause a high priority transition to be repeatedly sampled into the same mini-batch. This certainly happens on the first visit to the goal state in the 50 state chain. Uniform replay avoids this problem by design. Most reference implementations of PER sample with replacement. We hypothesize that duplicate transitions in the mini-batch reduces the sample efficiency of prioritized methods, effectively nagating the benefit of mini-batches.

We compare Naive PER with and without replacement sampling and uniform replay in the 50-state Markov chain prediction domain under tabular and neural net function approximation. We used a two layer network with 32 hidden units and ReLU activation, a batch size of 8000 and experimented with several mini-batch sizes (1, 8, 64, 256). With batch size 1, with and without replacement are identical. We report a representative result with learning rate of 8^{-4} in the tabular setting and 8^{-5} in the neural network setting and report MSVE under the target policy over training time.

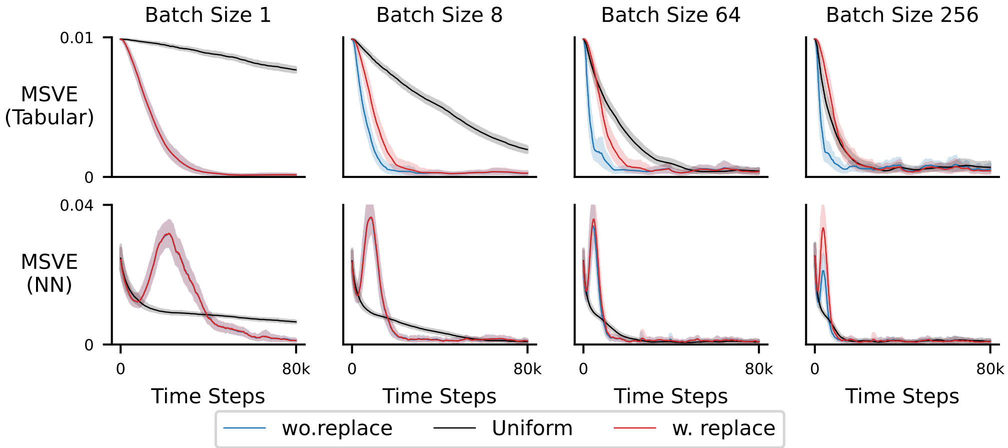

Figure 7: Sampling without replacement improves the performance of Naive PER in the tabular setting but not with neural nets. Results are averaged over 50 seeds and shaded regions are 95% bootstrap CI.

Figure 7 shows that sampling without replacement provides a minor improvement on the performance of Naive PER in tabular prediction, where Naive PER was already working well, but does not help when combined with NN function approximation. In fact, we again see Naive PER's characteristic spike due to over-generalization and bootstrapping. This poor performance is somewhat mitigated by larger batch sizes, but still uniform replay is better. Note, as expected, the performance of uniform replay suffers with smaller batch sizes.

Now we turn to the control setting to evaluate the impact of sampling without replacement. We tested tabular Q-learning and neural network DQN settings. The DQN agent has a two layer network with 32 hidden units and ReLU activation with target refresh rate 100. All agents have buffer size 10000 and a series of batch sizes similar to the previous experiment. The learning rate of each agent is selected by sweeping over a range of step sizes and maximizing over average performance (sweep details in the Appendix). Figure 8 summarizes the results.

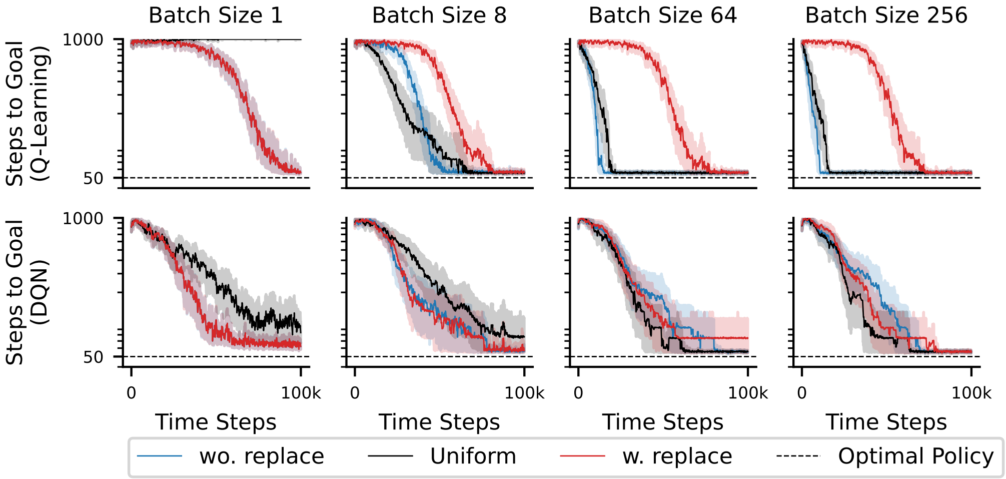

Figure 8: Sampling with and without replacement in control using Naive PER with tabular and NN representations. Without replacement sampling only helps in the tabular setting. Results averaged over 50 seeds; shaded regions are 95% bootstrap CI.

In tabular control we see a significant improvement in Naive PER when sampling without replacement, whereas with function approximation the result is less clear. In tabular, the gap in performance between with and without replacement steadily increases and eventually Naive PER becomes nearly statistically better than uniform. With DQN (function approximation), larger batch sizes mostly result in ties, though Naive PER without replacement is the only method to always reach optimal on average.

Taken together, the results which use sampling without replacement suggest a minor benefit. It always helps in the tabular case, at times outperforming uniform replay, and with function approximation it mostly does not hurt performance.

Another factor that can potentially limit the benefit of prioritization is non-informative and outdated priorities in the buffer. The priority of a transition is updated only when the transition is sampled. This means that at any given time the priority of almost all items in the replay buffer are outdated with respect to the current value function. We can update the priority of all transitions in the buffer by recomputing their TD error using the current value function estimate periodically.

We tested this idea in prediction setting in the 50 state chain. We compared the performance of Naive PER, EPER, and DM-PER recomputing the priorities every 10 and 1000 steps. Again we looked at tabular and NN representations with a two layer neural net of size 32 with ReLU activation for the latter. The buffer size is fixed to 8000, batch size to 8, and learning rate to 8^{-4} for tabular and 8^{-5} for neural net agents. Figure 9 summarizes the results. In short, we see no benefit from recomputing priorities in the function approximation settings and marginal benefit in the tabular case with Naive PER. Interestingly, for DM-PER recomputing too often, every 1000 steps vs every 10 steps, hurts compared to the default—updating only when a transition is first added or resampled. Note the over-generalization of Naive PER with neural nets is also not reduced more by up-to-date priorities.

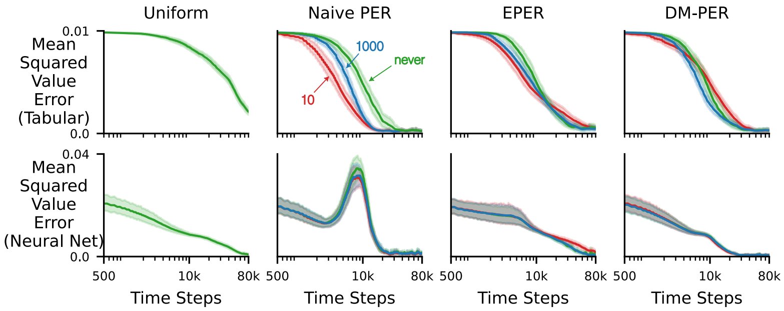

Figure 9: Recomputing priorities in chain prediction using Naive PER, EPER, and DM-PER with tabular (top) and NN (bottom) representations. Generally, recomputing does not help. Results averaged over 30 seeds; shaded regions are 95% bootstrap CI.

As a final experiment in the chain problem, we investigate if combining sampling without replacement and recomputing priorities every 10 steps, together, can improve the performance of Naive PER. We conduct this experiment in the control chain problem and repeat the experiment for tabular Q-learning and DQN with a two layer neural net with 32 hidden units and ReLU activation with target refresh rate of 100. The buffer size is fixed to 10000 and batch size is 64, we select the learning rate over a range of values that attained the best average performance. As we see in Figure 10, in the tabular case, Naive PER with both modifications achieves the best performance, but barely more than either modification in isolation. Unfortunately, as expected, there is no clear benefit in the function approximation setting.

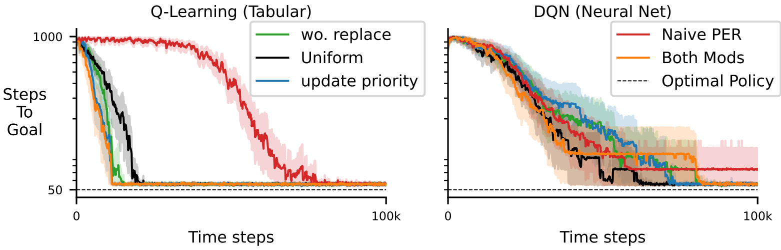

Figure 10: Combining recomputing priorities and without replacement sampling for tabular (left) and NN (right) control in the chain. Results averaged over 50 seeds; shaded regions are 95% bootstrap CI.

4.5 Comparing Sample Efficiency in Classic Control Domains

Our previous experiments in the chain were designed to represent an idealized problem to highlight the benefits of smarter replay, and in this section we consider slightly more complex, less ideal tasks. In the chain, we only saw clear advantages for prioritization in prediction and also in control with small batch sizes. The main question here is do these benefits persist or perhaps prioritization will be worse supporting the common preference for uniform replay in deep RL.

We consider four episodic environments which are significantly more complex than the chain, but are small enough that smaller NNs can be used and extensive experimentation is possible. The first three environments, often refered to as classic control feature low dimensional continuous state and discrete actions. MountainCar (Moore, 1990) and Acrobot (Sutton, 1995) are two control tasks where the goal is to manipulate a physical system to get to a goal at the end of a long trajectory. We also include Cartpole due to the unstable dynamics of the balanced position (Barto et al., 1983). Finally we include the tabular Cliffworld (Sutton & Barto, 2018) because the reward for falling off the cliff is a large negative value which causes rare but large spikes in the TD error, which might showcase the benefit of EPER. The details about these environments can be found in the Appendix. We set the discount factor $\gamma = 0.99$. The episodes in MountainCar, Acrobot, and Cartpole are cutoff every 500 steps, but there is no episode cutoff in Cliffworld.

In this experiment we use DQN with a two layer network of size 64 with ReLU activation and target refresh rate 128. Batch size and buffer size are fixed to 64 and 10000 respectively and the learning rate is selected using a two stage approach to avoid maximization bias (Patterson et al., 2023). First each agent is run for 30 seeds sweeping over many learning rate parameter settings, then the hyperparameter which achieved the best average performance is run for 100 new seeds (see details in the Appendix). We include *Modified PER*, which combines Naive PER with without-replacement sampling and recomputes the priorities every 10 steps. Figure 11 summarizes the results. Unsurprisingly, prioritization does not improve the sample efficiency over uniform replay in any of the four domains.

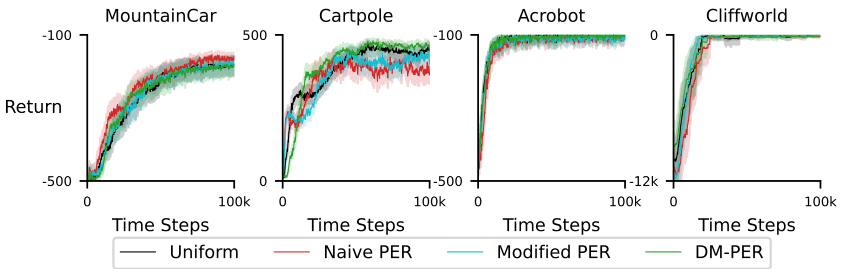

Figure 11: Performance of DQN replay agents on classic control problems. No clear benefit for prioritization. Results averaged over 100 seeds; shaded regions are 95% bootstrap CI.

Looking closely at the learning curve for Cliffworld in Figure 11 we see a small blip in the performance with uniform replay. Recall, we suspected that EPER might show benefit in this MDP due to outlier rewards when the agent falls off the cliff. Average learning curves can hide the stucture of individual runs, so we plotted all the runs individually for each method in Figure 12. Here we see DQN with uniform replay periodically performs quite poorly, even late in learning. This is true to a less extent for DM-PER, Naive PER, and Modified PER. Interestingly, Naive PER variants based on EPER appear substantially more stable with less collapses in performance.

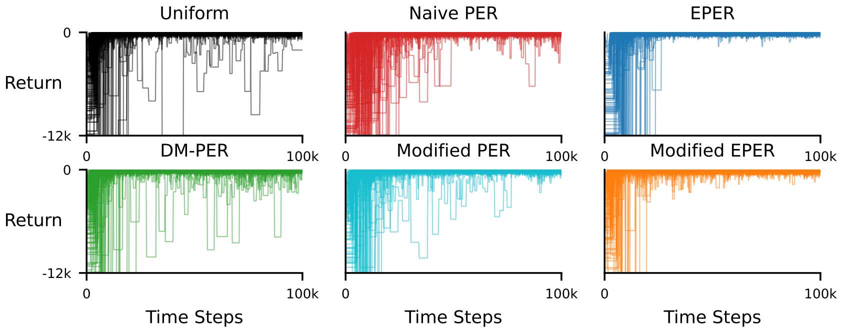

Figure 12: Performance of 100 individual runs in the Cliffworld shows performance dips using uniform replay, Naive PER, and DM-PER. EPER-based methods appear to have more stable performance.

5 Conclusion

In this paper, we conducted a series of carefully designed experiments with prioritized replay under tabular and neural network settings. We found that prioritization combined with non-linear generalization can overestimate values during early learning. It appears that a combination of bootstrapping and neural network generalization is the reason behind this overestimation. Furthermore, we showed in a simple chain domain, several variants of PER outperform i.i.d replay in the prediction setting but have poor sample efficiency in control. Unsurprisingly, no variant of PER improves upon i.i.d replay in classic control domains.

We introduced EPER as a simple modification prioritizing transitions according to a learned estimate of the expected error inspired by gradient TD methods. We showed that EPER can be more robust in noisy reward domains and perform more reliably than PER or i.i.d replay in Cliffworld. Finally, we explored two design decisions in PER, recomputing outdated priorities and sampling batches without replacement, discovering that these additions can improve PER in the tabular setting but have little to no effect when using neural networks.

References

Nishanth Anand and Doina Precup. Prediction and control in continual reinforcement learning. *Advances in Neural Information Processing Systems*, 36, 2024.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight Experience Replay. 2018.

Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.

William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting Fundamentals of Experience Replay. 2020.

Yuwei Fu, Di Wu, and Benoit Boulet. Benchmarking sample selection strategies for batch reinforcement learning, 2022.

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. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Zhang-Wei Hong, Tao Chen, Yen-Chen Lin, Joni Pajarinen, and Pulkit Agrawal. Topological Experience Replay, 2023.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

Hideyoshi Igata, Yuji Ikegaya, and Takuya Sasaki. Prioritized experience replays on a hippocampal predictive map for learning. 118(1):e2011266118, 2021.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, San Diega, CA, USA, 2015.

Taisuke Kobayashi. Revisiting experience replayable conditions. *arXiv preprint arXiv:2402.10374*, 2024.

Ramnath Kumar and Dheeraj Nagaraj. Introspective Experience Replay: Look Back When Surprised, 2023.

Su Young Lee, Choi Sungik, and Sae-Young Chung. Sample-Efficient Deep Reinforcement Learning via Episodic Backward Update. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Ang A Li, Zongqing Lu, and Chenglin Miao. Revisiting prioritized experience replay: A value perspective. *arXiv preprint arXiv:2102.03261*, 2021.

Min Li, Tianyi Huang, and William Zhu. Clustering experience replay for the effective exploitation in reinforcement learning. *Pattern Recognition*, 131:108875, 2022.

Long-Ji Lin. Programming robots using reinforcement learning and teaching. In *Proceedings of the Ninth National Conference on Artificial Intelligence - Volume 2*, AAAI'91, pp. 781–786. AAAI Press, 1991.

Cong Lu, Philip Ball, Yee Whye Teh, and Jack Parker-Holder. Synthetic experience replay. *Advances in Neural Information Processing Systems*, 36, 2024.

Guozheng Ma, Lu Li, Sen Zhang, Zixuan Liu, Zhen Wang, Yixin Chen, Li Shen, Xueqian Wang, and Dacheng Tao. Revisiting Plasticity in Visual Reinforcement Learning: Data, Modules and Training Stages, 2023.

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

Andrew W. Moore and Christopher G. Atkeson. Prioritized Sweeping: Reinforcement Learning with Less Data and Less Time. 13(1):103–130, 1993. ISSN 0885-6125.

Andrew William Moore. Efficient memory-based learning for robot control. Technical report, University of Cambridge, 1990.

Yangchen Pan, Muhammad Zaheer, Adam White, Andrew Patterson, and Martha White. Organizing Experience: A Deeper Look at Replay Mechanisms for Sample-based Planning in Continuous State Domains, 2018.

Andrew Patterson, Adam White, and Martha White. A generalized projected bellman error for off-policy value estimation in reinforcement learning. 23(1):145:6463–145:6523, 2022. ISSN 1532-4435.

Andrew Patterson, Samuel Neumann, Martha White, and Adam White. Empirical Design in Reinforcement Learning, 2023.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal Value Function Approximators. In *Proceedings of the 32nd International Conference on Machine Learning*, pp. 1312–1320. PMLR, 2015.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized Experience Replay. 2016.

Peiquan Sun, Wengang Zhou, and Houqiang Li. Attentive Experience Replay. 34(04):5900–5907, 2020. ISSN 2374-3468.

Richard Sutton, Hamid Maei, Doina Precup, Shalabh Bhatnagar, David Silver, Csaba Szepesvári, and Eric Wiewiora. Fast gradient-descent methods for temporal-difference learning with linear function approximation. volume 382, pp. 125, 2009.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In D. Touretzky, M.C. Mozer, and M. Hasselmo (eds.), *Advances in Neural Information Processing Systems*, volume 8. MIT Press, 1995.

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

Hado P Van Hasselt, Matteo Hessel, and John Aslanides. When to use parametric models in reinforcement learning? *Advances in Neural Information Processing Systems*, 32, 2019.

Han Wang, Erfan Miahi, Martha White, Marlos C Machado, Zaheer Abbas, Raksha Kumaraswamy, Vincent Liu, and Adam White. Investigating the properties of neural network representations in reinforcement learning. *Artificial Intelligence*, pp. 104100, 2024.

Lennart Wittkuhn, Samson Chien, Sam Hall-McMaster, and Nicolas W. Schuck. Replay in minds and machines. 129:367–388, 2021. ISSN 0149-7634.

A Algorithms

Here we present pseudo code for the DQN algorithm used in this paper. The behavior policy of DQN, denoted as $\pi_{\mathbf{w}}(S_t)$, is an ϵ-greedy policy over the current action values $q(S_t, .)$. Algorithm 1 shows the uniform replay variant and algorithm 2 shows the pseudo code for Naive PER. In both variants, size of the replay buffer is a hyperparameter.

Algorithm 1 DQN with Uniform Replay

```
Input: mini-batch size $b$, learning-rate $\alpha$, training time $T$, target refresh rate $\tau$.
Initialize: $q$ network parameters $\mathbf{w}$, target network parameters $\mathbf{w}_{\text{target}} = \mathbf{w}$, buffer $B$, $\Delta = 0$.
Observe $S_0$ and choose $A_0 \sim \pi_{\mathbf{w}}(S_0)$
for $t = 1$ to $T$ do
  Observe $R_t, S_t, \gamma_t$
  Store transition $(S_{t-1}, A_{t-1}, R_t, S_t, \gamma_t)$ in buffer $B$
  for $j = 1$ to $b$ do
    Sample transition $(S_j, A_j, R_j, S_{j+1}, \gamma_{j+1})$ from buffer $B$ with probability $1/|B|$
    Compute TD-error $\delta_j = R_j + \gamma_j \max_a q(S_{j+1}, a, \mathbf{w}_{\text{target}}) - q(S_j, A_j, \mathbf{w})$
    Accumulate gradient $\Delta \leftarrow \Delta - \delta_j \nabla_{\mathbf{w}} q(S_j, A_j, \mathbf{w})$
  end for
  Update $\mathbf{w} \leftarrow \text{adam}(\mathbf{w}, \frac{\Delta}{b}, \alpha)$; Reset $\Delta = 0$
  if $t\%\tau = 0$ then
    Refresh target network $\mathbf{w}_{\text{target}} \leftarrow \mathbf{w}$
  end if
  Choose action $A_t \sim \pi_{\mathbf{w}}(S_t)$
end for
```

Algorithm 2 DQN with Naive PER

```
Input: mini-batch size $b$, learning-rate $\alpha$, training time $T$, target refresh rate $\tau$.
Initialize: $q$ network parameters $\mathbf{w}$, target network parameters $\mathbf{w}_{\text{target}} = \mathbf{w}$, prioritized buffer $B$, $\Delta = 0$.
Observe $S_0$ and choose $A_0 \sim \pi_{\mathbf{w}}(S_0)$
for $t = 1$ to $T$ do
  Observe $R_t, S_t, \gamma_t$
  Store transition $(S_{t-1}, A_{t-1}, R_t, S_t, \gamma_t)$ in buffer $B$ with priority $p_{t-1} = |\delta_{t-1}| = |R_t + \gamma_t \max_a q(S_t, a, \mathbf{w}_{\text{target}}) - q(S_{t-1}, A_{t-1}, \mathbf{w})|$
  for $j = 1$ to $b$ do
    Sample transition $(S_j, A_j, R_j, S_{j+1}, \gamma_{j+1})$ from buffer $B$ with probability $\frac{p_j}{\sum_i p_i}$
    Compute TD-error $\delta_j = R_j + \gamma_j \max_a q(S_{j+1}, a, \mathbf{w}_{\text{target}}) - q(S_j, A_j, \mathbf{w})$
    Accumulate gradient $\Delta \leftarrow \Delta - \delta_j \nabla_{\mathbf{w}} q(S_j, A_j, \mathbf{w})$
  end for
  for $j = 1$ to $b$ do
    Update priority $p_j = |\delta_j|$
  end for
  Update $\mathbf{w} \leftarrow \text{adam}(\mathbf{w}, \frac{\Delta}{b}, \alpha)$; Reset $\Delta = 0$
  if $t\%\tau = 0$ then
    Refresh target network $\mathbf{w}_{\text{target}} \leftarrow \mathbf{w}$
  end if
  Choose action $A_t \sim \pi_{\mathbf{w}}(S_t)$
end for
```

B DM-PER Hyperparameters

In all the experiments with DM-PER, its additional hyperparameters are set to the values from (Schaul et al., 2016) and are listed in table 1.

C Prediction Experiments in the Chain

In This section, we document the hyperparameter sweep details and additional results from the Markov chain prediction experiment. Table 2 lists the hyperparameter selections for the prediction

Priority exponent	Importance sampling exponent	i.i.d mix-in ratio
0.6	$0.4 \to 1.0$	10^{-3}

Table 1: Hyperparameters specific to DM-PER. Arrow indicates linear schedule over training time.

agents. The agents in figure 3 use buffer size 8000, batch size 8, and learning rate 8^{-4} for tabular and 8^{-5} for neural network agents.

	Tabular agents	Neural network agents
Learning rate	$[8^{-6}, 8^{-5}, 8^{-4}, 8^{-3}, 8^{-2}]$	$[8^{-6}, 8^{-5}/4, 8^{-5}, 8^{-4}/4, 8^{-4}, 8^{-3}/4, 8^{-3}]$
Adam optimizer β_1	0.9	0.9
Adam optimizer β_2	0.999	0.999
Batch size	$[1, 8, 64]$	$[1, 8, 64]$
Buffer size	$[800, 8000, 80000]$	$[800, 8000, 80000]$
Network size	-	2×32 network with ReLU activation
Training time	80000	80000

Table 2: Hyperparameters of prediction agents in Markov chain

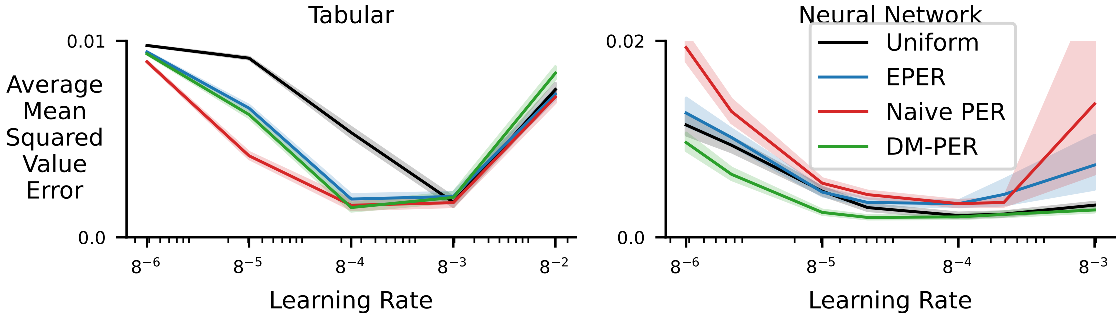

Figure 13: Sensitivity to learning rate in prediction chain task. Results are averaged over 30 seeds and shaded region is 95% bootstrap CI.

Figure 13 shows the sensitivity of replay methods to learning rate for batch size 8 and buffer size 8000 in the chain prediction problem. Prioritized replay is more sample efficient than uniform replay in the tabular setting, especially with smaller step sizes. But when using neural networks, the early increase in MSVE of Naive PER, reduces its average performance below other algorithms.

Now we show additional results for those meta-parameter choices in the chain prediction domain that are omitted from the main text. Figures 14 and 15 show the learning curves for tabular and neural network agents respectively. The learning rate is tuned by maximizing over average performance across 30 seeds.

As part of our investigation into PER, we experimented with sampling mini-batches without replacement (see figure 7). Here we present results for all PER variants, Naive PER, DM-PER, and EPER in the aforementioned prediction experiment. The hyperparameters of this experiment are chosen according to table 2 sweeping over a range of batch sizes $[1, 8, 64, 256]$. We chose the same learning rate as in figure 3, namely, 8^{-4} for tabular agents and 8^{-5} for neural network agents. Figure 16 shows tabular agents and figure 17 shows the neural network agents MSVE over training time.

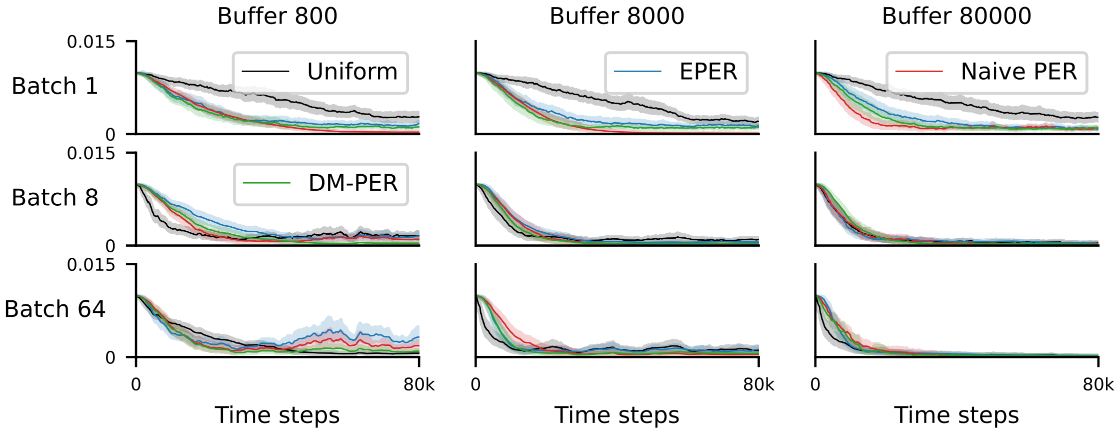

Figure 14: Performance of tabular replay agents in the prediction chain task. Results are averaged over 30 seeds; shaded region is 95% bootstrap CI.

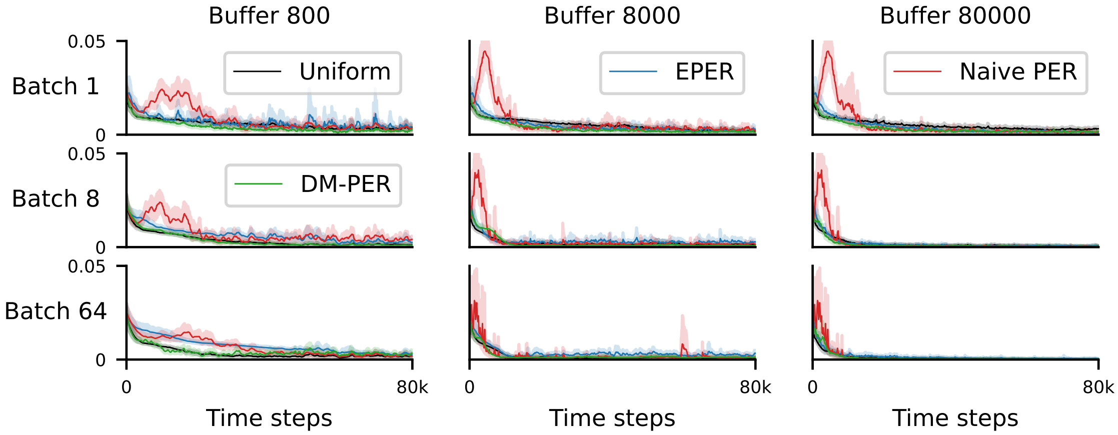

Figure 15: Performance of neural network replay agents in the prediction chain task. Results are averaged over 30 seeds; shaded region is 95% bootstrap CI.

D Control Markov Chain Experiment Details

The hyperparameters used in control chain experiments are given in table 3. We chose the learning rate for each agent by maximizing over average performance across a range of learning rates.

In the without replacement control experiment, we use a buffer size of 10000 and experiment with 4 different batch sizes $[1, 8, 64, 256]$. For each setting, the learning-rate is selected via maximizing over a range of learning-rates. All other meta-parameters are based on table 3. Figure 18 shows results for the tabular Q-learning setting, figure 19 shows the DQN results, and figure 20 shows the EQRC results.

E Classic Control Experiment Details

E.1 Environment Description

In the last part of the paper, we experiment with classic control problems that are more difficult than the chain. We experiment with MountainCar, Acrobot, Cartpole, and Cliffworld. In MountainCar,

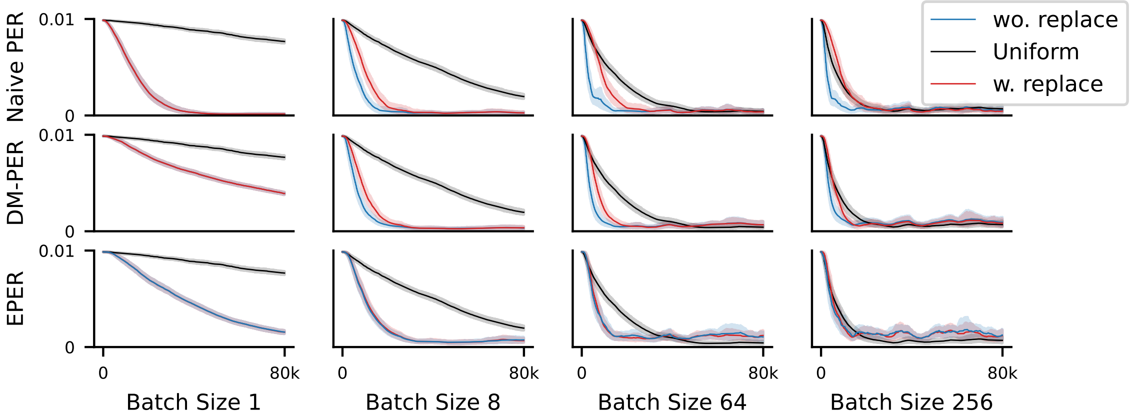

Figure 16: Sampling without replacement improves performance in the tabular prediction chain problem for Naive PER and DM-PER. Results averaged over 50 seeds with 95% bootstrap CI.

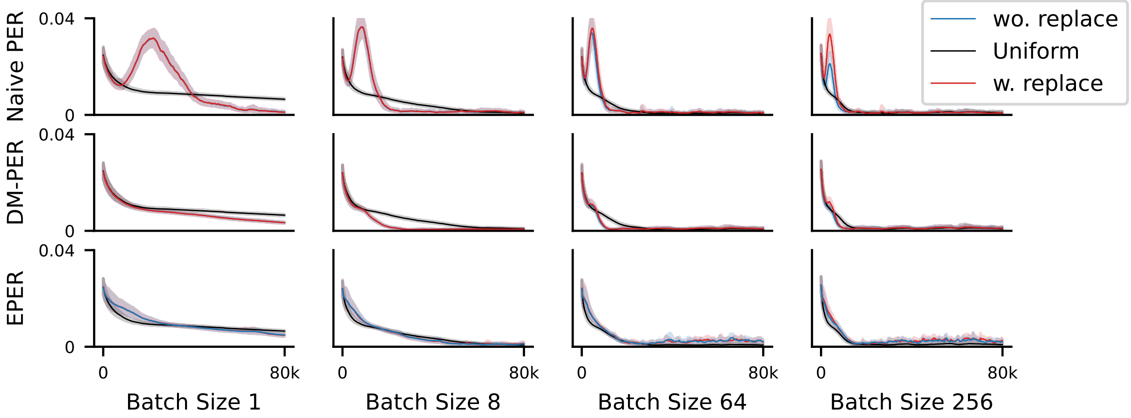

Figure 17: Sampling without replacement does not improve performance in the prediction chain problem under neural network function approximation. Results averaged over 50 seeds with 95% bootstrap CI.

the goal is to drive an under powered car up a hill in a simulated environment with simplified physics by taking one of three actions, accelerate left, accelerate right, do not accelerate. The observations are position and speed values of the car. The reward is -1 per step and episodes terminates when the car crosses a threshold at the top of hill with reward 0.

	Q-Learning agents (tabular)	DQN and EQRC agents (neural network)
Learning rate	$[8^{-7}, 8^{-6}, 8^{-5}, 8^{-4}, 8^{-3}, 8^{-2}]$	$[8^{-5}, 8^{-4}, 8^{-3}, 8^{-2}, 8^{-1}]$
Adam optimizer β_1	0.9	0.9
Adam optimizer β_2	0.999	0.999
Batch size	8	8
Buffer size	10000	10000
Network size	-	2 × 32 network with ReLU activation
Target refresh	-	100 (only DQN)
Exploration ϵ	0.1	0.1
Training time	100000	100000

Table 3: Hyperparameters of control agents in Markov chain

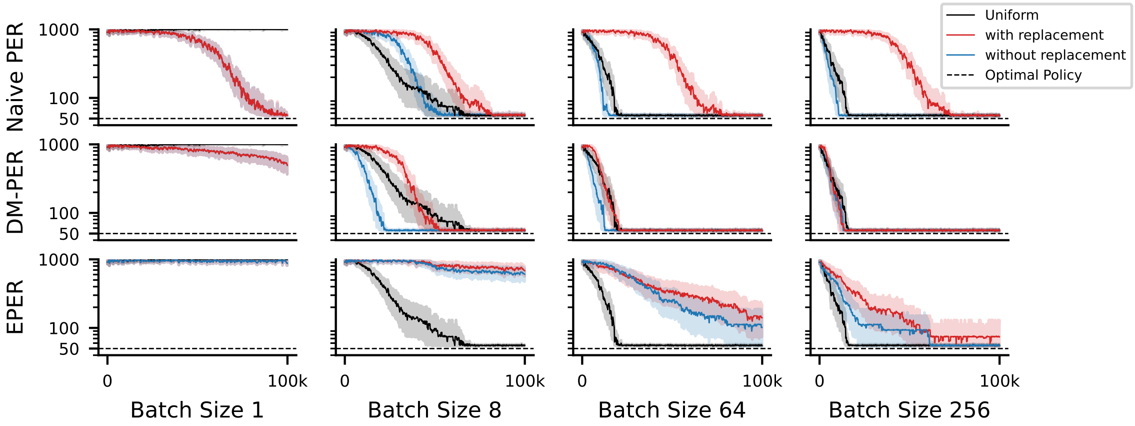

Figure 18: Sampling without replacement improves performance in the control chain task when using tabular Q-learning. Results are averaged over 50 seeds; shaded region is 95% boostrap CI.

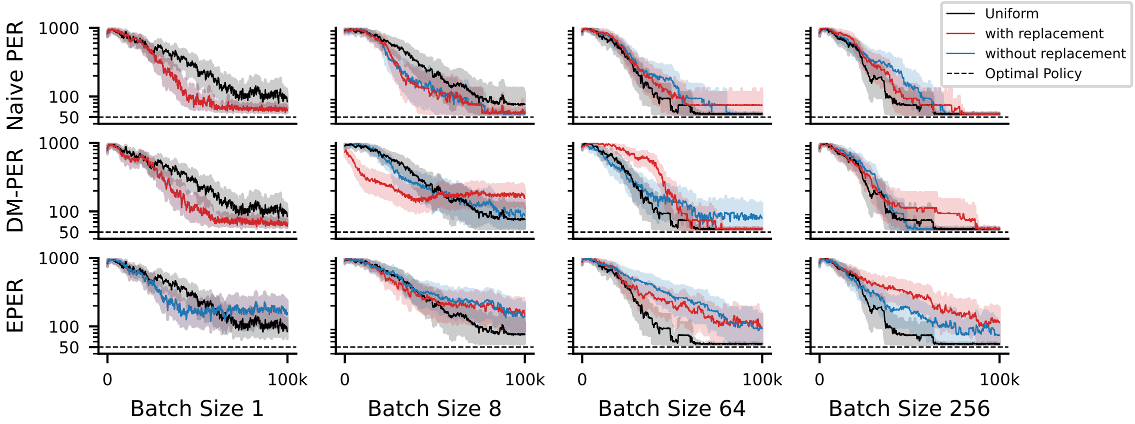

Figure 19: Sampling without replacement does not improve performance in control chain task when using DQN. Results are averaged over 50 seeds; shaded region is 95% boostrap CI.

In Acrobot the agent controls a system of two linear links connected by a movable joint. The goal is to move the links, by applying torque to the joint, such that the bottom part of the link rises to the level of its highest point upon which the episode terminates with reward 0. The reward of all other transitions is -1 per step.

The goal of a Cartpole agent is to balance a pole on top of a moving cart by accelerating the cart to either left or right. The reward is +1 per step if the pole is properly balanced. If the pole falls more than 12 degrees the episodes is terminated and the pole is reset to its upright position. The episode cutoff length is 500.

Cliffworld is a gridworld where agents start at a fixed state and pick any of cardinal directions and move to corresponding neighbor state. The goal is to reach to the final state on the opposite side of the starting state while avoiding a cliff near the optimal path. The reward is -1 per step except when falling off the cliff that gives -100 reward upon which the agent is reset back to start (without episode termination).

E.2 Hyperparameter selection

In the classic control experiments we use the hyperparameters from table 4 and tune the step size using the two stage hyperparameter selection method (Patterson et al., 2023). For each agent we

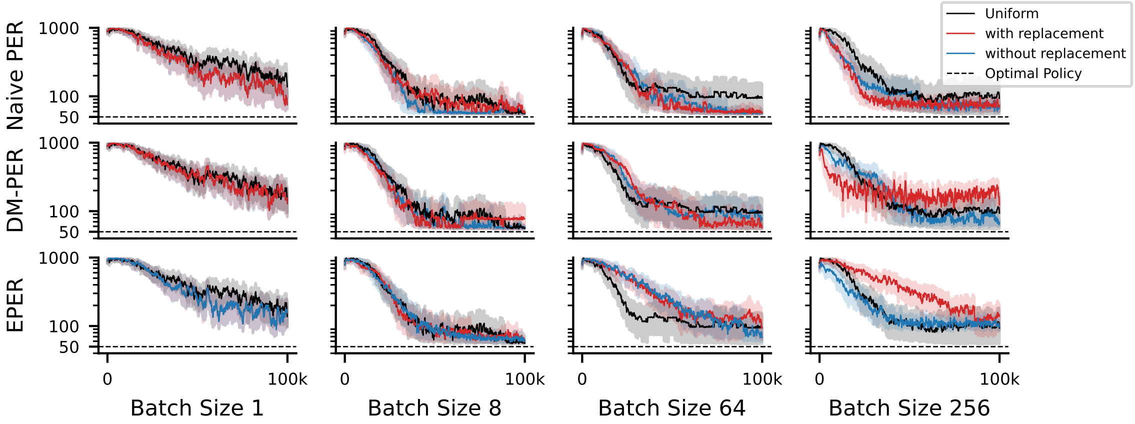

Figure 20: Sampling without replacement does not improve performance in the control chain task when using EQRC. Results are averaged over 50 seeds; shaded region is 95% boostrap CI.

run all learning rates for 30 seeds, selecting the value with maximum average performance, then running the tuned agent for 100 new seeds to avoid maximization bias.

	DQN agents
Learning rate	$[4^{-8}, 4^{-7}, 4^{-6}, 4^{-5}, 4^{-4}, 4^{-3}, 4^{-2}]$
Adam optimizer β_1	0.9
Adam optimizer β_2	0.999
Batch size	64
Buffer size	10000
Network size	2 × 64 dense network with ReLU activation
Target refresh	128
Exploration ϵ	0.1
Training time	100000

Table 4: Hyperparameters of classic control experiments