Prioritizing States with Action Sensitive Return in Experience Replay

Alexander Keijzer[*]
Cognitive Robotics
Delft University of Technology
Delft, The Netherlands
alexanderkeijzer@gmail.com

Bas van der Heijden[*]
Cognitive Robotics
Delft University of Technology
Delft, The Netherlands
d.s.vanderheijden@tudelft.nl

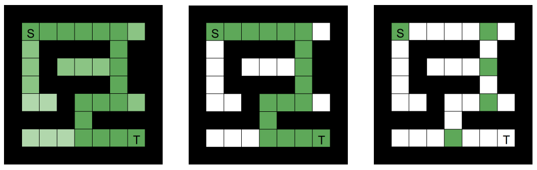
Jens Kober
Cognitive Robotics
Delft University of Technology
Delft, The Netherlands
j.kober@tudelft.nl

Abstract

Experience replay for off-policy reinforcement learning has been shown to improve sample efficiency and stabilize training. However, typical uniformly sampled replay includes many irrelevant samples for the agent to reach good performance. We introduce Action Sensitive Experience Replay (ASER), a method to prioritize samples in the replay buffer and selectively model parts of the state-space more accurately where choosing sub-optimal actions has a larger effect on the return. We experimentally show that this can make training more sample efficient and that similar performance can be reached with smaller parametric function approximators – like neural networks with few neurons – in environments where they would otherwise struggle.

1 Introduction

Reinforcement learning aims to find a policy that maximizes cumulative reward. This is often done through learning a compact representation of the value of states or state-action pairs in an environment by interacting with it. However, environment interactions can be costly and therefore necessitate efficient data use. Lin's introduction of experience replay in reinforcement learning [13] promotes efficient use and reuse of collected experience. A replay buffer stores transition samples from environment interactions, aiding sample efficiency and stabilizing training [14, 15]. Typically, learning of the value function in off-policy reinforcement learning is approached like a supervised learning problem where stochastic gradient descent is used to train



(a) Replay buffer data
(b) Only on-policy buffer data
(c) Only decision point buffer data

Figure 1: Toy maze environment where an agent moves from start S to termination T. The agent can choose which adjacent square to move to but can only move back to the square where it came from if there is no other valid action. The states sampled from a buffer of transitions are shown in green. After convergence, with normal experience replay the buffer will likely be filled with states from the whole maze. Previous work focused function approximator expressiveness on relevant states by prioritizing on-policy data. We introduce a method to prioritize "decision points" for the agent to further focus this expressiveness.

[*]These authors contributed equally to this work

function approximations such as neural networks. In supervised learning, data is usually sampled uniformly, however, for imbalanced datasets there are techniques to change the sampling distribution [7]. With reinforcement learning, we have a similar issue as not every sample is equally relevant for learning a good policy. Lambert et al. [12] showed that in model-based reinforcement learning, a model that better explains the transition data (with higher likelihood) does not necessarily allow a better policy to be found on it. There is an objective mismatch when training. This is also the case for a value function in off-policy reinforcement learning with function approximators. A lower value function error on the contents of a uniformly sampled buffer does not necessarily mean a better policy will result from it.

When using localized representations for the value network, such as in tabular reinforcement learning, replaying these irrelevant samples leads to computational overhead, but it does not impact the accuracy in the vital parts of the state-action space. However, many popular off-policy algorithms use global parametric function approximators like neural networks instead. Updates in irrelevant parts of the state-action space then yield a global effect, as these are often trained with stochastic gradient descent and minimize the mean square error across a batch of experiences [25]. Particularly in the case of smaller function approximators, such as neural networks with few neurons, the limited expressiveness that could have been allocated to model performance-critical areas is spent on inconsequential regions of the state-action space.

Off-policy algorithms may therefore, despite their ability to learn from experiences under different policies, still be influenced by the sampling distribution and buffer content when using global function approximators. To illustrate this, consider an extreme case: an agent would not see any performance improvement if it is trained using a replay buffer that contains states that the agent would never come across during rollouts. Other works have limited replay of states and/or actions unlikely under the current policy, the "off-policyness", which will reduce the replay of irrelevant samples and can improve performance [17, 23, 24]. However, just increased sampling of on-policy data does not prioritize what is really relevant: the importance of modeling these experiences accurately for the agent's performance during evaluation. A toy example that shows how we would ideally focus the approximators' expressiveness is shown in Fig. 1. In this example, an optimal policy could be learned with data from only 5 states while it is likely the replay buffer will contain data from all 30 states.

In this paper, we define a modeling importance criterion that measures sensitivity on return of taking a suboptimal action and introduce Action Sensitive Experience Replay (ASER), a method to change the replay distribution to match this criterion for off-policy reinforcement learning algorithms that use a state-action value function. This allows us to focus the expressiveness of function approximators on important parts of the state-space. We argue that while fitting the observation distribution (i.e. minimizing the total modeling error of the data collected by the agent along its rollouts) might seem logical, it is likely not the best strategy to achieve the maximum return efficiently. Since this leaves deprioritized parts of the state-space with lower accuracy, we use n-step returns to bootstrap only to states that are adequately modeled. We experimentally show that we find sample efficiency and/or final performance gains: (i) in a simple case, such as the maze in Fig. 1 where importance is given, this effective reduction of the state-space allows for significant improvements, (ii) when we transfer a learned importance criterion from a previously trained policy, (iii) in some cases, when learning the importance criterion during training.

2 Related Work

There have been several different proposals for non-uniform sampling or reweighting of replay buffers. Some, like CER [27], add extra samples to a uniform sampled batch while others change the sampling distribution altogether. Most techniques are aimed to satisfy one of the following four objectives.

Firstly, a common aim is to reduce the off-policyness of the selected samples for replay. This may improve learning speed and stability. Previous works estimate the off-policyness using importance weights [17] or multiple buffers [23] and then clip gradients or reweight experiences. In [24], off-policyness is instead reduced by sampling multiple batches and taking the most similar state distribution. Possible drawbacks of selecting mostly on-policy data are reduced robustness to policy or environment changes. Our approach does not prioritize on-policyness specifically, however, these approaches may work well together by further reducing the amount of modeled states.

Another objective may be to prioritize samples that are modeled poorly by the value function. Previous works [4, 22] propose prioritizing the replay of experiences based on their TD-error, which is analogous to how unexpected the transition is to the value function. This integrates new experience faster and, like our approach, will prioritize states where the value function is volatile but can potentially focus the limited expressiveness on irrelevant parts of the state-space (reducing the final performance and convergence rate).

Heuristics can also be used to enforce good coverage of transition dynamics. For example, [20] introduce prioritized sampling based on Shannon's entropy of the state space vector. Other variants include selecting on reward [10] and state coverage [9]. Some of these works may do the opposite of our approach and the works described before, as, instead of focusing replay, they often enforce a broader coverage of the state space. In some cases, this may be preferred to increase robustness to environment changes, avoid catastrophic collapse, or for transfer learning.

Lastly, instead of using rules-based strategies to do prioritization, a learning-based approach to select experience from the buffer can be used. In [18, 26], training a replay policy to maximize the increase in cumulative reward by selecting transitions to train the agents' policy on is proposed. An interesting observation in these works is that these learning methods prefer replaying recent, likely on-policy, data. These methods, however, cannot learn a prioritization similar to our approach as the learning methods do not have the information needed as input.

Next to what the goal of prioritization is previous works differ in how they implement prioritization. The approach used by some works discussed here [17, 23] is reweighting errors to, for example, take larger update steps for more prioritized data. Instead, the sampling distribution can also be changed so more prioritized samples are seen more often [4, 20, 22]. With very large batch sizes and many update steps per new sample, these approaches will converge to effectively the same. Since this is often not the case we use the latter approach, however, reweighting may work too.

Most works discussed here prioritize samples from a first-in-first-out buffer. Instead, works such as [9] change in what order data is removed from the replay buffer as they often have a limited capacity. While this may also benefit our approach, we only change how a fixed-size buffer is sampled, not how data is removed.

Finally, our method makes use of n-step returns for off-policy reinforcement learning algorithms. A common way of accounting for the bias towards off-policy data introduced by n-step returns is by using importance sampling or tree backup [25]. We do not use these approaches as we argue our method naturally limits the impact of this bias (see Sect. 3.3).

3 Action Sensitive Experience Replay (ASER)

ASER aims to focus the expressiveness of function approximators used in off-policy reinforcement learning by changing the sampling distribution of the replay buffer. To do this, our method consists of three additions to standard algorithms. Firstly, a formal definition of the modeling importance criterion based on the Q-function of the algorithm is given in Sect. 3.1. Then, this criterion is used to change the sampling distribution as described in Sect. 3.2. Lastly, due to the changed sampling distribution, we need to skip over some states when bootstrapping as described in Sect. 3.3. In Sect. 3.4 we show the implementation of ASER for SAC [8].

3.1 Modeling Importance Criterion

The modeling importance could be formalized in many ways, depending on reward distribution, action authority, etc. What is explored here is a measure to find areas of the state-action space where choosing a wrong action greatly affects the expected return for continuous state and action spaces. We define a function $p\colon \mathbb{R}^n \to \mathbb{R}^+$ that maps the state space s to the modeling importance. A possible modeling importance could be the norm of the second derivative of the Q-function to the action a at the optimal action. A highly negative concavity at the peak of the Q-function gives a local proxy for the rate of decrease in expected return when choosing a sub-optimal action. For multi-dimensional actions, a matrix norm of the Hessian could be used instead. However, there are multiple issues with using this definition as the modeling importance. It is often not trivial to find the peak of the Q-function in continuous domains. Taking the estimated optimal action in actor-critic methods instead, for example, may not exactly coincide with the peak of the Q-function and therefore give inaccurate values. Even if the peak of the Q-function could be found, the concavity is a local metric. The second-order derivatives may be large while the total value decrease is small.

Therefore, a less local metric for this value loss is needed. A solution to this is to approximate the Q-function over actions for every state with a tractable function where we could use metrics like entropy or variance. Fig. 2 shows examples of Q-functions where we would like a high difference in importance. The assumption here, however, is that the reinforcement learning algorithm used will model the reduction in expected return accurately. From experiments, we found that this is not generally the case. For example, we see that with ϵ-greedy exploration, DQN may find what the best action is but will likely not accurately model the value of selecting a sub-optimal action. On the other hand, maximum entropy reinforcement learning algorithms, such as SAC [8], have a built-in incentive to find the sensitivity of their actions and perform more randomly in parts of the state space where the effect on the expected return is smaller.

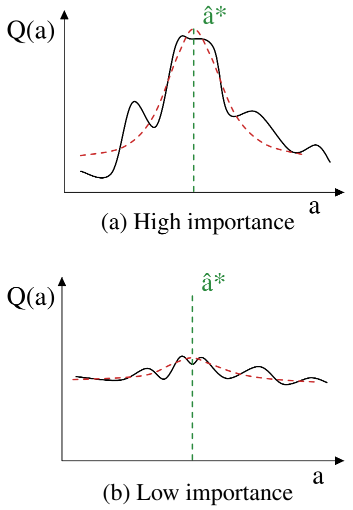


Figure 2: Visualization of Q-functions over actions where $\hat{a}^*$ is the estimate of the optimal action according to the policy. A good importance metric differentiates these, even though metrics like a second derivative may locally be similar. The dotted red line shows an approximation of this function, where a scaled entropy or variance could be a good metric, for example.

In this paper, we investigate the effectiveness of using the learned entropy of a policy trained via SAC as the criterion for modeling importance. We motivate this choice in Sec. 3.4.

3.2 Sampling Prioritization

The modeling importance criterion is used to select what priority samples should be selected for replay. Similarly to PER [22], the selection is stochastic depending on the criterion and we introduce a hyperparameter α that scales the prioritization of samples. The chance of selecting a sample i is:

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

Sweeping the entire buffer to select data is too computationally expensive, therefore their prioritization will be updated with the current criterion when they are sampled. New experience samples are introduced into the buffer with the maximum prioritization that exists inside the buffer. This prioritizes integrating new experiences quickly, although this effect is limited.

Since PER's prioritization criterion is based on the TD-error, which depends on what next state the transition ends up in, a bias will be introduced that needs to be annealed during training. PER does this using importance sampling. The modeling importance criterion we introduce only depends on the current state, which is independent of any stochasticity in the environment, which means we do not need this correction. We change the distribution of states that is replayed compared to the distribution of states that is seen by the agent during rollouts, this will introduce a bias in modeling errors but not in value convergence.

3.3 Bootstrapping

Due to prioritization, some states get replayed more frequently, causing less precise Q-function estimates in less-replayed areas. In such cases with our prioritization scheme, states where actions minimally impact expected returns are less accurately estimated. In essence, lower accuracy should not be a problem, precisely because actions have minimal effect here. However, it can still cause issues due to how off-policy algorithms typically use bootstrapping to update the Q-function. The commonly used TD(0) update rule has the form

$$Q(\mathbf{s}_t, \mathbf{a}_t) \leftarrow Q(\mathbf{s}_t, \mathbf{a}_t) + \beta(\underbrace{r_t + \gamma Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})}_{Q_{\text{targ}}(\mathbf{s}_t, \mathbf{a}_t)} - Q(\mathbf{s}_t, \mathbf{a}_t)), \tag{2}$$

with learning rate β, discount rate γ, state $\mathbf{s}_t$, action $\mathbf{a}_t$, reward r_t received after transitioning from state $\mathbf{s}_t$, and action $\mathbf{a}_{t+1}$ usually derived from "greedy" policy $\bar{\pi}(\mathbf{s}_{t+1})$. This rule adjusts the Q-function towards the target $Q_{\text{targ}}(\mathbf{s}_t, \mathbf{a}_t)$, an estimate of the true $Q(\mathbf{s}_t, \mathbf{a}_t)$ value. Bootstrapping helps lower variance and speeds up convergence. However, it also introduces a bias due to any estimation error in $Q(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})$. If not addressed, our prioritization scheme could worsen this bias. Prioritized states that transition to unimportant ones may use these unimportant states' inaccurate Q-function estimates as bootstrap targets, hindering the convergence of important states to their true values. Fig. 3 illustrates this, where important (green) states are separated by unimportant (white) ones. Since the unimportant states are rarely sampled, their Q-function values remain static and are likely inaccurate. Using these inaccurate Q-function values for bootstrapping introduces considerable bias into the Q-function updates of important states.

To counteract this effect, we propose an adaptive n-step returns scheme. An n-step returns scheme extends the TD(0) update rule in Equation (2) by aggregating rewards over n future steps, instead of using only the immediate reward, as defined in

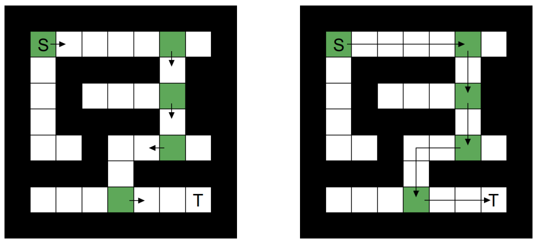
$$Q_{\text{targ}}(\mathbf{s}_t, \mathbf{a}_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ... + \gamma^{n-1} r_{t+n-1} + \gamma^n Q(\mathbf{s}_{t+n}, \mathbf{a}_{t+n}), \tag{3}$$

with action $\mathbf{a}_{t+n}$ usually derived from "greedy" policy $\bar{\pi}(\mathbf{s}_{t+n})$. Rather than bootstrapping to the immediate next state, which may be unimportant and inaccurately modeled, we unroll n-steps forward until we encounter a well-modeled state, as visualized in Fig. 3. Since it is difficult to know if a state is accurately modeled, we use our importance criterion as a proxy to gauge the expected accuracy of its Q-function. Thus, a state qualifies as a bootstrap target if its importance criterion exceeds a threshold b. As a result, n is not constant but varies with each experience sample. A maximum bootstrap length h limits the unrolling distance if many transitions in a trajectory have low importance.

Selecting a fixed threshold b for every task is challenging and requires tuning. Instead, we dynamically set the threshold based on the importance criterion distribution in an unprioritized sample batch. Specifically, we set b to the k-th percentile of the importance criterion distribution. Hence, a state is deemed important if its importance criterion surpasses k percent of states in the sampled batch. During training, we start with a low percentile k and gradually increase it, recognizing that early importance metrics may be unreliable. The increment in k is governed by a bounded linear function,

$$k = \min(k_m, k_s s), \tag{4}$$

where k_m is the maximum percentile, k_s is the slope, and s denotes the number of training timesteps.



(a) Standard bootstrapping (b) Importance n-step bootstrapping

Figure 3: Bootstrapping visualized with arrows in the toy maze environment. White states, which are not modeled accurately due to the prioritized sampling, are skipped when boostrapping to avoid pulling this badly modeled data into the well-modeled green states.

It is important to note this n-step bootstrapping will cause the bootstrap target to be dependent on the policy that collected it. A different policy may have taken a different trajectory by taking different actions and have collected different rewards. This will result in a bias towards older policies that are still in the buffer with off-policy reinforcement learning algorithms. There are ways to solve this, for example, with importance sampling or tree backup [25]. We argue, however, that this effect is limited since the policy dependency is only in areas where actions have a limited effect on the expected return. As soon as a state is encountered where a different policy would have a significant effect, it is used to bootstrap instead. A different policy would therefore only have a limited effect on the value of the bootstrap target.

3.4 Combining ASER with SAC

We integrate ASER with the Soft Actor-Critic (SAC) algorithm [8], using SAC's learned policy entropy as our importance criterion. SAC is an off-policy algorithm that optimizes a stochastic policy. A central feature of SAC is entropy regularization. The policy is trained to maximize a trade-off between expected return and entropy, a measure of randomness in the policy. This encourages the agent to explore actions around its perceived optimal strategy, provided that this exploration does not significantly compromise the expected return. If being very accurate is crucial for reaching its goal, the agent will act in a more predictable way, with low randomness. However, if the agent can act

Algorithm 1: ASER

```
Input: Replay buffer $\mathcal{D}$ of samples $T = (\mathbf{s}, \mathbf{a}, r, \mathbf{s}', d)$ with prioritization $p$, training timesteps $t$
Parameters : Maximum horizon $h$, maximum target percentile $k_m$, target percentile slope $k_s$,
             discount factor $\gamma$
1  for each gradient step do
2      Sample with $P(i) = p_i^\alpha / \sum_j p_j^\alpha$ a batch of transitions $B = \{(\mathbf{s}, \mathbf{a}, r, \mathbf{s}', d)_i\}$ from $\mathcal{D}$;
3      $b \leftarrow$ top $k$-th percentile of $p(\mathbf{s}')$ for every $T \in B$, where $k = \min(k_m, k_s s)$;
4      for each $T = (\mathbf{s}, \mathbf{a}, r, \mathbf{s}', d)_i \in B$ do
5          $n \leftarrow 1$;
6          while $p(\mathbf{s}') < b$ and not $d$ and $n < h$ do
7              $(\hat{r}, \hat{\mathbf{s}}', \hat{d}) \leftarrow$ from next transition in trajectory from buffer $\mathcal{D}$;
8              Modify $T$ with: $r \leftarrow r + \gamma^n \hat{r}$;   $\mathbf{s}' \leftarrow \hat{\mathbf{s}}'$;   $d \leftarrow \hat{d}$;
9              $n \leftarrow n + 1$;
10         end
11     end
12     Normal gradient step with batch $B$ of modified transitions $T$ with bootstrap discount $\gamma^n$;
13     Update in $\mathcal{D}$ prioritization $p$ for every $T \in B$ with $p(\mathbf{s})$;
14 end
```

freely in certain situations without hurting its overall performance, it will do so. Hence, if the agent finds that its actions in a certain state have no impact on the expected return, it will act completely random. We argue that such states are less important for an agent to replay, primarily because the agent's behavior in these states does not impact overall performance; therefore, there is little to learn. This rationale motivates our choice to use policy entropy as the modeling importance criterion in ASER. Specifically, we use the probability of the mode of SAC's Gaussian policy as a proxy for the policy's entropy, defined as

$$p(\mathbf{s}) = \pi(\mathbf{a}^*|\mathbf{s}) \tag{5}$$

where $\mathbf{a}^*$ is the greedy (or mode) of the policy π. It provides a scalar metric for multi-dimensional actions. Moreover, this can be efficiently determined through a single evaluation of the policy, unlike metrics that necessitate the computationally expensive calculation of the Q-function's Hessian. For consistency, we additionally incorporate the entropy terms from SAC's Q-value targets into the rewards specified in Equation (3) when employing n-step returns.

The complete algorithm for ASER is described in Algorithm 1. ASER can be used as a replacement for uniform sampling used in other reinforcement learning algorithms with a suitable importance criterion. We use a replay buffer of transition tuples state $\mathbf{s}$, actions $\mathbf{a}$, reward r, next state $\mathbf{s}'$ and end of episode (or done flag) d with prioritization p. For every gradient step, ASER modifies the batch of transitions before they are used in the normal gradient step of the algorithm. The only change that needs to be done to the algorithm it is implemented on is that any bootstrapping will need to be discounted according to the bootstrap length used by ASER, so with γ^n instead of simply γ. Other than updating the prioritization, the buffer contents are not modified.

4 Results

In the following section, we introduce the setup of our experiments and the Maze environment in Sect. 4.1 and then show the effect of ASER on this environment in Sect. 4.2. Results that show the dependency on reward function or environment definition and an ablation study of the sampling prioritization and n-step bootstrapping are given in Sect. 4.3.

4.1 Maze environment and experiment setup

We test ASER within a maze four time larger than that represented in Fig. 1 and 3. In this environment, the agent must move from start S to termination T. The agent can move to adjacent squares, but may only return to the square it came from if there is no other valid action, i.e., when reaching a dead-end. Therefore, in many states, the chosen action will have no effect, as every action leads to the same next state, but in some choosing the wrong action will result in the agent eventually reaching a dead end. This means that the neural network only needs to model these decision-sensitive areas and can ignore the other states. SAC is continuous in state and action space and this environment is discrete. Therefore, states are mapped uniformly onto a continuous axis between 0 and 1. For the action space, four valid actions [UP, DOWN, LEFT, RIGHT] are defined and mapped to the values [0, 0.25, 0.5, 0.75]. The agent can opt for an action as a continuous value within the range [0, 1], which is then mapped to the nearest valid action. For instance, should the agent choose a value of 0.4 with only [UP, RIGHT] as valid options, the outcome will be a RIGHT action. We initially show the maze with a reward function where a penalty is applied every time the agent reaches a dead end. Since the reward function has a significant effect on ASER, in Sect. 4.3 we also show the effect when the reward is given when reaching the end of the maze linearly decreasing with the timesteps spent to get to the end.

To show the effectiveness of only learning action-sensitive parts of the state space, we also compare the effect of our method with prior information about the sensitivity of actions. To do this, we create an "oracle" that has perfect information about the sensitivity of actions. This oracle is used instead of the modeling importance metric $p(\mathbf{s})$ from Equation (5), where a decision point has an importance of 1 and all other states have an importance of 0. Another way to use prior information is to use the importance criterion from a previously trained policy instead of learning it while training.

We build upon the algorithm implementations of Stable Baselines 3 [19]. To create a fair comparison we run hyperparameter optimization for every algorithm on the shown environments. For this, we use Optuna [2] and maximize the sum of the average total reward of multiple rollouts in the evaluation environment during training to optimize for speed of convergence and final performance. Tables with all hyperparameters, both optimized values and SB3 defaults, can be found in Appendix A. We present all the sample efficiency graphs according to the recommendations of Agarwal et al. [1], using the interquartile mean with a 95% stratified bootstrap confidence interval from 16 training runs.

4.2 ASER on Maze environment

In Fig. 4 we show the sample efficiency curves for SAC with standard experience replay, Prioritised Experience Replay, ASER with a learned selection criterion, and ASER with the oracle. All of these agents use a small, 24-neuron policy and critic network. SAC with standard replay struggles to reliably converge in the Maze environment with these small networks, even after hyperparameter optimization. It appears that at some point additional training samples no longer improve the agent's performance. A possible reason for this is that too much of the limited expressiveness of the neural network is used to model parts of the state-space with no impact on performance, as actions do not affect return there. Filling the replay buffer with more data will not be beneficial, as the neural network needs to filter out portions of these data to effectively incorporate it. When ASER has access to the oracle it only replays and bootstraps to decision points in the maze. With this oracle, we see a large increase in sample efficiency.

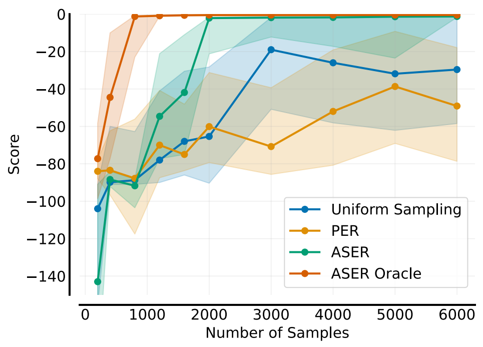


Figure 4: Sample efficiency comparison between SAC with standard replay, PER, ASER with an online learned importance criterion, and ASER with an importance criterion from an "oracle" on the Maze environment with a small, 24-neuron actor and critic network

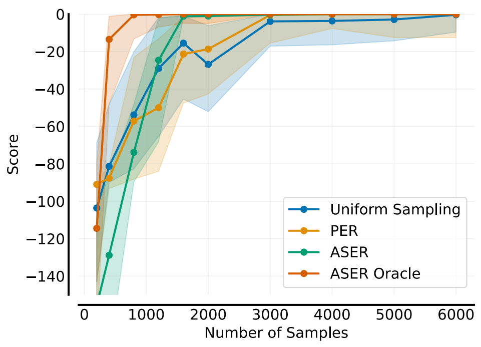


Figure 5: Sample efficiency comparison between SAC with uniform sampling, PER, and ASER with an online learned importance criterion on the Maze environment with a two-layer 256-neuron actor and critic network.

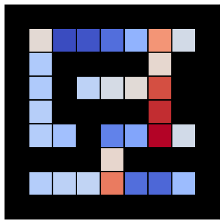


Figure 6: Sampling count relative to uniform with ASER in the Maze environment where blue is less sampling and red is more sampling than uniform.

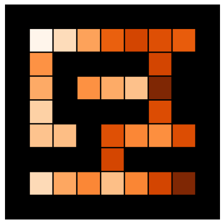


Figure 7: Bootstrap distance with ASER for the Maze environment where darker is a shorter distance. E.g. in the top row, bootstrapping distance decreases with the distance to the decision point.

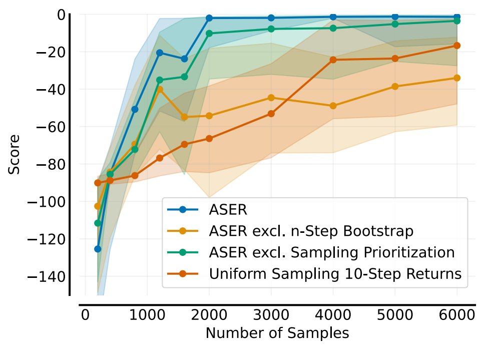


Figure 8: Sample efficiency comparison between ablations of SAC ASER on the Maze environment.

When the modeling importance criterion is not given, and therefore learned while training, we still see an improvement in performance against uniform sampling. ASER more quickly and consistently converges to the best policy. Since the importance now needs to be estimated by interacting with the environment performance picks up later compared to the oracle. PER struggles in this environment, with performance slightly below uniform sampling. Fig. 5 shows that when training a larger network of two layers of 256 neurons there is still a significant sample efficiency, stability, and final performance gain for ASER compared to standard experience replay and PER, but the effect is smaller than with a small network. In general, this increased expressiveness allows for more sample-efficient and stable learning compared to the smaller network.

In Fig. 6 and 7 we show, respectively, the sampling prioritization and the variable step bootstrapping of ASER with learned importance. In the best case, these should reflect Fig. 1c and 3b. We see very similar behavior in the sampling, where decision points have been learned and are prioritized, although with some effect on nearby states. The bootstrapping figure is less clear but gives a similar picture, especially in the bottom and top rows it is clear that it is bootstrapping to decision points and termination as the distance lowers the closer we get to these points.

4.3 Ablation and limitations

Sample efficiency curves for ablation of both prioritization and n-step bootstrapping with a predetermined importance criterion are shown in Fig. 8. We see that the most significant efficiency improvement comes from variable step bootstrapping, with a smaller improvement from the prioritization. Both features are needed to properly "compress" the environment and achieve stable convergence. We also show the effect of fixed 10-step returns to make sure the performance improvements shown are not only caused by this longer return. 10-step returns were chosen using parameter optimization as explained in Sect. 4.1. Somewhat surprisingly 10-step SAC with uniform sampling still performs well even though this will introduce bias in the value estimation to older policies in the buffer without importance sampling or tree backup.

In ASER, the value difference between actions is crucial to learn a good importance. This is heavily influenced by the reward function design. Fig. 9 displays identical mazes with differing

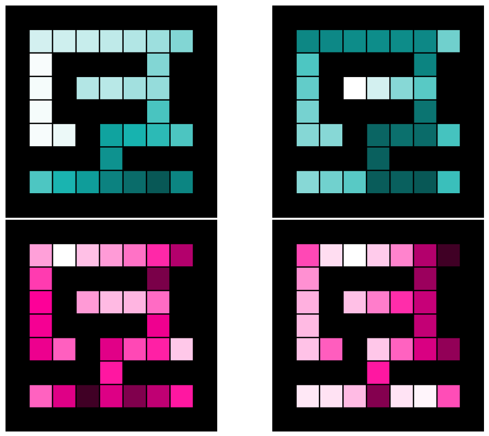

(a) Sparse reward (b) Denser reward

Figure 9: Comparison of the value function (top) and probability of the mode (bottom) between a sparse discounted reward at termination and a denser reward with a penalty each time the agent encounters a dead end. In the denser reward scenario figures (right), there is more immediate feedback resulting in a clearer difference in next state value at decision points and therefore a more accurate importance metric.

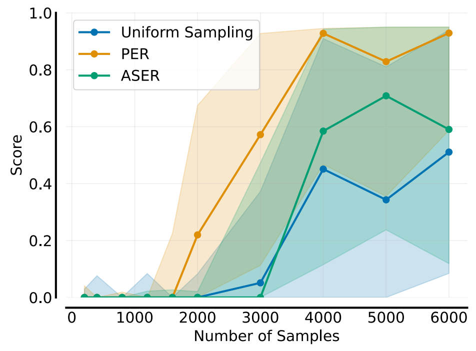


Figure 10: Sample efficiency comparison between SAC with uniform sampling, PER, and ASER with an online learned importance criterion on the Maze environment with a sparse reward function.

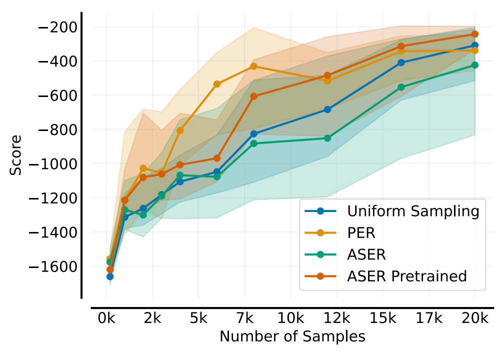


Figure 11: Sample efficiency comparison between SAC with uniform sampling, PER, ASER with an online learned importance criterion, and ASER with a pretrained importance criterion on the Pendulum environment with a small, 24-neuron actor and critic network.

rewards: one awards only at the end, the other penalizes for turning around when reaching a dead-end. In the former, ASER's performance stagnates as indicated in Fig. 10, due to a less clear differences in the importance criterion.

In the Pendulum environment from OpenAI Gym [5], performance differences are smaller, however, some increased sample efficiency can still be achieved with pretraining, as shown in Fig. 11. Online learned ASER is unable to achieve performance gains, possibly due to being unable to stably learn the importance criterion. In classical control environments, such as the shown Pendulum environment, the difference in importance between states is less clear than in the Maze environment. In these environments, the importance can also depend more on the policy. Especially when pretraining this may have a detrimental effect on performance as the newly trained policy may find a different path through the state space making different states important to the agent.

5 Conclusion

In this paper, we have presented Action Sensitive Experience Replay (ASER) which changes the experience replay distribution to prioritize states where the return is especially sensitive to non-optimal actions. This strategic reallocation of modeling resources enables parametric function approximators – such as neural networks – to focus their limited expressiveness on regions of the state-space that are most relevant for the agent's performance.

Our findings show improvements in sample efficiency, stability, and final performance, particularly when the action's sensitivity is either known in advance. When learning the sensitivity during training or carrying the sensitivity over from a previously trained policy we still observed some improvements. The improvements are more pronounced when the number of parameters in the function approximator is constrained. In order to achieve these performance improvements, our method does require environments to have rewards shaped in a way that results in clear changes in the value in some states

By focusing RL agents on key state-space areas, they can learn effective policies in larger domains without expanding function approximators. This could enhance performance with smaller neural networks, aiding in multi-task RL. For instance, our method may obviate the need for policy distillation [21]. It may also be useful in transfer learning, particularly sim-to-real transfer. A pretrained importance criterion could be more robust than a policy that is prone to simulator overfitting [6, 11, 16].

We realize that the evaluation of our method is limited to a small number of environments in this paper. A larger study may give a better overview of the implications and possible limitations of our method. As shown, environments with clear differences in importance benefit most from our method. Therefore, we expect that some environments in the Atari 2600 benchmark suite [3] would work well with our method. This, along with an implementation on a discrete action-space algorithm like DQN, would be an interesting direction for future development.

Acknowledgments and Disclosure of Funding

This work was supported by the European Union's H2020 project Open Deep Learning Toolkit for Robotics (OpenDR) under grant agreement No 871449.

References

[1] Rishabh Agarwal et al. "Deep Reinforcement Learning at the Edge of the Statistical Precipice". In: *Advances in Neural Information Processing Systems* (2021).

[2] Takuya Akiba et al. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2019.

[3] Marc G. Bellemare et al. "The Arcade Learning Environment: An Evaluation Platform for General Agents". In: *CoRR* abs/1207.4708 (2012). arXiv: `1207.4708`. URL: `http://arxiv.org/abs/1207.4708`.

[4] Marc Brittain et al. "Prioritized Sequence Experience Replay". In: *CoRR* abs/1905.12726 (2019). arXiv: `1905.12726`. URL: `http://arxiv.org/abs/1905.12726`.

[5] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: `arXiv:1606.01540`.

[6] Rodney A Brooks. "Artificial life and real robots". In: *Proceedings of the First European Conference on artificial life*. 1992, pp. 3–10.

[7] Mikel Galar et al. "A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.4 (2012), pp. 463–484.

[8] Tuomas Haarnoja et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[9] David Isele and Akansel Cosgun. "Selective experience replay for lifelong learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.

[10] Max Jaderberg et al. "Reinforcement Learning with Unsupervised Auxiliary Tasks". In: *International Conference on Learning Representations*. 2017.

[11] Nick Jakobi, Phil Husbands, and Inman Harvey. "Noise and the reality gap: The use of simulation in evolutionary robotics". In: *Advances in Artificial Life: Third European Conference on Artificial Life Granada, Spain, June 4–6, 1995 Proceedings 3*. Springer. 1995, pp. 704–720.

[12] Nathan Lambert et al. "Objective Mismatch in Model-based Reinforcement Learning". In: *Learning for Dynamics and Control*. PMLR. 2020, pp. 761–770.

[13] Long-Ji Lin. "Self-Improving Reactive Agents Based On Reinforcement Learning, Planning and Teaching". In: *Machine learning* 8 (1992), pp. 293–321.

[14] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518 (2015).

[15] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602 (2013). arXiv: `1312.5602`. URL: `http://arxiv.org/abs/1312.5602`.

[16] Stefano Nolfi and Dario Floreano. *Evolutionary robotics: The biology, intelligence, and technology of self-organizing machines*. MIT press, 2000.

[17] Guido Novati and Petros Koumoutsakos. "Remember and forget for experience replay". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 4851–4860.

[18] Youngmin Oh et al. "Learning to Sample with Local and Global Contexts in Experience Replay Buffer". In: *CoRR* abs/2007.07358 (2020). arXiv: `2007.07358`. URL: `https://arxiv.org/abs/2007.07358`.

[19] Antonin Raffin et al. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8.

[20] Mirza Ramicic and Andrea Bonarini. "Entropy-based prioritized sampling in deep Q-learning". In: *2017 2nd international conference on image, vision and computing (ICIVC)*. IEEE. 2017, pp. 1068–1072.

[21] Andrei A Rusu et al. "Policy distillation". In: *arXiv preprint arXiv:1511.06295* (2015).

[22] Tom Schaul et al. “Prioritized Experience Replay”. In: *International Conference on Learning Representations (ICLR)*. 2016.

[23] Samarth Sinha et al. “Experience replay with likelihood-free importance weights”. In: *Learning for Dynamics and Control Conference*. PMLR. 2022, pp. 110–123.

[24] Peiquan Sun, Wengang Zhou, and Houqiang Li. “Attentive experience replay”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 5900–5907.

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

[26] Daochen Zha et al. “Experience replay optimization”. In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. 2019, pp. 4243–4249.

[27] Shangtong Zhang and Richard S. Sutton. “A Deeper Look at Experience Replay”. In: *CoRR* abs/1712.01275 (2017). arXiv: `1712.01275`. URL: `http://arxiv.org/abs/1712.01275`.

A Hyperparameters

Table 1: Optimized Hyperparameters Values Maze Small Network

		Standard	PER	ASER	Oracle ASER
λ	Learning Rate	$2.480\text{e}{-2}$	$5.192\text{e}{-2}$	$1.594\text{e}{-2}$	$3.215\text{e}{-2}$
γ	Discount Factor	$9.770\text{e}{-1}$	$9.549\text{e}{-1}$	$9.358\text{e}{-1}$	$9.5267\text{e}{-1}$
τ	Target Smoothing Coefficient	$1.914\text{e}{-3}$	$9.203\text{e}{-2}$	$1.965\text{e}{-2}$	$7.399\text{e}{-2}$
α	Sample Prioritization	-	$3.414\text{e}{-1}$	$1.409\text{e}{-1}$	1.032
h	Max Bootstrap Length	-	-	6	8
k_m	Bootstrap Target Percentile Maximum	-	-	73.54%	85.06%
k_s	Bootstrap Target Percentile Slope	-	-	$5.457\text{e}{-1}$	$4.457\text{e}{-1}$

Table 2: Optimized Hyperparameters Values Maze Large Network

		Standard	PER	ASER	Oracle ASER
λ	Learning Rate	$1.439\text{e}{-2}$	$1.350\text{e}{-2}$	$3.904\text{e}{-3}$	$1.725\text{e}{-2}$
γ	Discount Factor	$9.696\text{e}{-1}$	$9.097\text{e}{-1}$	$9.827\text{e}{-1}$	$9.226\text{e}{-1}$
τ	Target Smoothing Coefficient	$1.013\text{e}{-2}$	$3.151\text{e}{-2}$	$7.655\text{e}{-2}$	$5.741\text{e}{-2}$
α	Sample Prioritization	-	$9.473\text{e}{-2}$	$3.593\text{e}{-1}$	$5.384\text{e}{-1}$
h	Max Bootstrap Length	-	-	8	11
k_m	Bootstrap Target Percentile Maximum	-	-	71.87%	39.73%
k_s	Bootstrap Target Percentile Slope	-	-	$1.739\text{e}{-2}$	$9.736\text{e}{-1}$

Table 3: Optimized Hyperparameters Values Maze Reward at Termination

		Standard	PER	ASER
λ	Learning Rate	$1.246\text{e}{-2}$	$3.106\text{e}{-2}$	$1.935\text{e}{-3}$
γ	Discount Factor	$9.599\text{e}{-1}$	$9.821\text{e}{-1}$	$9.720\text{e}{-1}$
τ	Target Smoothing Coefficient	$1.536\text{e}{-2}$	$8.151\text{e}{-2}$	$6.833\text{e}{-2}$
α	Sample Prioritization	-	$3.908\text{e}{-1}$	$1.501\text{e}{-1}$
h	Max Bootstrap Length	-	-	17
k_m	Bootstrap Target Percentile Maximum	-	-	43.31%
k_s	Bootstrap Target Percentile Slope	-	-	$6.508\text{e}{-4}$

Table 4: Optimized Hyperparameters Values Pendulum

		Standard	PER	ASER	Pretrained ASER
λ	Learning Rate	$3.281\text{e}{-2}$	$1.200\text{e}{-2}$	$2.196\text{e}{-2}$	$5.365\text{e}{-2}$
γ	Discount Factor	$9.357\text{e}{-1}$	$9.447\text{e}{-1}$	$9.244\text{e}{-1}$	$9.273\text{e}{-1}$
τ	Target Smoothing Coefficient	$7.518\text{e}{-2}$	$2.538\text{e}{-3}$	$6.764\text{e}{-2}$	$4.648\text{e}{-2}$
α	Sample Prioritization	-	$2.538\text{e}{-1}$	$1.829\text{e}{-1}$	$1.961\text{e}{-1}$
h	Max Bootstrap Length	-	-	2	1
k_m	Bootstrap Target Percentile Maximum	-	-	77.05%	69.57%
k_s	Bootstrap Target Percentile Slope	-	-	$1.444\text{e}{-3}$	$1.064\text{e}{-4}$

Table 5: Stable Baselines 3 Defaults

Replay Buffer Size	$1\text{e}6$
Env. Steps Before Learning Starts	100
Batch Size	256
Gradient Steps Per Env. Step	1
Entropy Coefficient	Learned
Target Entropy	-1