Prioritizing Samples in Reinforcement Learning with Reducible Loss

Anonymous Author(s)
Affiliation
Address
email

Abstract

Most reinforcement learning algorithms take advantage of an experience replay buffer to repeatedly train on samples the agent has observed in the past. This prevents catastrophic forgetting, however simply assigning equal importance to each of the samples is a naive strategy. In this paper, we propose a method to prioritize samples based on how much we can learn from a sample. We define the learn-ability of a sample as the steady decrease of the training loss associated with this sample over time. We develop an algorithm to prioritize samples with high learn-ability, while assigning lower priority to those that are hard-to-learn, typically caused by noise or stochasticity. We empirically show that our method is more robust than random sampling and also better than just prioritizing with respect to the training loss, i.e. the temporal difference loss, which is used in vanilla prioritized experience replay.

1 Introduction

Deep reinforcement learning has shown great promise in recent years, particularly with its ability to solve difficult games such as Go Silver et al. [2016], chess Silver et al. [2018], and Atari Mnih et al. [2015]. However, online Reinforcement Learning (RL) suffers from sample inefficiency because updates to network parameters take place at every time-step with the data being discarded immediately. One of the landmarks in the space of online RL learning has been Deep Q Learning (DQN) Mnih et al. [2015], where the agent learns to achieve human-level performance in Atari 2600 games. A key feature of that algorithm was the use of batched data for online learning. Observed transitions are stored in a buffer called the *experience replay* Lin [2004], from which one randomly samples batches of transitions for updating the RL agent. This way, the agent is trained on previously visited samples to prevent catastrophic forgetting.

Instead of randomly sampling from the experience replay, we propose to sample based on the *learn-ability* of the samples. We consider a sample to be learnable if there is a potential for reducing the agent's loss with respect to that sample. We term the amount by which we can reduce the loss of a sample to be its *reducible loss* (ReLo). This is different from vanilla prioritization in Schaul et al. [2016] which just assigns high priority to samples with high loss, which can potentially lead to repeated sampling of data points which can not be learned from due to noise.

In our paper, we first briefly describe the current methods for prioritization while sampling from the buffer, followed by the intuition for reducible loss in reinforcement learning. We demonstrate the performance of our approach empirically on the DeepMind Control Suite Tassa et al. [2018], MinAtar Young and Tian [2019] and Arcade Learning Environment Bellemare et al. [2013] benchmarks. These experiments show how prioritizing based on the reducible loss is a more robust approach compared to just the loss term Schaul et al. [2016] used in Hessel et al. [2017] and that it can be integrated without adding any additional computational complexity.

Submitted to the Deep RL Workshop, NeurIPS 2022.

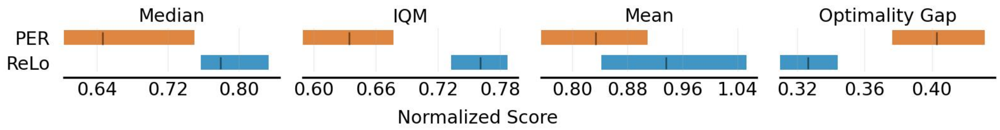


Figure 1: Performance difference between vanilla PER and ReLo aggregated across 21 benchmarks, from DMC, MinAtar and ALE suites with 5 runs each, based on proposals from Agarwal et al. [2021]. ReLo clearly outperforms PER with a higher interquartile mean (IQM) and median as well as a lower optimality gap.

2 Background and Related Work

In Reinforcement Learning (RL), an agent is tasked with maximizing the expected total reward it receives from an environment via interaction with it. This problem is formulated using a Markov Decision Process (MDP) Bellman [1957] that is described by $< \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P} >$, where $\mathcal{S}$, $\mathcal{A}$, $\mathcal{R}$ and $\mathcal{P}$ represent the state space, the action space, the reward function, and the transition function of the environment, respectively. The objective of RL is to learn an optimal policy π^*, which is a mapping from states to actions that maximizes the expected discounted sum of rewards it receives from the environment, that is

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t | S_t = s, A_t = a], \tag{1}$$

where $\gamma \in [0, 1]$ is the discount factor. Action value methods obtain a policy by learning the action value ($Q^\pi(s_t, a_t)$) of a policy which is the expected return by taking action a_t in state s_t and then following the policy π to choose further actions. This is done using the Bellman equation, which defines a recursive relationship in terms of the Q value function, as follows

$$Q^\pi(s_t, a_t) = r_t + \gamma \underset{a}{\operatorname{argmax}}\, Q^\pi(s_{t+1}, a) \tag{2}$$

The difference between the left and right sides of Eq. 2 is called the temporal difference error (TD error), and Q value methods minimize the TD error of the learned Q function Q^θ (implemented as a neural network) using stochastic gradient descent. That is, the loss for the Q network is

$$L_\theta = (Q^\theta(s_t, a_t) - (r_t + \gamma \underset{a}{\operatorname{argmax}}\, Q^\theta(s_{t+1}, a)))^2. \tag{3}$$

We can then use the Q value to implicitly represent a policy by choosing actions with high Q values. While this is easy in discrete control tasks which have a small action space, it can be difficult in continuous action spaces because finding the action that maximizes the Q value can be an optimization problem in itself. This can be computationally expensive to do at every instant, so recent methods alleviate this problem through an actor network μ_θ that learns the action that produces the maximum Q value through stochastic gradient ascent, that is

$$\mu_\theta = \underset{\theta}{\operatorname{argmax}}\, Q^\theta(s_t, \mu_\theta(s_t)). \tag{4}$$

The loss for the Q network in Eq. 3 is then modified so that the argmax is evaluated using the actor network,

$$L_\theta = (Q^\theta(s_t, a_t) - (r_t + \gamma\, Q^\theta(s_{t+1}, \mu_\theta(s_t))))^2 \tag{5}$$

2.1 Experience Replay

Online RL algorithms perform updates immediately after observing a transition. However, these not only make learning inefficient but also lead to catastrophic forgetting as some transitions can be sparsely visited. To eliminate this problem, Lin [2004] introduced experience replay, which stores the observed transitions and provides an interface to sample batches of transitions. This has been successfully used in DQN Mnih et al. [2015] to play Atari 2600 games.

Since Eqs. 3 and 5 do not require that the states and actions are generated from the current policy, algorithms trained this way are called off-policy RL algorithms. During training, data is collected from the environment and stored in a replay buffer from which mini-batches are sampled to be trained on.

A naive method of sampling is to uniformly sample all data in the buffer, however, this is inefficient because not all data is necessarily equally important. Schaul et al. [2016] proposes Prioritized Experience Replay (PER), that samples points with probabilities proportional to their TD error – which has been shown to have a positive effect on performance by efficiently replaying samples that the model has not yet learned, i.e., data points with high TD error. Each transition in the replay buffer is assigned a priority p_i, and the transitions are sampled based on this priority. To ensure that data points, even with low TD error, are sampled sometimes by the agent, instead of greedy sampling based on TD error, the replay buffer in PER stochastically samples points with probability P_i.

$$P_i = \frac{p_i^\alpha}{\sum_j p_j^\alpha} \tag{6}$$

where $\alpha \in [0, 1)$ is a hyper-parameter introduced to smoothen out very high TD errors. Setting α to 0 makes it equivalent to uniform sampling. Since sampling points non-uniformly changes the expected gradient of a mini-batch, PER corrects for this by using importance sampling (IS) weights w

$$w_i = \left(\frac{p_{uniform}}{P_i}\right)^\beta \tag{7}$$

where $\beta \in [0, 1]$ controls the amount by which the change in gradient should be corrected and $p_{uniform} = \frac{1}{N}$ where N is the number of samples in the replay buffer. The loss attributed to each sample is weighed by the corresponding w_i before the gradient is computed. In practice, β is either set to 0.5 or linearly annealed from 0.4 to 1 during training.

While PER was initially proposed as an addition to DQN-style agents, Hou et al. [2017] have shown that PER can be a useful strategy for improving performance in Deep Deterministic Policy Gradients (DDPG) Lillicrap et al. [2016]. Another recent strategy to improve sample efficiency was to introduce losses from the transition dynamics along with the TD error as the priority Oh et al. [2022]. Although this has shown improvements, it involves additional computational complexity since it also requires learning a reward predictor and transition predictor for the environment. Our proposal does not require training additional networks and hence is similar in computational complexity to vanilla PER. This makes it very simple to integrate into any existing algorithm. Wang and Ross [2019] propose an algorithm to dynamically reduce the replay buffer size during training of SAC so that the agent prioritizes recent experience while also ensuring that updates performed using newer data are not overwritten by updates from older data. However, they do not distinguish between points based on learn-ablity and only assume that newer data is more useful for the agent to learn.

2.2 Target Networks

In Eqs. 3 and 5, the target action value depends not only on the rewards but also on the value of the next state, which is not known. So, the value of the next state is approximated by feeding the next state to the same network used for generating the current Q values. As mentioned in DQN Mnih et al. [2015], this leads to a very unstable target for learning due to the frequent updates of the Q network. To alleviate this issue, Mnih et al. [2015] introduce target networks, where the target Q value is obtained from a lagging copy of the Q network used to generate the current Q value. This prevents the target from changing rapidly and makes learning much more stable. So Eqs. 3 and 5 can be suitably modified to

$$L_\theta = (Q^\theta(s_t, a_t) - (r_t + \gamma \operatorname*{argmax}_a Q^{\theta_{tgt}}(s_{t+1}, a)))^2 \tag{8}$$

and

$$L_\theta = (Q^\theta(s_t, a_t) - (r_t + \gamma\, Q^{\theta_{tgt}}(s_{t+1}, \mu_\theta(s_t))))^2, \tag{9}$$

respectively, where θ_{tgt} are the parameters of the target network, which are updated at a low frequency.

Mnih et al. [2015] copies the entire training network θ to the target network, whereas Haarnoja et al. [2018] performs a soft update, where the new target network parameters are an exponential moving average (with a parameter τ) of the old target network parameters and the online network parameters.

2.3 Off-Policy Algorithms

Off-policy algorithms are those that can learn a policy by learning from data not generated from the current policy. This improves sample efficiency by reusing data collected by old versions of the policy. This is in contrast to on-policy algorithms such as PPO Schulman et al. [2017], which after collecting a batch of data and training on it, discard those samples and start data collection from scratch. Recent state-of-the-art off-policy algorithms for continuous control include Soft Actor Critic (SAC) Haarnoja et al. [2018] and Twin Delayed DDPG (TD3) Fujimoto et al. [2018]. SAC learns two Q networks together and uses the minimum of the Q values generated by these networks for the Bellman update equation to avoid over estimation bias. The Q target update also includes a term to maximize the entropy of the policy to encourage exploration, a formulation that comes from Maximum Entropy RL Ziebart et al. [2008]. TD3 is a successor to DDPG Lillicrap et al. [2016] which addresses the overestimation bias present in DDPG in a similar fashion to SAC, by learning two Q networks in parallel, which explains the "twin" in the name. It learns an actor network μ following Eq. 4 to compute the maximum over Q values. TD3 proposes that the actor networks be updated at a less frequent interval than the Q networks, which gives rise to the "delayed" name. In discrete control, Rainbow Hessel et al. [2017] combines several previous improvements over DQN, such as Double DQN van Hasselt et al. [2016], PER Schaul et al. [2016], Dueling DQN Wang et al. [2016], Distributional RL Bellemare et al. [2017] and Noisy Nets Fortunato et al. [2018].

2.4 Reducible Loss

The work of Mindermann et al. [2022] proposes prioritized training for supervised learning tasks based on focusing on data points that reduce the model's generalization loss the most. Prioritized training keeps a held-out subset of the training data to train a small capacity model, θ_{ho} at the beginning of training. During training, this hold-out model is used to provide a measure of whether a data point could be learned without training on it. The loss of the hold-out model's prediction, $\hat{y}_{ho}$ on a data point x could be considered an estimate of the remaining loss after training on data other than (x, y), termed the *irreducible loss*. This estimate becomes more accurate as one increases the size of the held-out dataset. The difference between the losses of the main model, θ, and the hold-out model on the actual training data is called the *reducible loss*, L_r which is used for prioritizing training data in mini-batch sampling.

$$L_r = Loss(\hat{y} \mid x, \theta) - Loss(\hat{y} \mid x, \theta_{ho}) \tag{10}$$

L_r can be thought of as a measure of information gain by also training on data point (x, y).

3 Reducible Loss for Reinforcement Learning

While PER helps the agent to prioritize points that the model has not yet learned based on high TD error, we argue that there are some drawbacks. Data points could have *high* TD error because they are noisy or not learnable by the model. It might not be the case that a data point with high TD error is also a sample that the model can actually learn or get a useful signal from. Instead of prioritization based on the TD error, we propose that the agent should focus on samples that have higher *reducible* TD error. This means that instead of the TD error, we should use a measure of how much the TD error can be potentially decreased, as the priority p_i term in Eq. 6. We contend that this is better because it means that the algorithm can avoid repeatedly sampling points that the agent has been unable to learn from and can focus on minimizing error on points that are learnable, thereby improving sample efficiency. Motivated by prioritized training, we propose a scheme of prioritization tailored to the RL problem.

In contrast to supervised learning, the concepts of a hold-out dataset or model are not well defined in the RL paradigm. In Q learning based RL methods, a good proxy for the hold-out model is the target network used in the Bellman update in Eq. 8. Since the target network is only periodically updated with the online model parameters and retains the performance of the agent on older data which are trained with outdated policies. Schaul et al. [2022] demonstrates how the policies keep changing with more training even when the agent receives close to optimal rewards. Thus, the target network can be easily used as an approximation of the hold out model that was not trained on the sample. In this way, we define the Reducible Loss (ReLo) for RL as the difference between the loss of the data point with respect to the online network (with parameters θ) and with respect to the target network (with parameters θ_{tgt}). So the Reducible Loss (ReLo) can be computed as

$$\text{ReLo} = L_\theta - L_{\theta_{tgt}} \tag{11}$$

When using ReLo as p_i, there are similarities in the sampling behavior of low priority points when compared to PER. Data points that were not important under PER, i.e. they have low L_θ, will also remain unimportant in ReLo. This is because if L_θ is low, then as per Eq. 11, ReLo will also be low. This ensures that we retain the desirable behavior of PER, which is to not repeatedly sample points that have already been learned.

However, there is a difference in sampling points that have high TD error. PER would assign high priority to data points with high TD error, regardless of whether or not those data points are noisy or unlearnable. For example, a data point can have a high TD error which continues to remain high even after being sampled several times due to the inherent noise of the transition itself, but it would continue to have high priority with PER. Thus, PER would continue to sample it, leading to inefficient learning. But, its priority should be reduced since there might be other data points that are worth sampling more because they have useful information which would enable faster learning. The ReLo of such a point would be low because both L_θ and $L_{\theta_{tgt}}$ would be high. In case a data point is forgotten, then the L_θ would be higher than $L_{\theta_{tgt}}$, and the ReLo would ensure that these points are revisited.

3.1 Implementation

The probability of sampling a data point is related to the priority through Eq. 6 and requires the priority to be non-negative. Since Q value methods use the mean-squared error (MSE) loss, the priority is guaranteed to be non-negative. However, ReLo computes the difference between the MSE losses and it does not have the same property. Hence, we should create a mapping f_{map} for the ReLo error that is monotonically increasing and non-negative for all values. In practice, we found that clipping the negative values to zero, followed by adding a small ϵ to ensure samples had some minimum probability, worked well. That is, $p_i = max(ReLo, 0) + \epsilon$. This is not the only way we can map the negative values and we have studied one other mapping in Sec. 4.4. ReLo is not computationally expensive since it does not require any additional training. It only involves one additional forward pass of the states through the target network. This is because the Bellman backup (i.e., the right hand side of Eq. 2) is the same for L_θ and $L_{\theta_{tgt}}$. The only additional term that needs to be computed for ReLo is $Q_{tgt}(s_t, a_t)$ to compute $L_{\theta_{tgt}}$.

In our implementation, we saw a negligible change in the computational time between PER and ReLo. ReLo also does not introduce any additional hyper-parameters that need to be tuned and works well with the default hyper-parameters of α and β in vanilla PER. An important point to note is that ReLo does not necessarily depend on the exact loss formulation given in Eq. 8 and can be used with the loss function L_θ^{alg} of any off-policy Q value learning algorithm. In order to use ReLo, we only have to additionally compute L^{alg} with respect to the target network parameters θ_{tgt}. Our experiments also show that ReLo is robust to the target network update mechanism, whether it is a hard copy of online parameters at a fixed frequency (as in DQN Mnih et al. [2015], and Rainbow Hessel et al. [2017]) or if the target network is an exponential moving average of the online parameters (as in Soft Actor Critic Haarnoja et al. [2018]).

Algorithm 1 Computing ReLo for prioritization

Given off-policy algorithm A with loss function L^{alg}, online Q network parameters θ, target Q network parameters θ_{tgt}, replay buffer B, max priority p_{max}, ReLo mapping f_{map}, epsilon priority ϵ, training timesteps T, gradient steps per timestep T_{grad}, batch size b.
for t in 1, 2, 3, . . . T **do**
  Get current state s_t from the environment
  Compute action a_t from the agent
  Store the transition $< s_t, a_t, r_t, s_{t+1} >$ in the replay buffer B with priority p_{max}.
  for steps in 1, 2, 3, . . . T_{grad} **do**
    Sample minibatch of size b from replay buffer
    Compute the loss L_θ^{alg} and update the agent parameters θ
    Compute $L_{\theta tgt}^{alg}$ and calculate ReLo as per Eq. 11
    Update priorities of the samples in mini-batch with the newly computed ReLo values as f_{map}(ReLo$_i$) $+ \epsilon$
  end for
  Update target network following the original RL algorithm A
end for

4 Results

We study the effectiveness of ReLo on several continuous and discrete control tasks. For continuous control, we evaluate on 9 environments from the DeepMind Control (DMC) benchmark Tassa et al. [2018] as they present a variety of challenging robotic control tasks, with high dimensional state and action spaces. For discrete control, we use the MinAtar suite Young and Tian [2019] which consists of visually simpler versions of games from the Arcade Learning Environment (ALE) Bellemare et al. [2013]. The goal of MinAtar is to provide a benchmark that does not require the vast amounts of compute needed for the full ALE evaluation protocol, which involves training for 200M frames usually for 5 runs per game. This can be prohibitively expensive for researchers and thereby the MinAtar benchmark reduces the barriers present in studying deep RL research. We include scores on a few games from the ALE benchmark for a reduced number of steps to observe if there are signs of improvement when using ReLo over PER. We provide full training curves for each environment in the supplementary material.

In addition to the per environment scores and training curves, we report metrics aggregated across environments based on recommendations from Agarwal et al. [2021] in Fig. 2. They treat performance across runs as a random variable and suggest that authors report statistical measures on these random variables. The mean and the median in Fig. 2 are the respective measures of the random variables. The interquartile mean (IQM) computes the mean of the middle 50% of runs while the optimality gap is a measure of how far an algorithm is from optimal performance aggregated across environments[1]. In the DMC benchmark, the optimal score for each environment is 1000, while we use the highest reported scores for each environment from the MinAtar paper for calculating the optimality gap for the benchmark. For the ALE benchmark, we normalize the scores of each game with respect to reported random and human level scores, i.e. norm score $= \frac{\text{score}-\text{random}}{\text{human}-\text{random}}$.

We also aggregated the normalized scores across benchmarks and show the IQM and optimality gap of ReLo and PER in Fig. 1. The scores are aggregated across 21 environments (9 from DMC, 5 from MinAtar, and 7 from ALE) and 5 seeds. We can clearly see that ReLo has a significantly higher IQM with a smaller interval. This highlights the generality of ReLo since it performs better than PER across a diverse set of tasks.

4.1 DMC

In the continuous control tasks, Soft Actor Critic (SAC) Haarnoja et al. [2018] is used as the base off-policy algorithm to which we add ReLo. SAC has an online and an exponential moving average target Q network which we use to generate the ReLo priority term as given in Eq. 11. For comparison, we also include SAC with vanilla PER to showcase the differences in performance characteristics of PER and ReLo. The results are given in Table 1 and Fig. 2. On 6 of the 9 environments, ReLo outperforms the baseline SAC as well as SAC with PER. There is also a general trend where PER leads to worse performance when compared to the baseline algorithm, in line with previous work by Wang and Ross [2019] who show that the addition of vanilla PER to SAC hurts performance. However, this is not the case when using ReLo as a prioritization scheme. This trend in performance is visible in the aggregated scores in Fig. 2 where ReLo has a higher mean, median and IQM score along with a lower optimality gap when compared to SAC and SAC with PER.

Table 1: Comparison of PER and ReLo on the DMC benchmark

	Baseline	PER	ReLo
cheetah run	761.9 ± 112.3	**831.9 ± 38.9**	660.3 ± 141.2
finger spin	966.7 ± 29.3	975.4 ± 6.7	**978.8 ± 14.4**
hopper hop	**264.7 ± 37.8**	217.4 ± 113.7	247.8 ± 51.0
quadruped run	612.7 ± 143.9	496.4 ± 216.0	**833.9 ± 81.0**
quadruped walk	831.9 ± 74.3	766.3 ± 200	**942.6 ± 9.7**
reacher easy	**983.1 ± 2.7**	981.6 ± 6.3	979.1 ± 11.0
reacher hard	955.1 ± 38.5	935.1 ± 47.9	**956.8 ± 38.7**
walker run	759.1 ± 23.9	755.5 ± 64.3	**795.1 ± 42.5**
walker walk	943.7 ± 30.2	957.4 ± 8.2	**963.3 ± 5.0**

[1]Lower optimality gap is better.

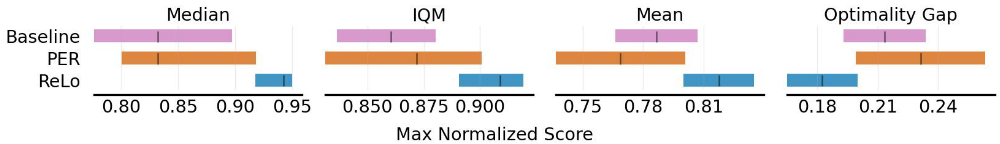


Figure 2: Metrics aggregated across 9 environments and 5 seeds in DMC based on proposed metrics from Agarwal et al. [2021]

4.2 MinAtar

In the MinAtar benchmark, we use DQN Mnih et al. [2015] as a baseline algorithm and compare its performance with PER and ReLo on the 5 environments in the benchmark. DQN does not have a moving average target Q network and instead performs a hard copy of the online network parameters to the target network at a fixed interval. Similar to the implementation of ReLo in SAC, we use the online and hard copy target Q network in the ReLo equation for calculating priorities. The results on the benchmark are given in Table 2 and Fig. 3. Vanilla PER performs poorly on Seaquest and SpaceInvaders, with scores lower than the baseline DQN. These results are consistent with observations by Obando-Ceron and Castro [2021] which analysed the effect of the components of Rainbow in the MinAtar environment. In contrast, ReLo consistently outperforms PER and is comparable to or better than the baseline. Our previous observation that ReLo tends to help improve performance in situations where PER hurts performance is also true here.

Table 2: Comparison of PER and ReLo on the MinAtar benchmark

	Baseline	PER	ReLo
Asterix	12.5 ± 1.0	$\mathbf{16.2 \pm 1.0}$	16.1 ± 0.5
Breakout	$\mathbf{9.4 \pm 0.2}$	8.9 ± 0.7	$\mathbf{9.4 \pm 0.8}$
Freeway	52.8 ± 0.3	52.8 ± 0.2	$\mathbf{53.2 \pm 0.4}$
Seaquest	16.1 ± 2.8	6 ± 1.9	$\mathbf{19.5 \pm 0.6}$
Space Invaders	$\mathbf{45.4 \pm 1.6}$	37.4 ± 4.4	39.4 ± 3.1

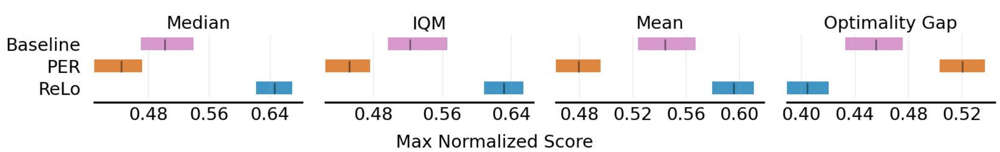


Figure 3: Metrics aggregated across 5 environments and 5 seeds in MinAtar based on proposed metrics from Agarwal et al. [2021]

4.3 ALE

As an additional test, we modified the Rainbow Hessel et al. [2017] algorithm, which uses PER by default, to instead use ReLo as the prioritization scheme and compared it against vanilla Rainbow on a subset of environments from the ALE benchmark. Instead of the usual 200M frames of evaluation, we trained each agent for 2M frames to study if there are gains that can be observed in this compute-constrained setting. As shown in Fig. 4 and Table 3, we see that Rainbow with ReLo achieves better performance than vanilla Rainbow in nearly all the tested environments. These experiments show the versatility of ReLo as a prioritization scheme.

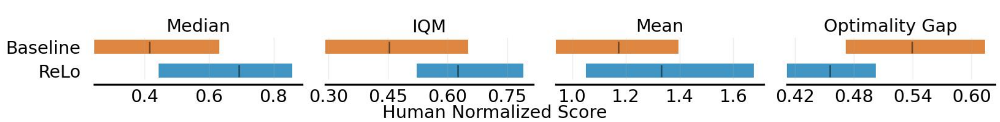


Figure 4: Metrics aggregated across 7 environments and 5 seeds in the ALE Benchmark based on proposed metrics from Agarwal et al. [2021]

Table 3: Comparison of Rainbow with PER and Rainbow with ReLo on the ALE benchmark

	Rainbow w/ PER	Rainbow w/ ReLo
Alien	1217.2 $\pm$ 207.2	**1544.0** $\pm$ **685.6**
Amidar	**445.3** $\pm$ **47.3**	393.7 $\pm$ 111.7
Assault	**2531.5** $\pm$ **444.7**	2506.9 $\pm$ 683.9
BankHeist	452.8 $\pm$ 131.2	**525.4** $\pm$ **201.3**
Frostbite	1842.0 $\pm$ 1450.5	**3366.4** $\pm$ **1613.7**
Jamesbond	663.0 $\pm$ 429.6	**851.0** $\pm$ **580.6**
Seaquest	1412.8 $\pm$ 402.6	**1755.2** $\pm$ **262.0**

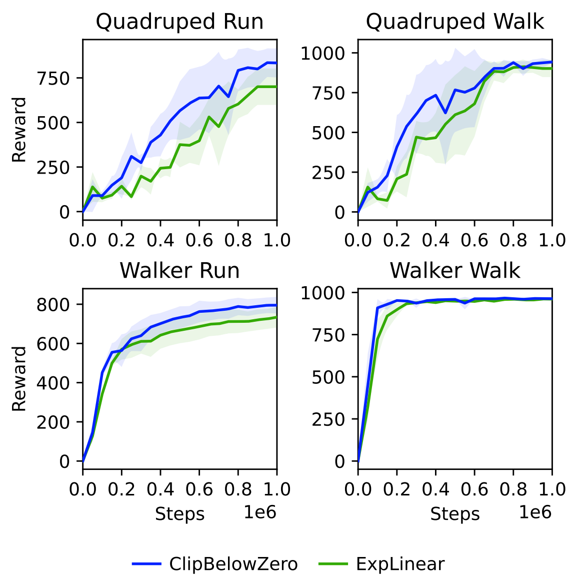


Figure 5: Comparison of different mapping functions from ReLo to p_i on a subset of environments from the DMC benchmark. Performance is evaluated for 10 episodes over 3 seeds.

4.4 Mapping functions for ReLo

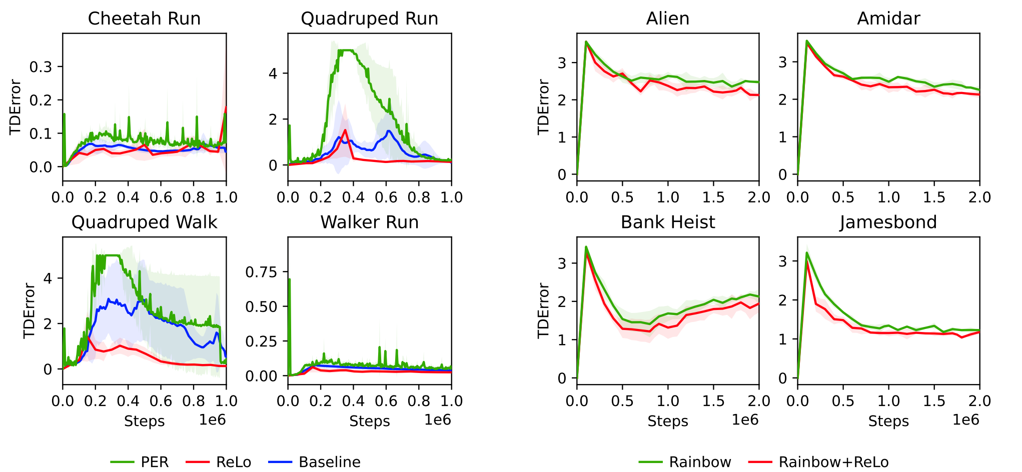
Prioritized experience replay buffers expect the priorities assigned to data points to be non-negative. While the MSE version of the TD error used in vanilla PER satisfies this constraint, ReLo does not. Therefore, there must be a non-negative, monotonically increasing mapping from ReLo to p_i. In the main experiments above we clipped negative ReLo values to zero. Another mapping we tried was to set $p_i = e^{ReLo}$, in which case the probability of sampling a data point P_i, from Eq. 6, corresponds to the softmax over ReLo scores. However, for this choice the priority would explode if the ReLo crossed values above 40 which happened occasionally during the initial stages of learning in Rainbow. The second mapping function candidate was exponential when ReLo is negative and linear otherwise, that is,

$$f_{ExpLinear} = \begin{cases} e^{\text{ReLo}} & \text{if ReLo} < 0 \\ \text{ReLo} + 1 & \text{otherwise} \end{cases} \tag{12}$$

The linear portion is shifted so that the mapping is smooth around ReLo = 0. As shown in Fig. 5, ExpLinear performs worse compared to just clipping ReLo below zero. When the ReLo values during training are analysed, we observe that the average of ReLo values (before the mapping) tends to be positive, so clipping does not lead to a large loss in information.

4.5 Analysis of TD Loss Minimization

To verify if using ReLo as a prioritization scheme leads to lower loss values during training, we logged the TD error of each agent over the course of training and these loss curves are presented in Figs. 6b and 6a. As we can see, ReLo does indeed lead to lower TD errors, empirically validating our



(a) (b)

Figure 6: Comparison of temporal difference loss curves for a) DMC and b) ALE. ReLo achieves lower loss compared to the baseline and PER, showing that ReLo is able to prioritize samples with reducible loss. Dark line represents the mean and the shaded region is the standard deviation over 3 seeds.

claims that using ReLo helps the algorithm focus on samples where the loss can be reduced. Another interesting point is that in Fig. 6a, SAC with PER has the highest reported TD errors throughout training. This is due to PER prioritizing data points with high TD error, however, as we noted these points need not necessarily be learnable. But since they have higher TD error, they repeatedly keep getting sampled making the overall losses during training higher. ReLo addresses this issue and is able to sample those data points which can be readily learned from, leading to the lowest TD errors during training.

5 Conclusion

In this paper, we have proposed a new prioritization scheme for experience replay, Reducible Loss (ReLo), which is based on the principle of frequently sampling data points that have potential for loss reduction. We obtain a measure of the reducible loss through the difference in loss of the online model and a hold-out model on a data point. In practice, we use the target network in Q value methods as a proxy for a hold-out model.

ReLo avoids the pitfall that comes with naively sampling points based only on the magnitude of the loss since having a high loss does not imply that the data point is actually learnable. While alleviating this issue, ReLo retains the positive aspects of vanilla PER, thereby improving the performance of deep RL algorithms. This has been empirically verified on both continuous and discrete control tasks using a variety of algorithms: SAC, DQN, and Rainbow. It is very simple to implement, requiring just the addition of a few lines of code to vanilla PER. It is also general and can be applied to any off-policy algorithm and is agnostic to the choice of target network update mechanism. Since it requires only one additional forward pass through the target network, the computational cost of ReLo is minimal, and there is very little overhead in integrating it into an algorithm.

While the reducible loss can be intuitively reasoned about and has been tested empirically, future work should theoretically analyse the sampling differences between ReLo and vanilla PER about the kind of samples that they tend to prioritize or ignore. This deeper insight would allow us to find flaws in how we approach non-uniform sampling in deep RL algorithms similar to work done in Fujimoto et al. [2020].

References

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021.

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

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

Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5): 679–684, 1957. URL `http://www.jstor.org/stable/24900506`.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Matteo Hessel, Ian Osband, Alex Graves, Volodymyr Mnih, Rémi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg. Noisy networks for exploration. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL `https://openreview.net/forum?id=rywHCPkAW`.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 10-15 Jul 2018. URL `https://proceedings.mlr.press/v80/fujimoto18a.html`.

Scott Fujimoto, David Meger, and Doina Precup. An equivalence between loss functions and non-uniform sampling in experience replay. *Advances in neural information processing systems*, 33: 14219–14230, 2020.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR, 2018. URL `http://proceedings.mlr.press/v80/haarnoja18b.html`.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Daniel Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298, 2017. URL `http://arxiv.org/abs/1710.02298`.

Yuenan Hou, Lifeng Liu, Qing Wei, Xudong Xu, and Chunlin Chen. A novel ddpg method with prioritized experience replay. In *Systems, Man, and Cybernetics (SMC), 2017 IEEE International Conference on*, pages 316–321. IEEE, 2017.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL `http://arxiv.org/abs/1509.02971`.

Longxin Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8:293–321, 2004.

Sören Mindermann, Jan M Brauner, Muhammed T Razzak, Mrinank Sharma, Andreas Kirsch, Winnie Xu, Benedikt Höltgen, Aidan N Gomez, Adrien Morisot, Sebastian Farquhar, and Yarin Gal. Prioritized training on points that are learnable, worth learning, and not yet learnt. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 15630–15649. PMLR, 17–23 Jul 2022. URL `https://proceedings.mlr.press/v162/mindermann22a.html`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

Johan Samir Obando-Ceron and Pablo Samuel Castro. Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 1373–1383. PMLR, 2021. URL http://proceedings.mlr.press/v139/ceron21a.html.

Youngmin Oh, Jinwoo Shin, Eunho Yang, and Sung Ju Hwang. Model-augmented prioritized experience replay. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=WuEiafqdy9H.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.05952.

Tom Schaul, André Barreto, John Quan, and Georg Ostrovski. The phenomenon of policy churn, 2022. URL https://arxiv.org/abs/2206.00730.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv: Arxiv-1707.06347*, 2017.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. doi: 10.1038/nature16961. URL https://doi.org/10.1038/nature16961.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. doi: 10.1126/science.aar6404. URL https://www.science.org/doi/abs/10.1126/science.aar6404.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. Deepmind control suite. *arXiv preprint arXiv: Arxiv-1801.00690*, 2018.

Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In Dale Schuurmans and Michael P. Wellman, editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 2094–2100. AAAI Press, 2016. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12389.

Che Wang and Keith Ross. Boosting soft actor-critic: Emphasizing recent experience without forgetting the past. *arXiv preprint arXiv: Arxiv-1906.04009*, 2019.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In Maria-Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1995–2003. JMLR.org, 2016. URL http://proceedings.mlr.press/v48/wangf16.html.

Denis Yarats and Ilya Kostrikov. Soft actor-critic (sac) implementation in pytorch. https://github.com/denisyarats/pytorch_sac, 2020.

Kenny Young and Tian Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019.

Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

A Implementation Details

We build our experiments on top of existing implementations of SAC, DQN and Rainbow. For the DeepMind Control Suite experiments, we modify Yarats and Kostrikov [2020], adding a prioritized replay buffer and the ReLo version. We use an open source implementation of Rainbow[2] for the Arcade Learning Environment and the DQN implementation from the MinAtar authors. Aside from the collected frames and number of seeds, we have not modified any of the hyper-parameters from these original implementations. The hyper-parameters as well as hardware and software used are given in Table 4.

Table 4: Hyper-Parameters of all experiments

Environments	Algorithm	Algorithm Parameters	Hardware & Software
ALE	Rainbow	Frames = 2×10^6 seeds = 5 Remaining hyper-parameters same as Hessel et al. [2017]	Hardware- CPU: 6 Intel Gold 6148 Skylake GPU: 1 NVidia V100 RAM: 32 GB Software- Pytorch: 1.10.0 Python: 3.8
DeepMind Control Suite	SAC	Frames = 1×10^6 seeds = 5 Remaining hyper-parameters same as Haarnoja et al. [2018]	Hardware- CPU: 6 Intel Gold 6148 Skylake GPU: 1 NVidia V100 RAM: 32 GB Software- Pytorch: 1.10.0 Python: 3.8
MinAtar	DQN	Frames = 5×10^6 seeds = 5 Remaining hyper-parameters same as Mnih et al. [2015]	Hardware- CPU: 6 Intel Gold 6148 Skylake GPU: 1 NVidia V100 RAM: 32 GB Software- Pytorch: 1.10.0 Python: 3.8

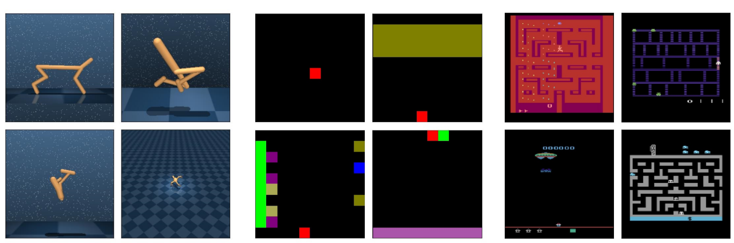


Figure 7: Visualization of a few environments from each benchmark. Left to right: DeepMind Control Suite, MinAtar, Arcade Learning Environment

B DeepMind Control Suite

We choose 9 environments from the DeepMind Control Suite Tassa et al. [2018] for testing the performance of ReLo on continuous control tasks. Each agent was trained on proprioceptive inputs from the environment for 1M frames with an action repeat of 1. The training curves for the baselines and ReLo are given in Fig. 8.

[2] https://github.com/Kaixhin/Rainbow

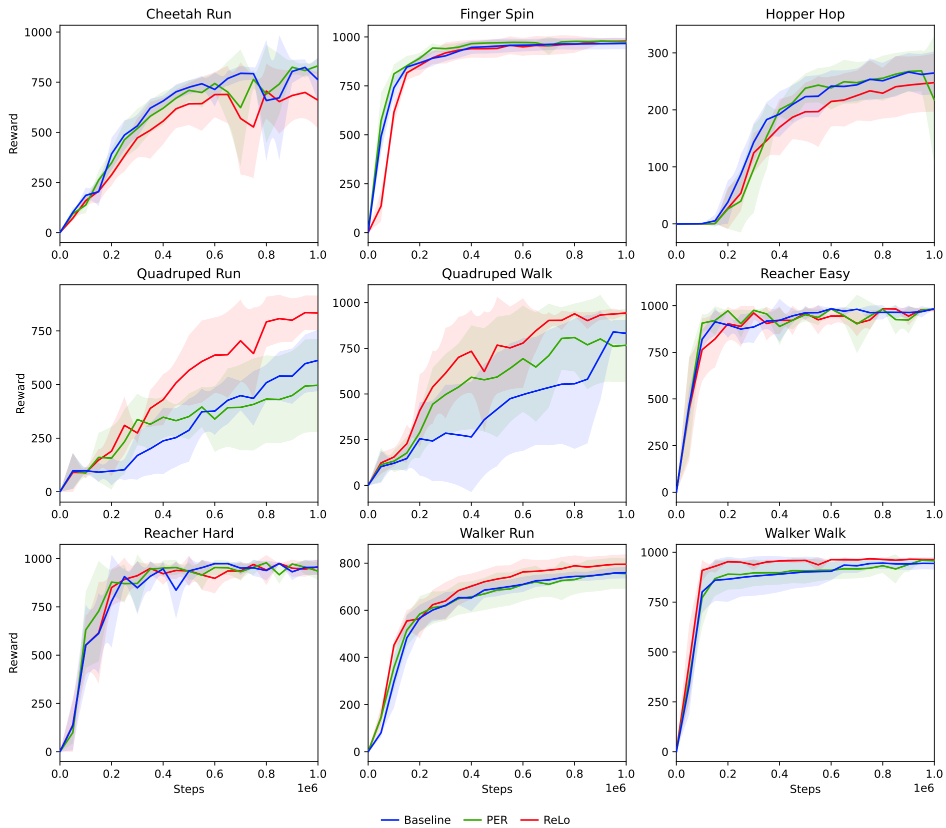


Figure 8: Training curves of environments from the DeepMind Control Suite. Performance is evaluated for 10 episodes over 5 random seeds.

C MinAtar

We evaluate the baselines against all 5 environments in the MinAtar suite Young and Tian [2019]. A visualization of a few environments from the suite is presented in Fig. 7. Each agent receives the visual observations from the environment and is trained for 5M frames following the evaluation methodology outlined in Young and Tian [2019]. The training curves are given in Fig. 9.

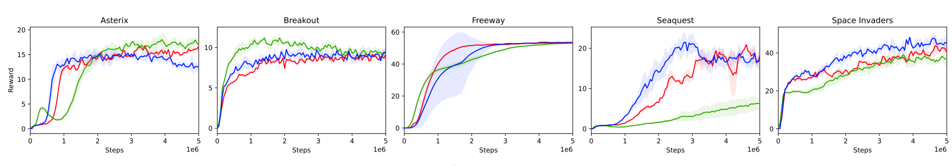


Figure 9: Training curves of environments from the MinAtar benchmark. Performance is evaluated using a running average over the last 1000 episodes over 5 random seeds.

D Arcade Learning Environment

We evaluate agents on a compute-constrained version of the Arcade Learning Environment Bellemare et al. [2013], training each agent for 2M frames. We chose a subset of 7 games from the suite for our evaluation. ReLo performs on par or better than vanilla PER Schaul et al. [2016] in each environment. The training curves are given in Fig. 10.

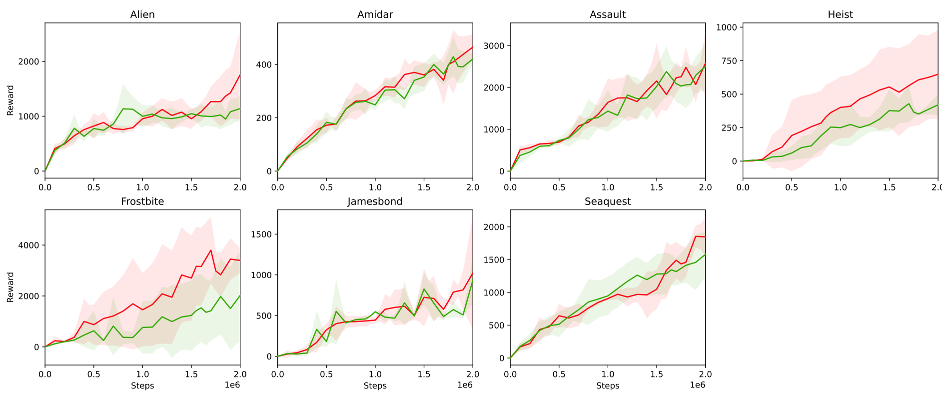


Figure 10: Training curves of 7 environments from the ALE benchmark. Performance is evaluated for 10 episodes over 5 random seeds.