

Prioritized Experience Replay

Tom Schaul, John Quan, Ioannis Antonoglou and David Silver
Google DeepMind

{schaul,johnquan,ioannisa,davidsilver}@google.com

Abstract

Experience replay lets online reinforcement learning agents remember and reuse experiences from the past. In prior work, experience transitions were uniformly sampled from a replay memory. However, this approach simply replays transitions at the same frequency that they were originally experienced, regardless of their significance. In this paper we develop a framework for prioritizing experience, so as to replay important transitions more frequently, and therefore learn more efficiently. We use prioritized experience replay in Deep Q-Networks (DQN), a reinforcement learning algorithm that achieved human-level performance across many Atari games. DQN with prioritized experience replay achieves a new state-of-the-art, outperforming DQN with uniform replay on 41 out of 49 games.

1 Introduction

Online reinforcement learning (RL) agents incrementally update their parameters (of the policy, value function or model) while they observe a stream of experience. In their simplest form, they discard incoming data immediately, after a single update. Two issues with this are (a) strongly correlated updates that break the i.i.d. assumption of many popular stochastic gradient-based algorithms, and (b) the rapid forgetting of possibly rare experiences that would be useful later on.

Experience replay (Lin, 1992) addresses both of these issues: with experience stored in a replay memory, it becomes possible to break the temporal correlations by mixing more and less recent experience for the updates, and rare experience will be used for more than just a single update. This was demonstrated in the Deep Q-Network (DQN) algorithm (Mnih et al., 2013; 2015), which stabilized the training of a value function, represented by a deep neural network, by using experience replay. Specifically, DQN used a large sliding window replay memory, sampled from it uniformly at random, and

revisited each transition¹

¹ A transition is the atomic unit of interaction in RL, in our case a tuple of (state S_{t-1} , action A_{t-1} , reward R_t , discount γ_t , next state S_t). We choose this for simplicity, but most of the arguments in this paper also hold for a coarser ways of chunking experience, e.g. into sequences or episodes.

eight times on average. In general, experience replay can reduce the amount of experience required to learn, and replace it with more computation and more memory – which are often cheaper resources than the RL agent’s interactions with its environment.

In this paper, we investigate how *prioritizing* which transitions are replayed can make experience replay more efficient and effective than if all transitions are replayed uniformly. The key idea is that an RL agent can learn more effectively from some transitions than from others. Transitions may be more or less surprising, redundant, or task-relevant. Some transitions may not be immediately useful to the agent, but might become so when the agent competence increases (Schmidhuber, 1991). Experience replay liberates online learning agents from processing transitions in the exact order they are experienced. Prioritized replay further liberates agents from considering transitions with the same frequency that they are experienced.

In particular, we propose to more frequently replay transitions with high expected learning progress, as measured by the magnitude of their temporal-difference (TD) error. This prioritization can lead to a loss of diversity, which we alleviate with stochastic prioritization, and introduce bias, which we correct with importance sampling. Our resulting algorithms are robust and scalable, which we demonstrate on the Atari 2600 benchmark suite, where we obtain faster learning and state-of-the-art performance.

2 Background

Numerous neuroscience studies have identified evidence of experience replay in the hippocampus of rodents, suggesting that sequences of prior experience are replayed, either during awake resting or sleep. Sequences associated with rewards appear to be replayed more frequently (Atherton et al., 2015; Ólafsdóttir et al., 2015; Foster & Wilson, 2006). Experiences with high magnitude TD error also appear to be replayed more often (Singer & Frank, 2009; McNamara et al., 2014).

It is well-known that planning algorithms such as value iteration can be made more efficient by prioritizing updates in an appropriate order. *Prioritized sweeping* (Moore & Atkeson, 1993; Andre et al., 1998) selects which state to update next, prioritized according to the change in value, if that update was executed. The TD error provides

one way to measure these priorities (van Seijen & Sutton, 2013). Our approach uses a similar prioritization method, but for model-free RL rather than model-based planning. Furthermore, we use a stochastic prioritization that is more robust when learning a function approximator from samples.

TD-errors have also been used as a prioritization mechanism for determining where to focus resources, for example when choosing where to explore (White et al., 2014) or which features to select (Geramifard et al., 2011; Sun et al., 2011).

In supervised learning, there are numerous techniques to deal with imbalanced datasets when class identities are known, including re-sampling, under-sampling and over-sampling techniques, possibly combined with ensemble methods (for a review, see Galar et al., 2012). A recent paper introduced a form of re-sampling in the context of deep RL with experience replay (Narasimhan et al., 2015); the method separates experience into two buckets, one for positive and one for negative rewards, and then picks a fixed fraction from each to replay. This is only applicable in domains that (unlike ours) have a natural notion of ‘positive/negative’ experience. Furthermore, Hinton (2007) introduced a form of non-uniform sampling based on error, with an importance sampling correction, which led to a 3x speed-up on MNIST digit classification.

There have been several proposed methods for playing Atari with deep reinforcement learning, including deep Q-networks (DQN) (Mnih et al., 2013; 2015; Guo et al., 2014; Stadie et al., 2015; Nair et al., 2015; Bellemare et al., 2016), and the *Double DQN* algorithm (van Hasselt et al., 2016), which is the current published state-of-the-art. Simultaneously with our work, an architectural innovation that separates advantages from the value function (see the co-submission by Wang et al., 2015) has also led to substantial improvements on the Atari benchmark.

3 Prioritized Replay

Using a replay memory leads to design choices at two levels: which experiences to store, and which experiences to replay (and how to do so). This paper addresses only the latter: making the most effective use of the replay memory for learning, assuming that its contents are outside of our control (but see also Section 6).

3.1 A Motivating Example

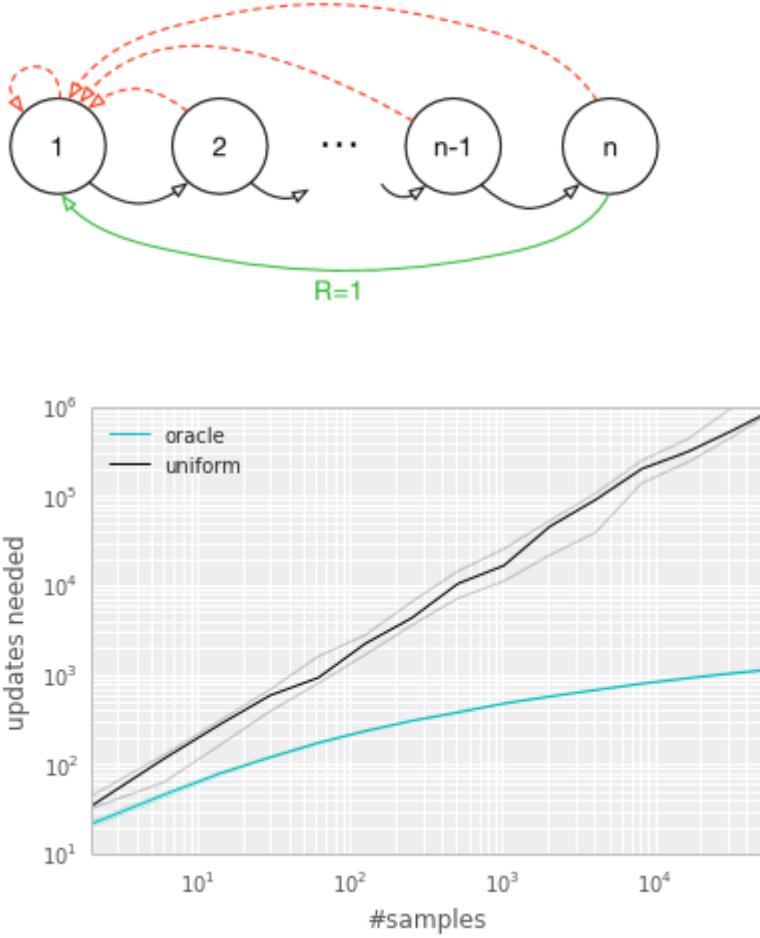


Figure 1: **Left:** Illustration of the ‘Blind Cliffwalk’ example domain: there are two actions, a ‘right’ and a ‘wrong’ one, and the episode is terminated whenever the agent takes the ‘wrong’ action (dashed red arrows). Taking the ‘right’ action progresses through a sequence of n states (black arrows), at the end of which lies a final reward of 1 (green arrow); reward is 0 elsewhere. We chose a representation such that generalizing over what action is ‘right’ is not possible. **Right:** Median number of learning steps required to learn the value function as a function of the size of the total number of transitions in the replay memory. Note the log-log scale, which highlights the exponential speed-up from replaying with an oracle (bright blue), compared to uniform replay (black); faint lines are min/max values from 10 independent runs.

To understand the potential gains of prioritization, we introduce an artificial ‘Blind Cliffwalk’ environment (described in Figure 1, left) that exemplifies the challenge of exploration when rewards are rare. 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} . Furthermore, the most relevant transitions (from rare successes) are hidden in a mass of highly

redundant failure cases (similar to a bipedal robot falling over repeatedly, before it discovers how to walk).

We use this example to highlight the difference between the learning times of two agents. Both agents perform Q-learning updates on transitions drawn from the same replay memory. The first agent replays transitions uniformly at random, while the second agent invokes an oracle to prioritize transitions. This oracle greedily selects the transition that maximally reduces the global loss in its current state (in hindsight, after the parameter update). For the details of the setup, see Appendix B.1. Figure 1 (right) shows that picking the transitions in a good order can lead to exponential speed-ups over uniform choice. Such an oracle is of course not realistic, yet the large gap motivates our search for a practical approach that improves on uniform random replay.

3.2 Prioritizing with TD-error

The central component of prioritized replay is the criterion by which the importance of each transition is measured. One idealised criterion would be the amount the RL agent can learn from a transition in its current state (expected learning progress). While this measure is not directly accessible, a reasonable proxy is the magnitude of a transition’s TD error δ , which indicates how ‘surprising’ or unexpected the transition is: specifically, how far the value is from its next-step bootstrap estimate (Andre et al., 1998). This is particularly suitable for incremental, online RL algorithms, such as SARSA or Q-learning, that already compute the TD-error and update the parameters in proportion to δ . The TD-error can be a poor estimate in some circumstances as well, e.g. when rewards are noisy; see Appendix A for a discussion of alternatives.

To demonstrate the potential effectiveness of prioritizing replay by TD error, we compare the uniform and oracle baselines in the Blind Cliffwalk to a ‘greedy TD-error prioritization’ algorithm. This algorithm stores the last encountered TD error along with each transition in the replay memory. The transition with the largest absolute TD error is replayed from the memory. A Q-learning update is applied to this transition, which updates the weights in proportion to the TD error. New transitions arrive without a known TD-error, so we put them at maximal priority in order to guarantee that all experience is seen at least once. Figure 2 (left), shows that this algorithm results in a substantial reduction in the effort required to solve the Blind Cliffwalk task.²

² Note that a random (or optimistic) initialization of the Q-values is necessary with greedy prioritization. If initializing with zero instead, unrewarded transitions would appear to have zero error initially, be placed at the bottom of the queue, and not be revisited until the error on other transitions drops below numerical precision.

Implementation: To scale to large memory sizes N , we use a binary heap data structure for the priority queue, for which finding the maximum priority transition when sampling is $O(1)$ and updating priorities (with the new TD-error after a learning step) is $O(\log N)$. See Appendix B.2.1 for more details.

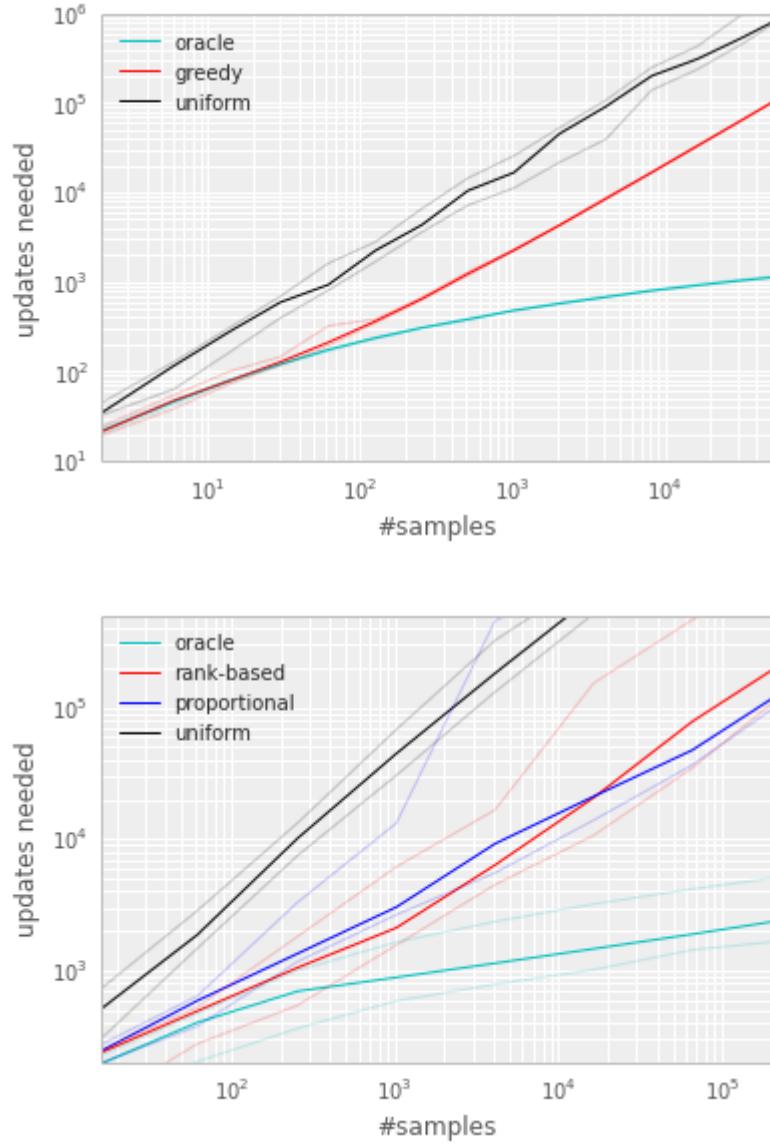


Figure 2: Median number of updates required for Q-learning to learn the value function on the Blind Cliffwalk example, as a function of the total number of transitions (only a single one of which was successful and saw the non-zero reward). Faint lines are min/max values from 10 random initializations. Black is uniform random replay, cyan uses the hindsight-oracle to select transitions, red and blue use prioritized replay (rank-based and proportional respectively). The results differ by multiple orders of magnitude, thus the need for a log-log plot. In both subplots it is evident that replaying experience in the right order makes an enormous difference to the number of updates required. See Appendix B.1 for details. **Left:** Tabular representation, greedy

prioritization. **Right:** Linear function approximation, both variants of stochastic prioritization.

3.3 Stochastic Prioritization

However, greedy TD-error prioritization has several issues. First, to avoid expensive sweeps over the entire replay memory, TD errors are only updated for the transitions that are replayed. One consequence is that transitions that have a low TD error on first visit may not be replayed for a long time (which means effectively never with a sliding window replay memory). Further, it is sensitive to noise spikes (e.g. when rewards are stochastic), which can be exacerbated by bootstrapping, where approximation errors appear as another source of noise. Finally, greedy prioritization focuses on a small subset of the experience: errors shrink slowly, especially when using function approximation, meaning that the initially high error transitions get replayed frequently. This lack of diversity that makes the system prone to over-fitting.

To overcome these issues, we introduce a stochastic sampling method that interpolates between pure greedy prioritization and uniform random sampling. We ensure that the probability of being sampled is monotonic in a transition’s priority, while guaranteeing a non-zero probability even for the lowest-priority transition. Concretely, we define the probability of sampling transition i as

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \quad (1)$$

where $p_i > 0$ is the priority of transition i . The exponent α determines how much prioritization is used, with $\alpha = 0$ corresponding to the uniform case.

The first variant we consider is the direct, proportional prioritization where $p_i = |\delta_i| + \epsilon$, where ϵ is a small positive constant that prevents the edge-case of transitions not being revisited once their error is zero. The second variant is an indirect, rank-based prioritization where $p_i = \frac{1}{\text{rank}(i)}$, where $\text{rank}(i)$ is the rank of transition i when the replay memory is sorted according to $|\delta_i|$. In this case, P becomes a power-law distribution with exponent α . Both distributions are monotonic in $|\delta|$, but the latter is likely to be more robust, as it is insensitive to outliers. Both variants of stochastic prioritization lead to large speed-ups over the uniform baseline on the Cliffwalk task, as shown on Figure 2 (right).

Implementation: To efficiently sample from distribution (1), the complexity cannot depend on N . For the rank-based variant, we can approximate the cumulative density function with a piecewise linear function with k segments of equal probability. The segment boundaries can be precomputed (they change only when N or α change). At

runtime, we sample a segment, and then sample uniformly among the transitions within it. This works particularly well in conjunction with a minibatch-based learning algorithm: choose k to be the size of the minibatch, and sample exactly one transition from each segment – this is a form of stratified sampling that has the added advantage of balancing out the minibatch (there will always be exactly one transition with high magnitude δ , one with medium magnitude, etc). The proportional variant is different, also admits an efficient implementation based on a ‘sum-tree’ data structure (where every node is the sum of its children, with the priorities as the leaf nodes), which can be efficiently updated and sampled from. See Appendix B.2.1 for more additional details.

3.4 Annealing the Bias

The estimation of the expected value with stochastic updates relies on those updates corresponding to the same distribution as its expectation. Prioritized replay introduces bias because it changes this distribution in an uncontrolled fashion, and therefore changes the solution that the estimates will converge to (even if the policy and state distribution are fixed). We can correct this bias by using importance-sampling (IS) weights

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)} \right)^\beta$$

that fully compensates for the non-uniform probabilities $P(i)$ if $\beta = 1$. These weights can be folded into the Q-learning update by using $w_i \delta_i$ instead of δ_i (this is thus *weighted* IS, not ordinary IS, see e.g. Mahmood et al., 2014). For stability reasons, we always normalize weights by $1 / \max_i w_i$ so that they only scale the update downwards.

In typical reinforcement learning scenarios, the unbiased nature of the updates is most important near convergence at the end of training, as the process is highly non-stationary anyway, due to changing policies, state distributions and bootstrap targets; we hypothesize that a small bias can be ignored in this context (see also Figure 12 in the appendix for a case study of full IS correction on Atari). We therefore exploit the flexibility of *annealing* the amount of importance-sampling correction over time, by defining a schedule on the exponent β that reaches 1 only at the end of learning. In practice, we linearly anneal β from its initial value β_0 to 1. Note that the choice of this hyperparameter interacts with choice of prioritization exponent α ; increasing both simultaneously prioritizes sampling more aggressively at the same time as correcting for it more strongly.

Importance sampling has another benefit when combined with prioritized replay in the context of non-linear function approximation (e.g. deep neural networks): here large steps can be very disruptive, because the first-order approximation of the gradient is only reliable locally, and have to be prevented with a smaller global step-size. In our

approach instead, prioritization makes sure high-error transitions are seen many times, while the IS correction reduces the gradient magnitudes (and thus the effective step size in parameter space), and allowing the algorithm follow the curvature of highly non-linear optimization landscapes because the Taylor expansion is constantly re-approximated.

We combine our prioritized replay algorithm into a full-scale reinforcement learning agent, based on the state-of-the-art Double DQN algorithm. Our principal modification is to replace the uniform random sampling used by Double DQN with our stochastic prioritization and importance sampling methods (see Algorithm 1).

Algorithm 1 Double DQN with proportional prioritization

```

1: | Input: minibatch  $k$ , step-size  $\eta$ , replay period  $K$  and size  $N$ , exponents  $\alpha$  and  $\beta$ , budget  $T$ .
2: | Initialize replay memory  $\mathcal{H} = \emptyset$ ,  $\Delta = 0$ ,  $p_1 = 1$ 
3: | Observe  $S_0$  and choose  $A_0 \sim \pi_\theta(S_0)$ 
4: | for  $t = 1$  to  $T$  do
5: |   Observe  $S_t, R_t, \gamma_t$ 
6: |   Store transition  $(S_{t-1}, A_{t-1}, R_t, \gamma_t, S_t)$  in  $\mathcal{H}$  with maximal priority  $p_t = \max_i p_i$ 
7: |   if  $t \equiv 0 \bmod K$  then
8: |     for  $j = 1$  to  $k$  do
9: |       Sample transition  $j \sim P(j) = p_j^\alpha / \sum_i p_i^\alpha$ 
10: |      Compute importance-sampling weight  $w_j = (N \cdot P(j))^{-\beta} / \max_i w_i$ 
11: |      Compute TD-error  $\delta_j = R_j + \gamma_j Q_{\text{target}}(S_j, \arg \max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$ 
12: |      Update transition priority  $p_j \leftarrow |\delta_j|$ 
13: |      Accumulate weight-change  $\Delta \leftarrow \Delta + w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$ 
14: |    end for
15: |    Update weights  $\theta \leftarrow \theta + \eta \cdot \Delta$ , reset  $\Delta = 0$ 
16: |    From time to time copy weights into target network  $\theta_{\text{target}} \leftarrow \theta$ 
17: |  end if
18: |  Choose action  $A_t \sim \pi_\theta(S_t)$ 
19: | end for

```

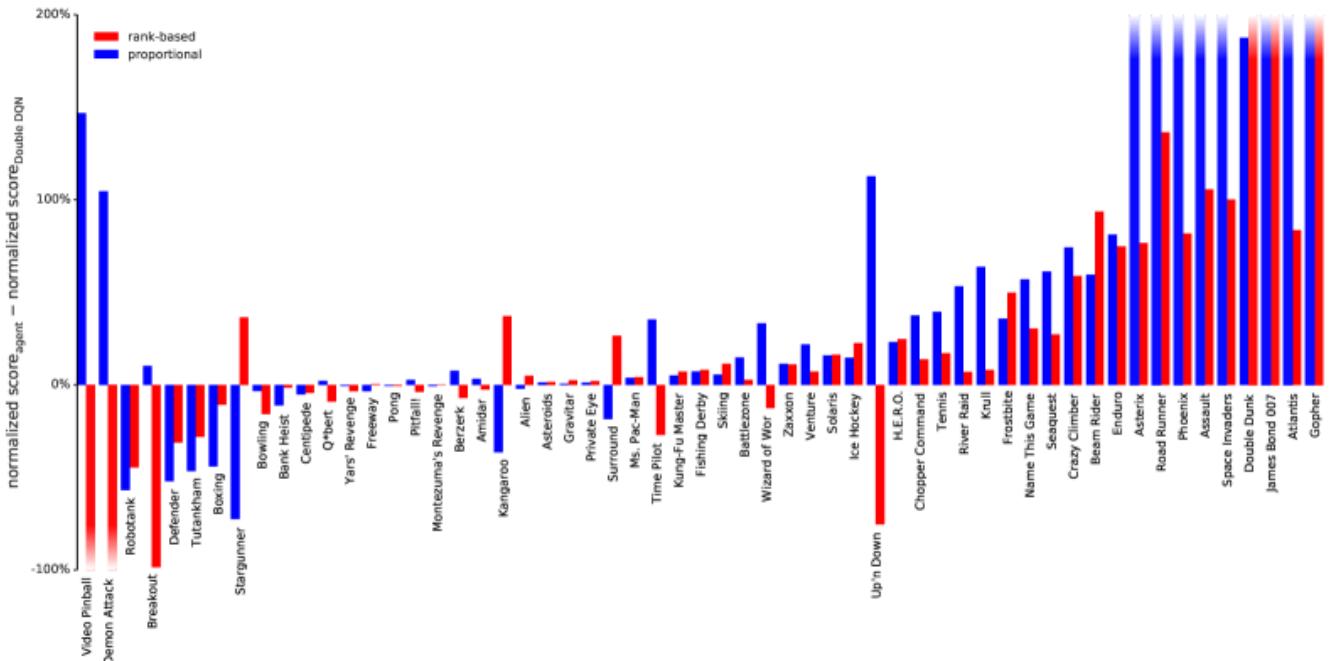


Figure 3: Difference in normalized score (the gap between random and human is 100%) on 57 games with human starts, comparing Double DQN with and without prioritized replay (rank-based variant in red, proportional in blue), showing substantial improvements in most games. Exact scores are in Table 6. See also Figure 9 where regular DQN is the baseline.

4 Atari Experiments

With all these concepts in place, we now investigate to what extent replay with such prioritized sampling can improve performance in realistic problem domains. For this, we chose the collection of Atari benchmarks (Bellemare et al., 2012) with their end-to-end RL from vision setup, because they are popular and contain diverse sets of challenges, including delayed credit assignment, partial observability, and difficult function approximation (Mnih et al., 2015; van Hasselt et al., 2016). Our hypothesis is that prioritized replay is generally useful, so that it will make learning with experience replay more efficient without requiring careful problem-specific hyperparameter tuning.

We consider two baseline algorithms that use uniform experience replay, namely the version of the DQN algorithm from the Nature paper (Mnih et al., 2015), and its recent extension Double DQN (van Hasselt et al., 2016) that substantially improved the state-of-the-art by reducing the over-estimation bias with Double Q-learning (van Hasselt, 2010). Throughout this paper we use the tuned version of the Double DQN algorithm. For this paper, the most relevant component of these baselines is the replay mechanism: all

experienced transitions are stored in a sliding window memory that retains the last 10^6 transitions. The algorithm processes minibatches of 32 transitions sampled uniformly from the memory. One minibatch update is done for each 4 new transitions entering the memory, so all experience is replayed 8 times on average. Rewards and TD-errors are clipped to fall within $[-1, 1]$ for stability reasons.

We use the identical neural network architecture, learning algorithm, replay memory and evaluation setup as for the baselines (see Appendix B.2). The only difference is the mechanism for sampling transitions from the replay memory, which is now done according to Algorithm 1 instead of uniformly. We compare the baselines to both variants of prioritized replay (rank-based and proportional).

Only a single hyperparameter adjustment was necessary compared to the baseline: Given that prioritized replay picks high-error transitions more often, the typical gradient magnitudes are larger, so we reduced the step-size η by a factor 4 compared to the (Double) DQN setup. For the α and β_0 hyperparameters that are introduced by prioritization, we did a coarse grid search (evaluated on a subset of 8 games), and found the sweet spot to be $\alpha = 0.7$, $\beta_0 = 0.5$ for the rank-based variant and $\alpha = 0.6$, $\beta_0 = 0.4$ for the proportional variant. These choices are trading off aggressiveness with robustness, but it is easy to revert to a behavior closer to the baseline by reducing α and/or increasing β .

We produce the results by running each variant with a single hyperparameter setting across all games, as was done for the baselines. Our main evaluation metric is the *quality of the best policy*, in terms of average score per episode, given start states sampled from human traces (as introduced in Nair et al., 2015 and used in van Hasselt et al., 2016, which requires more robustness and generalization as the agent cannot rely on repeating a single memorized trajectory). These results are summarized in Table 1 and Figure 3, but full results and raw scores can be found in Tables 7 and 6 in the Appendix. A secondary metric is the *learning speed*, which we summarize on Figure 4, with more detailed learning curves on Figures 7 and 8.

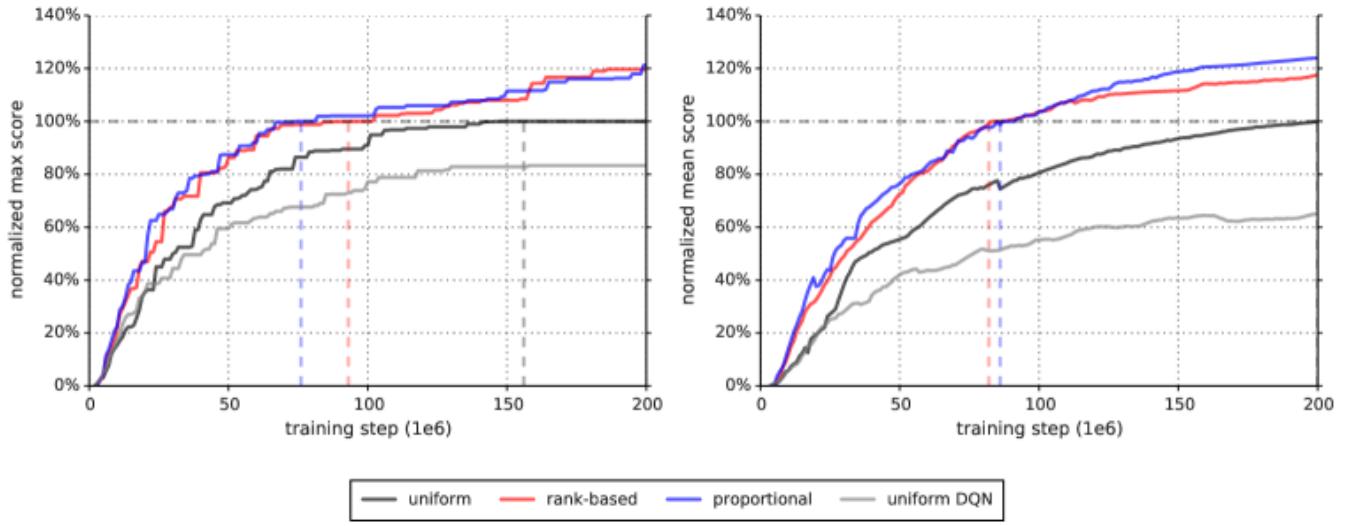


Figure 4: Summary plots of learning speed. **Left:** median over 57 games of the maximum baseline-normalized score achieved so far. The baseline-normalized score is calculated as in Equation 4 but using the maximum Double DQN score seen across training is used instead of the human score. The equivalence points are highlighted with dashed lines; those are the steps at which the curves reach 100%, (i.e., when the algorithm performs equivalently to Double DQN in terms of median over games). For rank-based and proportional prioritization these are at 47% and 38% of total training time. **Right:** Similar to the left, but using the mean instead of maximum, which captures cumulative performance rather than peak performance. Here rank-based and proportional prioritization reach the equivalence points at 41% and 43% of total training time, respectively. For the detailed learning curves that these plots summarize, see Figure 7.

	DQN	Double DQN (tuned)			
	baseline	rank-based	baseline	rank-based	proportional
Median	48%	106%	111%	113%	128%
Mean	122%	355%	418%	454%	551%
> baseline	–	41	–	38	42
> human	15	25	30	33	33
# games	49	49	57	57	57

Table 1: Summary of normalized scores. See Table 6 in the appendix for full results.

We find that adding prioritized replay to DQN leads to a substantial improvement in score on 41 out of 49 games (compare columns 2 and 3 of Table 6 or Figure 9 in the appendix), with the median normalized performance across 49 games increasing from 48% to 106%. Furthermore, we find that the boost from prioritized experience replay is *complementary* to the one from introducing Double Q-learning into DQN: performance increases another notch, leading to the current state-of-the-art on the Atari benchmark (see Figure 3). Compared to Double DQN, the median performance across 57 games increased from 111% to 128%, and the mean performance from 418% to 551% bringing additional games such as River Raid, Seaquest and Surround to a human level for the first time, and making large jumps on others (e.g. Gopher, James Bond 007 or Space Invaders). Note that mean performance is not a very reliable metric because a single game (Video Pinball) has a dominant contribution. Prioritizing replay gives a performance boost on almost all games, and on aggregate, learning is twice as fast (see Figures 4 and 8). The learning curves on Figure 7 illustrate that while the two variants of prioritization usually lead to similar results, there are games where one of them remains close to the Double DQN baseline while the other one leads to a big boost, for example Double Dunk or Surround for the rank-based variant, and Alien, Asterix, Enduro, Phoenix or Space Invaders for the proportional variant. Another observation from the learning curves is that compared to the uniform baseline, prioritization is effective at reducing the delay until performance gets off the ground in games that otherwise suffer from such a delay, such as Battlezone, Zaxxon or Frostbite.

5 Discussion

In the head-to-head comparison between rank-based prioritization and proportional prioritization, we expected the rank-based variant to be more robust because it is not affected by outliers nor error magnitudes. Furthermore, its heavy-tail property also guarantees that samples will be diverse, and the stratified sampling from partitions of different errors will keep the total minibatch gradient at a stable magnitude throughout training. On the other hand, the ranks make the algorithm blind to the relative error scales, which could incur a performance drop when there is structure in the distribution of errors to be exploited, such as in sparse reward scenarios. Perhaps surprisingly, both variants perform similarly in practice; we suspect this is due to the heavy use of clipping (of rewards and TD-errors) in the DQN algorithm, which removes outliers. Monitoring the distribution of TD-errors as a function of time for a number of games (see Figure 10 in the appendix), and found that it becomes close to a heavy-tailed distribution as learning progresses, while still differing substantially across games; this empirically validates the form of Equation 1. Figure 11, in the appendix, shows how this distribution interacts with Equation 1 to produce the effective replay probabilities.

While doing this analysis, we stumbled upon another phenomenon (obvious in retrospect), namely that some fraction of the visited transitions are never replayed

before they drop out of the sliding window memory, and many more are replayed for the first time only long after they are encountered. Also, uniform sampling is implicitly biased toward out-of-date transitions that were generated by a policy that has typically seen hundreds of thousands of updates since. Prioritized replay with its bonus for unseen transitions directly corrects the first of these issues, and also tends to help with the second one, as more recent transitions tend to have larger error – this is because old transitions will have had more opportunities to have them corrected, and because novel data tends to be less well predicted by the value function.

We hypothesize that deep neural networks interact with prioritized replay in another interesting way. When we distinguish learning the value given a representation (i.e., the top layers) from learning an improved representation (i.e., the bottom layers), then transitions for which the representation is good will quickly reduce their error and then be replayed much less, increasing the learning focus on others where the representation is poor, thus putting more resources into distinguishing aliased states – if the observations and network capacity allow for it.

6 Extensions

Prioritized Supervised Learning: The analogous approach to prioritized replay in the context of supervised learning is to sample non-uniformly from the dataset, each sample using a priority based on its last-seen error. This can help focus the learning on those samples that can still be learned from, devoting additional resources to the (hard) boundary cases, somewhat similarly to boosting (Galar et al., 2012). Furthermore, if the dataset is imbalanced, we hypothesize that samples from the rare classes will be sampled disproportionately often, because their errors shrink less fast, and the chosen samples from the common classes will be those nearest to the decision boundaries, leading to an effect similar to hard negative mining (Felzenszwalb et al., 2008). To check whether these intuitions hold, we conducted a preliminary experiment on a class-imbalanced variant of the classical MNIST digit classification problem (LeCun et al., 1998), where we removed 99% of the samples for digits 0, 1, 2, 3, 4 in the training set, while leaving the test/validation sets untouched (i.e., those retain class balance). We compare two scenarios: in the informed case, we reweight the errors of the impoverished classes artificially (by a factor 100), in the uninformed scenario, we provide no hint that the test distribution will differ from the training distribution. See Appendix B.3 for the details of the convolutional neural network training setup. Prioritized sampling (uninformed, with $\alpha = 1$, $\beta = 0$) outperforms the uninformed uniform baseline, and approaches the performance of the informed uniform baseline in terms of generalization (see Figure 5); again, prioritized training is also faster in terms of learning speed.

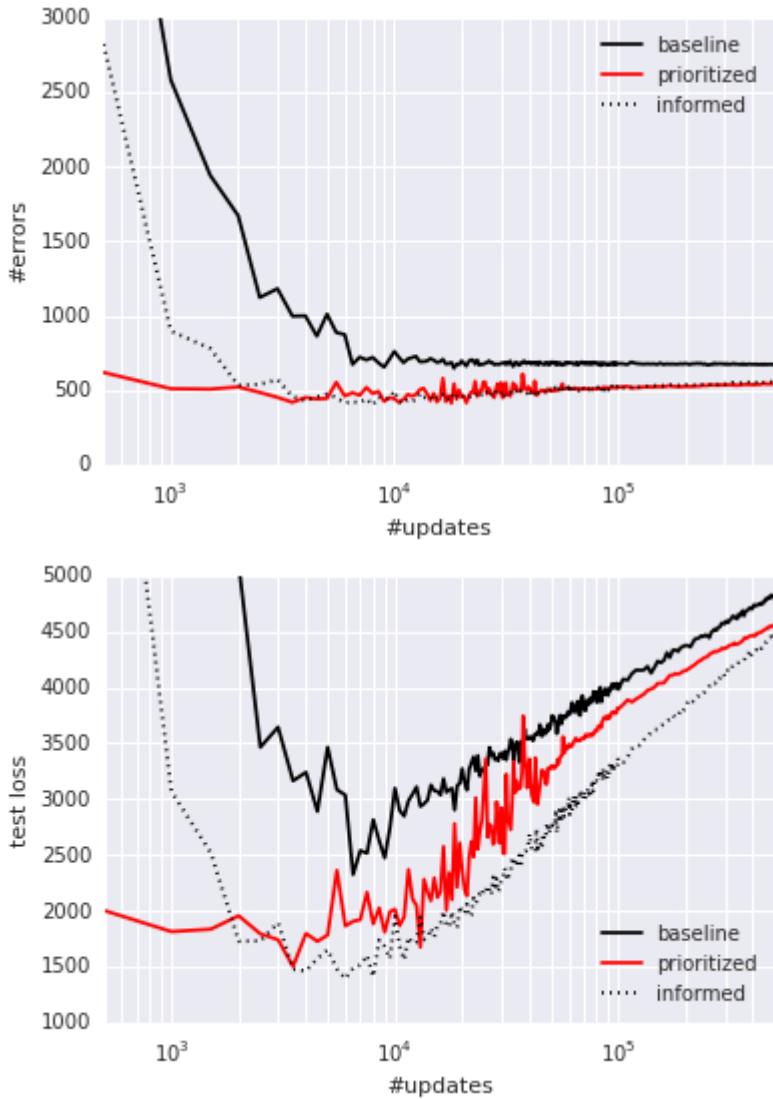


Figure 5: Classification errors as a function of supervised learning updates on severely class-imbalanced MNIST. Prioritized sampling improves performance, leading to comparable errors on the test set, and approaching the imbalance-informed performance (median of 3 random initializations). **Left:** Number of misclassified test set samples. **Right:** Test set loss, highlighting overfitting.

Off-policy Replay: Two standard approaches to off-policy RL are rejection sampling and using importance sampling ratios ρ to correct for how likely a transition would have been on-policy. Our approach contains analogues to both these approaches, the replay probability P and the IS-correction w . It appears therefore natural to apply it to off-policy RL, if transitions are available in a replay memory. In particular, we recover weighted IS with $w = \rho$, $\alpha = 0$, $\beta = 1$ and rejection sampling with $p = \min(1; \rho)$, $\alpha = 1$, $\beta = 0$, in the proportional variant. Our experiments indicate that intermediate variants, possibly with annealing or ranking, could be more useful in practice – especially when IS ratios introduce high variance, i.e., when the policy of interest differs substantially

from the behavior policy in some states. Of course, off-policy correction is complementary to our prioritization based on expected learning progress, and the same framework can be used for a hybrid prioritization by defining $p = \rho \cdot |\delta|$, or some other sensible trade-off based on both ρ and δ .

Feedback for Exploration: An interesting side-effect of prioritized replay is that the total number M_i that a transition will end up being replayed varies widely, and this gives a rough indication of how useful it was to the agent. This potentially valuable signal can be fed back to the exploration strategy that generates the transitions. For example, we could sample exploration hyperparameters (such as the fraction of random actions ϵ , the Boltzmann temperature, or the amount of intrinsic reward to mix in) from a parametrized distribution at the beginning of each episode, monitor the usefulness of the experience via M_i , and update the distribution toward generating more useful experience. Or, in a parallel system like the Gorila agent (Nair et al., 2015), it could guide resource allocation between a collection of concurrent but heterogeneous ‘actors’, each with different exploration hyperparameters.

Prioritized Memories: Considerations that help determine which transitions to replay are likely to also be relevant for determining which memories to store and when to erase them (e.g. when it becomes likely that they will never be replayed anymore). An explicit control over which memories to keep or erase can help reduce the required total memory size, because it reduces redundancy (frequently visited transitions will have low error, so many of them will be dropped), while automatically adjusting for what has been learned already (dropping many of the ‘easy’ transitions) and biasing the contents of the memory to where the errors remain high. This is a non-trivial aspect, because memory requirements for DQN are currently dominated by the size of the replay memory, no longer by the size of the neural network. Erasing is a more final decision than reducing the replay probability, thus an even stronger emphasis of diversity may be necessary, for example by tracking the age of each transitions and using it to modulate the priority in such a way as to preserve sufficient old experience to prevent cycles (related to ‘hall of fame’ ideas in multi-agent literature, Rosin & Belew, 1997). The priority mechanism is also flexible enough to permit integrating experience from *other sources*, such as from a planner or from human expert trajectories (Guo et al., 2014), since knowing the source can be used to modulate each transition’s priority, e.g. in such a way as to preserve a sufficient fraction of external experience in memory.

7 Conclusion

This paper introduced prioritized replay, a method that can make learning from experience replay more efficient. We studied a couple of variants, devised implementations that scale to large replay memories, and found that prioritized replay speeds up learning by a factor 2 and leads to a new state-of-the-art of performance on

the Atari benchmark. We laid out further variants and extensions that hold promise, namely for class-imbalanced supervised learning.

Acknowledgments

We thank our Deepmind colleagues, in particular Hado van Hasselt, Joseph Modayil, Nicolas Heess, Marc Bellemare, Razvan Pascanu, Dharshan Kumaran, Daan Wierstra, Arthur Guez, Charles Blundell, Alex Pritzel, Alex Graves, Balaji Lakshminarayanan, Ziyu Wang, Nando de Freitas, Remi Munos and Geoff Hinton for insightful discussions and feedback.

References

- Andre et al. (1998) Andre, David, Friedman, Nir, and Parr, Ronald.
Generalized prioritized sweeping.
In *Advances in Neural Information Processing Systems*. Citeseer, 1998.
- Atherton et al. (2015) Atherton, Laura A, Dupret, David, and Mellor, Jack R.
Memory trace replay: the shaping of memory consolidation by neuromodulation.
Trends in neurosciences, 38(9):560–570, 2015.
- Bellemare et al. (2012) Bellemare, Marc G, Naddaf, Yavar, Veness, Joel, and Bowling, Michael.
The arcade learning environment: An evaluation platform for general agents.
arXiv preprint arXiv:1207.4708, 2012.
- Bellemare et al. (2016) Bellemare, Marc G., Ostrovski, Georg, Guez, Arthur, Thomas, Philip S., and Munos, Rémi.
Increasing the action gap: New operators for reinforcement learning.
In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016.
URL
<http://arxiv.org/abs/1512.04860>.

- Collobert et al. (2011)
Collobert, Ronan, Kavukcuoglu, Koray, and Farabet, Clément.
Torch7: A matlab-like environment for machine learning.
In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.
- Diamond (1994)
Diamond, Jared.
Zebras and the Anna Karenina principle.
Natural History, 103:4–4, 1994.
- Felzenszwalb et al. (2008)
Felzenszwalb, Pedro, McAllester, David, and Ramanan, Deva.
A discriminatively trained, multiscale, deformable part model.
In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pp. 1–8. IEEE, 2008.
- Foster & Wilson (2006)
Foster, David J and Wilson, Matthew A.
Reverse replay of behavioural sequences in hippocampal place cells during the awake state.
Nature, 440(7084):680–683, 2006.
- Galar et al. (2012)
Galar, Mikel, Fernandez, Alberto, Barrenechea, Edurne, Bustince, Humberto, and Herrera, Francisco.
A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches.
Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on, 42(4):463–484, 2012.
- Geramifard et al. (2011)
Geramifard, Alborz, Doshi, Finale, Redding, Joshua, Roy, Nicholas, and How, Jonathan.
Online discovery of feature dependencies.
In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 881–888, 2011.

- Guo et al. (2014)
Guo, Xiaoxiao, Singh, Satinder, Lee, Honglak, Lewis, Richard L, and Wang, Xiaoshi.
Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning.
In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., and Weinberger, K.Q. (eds.), *Advances in Neural Information Processing Systems 27*, pp. 3338–3346. Curran Associates, Inc., 2014.
- Hinton (2007)
Hinton, Geoffrey E.
To recognize shapes, first learn to generate images.
Progress in brain research, 165:535–547, 2007.
- Kingma & Ba (2014)
Kingma, Diederik P. and Ba, Jimmy.
Adam: A method for stochastic optimization.
CoRR, abs/1412.6980, 2014.
- Lecun et al. (1998)
Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P.
Gradient-based learning applied to document recognition.
Proceedings of the IEEE, 86(11):2278–2324, Nov 1998.
ISSN 0018-9219. doi: [10.1109/5.726791](https://doi.org/10.1109/5.726791).
- LeCun et al. (1998)
LeCun, Yann, Cortes, Corinna, and Burges, Christopher JC.
The MNIST database of handwritten digits, 1998.
- Lin (1992)
Lin, Long-Ji.
Self-improving reactive agents based on reinforcement learning, planning and teaching.
Machine learning, 8(3-4):293–321, 1992.

- Mahmood et al. (2014)
Mahmood, A Rupam, van Hasselt, Hado P, and Sutton, Richard S.
Weighted importance sampling for off-policy learning with linear function approximation.
In *Advances in Neural Information Processing Systems*, pp. 3014–3022, 2014.
- McNamara et al. (2014)
McNamara, Colin G, Tejero-Cantero, Álvaro, Trouche, Stéphanie, Campo-Urriza, Natalia, and Dupret, David.
Dopaminergic neurons promote hippocampal reactivation and spatial memory persistence.
Nature neuroscience, 2014.
- Mnih et al. (2013)
Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin.
Playing atari with deep reinforcement learning.
arXiv preprint arXiv:1312.5602, 2013.
- Mnih et al. (2015)
Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, Petersen, Stig, Beattie, Charles, Sadik, Amir, Antonoglou, Ioannis, King, Helen, Kumaran, Dharshan, Wierstra, Daan, Legg, Shane, and Hassabis, Demis.
Human-level control through deep reinforcement learning.
Nature, 518(7540):529–533, 2015.
- Moore & Atkeson (1993)
Moore, Andrew W and Atkeson, Christopher G.
Prioritized sweeping: Reinforcement learning with less data and less time.
Machine Learning, 13(1):103–130, 1993.

- Nair et al. (2015) Nair, Arun, Srinivasan, Praveen, Blackwell, Sam, Alcicek, Cagdas, Fearon, Rory, Maria, Alessandro De, Panneershelvam, Vedavyas, Suleyman, Mustafa, Beattie, Charles, Petersen, Stig, Legg, Shane, Mnih, Volodymyr, Kavukcuoglu, Koray, and Silver, David. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.
- Narasimhan et al. (2015) Narasimhan, Karthik, Kulkarni, Tejas, and Barzilay, Regina. Language understanding for text-based games using deep reinforcement learning. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2015.
- Ólafsdóttir et al. (2015) Ólafsdóttir, H Freyja, Barry, Caswell, Saleem, Aman B, Hassabis, Demis, and Spiers, Hugo J. Hippocampal place cells construct reward related sequences through unexplored space. *Elife*, 4:e06063, 2015.
- Riedmiller (1994) Riedmiller, Martin. Rprop-description and implementation details. 1994.
- Rosin & Belew (1997) Rosin, Christopher D and Belew, Richard K. New methods for competitive coevolution. *Evolutionary Computation*, 5(1):1–29, 1997.
- Schaul et al. (2013) Schaul, Tom, Zhang, Sixin, and Lecun, Yann. No more pesky learning rates.

- Schmidhuber (1991)
In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 343–351, 2013.
- Schmidhuber, Jürgen.
Curious model-building control systems.
In *Neural Networks, 1991. 1991 IEEE International Joint Conference on*, pp. 1458–1463. IEEE, 1991.
- Singer & Frank (2009)
Singer, Annabelle C and Frank, Loren M.
Rewarded outcomes enhance reactivation of experience in the hippocampus.
Neuron, 64(6):910–921, 2009.
- Stadie et al. (2015)
Stadie, Bradly C, Levine, Sergey, and Abbeel, Pieter.
Incentivizing exploration in reinforcement learning with deep predictive models.
arXiv preprint arXiv:1507.00814, 2015.
- Sun et al. (2011)
Sun, Yi, Ring, Mark, Schmidhuber, Jürgen, and Gomez, Faustino J.
Incremental basis construction from temporal difference error.
In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 481–488, 2011.
- van Hasselt (2010)
van Hasselt, Hado. Double Q-learning.
In *Advances in Neural Information Processing Systems*, pp. 2613–2621, 2010.
- van Hasselt et al. (2016)
van Hasselt, Hado, Guez, Arthur, and Silver, David.
Deep Reinforcement Learning with Double Q-learning.
In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
URL
<http://arxiv.org/abs/1509.06461>.

van Seijen & Sutton (2013)	van Seijen, Harm and Sutton, Richard. Planning by prioritized sweeping with small backups. In <i>Proceedings of The 30th International Conference on Machine Learning</i> , pp. 361–369, 2013.
Wang et al. (2015)	Wang, Z., de Freitas, N., and Lanctot, M. Dueling network architectures for deep reinforcement learning. Technical report, 2015. URL http://arxiv.org/abs/1511.06581
Watkins & Dayan (1992)	Watkins, Christopher JCH and Dayan, Peter. Q-learning. <i>Machine learning</i> , 8(3-4):279–292, 1992.
White et al. (2014)	White, Adam, Modayil, Joseph, and Sutton, Richard S. Surprise and curiosity for big data robotics. In <i>Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence</i> , 2014.

Appendix A Prioritization Variants

The absolute TD-error is only one possible proxy for the ideal priority measure of expected learning progress. While it captures the scale of potential improvement, it ignores inherent stochasticity in rewards or transitions, as well as possible limitations from partial observability or FA capacity; in other words, it is problematic when there are unlearnable transitions. In that case, its *derivative* – which could be approximated by the difference between a transition’s current $|\delta|$ and the $|\delta|$ when it was last replayed³

³Of course, more robust approximations would be a function of the history of all encountered δ values. In particular, one could imagine an RProp-style update (Riedmiller, 1994) to priorities that increase the priority while the signs match, and reduce it whenever consecutive errors (for the same transition) differ in sign.

– may be more useful. This measure is less immediately available, however, and is influenced by whatever was replayed in the meanwhile, which increases its variance. In

preliminary experiments, we found it did not outperform $|\delta|$, but this may say more about the class of (near-deterministic) environments we investigated, than about the measure itself.

An orthogonal variant is to consider the norm of the *weight-change* induced by replaying a transition – this can be effective if the underlying optimizer employs adaptive step-sizes that reduce gradients in high-noise directions (Schaul et al., 2013; Kingma & Ba, 2014), thus placing the burden of distinguishing between learnable and unlearnable transitions on the optimizer.

It is possible to modulate prioritization by not treating positive TD-errors the same than negative ones; we can for example invoke the Anna Karenina principle (Diamond, 1994), interpreted to mean that there are many ways in which a transition can be less good than expected, but only one in which can be better, to introduce an *asymmetry* and prioritize positive TD-errors more than negative ones of equal magnitude, because the former are more likely to be informative. Such an asymmetry in replay frequency was also observed in rat studies (Singer & Frank, 2009). Again, our preliminary experiments with such variants were inconclusive.

The evidence from neuroscience suggest that a prioritization based on episodic return rather than expected learning progress may be useful too Atherton et al. (2015); Ólafsdóttir et al. (2015); Foster & Wilson (2006). For this case, we could boost the replay probabilities of entire episodes, instead of transitions, or boost individual transitions by their observed return-to-go (or even their value estimates).

For the issue of preserving sufficient diversity (to prevent overfitting, premature convergence or impoverished representations), there are alternative solutions to our choice of introducing stochasticity, for example, the priorities could be modulated by a novelty measure in observation space. Nothing prevents a hybrid approach where some fraction of the elements (of each minibatch) are sampled according to one priority measure and the rest according to another one, introducing additional diversity. An orthogonal idea is to increase priorities of transitions that have not been replayed for a while, by introducing an explicit *staleness bonus* that guarantees that every transition is revisited from time to time, with that chance increasing at the same rate as its last-seen TD-error becomes stale. In the simple case where this bonus grows linearly with time, this can be implemented at no additional cost by subtracting a quantity proportional to the global step-count from the new priority on any update.⁴

⁴ If bootstrapping is used with policy iteration, such that the target values come from separate network (as is the case for DQN), then there is a large increase in uncertainty about the priorities when the target network is updated in the outer iteration. At these points, the staleness bonus is increased in proportion to the number of individual (low-level) updates that happened in-between.

In the particular case of RL with bootstrapping from value functions, it is possible to exploit the sequential structure of the replay memory using the following intuition: a transition that led to a large amount of learning (about its outgoing state) has the potential to change the bootstrapping target for all transitions leading into that state, and thus there is more to be learned about these. Of course we know at least one of these, namely the historic *predecessor* transition, and so boosting its priority makes it more likely to be revisited soon. Similarly to eligibility traces, this lets information trickle backward from a future outcome to the value estimates of the actions and states that led to it. In practice, we add $|\delta|$ of the current transition to predecessor transition’s priority, but only if the predecessor transition is not a terminal one. This idea is related to ‘reverse replay’ observed in rodents Foster & Wilson (2006), and to a recent extension of prioritized sweeping (van Seijen & Sutton, 2013).

Appendix B Experimental Details

B.1 Blind Cliffwalk

For the Blind Cliffwalk experiments (Section 3.1 and following), we use a straightforward Q-learning (Watkins & Dayan, 1992) setup. The Q-values are represented using either a tabular look-up table, or a linear function approximator, in both cases represented $Q(s, a) := \theta^\top \phi(s, a)$. For each transition, we compute its TD-error using:

$$\delta_t := R_t + \gamma_t \max_a Q(S_t, a) - Q(S_{t-1}, A_{t-1}) \quad (2)$$

and update the parameters using stochastic gradient ascent:

$$\theta \leftarrow \theta + \eta \cdot \delta_t \cdot \nabla_\theta Q \Big|_{S_{t-1}, A_{t-1}} = \theta + \eta \cdot \delta_t \cdot \phi(S_{t-1}, A_{t-1}) \quad (3)$$

For the linear FA case we use a very simple encoding of state as a 1-hot vector (as for tabular), but concatenated with a constant bias feature of value 1. To make generalization across actions impossible, we alternate which action is ‘right’ and which one is ‘wrong’ for each state. All elements are initialized with small values near zero, $\theta_i \sim \mathcal{N}(0, 0.1)$.

We vary the size of the problem (number of states n) from 2 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 used a fixed step-size of $\eta = \frac{1}{4}$ in all experiments.

The replay memory is 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. Given this, we can in principle learn until convergence by just increasing the amount of computation; here, convergence is defined as a mean-squared error (MSE) between the Q-value estimates and the ground truth below 10^{-3} .

B.2 Atari Experiments

B.2.1 Implementation Details

Prioritizing using a replay memory with $N = 10^6$ transitions introduced some performance challenges. Here we describe a number of things we did to minimize additional run-time and memory overhead, extending the discussion in Section 3.

Rank-based prioritization

Early experiments with Atari showed that maintaining a sorted data structure of 10^6 transitions with constantly changing TD-errors dominated running time. Our final solution was to store transitions in a priority queue implemented with an array-based binary heap. The heap array was then directly used as an approximation of a sorted array, which is infrequently sorted once every 10^6 steps to prevent the heap becoming too unbalanced. This is an unconventional use of a binary heap, however our tests on smaller environments showed learning was unaffected compared to using a perfectly sorted array. This is likely due to the last-seen TD-error only being a proxy for the usefulness of a transition and our use of stochastic prioritized sampling. A small improvement in running time came from avoiding excessive recalculation of partitions for the sampling distribution. We reused the same partition for values of N that are close together and by updating α and β infrequently. Our final implementation for rank-based prioritization produced an additional 2%-4% increase in running time and negligible additional memory usage. This could be reduced further in a number of ways, e.g. with a more efficient heap implementation, but it was good enough for our experiments.

Proportional prioritization

The ‘sum-tree’ data structure used here is very similar in spirit to the array representation of a binary heap. However, instead of the usual heap property, the value of a parent node is the sum of its children. Leaf nodes store the transition priorities and the internal nodes are intermediate sums, with the parent node containing the sum over all priorities, p_{total} . This provides an efficient way of calculating the cumulative sum of priorities, allowing $O(\log N)$ updates and sampling. To sample a minibatch of size k , the range $[0, p_{\text{total}}]$ is divided equally into k ranges. Next, a value is uniformly sampled from each range. Finally the transitions that correspond to each of these sampled values are retrieved from the tree. Overhead is similar to rank-based prioritization.

As mentioned in Section 3.4, whenever importance sampling is used, all weights w_i were scaled so that $\max_i w_i = 1$. We found that this worked better in practice as it kept all weights within a reasonable range, avoiding the possibility of extremely large updates. It is worth mentioning that this normalization interacts with annealing on β : as β approaches 1, the normalization constant grows, which reduces the effective average update in a similar way to annealing the step-size η .

B.2.2 Hyperparameters

Throughout this paper our baseline was DQN and the tuned version of Double DQN. We tuned hyperparameters over a subset of Atari games: Breakout, Pong, Ms. Pac-Man, Q*bert, Alien, Battlezone, Asterix. Table 2 lists the values tried and Table 3 lists the chosen parameters.

Hyperparameter	Range of values
α	0, 0.4, 0.5, 0.6, 0.7, 0.8
β	0, 0.4, 0.5, 0.6, 1
η	$\eta_{\text{baseline}}, \eta_{\text{baseline}} / 2, \eta_{\text{baseline}} / 4, \eta_{\text{baseline}} / 8$

Table 2: Hyperparameters considered in experiments. Here $\eta_{\text{baseline}} = 0.00025$.

	DQN	Double DQN (tuned)			
Hyperparameter	baseline	rank-based	baseline	rank-based	propo
α (priority)	0	0.5 → 0	0	0.7	
β (IS)	0	0	0	0.5 → 1	
η (step-size)	0.00025	$\eta_{\text{baseline}} / 4$	0.00025	$\eta_{\text{baseline}} / 4$	η_{ba}

Table 3: Chosen hyperparameters for prioritized variants of DQN. Arrows indicate linear annealing, where the limiting value is reached at the end of training. Note the rank-based variant with DQN as the baseline is an early version without IS. Here, the bias introduced by prioritized replay was instead corrected by annealing α to zero.

B.2.3 Evaluation

We evaluate our agents using the *human starts* evaluation method described in (van Hasselt et al., 2016). Human starts evaluation uses start states sampled randomly from human traces. The *test* evaluation that agents periodically undergo during training uses start states that are randomized by doing a random number of no-ops at the beginning of each episode. Human starts evaluation averages the score over 100 evaluations of 30 minutes of game time. All learning curve plots show scores under the test evaluation and were generated using the same code base, with the same random seed initializations.

Table 4 and Table 5 show evaluation method differences and the ϵ used in the ϵ -greedy policy for each agent during evaluation. The agent evaluated with the human starts evaluation is the best agent found during training as in (van Hasselt et al., 2016).

Evaluation method	Frames	Emulator time	Number of evaluations	Agent state
Human starts	108,000	30 mins	100	human starts
Test	500,000	139 mins	1	up to 30 no-ops

Table 4: Evaluation method comparison.

	DQN	Double DQN (tuned)			
Evaluation method	baseline	rank-based	baseline	rank-based	prob.
Human starts	0.05	0.01	0.001	0.001	
Test	0.05	0.05	0.01	0.01	

Table 5: The ϵ used in the ϵ -greedy policy for each agent, for each evaluation method.

Normalized score is calculated as in (van Hasselt et al., 2016):

$$\text{score}_{\text{normalized}} = \frac{\text{score}_{\text{agent}} - \text{score}_{\text{random}}}{|\text{score}_{\text{human}} - \text{score}_{\text{random}}|} \quad (4)$$

Note the absolute value of the denominator is taken. This only affects Video Pinball where the random score is higher than the human score. Combined with a high agent score, Video Pinball has a large effect on the mean normalized score. We continue to use this so our normalized scores are comparable.

B.3 Class-imbalanced MNIST

B.3.1 Dataset Setup

In our supervised learning setting we modified MNIST to obtain a new training dataset with a significant label imbalance. This new dataset was obtained by considering a small subset of the samples that correspond to the first 5 digits (0, 1, 2, 3, 4) and all of the samples that correspond to the remaining 5 labels (5, 6, 7, 8, 9). For each of the first 5 digits we randomly sampled 1% of the available examples, i.e., 1% of the available 0s, 1% of the available 1s etc. In the resulting dataset there are examples of all 10 different classes but it is highly imbalanced since there are 100 times more examples that correspond to the 5, 6, 7, 8, 9 classes than to the 0, 1, 2, 3, 4 ones. In all our experiments we used the original MNIST *test* dataset without removing any samples.

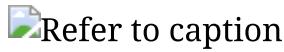


Figure 6:

The architecture of the feed-forward network used in the Prioritized Supervised Learning experiments.

B.3.2 Training Setup

In our experiments we used a 4 layer feed-forward neural network with an architecture similar to that of LeNet5 (Lecun et al., 1998). This is a 2 layer convolutional neural network followed by 2 fully connected layers at the top. Each convolutional layer is comprised of a pure convolution followed by a rectifier non-linearity and a subsampling max pooling operation. The two fully-connected layers in the network are also separated by a rectifier non-linearity. The last layer is a softmax which is used to obtain a normalized distribution over the possible labels. The complete architecture is shown

in Figure 6, and is implemented using Torch7 (Collobert et al., 2011). The model was trained using stochastic gradient descent with no momentum and a minibatch of size 60. In all our experiments we considered 6 different step-sizes (0.3, 0.1, 0.03, 0.01, 0.003 and 0.001) and for each case presented in this work, we selected the step-size that lead to the best (balanced) validation performance. We used the negative log-likelihood loss criterion and we ran experiments with both the weighted and unweighted version of the loss. In the weighted case the loss of the examples that correspond to the first 5 digits (0, 1, 2, 3, 4) was scaled by a factor of a 100 to accommodate the label imbalance in the training set described above.

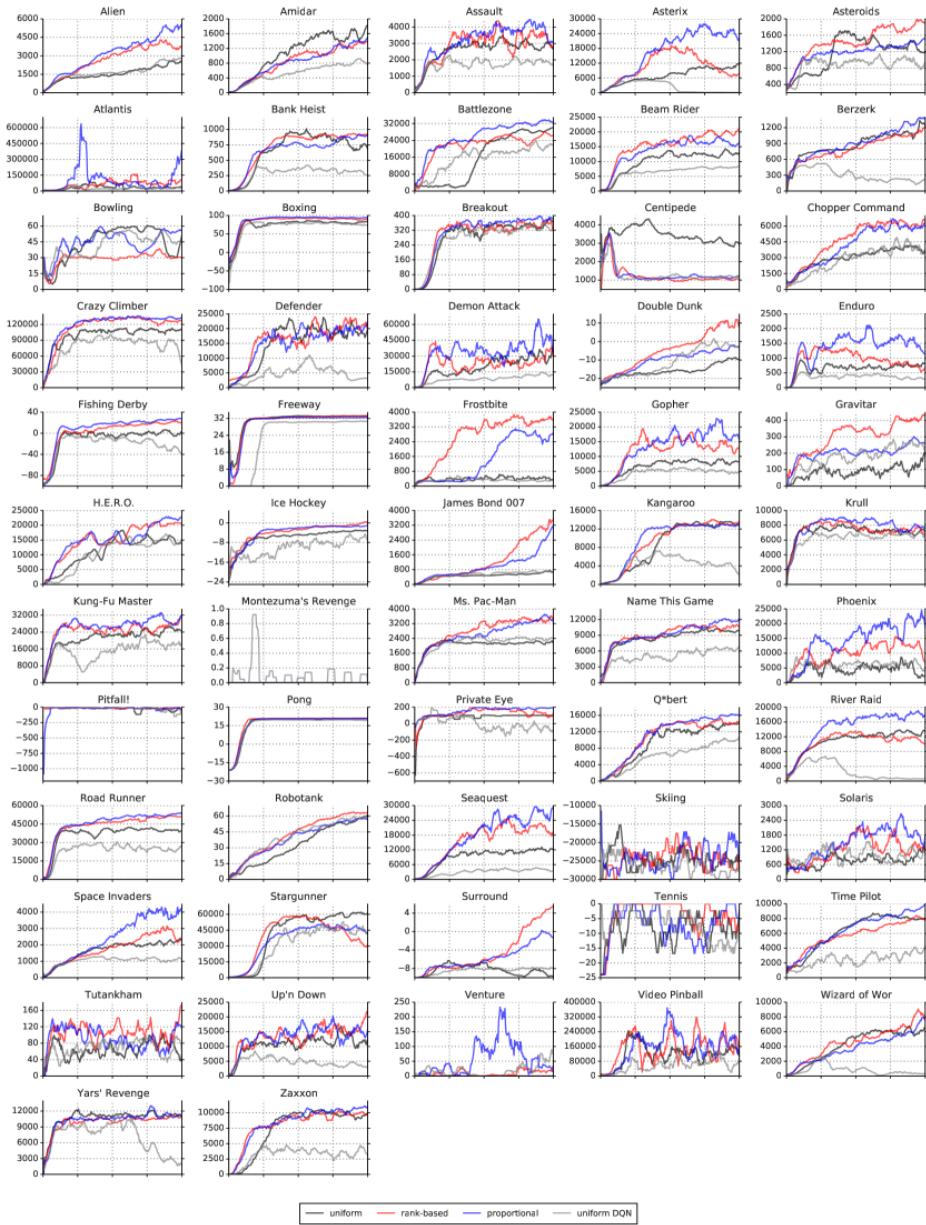


Figure 7: Learning curves (in raw score) for Double DQN (uniform baseline, in black), with rank-based prioritized replay (red), proportional prioritization (blue), for all 57 games of the Atari benchmark suite. Each curve corresponds to a single training run over 200 million unique frames, using test evaluation (see Section B.2.3), with a moving

average smoothed over 10 points. Learning curves for the original DQN are in gray. See Figure 8 for a more detailed view on a subset of these.

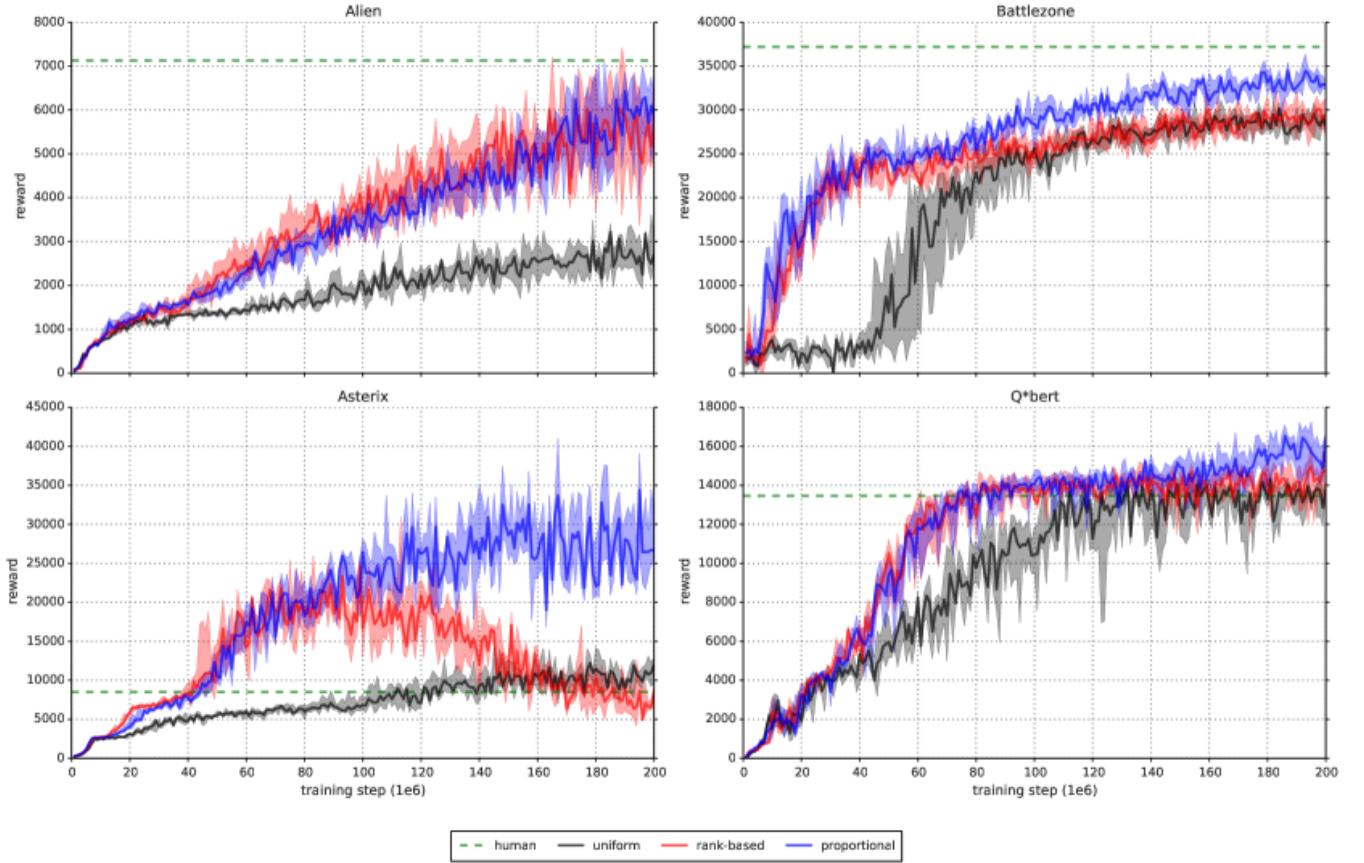


Figure 8: Detailed learning curves for rank-based (red) and proportional (blue) prioritization, as compared to the uniform Double DQN baseline (black) on a selection of games. The solid lines are the median scores, and the shaded area denotes the interquartile range across 8 random initializations. The dashed green lines are human scores. While the variability between runs is substantial, there are significant differences in final achieved score, and also in learning speed.

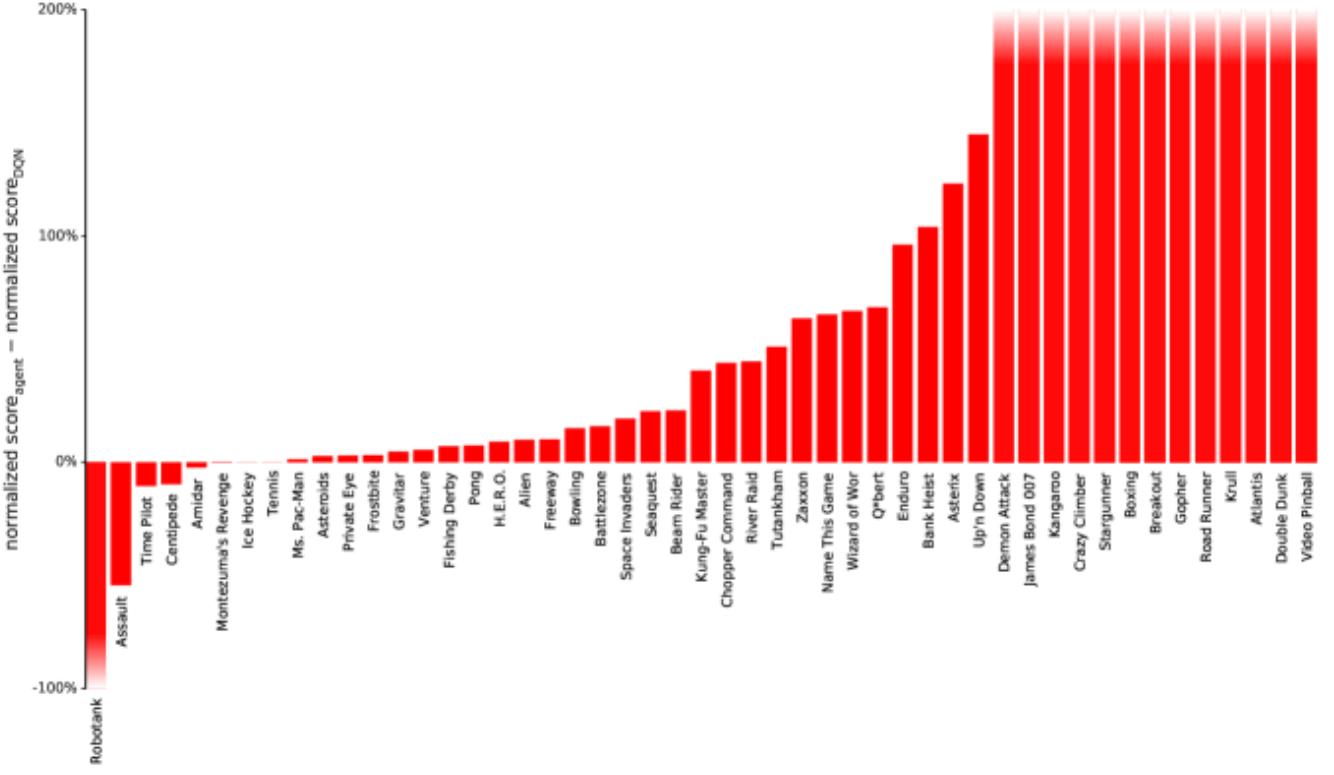


Figure 9: Difference in normalized score (the gap between random and human is 100%) on 49 games with human starts, comparing DQN with and without rank-based prioritized replay, showing substantial improvements in many games. Exact scores are in Table 6. See also Figure 3 where Double DQN is the baseline.

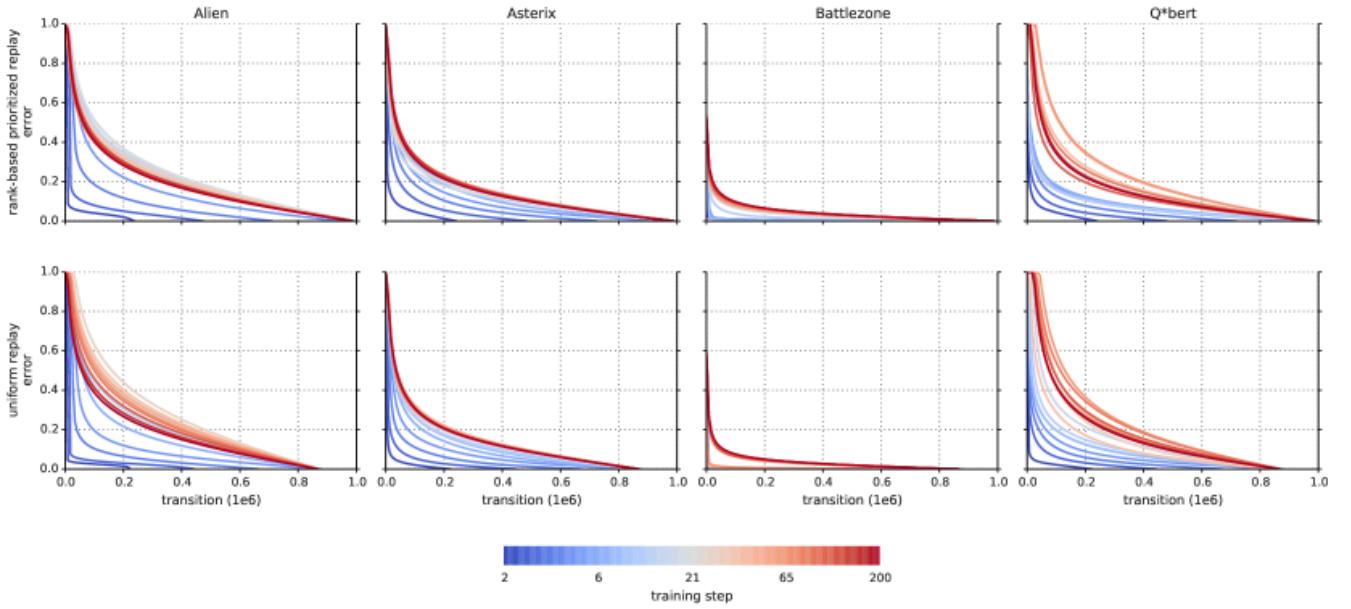


Figure 10: Visualization of the last-seen absolute TD-error for all transitions in the replay memory, sorted, for a selection of Atari games. The lines are color-coded by the time during learning, at a resolution of 10^6 frames, with the coldest colors in the beginning and the warmest toward the end of training. We observe that in some games it starts quite peaked but quickly becomes spread out, following approximately a heavy-tailed distribution. This phenomenon happens for both rank-based prioritized replay (top) and uniform replay (bottom) but is faster for prioritized replay.

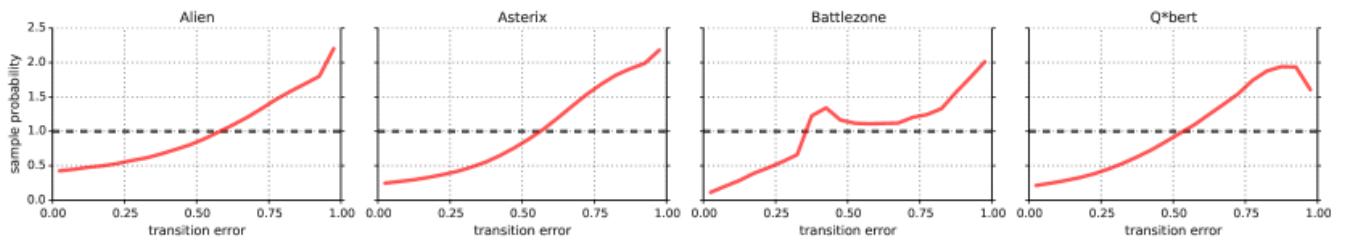


Figure 11: **Effective replay probability**, as a function of absolute TD-error, for the rank-based prioritized replay variant near the start of training. This shows the effect of Equation 1 with $\alpha = 0.7$ in practice, compared to the uniform baseline (dashed horizontal line). The effect is irregular, but qualitatively similar for a selection of games.

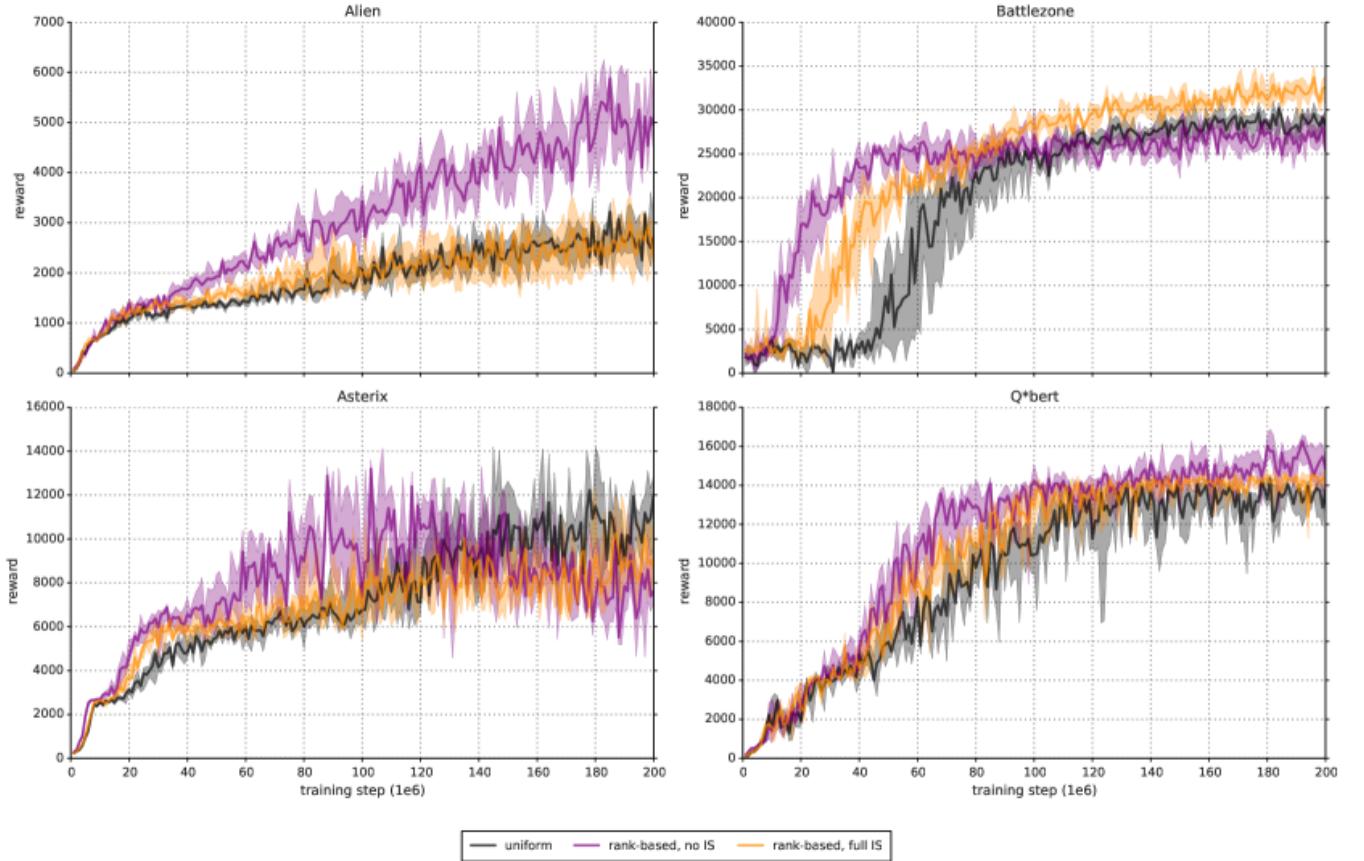


Figure 12: Effect of importance sampling: These learning curves (as in Figure 8) show how rank-based prioritization is affected by full importance-sampling correction (i.e., $\beta = 1$, in orange), as compared to the uniform baseline (black, $\alpha = 0$) and pure, uncorrected prioritized replay (violet, $\beta = 0$), on a few selected games. The shaded area corresponds to the interquartile range. The step-size for full IS correction is the same as for uniform replay. For uncorrected prioritized replay, the step-size is reduced by a factor of 4. Compared to uncorrected prioritized replay, importance sampling makes learning less aggressive, leading on the one hand to slower initial learning, but on the other hand to a smaller risk of premature convergence and sometimes better ultimate results. Compared to uniform replay, fully corrected prioritization is on average better.

	DQN	Double DQN (tuned)				
Game	baseline	rank-based	baseline	rank-based	proportional	
Alien	7%	17%	14%	19%	12%	
Amidar	8%	6%	10%	8%	14%	
Assault	685%	631%	1276%	1381%	1641%	
Asterix	-1%	123%	226%	303%	431%	
Asteroids	-0%	2%	1%	2%	2%	
Atlantis	478%	1480%	2335%	2419%	4425%	
Bank Heist	25%	129%	139%	137%	128%	
Battlezone	48%	63%	72%	75%	87%	
Beam Rider	57%	80%	117%	210%	176%	
Berzerk		22%	40%	33%	47%	
Bowling	5%	20%	31%	15%	27%	
Boxing	246%	641%	676%	665%	632%	
Breakout	1149%	1823%	1397%	1298%	1407%	
Centipede	22%	12%	23%	19%	18%	
Chopper	29%	73%	34%	48%	72%	
Command						
Crazy Climber	179%	429%	448%	507%	522%	
Defender		151%	207%	176%	155%	
Demon Attack	390%	596%	2152%	1888%	2256%	
Double Dunk	-350%	669%	981%	2000%	1169%	
Enduro	68%	164%	158%	233%	239%	
Fishing Derby	91%	98%	98%	106%	105%	
Freeway	101%	111%	113%	113%	109%	
Frostbite	2%	5%	33%	83%	69%	
Gopher	120%	836%	728%	1679%	2792%	
Gravitar	-1%	4%	-2%	1%	-1%	
H.E.R.O.	47%	56%	55%	80%	78%	
Ice Hockey	58%	58%	71%	93%	85%	
James Bond 007	94%	311%	161%	1172%	1038%	
Kangaroo	98%	339%	421%	458%	384%	
Krull	283%	1051%	590%	598%	653%	
Kung-Fu Master	56%	97%	146%	153%	151%	

Montezuma's Revenge	1%	0%	0%	1%	-0%
Ms. Pac-Man	4%	5%	7%	11%	11%
Name This Game	73%	138%	143%	173%	200%
Phoenix		270%	202%	284%	474%
Pitfall!		2%	3%	-1%	5%
Pong	102%	110%	111%	110%	110%
Private Eye	-1%	2%	-2%	0%	-1%
Q*bert	37%	106%	91%	82%	93%
River Raid	25%	70%	74%	81%	128%
Road Runner	136%	854%	643%	780%	850%
Robotank	863%	752%	872%	828%	815%
Seaquest	6%	29%	36%	63%	97%
Skiing		-122%	33%	44%	38%
Solaris		-21%	-14%	3%	2%
Space Invaders	99%	118%	191%	291%	693%
Stargunner	378%	660%	653%	689%	580%
Surround		29%	77%	103%	58%
Tennis	130%	130%	93%	110%	132%
Time Pilot	100%	89%	140%	113%	176%
Tutankham	16%	67%	63%	35%	17%
Up'n Down	28%	173%	200%	125%	313%
Venture	4%	9%	0%	7%	22%
Video Pinball	-5%	4042%	7221%	5727%	7367%
Wizard of Wor	-15%	52%	144%	131%	177%
Yars' Revenge		11%	10%	7%	10%
Zaxxon	4%	68%	102%	113%	113%

Table 6: Normalized scores on 57 Atari games (random is 0%, human is 100%), from a single training run each, using human starts evaluation (see Section B.2.3). Baselines are from van Hasselt et al. (2016), see Equation 4 for how normalized scores are calculated.

		DQN	Double DQN (tuned)					
Game	random	human		baseline	Gorila	rank-b.	baseline	r
Alien	128.3	6371.3		570.2	813.5	1191.0	1033.4	1
Amidar	11.8	1540.4		133.4	189.2	98.9	169.1	
Assault	166.9	628.9		3332.3	1195.8	3081.3	6060.8	1
Asterix	164.5	7536.0		124.5	3324.7	9199.5	16837.0	2
Asteroids	871.336517.3			697.1	933.6	1677.2	1193.2	1
Atlantis	13463.026575.0			76108.0629166.5207526.0	319688.0330			
Bank Heist	21.7	644.5		176.3	399.4	823.7	886.0	
Battlezone	3560.033030.0			17560.0	19938.0	22250.0	24740.0	2
Beam Rider	254.614961.0			8672.4	3822.1	12041.9	17417.2	3
Berzerk	196.1	2237.5				644.0	1011.1	
Bowling	35.2	146.5		41.2	54.0	58.0	69.6	
Boxing	-1.5	9.6		25.8	74.2	69.6	73.5	
Breakout	1.6	27.9		303.9	313.0	481.1	368.9	
Centipede	1925.510321.9			3773.1	6296.9	2959.4	3853.5	1
Chopper	644.0	8930.0		3046.0	3191.8	6685.0	3495.0	1
Command								
Crazy Climber	9337.032667.0			50992.0	65451.0109337.0	113782.012		
Defender	1965.514296.0					20634.0	27510.0	2
Demon Attack	208.3	3442.8		12835.2	14880.1	19478.8	69803.4	6
Double Dunk	-16.0	-14.4		-21.6	-11.3	-5.3	-0.3	
Enduro	-81.8	740.2		475.6	71.0	1265.6	1216.6	1
Fishing Derby	-77.1	5.1		-2.3	4.6	3.5	3.2	
Freeway	0.1	25.6		25.8	10.2	28.4	28.8	
Frostbite	66.4	4202.8		157.4	426.6	288.7	1448.1	1
Gopher	250.0	2311.0		2731.8	4373.0	17478.2	15253.0	3
Gravitar	245.5	3116.0		216.5	538.4	351.0	200.5	
H.E.R.O.	1580.325839.4			12952.5	8963.4	15150.9	14892.5	2
Ice Hockey	-9.7	0.5		-3.8	-1.7	-3.8	-2.5	
James Bond 007	33.5	368.5		348.5	444.0	1074.5	573.0	1
Kangaroo	100.0	2739.0		2696.0	1431.0	9053.0	11204.0	1
Krull	1151.9	2109.1		3864.0	6363.1	11209.5	6796.1	1
Kung-Fu Master	304.020786.8			11875.0	20620.0	20181.0	30207.0	3

Montezuma's Revenge	25.0	4182.0		50.0	84.0	44.0	42.0
Ms. Pac-Man	197.815375.0		763.5	1263.0	964.7	1241.3	:
Name This Game	1747.8	6796.0		5439.9	9238.5	8738.5	8960.3
Phoenix	1134.4	6686.2			16107.8	12366.5	10
Pitfall!	-348.8	5998.9			-193.7	-186.7	
Pong	-18.0	15.5		16.2	16.7	18.7	19.1
Private Eye	662.864169.1		298.2	2598.6	2202.3	-575.5	
Q*bert	183.012085.0		4589.8	7089.8	12740.5	11020.8	:
River Raid	588.314382.2		4065.3	5310.3	10205.5	10838.4	1:
Road Runner	200.0	6878.0		9264.0	43079.8	57207.0	43156.0
Robotank	2.4	8.9		58.5	61.8	51.3	59.1
Seaquest	215.540425.8		2793.9	10145.9	11848.8	14498.0	2:
Skiing	-15287.4	-3686.6			-29404.3	-11490.4	-10
Solaris	2047.211032.6				134.6	810.0	:
Space Invaders	182.6	1464.9		1449.7	1183.3	1696.9	2628.7
Stargunner	697.0	9528.0		34081.0	14919.2	58946.0	58365.0
Surround	-9.7	5.4			-5.3	-1.9	
Tennis	-21.4	-6.7		-2.3	-0.7	-2.3	-7.8
Time Pilot	3273.0	5650.0		5640.0	8267.8	5391.0	6608.0
Tutankham	12.7	138.3		32.4	118.5	96.5	92.2
Up'n Down	707.2	9896.1		3311.3	8747.7	16626.5	19086.9
Venture	18.0	1039.0		54.0	523.4	110.0	21.0
Video Pinball	20452.015641.1		20228.1112093.4214925.3	367823.729			
Wizard of Wor	804.0	4556.0		246.0	10431.0	2755.0	6201.0
Yars' Revenge	1476.947135.2				6626.7	6270.6	-
Zaxxon	475.0	8443.0		831.0	6159.4	5901.0	8593.0

Table 7: Raw scores obtained on the original 49 Atari games plus 8 additional games where available, evaluated on human starts. Human, random, DQN and tuned Double DQN scores are from van Hasselt et al. (2016). Note that the Gorila results from Nair et al. (2015) used much more data and computation, but the other methods are more directly comparable to each other in this respect.



Feeling
lucky?

Conversion
report (E)

Report
an issue

View original
on arXiv



[Copyright](#)

[Privacy Policy](#)

Generated on Sun Mar 3 09:26:44 2024 by [LATEXML](#)