Experience selection in deep reinforcement learning for control

De Bruin, Tim; Kober, Jens; Tuyls, Karl; Babuška, Robert

Publication date
2018

Document Version
Final published version

Published in
Journal of Machine Learning Research

Citation (APA)
De Bruin, T., Kober, J., Tuyls, K., & Babuška, R. (2018). Experience selection in deep reinforcement learning for control. *Journal of Machine Learning Research*, *19*(9), Article 9. https://www.jmlr.org/papers/v19/17-131.html

Journal of Machine Learning Research 19 (2018) 1-56 Submitted 3/17; Revised 07/18; Published 08/18

Experience Selection in Deep Reinforcement Learning for Control

Tim de Bruin T.D.DEBRUIN@TUDELFT.NL
Jens Kober J.KOBER@TUDELFT.NL
Cognitive Robotics Department
Delft University of Technology
Mekelweg 2, 2628 CD Delft, The Netherlands

Karl Tuyls KARLTUYLS@GOOGLE.COM
Deepmind
14 Rue de Londres, 75009 Paris, France

Department of Computer Science
University of Liverpool
Ashton Street, Liverpool L69 3BX, United Kingdom

Robert Babuška R.BABUSKA@TUDELFT.NL
Cognitive Robotics Department
Delft University of Technology
Mekelweg 2, 2628 CD Delft, The Netherlands

Editor: George Konidaris

Abstract

Experience replay is a technique that allows off-policy reinforcement-learning methods to reuse past experiences. The stability and speed of convergence of reinforcement learning, as well as the eventual performance of the learned policy, are strongly dependent on the experiences being replayed. Which experiences are replayed depends on two important choices. The first is which and how many experiences to retain in the experience replay buffer. The second choice is how to sample the experiences that are to be replayed from that buffer. We propose new methods for the combined problem of experience *retention* and experience *sampling*. We refer to the combination as experience *selection*. We focus our investigation specifically on the control of physical systems, such as robots, where exploration is costly. To determine which experiences to keep and which to replay, we investigate different proxies for their immediate and long-term utility. These proxies include age, temporal difference error and the strength of the applied exploration noise. Since no currently available method works in all situations, we propose guidelines for using prior knowledge about the characteristics of the control problem at hand to choose the appropriate experience replay strategy.

Keywords: reinforcement learning, deep learning, experience replay, control, robotics

1. Introduction

Reinforcement learning is a powerful framework that makes it possible to learn complex nonlinear policies for sequential decision making processes while requiring very little prior knowledge. Especially the subfield of deep reinforcement learning, where neural networks are used as function approximators, has recently yielded some impressive results. Among these results are learning to play Atari games (Mnih et al., 2015) and to control robots (Levine et al., 2016) straight from raw images, as well as beating the top human player in the game of Go (Silver et al., 2016).

Reinforcement learning methods can be divided into on-policy and off-policy methods. On-policy methods directly optimize the policy that is used to make decisions, while off-policy methods can learn about an optimal policy from data generated by another policy. Neither approach is without its problems, which has motivated work on methods that combine on and off-policy updates (Wang et al., 2017; Gu et al., 2017; O'Donoghue et al., 2017).

When a reinforcement learning method is either partially or entirely off-policy, past experiences can be stored in a buffer and reused for learning. Doing so not only reduces the sample complexity of the learning algorithm, but can also be crucial for the stability of reinforcement-learning algorithms that use deep neural networks as function approximators (Mnih et al., 2015; Lillicrap et al., 2016; Schaul et al., 2016; Wang et al., 2017).

If we have access to a buffer with past experiences, an interesting question arises: how should we *sample* the experiences to be replayed from this buffer? It has been shown by Schaul et al. (2016) that a good answer to this question can significantly improve the performance of the reinforcement-learning algorithm.

However, even if we know how to sample from the experience buffer, two additional questions arise: what should the buffer capacity be and, once it is full, how do we decide which experiences should be *retained* in the buffer and which ones can be overwritten with new experiences? These questions are especially relevant when learning on systems with a limited storage capacity, for instance when dealing with high-dimensional inputs such as images. Finding a good answer to the question of which experiences to retain in the buffer becomes even more important when exploration is costly. This can be the case for physical systems such as robots, where exploratory actions cause wear or damage and risks need to be minimized (Kober et al., 2013; Garcıa and Fernández, 2015; Tamar et al., 2016; Koryakovskiy et al., 2017). It is also the case for tasks where a minimum level of performance needs to be achieved at all times (Banerjee and Peng, 2004) or when the policy that generates the experiences is out of our control (Seo and Zhang, 2000; Schaal, 1999).

We will refer to the combined problem of experience *retention* and experience *sampling* as experience *selection*. The questions of which experiences to sample and which experiences to retain in the buffer are related, since they both require a judgment on the utility of the experiences. The difference between them is that determining which experiences to sample requires a judgment on the instantaneous utility: from which experiences can the agent learn the most at the moment of sampling? In contrast, a decision on experience retention should be based on the expected long term utility of experiences. Experiences need to be retained in a way that prevents insufficient coverage of the state action space in the future, as experiences cannot be recovered once they have been discarded.

To know the true utility of an experience, it would be necessary to foresee the effects of having the reinforcement-learning agent learn from the experience at any given time. Since this is not possible, we instead investigate *proxies* for the experience utility that are cheap to obtain.

In this work, we investigate age, surprise (in the form of the temporal difference error), and the amplitude of the exploration noise as proxies for the utility of experiences. To motivate the need for *multiple* proxies, we will start by showing the performance of different experience selection methods on control benchmarks that, at first sight, seem very closely related. As a motivating example we show how the current state-of-the-art experience selection method of Schaul et al. (2016), based on retaining a large number of experiences and sampling them according to their temporal difference error, compares on these benchmarks to sampling uniformly at random from the experiences of the most recent episodes. We show that the state-of-the-art method significantly outperforms the standard method on one benchmark while significantly *under*-performing on the other, seemingly similar benchmark.

The focus of this paper is on the control of physical systems such as robots. The hardware limitations of these systems can impose constraints on the exploration policy and the number of experiences that can be stored in the buffer. These factors make the correct choice of experience sampling strategy especially important. As we show on additional, more complex benchmarks, even when sustained exploration *is* possible, it can be beneficial to be selective about which and how many experiences to retain in the buffer. The costs involved in operating a robot mean that it is generally infeasible to rely on an extensive hyper-parameter search to determine which experience selection strategy to use. We therefore want to understand how this choice can be made based on prior knowledge of the control task.

With this in mind, the contributions of this work are twofold:

1. We investigate how the utility of different experiences is influenced by the aspects of the control problem. These aspects include properties of the system dynamics such as the sampling frequency and noise, as well as constraints on the exploration.

2. We describe how to perform experience retention and experience sampling based on experience utility proxies. We show how these two parts of experience selection work together under a range of conditions. Based on this we provide guidelines on how to use prior knowledge about the control problem at hand to choose an experience selection strategy.

Note that for many of the experiments in this work most of the hyper-parameters of the deep reinforcement-learning algorithms are kept fixed. While it would be possible to improve the performance through a more extensive hyper-parameter search, our focus is on showing the relationships between the performance of the different methods and the properties of the control problems. While we do introduce new methods to address specific problems, the intended outcome of this work is to be able to make more informed choices regarding experience selection, rather than to promote any single method.

The rest of this paper is organized as follows. Section 2 gives an overview of related work. In Section 3, the basics of reinforcement learning, as well as the deep reinforcement learning and experience replay methods used as a starting point are discussed. Section 4 gives a high-level overview of the simple benchmarks used in most of this work, with the mathematical details presented in Appendix 9.3. The notation we use to distinguish between different methods, as well as the performance criteria that we use, are discussed in Section 5. In

Section 6, we investigate what spread over the state-action the experiences ideally should have, based on the characteristics of the control problem to be solved. The proposed methods to select experiences are detailed in Section 7, with the results of applying these methods to the different scenarios in simple and more complex benchmarks are presented in Section 8. The conclusions, as well as our recommended guidelines for choosing the buffer size, retention proxy and sampling strategy are given in Section 9.

2. Related Work

When a learning system needs to learn a task from a set of examples, the order in which the examples are presented to the learner can be very important. One method to improve the learning performance on complex tasks is to gradually increase the difficulty of the examples that are presented. This concept is known as shaping (Skinner, 1958) in animal training and curriculum learning (Bengio et al., 2009) in machine learning. Sometimes it is possible to generate training examples of just the right difficulty on-line. Recent machine learning examples of this include generative adversarial networks (Goodfellow et al., 2014) and self play in reinforcement learning (see for example the work by Silver et al. 2017). When the training examples are fixed, learning can be sped up by repeating those examples that the learning system is struggling with more often than those that it finds easy, as was shown for supervised learning by, among others, Hinton (2007) and Loshchilov and Hutter (2015). Additionally, the eventual performance of supervised-learning methods can be improved by re-sampling the training data proportionally to the difficulty of the examples, as done in the boosting technique (Valiant, 1984; Freund et al., 1999)

In on-line reinforcement learning, a set of examples is generally not available to start with. Instead, an agent interacts with its environment and observes a stream of experiences as a result. The experience replay technique was introduced to save those experiences in a buffer and replay them from that buffer to the learning system (Lin, 1992). The introduction of an experience buffer makes it possible to choose which examples should be presented to the learning system again. As in supervised learning, we can replay those experiences that induced the largest error (Schaul et al., 2016). Another option that has been investigated in the literature is to replay more often those experiences that are associated with large immediate rewards (Narasimhan et al., 2015).

In off-policy reinforcement learning the question of which experiences to learn from extends beyond choosing how to sample from a buffer. It begins with determining which experiences should be in the buffer. Lipton et al. (2016) fill the buffer with successful experiences from a pre-existing policy before learning starts. Other authors have investigated criteria to determine which experiences should be retained in a buffer of limited capacity when new experiences are observed. In this context, Pieters and Wiering (2016) have investigated keeping only experiences with the highest immediate rewards in the buffer, while our previous work has focused on ensuring sufficient diversity in the state-action space (de Bruin et al., 2016a,b).

Experience replay techniques, including those in this work, often take the stream of experiences that the agent observes as given and attempt to learn from this stream in an optimal way. Other authors have investigated ways to instill the desire to seek out information that is useful for the learning process directly into the agent's behavior (Schmidhuber, 1991; Chentanez et al., 2004; Houthooft et al., 2016; Bellemare et al., 2016; Osband et al., 2016). Due to the classical exploration-exploitation dilemma, changing the agents behavior to obtain more informative experiences comes at the price of the agent acting less optimally according to the original reward function.

A safer alternative to actively seeking out *real* informative but potentially dangerous experiences is to learn, at least in part, from *synthetic* experiences. This can be done by using an a priori available environment model such as a physics simulator (Barrett et al., 2010; Rusu et al., 2016), or by learning a model from the stream of experiences itself and using that to generate experiences (Sutton, 1991; Kuvayev and Sutton, 1996; Gu et al., 2016; Caarls and Schuitema, 2016). The availability of a generative model still leaves the question of *which* experiences to generate. Prioritized sweeping bases updates again on surprise, as measured by the size of the change to the learned functions (Moore and Atkeson, 1993; Andre et al., 1997). Ciosek and Whiteson (2017) dynamically adjusted the distribution of experiences generated by a simulator to reduce the variance of learning updates.

Learning a model can reduce the sample complexity of a learning algorithm when learning the dynamics and reward functions is easy compared to learning the value function or policy. However, it is not straightforward to get improved performance in general. In contrast, the introduction of an experience replay buffer has shown to be both simple and very beneficial for many deep reinforcement learning techniques (Mnih et al., 2015; Lillicrap et al., 2016; Wang et al., 2017; Gu et al., 2017). When a buffer is used, we can decide which experiences to have in the buffer and which experiences to sample from the buffer. In contrast to previous work on this topic we investigate the combined problem of experience retention and sampling. We also look at several different proxies for the usefulness of experiences and how prior knowledge about the specific reinforcement learning problem at hand can be used to choose between them, rather than attempting to find a single universal experience-utility proxy.

3. Preliminaries

We consider a standard reinforcement learning setting (Section 3.1) in which an agent learns to act optimally in an environment, using the implementation by Lillicrap et al. (2016) of the off-policy actor-critic algorithm by Silver et al. (2014) (Section 3.2). Actor-critic algorithms make it possible to deal with the continuous action spaces that are often found in control applications. The off-policy nature of the algorithm enables the use of experience replay (Section 3.3), which helps to reduce the number of environment steps needed by the algorithm to learn a successful policy and improves the algorithms stability. Here, we summarize the deep reinforcement learning (Lillicrap et al., 2016) and experience replay (Schaul et al., 2016) methods that we use as a starting point.

3.1 Reinforcement Learning

In reinforcement learning, an agent interacts with an environment $\mathcal{E}$ with (normalized) state $s_\mathcal{E}$ by choosing (normalized) actions a according to its policy π: $a = \pi(s)$, where s is the agent's perception of the environment state.

To simplify the analysis in Section 6 and 7, and to aid learning, we normalize the state and action spaces in our benchmarks such that $s_\mathcal{E} \in [-1, 1]^n$ and $a_\mathcal{E} \in [-1, 1]^m$, where n

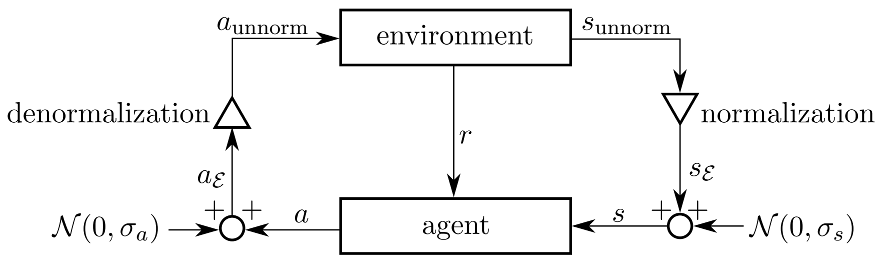


Figure 1: Reinforcement learning scheme and symbols used.

and m are the dimensions of the state and action spaces. We perform the (de)normalization on the connections between the agent and the environment, so the agent only deals with normalized states and actions.

We consider the dynamics of the environment to be deterministic: $s'_{\mathcal{E}} = f(s_{\mathcal{E}}, a_{\mathcal{E}})$. Here, $s'_{\mathcal{E}}$ is the state of the environment at the next time step after applying action $a_{\mathcal{E}}$ in state $s_{\mathcal{E}}$. Although the environment dynamics are deterministic, in some of our experiments we do consider sensor and actuator noise. In these cases, the state s that the agent perceives is perturbed from the actual environment state $s_{\mathcal{E}}$ by additive Gaussian noise

$$s = s_{\mathcal{E}} + \mathcal{N}(0, \sigma_s). \tag{1}$$

Similarly, actuator noise changes the actions sent to the environment according to:

$$a_{\mathcal{E}} = a + \mathcal{N}(0, \sigma_a). \tag{2}$$

A reward function ρ describes the desirability of being in an unnormalized state s_{unnorm} and taking an unnormalized action a_{unnorm}: $r_k = \rho(s^k_{\text{unnorm}}, a^k_{\text{unnorm}}, s^{k+1}_{\text{unnorm}})$, where k indicates the time step. An overview of the different reinforcement learning signals and symbols used is given in Figure 1.

The goal of the agent is to choose the actions that maximize the expected return from the current state, where the return is the discounted sum of future rewards: $\sum_{k=0}^{\infty} \gamma^k r_k$. The discount factor $0 \leq \gamma < 1$ keeps this sum finite and allows trading off short-term and long-term rewards.

Although we will come back to the effect of the sensor and actuator noise later on, in the remainder of this section we will look at the reinforcement learning problem from the perspective of the agent and consider the noise to be part of the environment. This makes the transition dynamics and reward functions stochastic: $\mathcal{F}(s'|s,a)$ and $\mathcal{P}(r|s,a,s')$.

3.2 Off-Policy Deep Actor-Critic Learning

In this paper we use the Deep Deterministic Policy Gradient (DDPG) reinforcement-learning method of Lillicrap et al. (2016), with the exception of Section 6.3, where we compare it to DQN (Mnih et al., 2015). In the DDPG method, based on the work of Silver et al. (2014), a neural network with parameters θ_π implements the policy: $a = \pi(s; \theta_\pi)$. A second neural network with parameters θ_Q, the critic, is used to approximate the Q function. The $Q^\pi(s,a)$ function gives the expected return when taking action a in state s and following the policy π from next time-step onwards

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_k | s^0 = s, a^0 = a\right]. \tag{3}$$

The critic function $Q\left(s, a; \theta_Q\right)$ is trained to approximate the true $Q^{\pi}(s,a)$ function by minimizing the squared temporal difference error δ for experience $\langle s_i, a_i, s'_i, r_i\rangle$

$$\begin{aligned} \delta_i &= \left[r_i + \gamma Q\left(s'_i, \pi(s'_i; \theta_\pi^-); \theta_Q^-\right)\right] - Q\left(s_i, a_i; \theta_Q\right), & (4)\\ L_i(\theta_Q) &= \delta_i^2, & \\ \Delta_{i\theta_Q} &\sim -\nabla_{\theta_Q} L_i(\theta_Q). & (5) \end{aligned}$$

The index i is a generic index for experiences that we will in the following use to indicate the index of an experience in a buffer. The parameter vectors θ_π^- and θ_Q^- are copies of θ_π and θ_Q that are updated with a low-pass filter to slowly track θ_π and θ_Q

$$\begin{aligned} \theta_\pi^- &\leftarrow (1-\tau)\theta_\pi^- + \tau\theta_\pi,\\ \theta_Q^- &\leftarrow (1-\tau)\theta_Q^- + \tau\theta_Q, \end{aligned}$$

with $\tau \in (0,1), \tau \ll 1$. This was found to be important for ensuring stability when using deep neural networks as function approximators in reinforcement learning (Mnih et al., 2015; Lillicrap et al., 2016).

The parameters θ_π of the policy neural network $\pi(s;\theta_\pi)$ are updated in the direction that changes the action $a = \pi(s;\theta_\pi)$ in the direction for which the critic predicts the steepest ascent in the expected sum of discounted rewards

$$\Delta_{\theta_\pi} \sim \nabla_a Q\left(s_i, \pi(s_i;\theta_\pi); \theta_Q\right) \nabla_{\theta_\pi} \pi(s_i;\theta_\pi). \tag{6}$$

3.3 Experience Replay

The actor and critic neural networks are trained by using sample-based estimates of the gradients ∇_{θ_Q} and ∇_{θ_π} in a stochastic gradient optimization algorithm such as ADAM (Kingma and Ba, 2015). These algorithms are based on the assumption of independent and identically distributed (i.i.d.) data. This assumption is violated when the experiences $\langle s_i, a_i, s'_i, r_i\rangle$ in (5) and (6) are used in the same order during the optimization of the networks as they were observed by the agent. This is because the subsequent samples are strongly correlated, since the world only changes slowly over time. To solve this problem, an experience replay (Lin, 1992) buffer $\mathcal{B}$ with some finite capacity $\mathcal{C}$ can be introduced.

Most commonly, experiences are written to this buffer in a First In First Out (FIFO) manner. When experiences are needed to train the neural networks, they are sampled uniformly at random from the buffer. This breaks the temporal correlations of the updates and restores the i.i.d. assumption of the optimization algorithms, which improves their performance (Mnih et al., 2015; Montavon et al., 2012). The increased stability comes in addition to the main advantage of experience replay, which is that experiences can be used multiple times for updates, increasing the sample efficiency of the algorithm.

3.3.1 Prioritized Experience Replay

Although sampling experiences uniformly at random from the experience buffer is an easy default, the performance of reinforcement-learning algorithms can be improved by choosing the experience samples used for training in a smarter way. Here, we summarize one of the variants of Prioritized Experience Replay (PER) that was introduced by Schaul et al. (2016). Our enhancements to experience replay are given in Section 7.

The PER technique is based on the idea that the temporal difference error (4) provides a good proxy for the instantaneous utility of an experience. Schaul et al. (2016) argue that, when the critic made a large error on an experience the last time it was used in an update, there is more to be learned from the experience. Therefore, its probability of being sampled again should be higher than that of an experience associated with a low temporal difference error.

In this work we consider the rank-based stochastic PER variant. In this method, the probability of sampling an experience i from the buffer is approximately given by:

$$P(i) \approx \frac{\left(\frac{1}{\text{rank}(i)}\right)^{\alpha}}{\sum_j \left(\frac{1}{\text{rank}(j)}\right)^{\alpha}}. \tag{7}$$

Here, rank(i) is the rank of sample i according to the absolute value of the temporal difference error $|\delta|$ according to (4), calculated when the experience was last used to train the critic. All experiences that have not yet been used for training have $\delta = \infty$, resulting in a large probability of being sampled. The parameter α determines how strongly the probability of sampling an experience depends on δ. We use $\alpha = 0.7$ as proposed by Schaul et al. (2016) and have included a sensitivity analysis for different buffer sizes in Appendix 9.3. Note that the relation is only approximate as sampling from this probability distribution directly is inefficient. For efficient sampling, (7) is used to divide the buffer $\mathcal{B}$ into S segments of equal cumulative probability, where S is taken as the number of experiences per training mini batch. During training, one experience is sampled uniformly at random from each of the segments.

3.3.2 Importance Sampling

The estimation of an expected value with stochastic updates relies on those updates corresponding to the same distribution as its expectation. Schaul et al. (2016) proposed to compensate for the fact that the changed sampling procedure can affect the value of the expectation in (3) by multiplying the gradients (5) with an Importance Sampling (IS) weight

$$\omega_i = \left(\frac{1}{\mathcal{C}} \frac{1}{P(i)}\right)^{\beta}. \tag{8}$$

Here, β allows scaling between not compensating at all ($\beta = 0$) to fully compensating for the changes in the sample distribution caused by the sampling strategy ($\beta = 1$). In our experiments, when IS is used, we follow Schaul et al. (2016) in scaling β linearly per episode from 0.5 at the start of a learning run to $\beta = 1$ at the end of the learning run. $\mathcal{C}$ indicates the capacity of the buffer.

Not all changes to the sampling distribution need to be compensated for. Since we use a deterministic policy gradient algorithm with a Q-learning critic, we do not need to compensate for the fact that the samples are obtained by a different policy than the one we are optimizing for (Silver et al., 2014). We can change the sampling distribution from the buffer, without compensating for the change, so long as these samples accurately represent the transition and reward functions.

Sampling based on the TD error can cause issues here, as infrequently occurring transitions or rewards will tend to be surprising. Replaying these samples more often will introduce a bias, which should be corrected through importance sampling.

However, the temporal difference error will also be partly caused by the function approximation error. These errors will be present even for a stationary sample distribution after learning has converged. The errors will vary over the state-action space and their magnitude will be related to the sample density. Sampling based on *this* part of the temporal difference error will make the function approximation accuracy more consistent over the state-space. This effect might be unwanted when the learned controller will be tested on the same initial state distribution as it was trained on. In that case, it is preferable to have the function approximation accuracy be highest where the sample density is highest. However, when the aim is to train a controller that generalizes to a larger part of the state space, we might *not* want to use importance sampling to correct this effect. Note that importance sampling based on the sample distribution over the state space is heuristically motivated and based on function approximation considerations. The motivation does not stem from the reinforcement learning theory, where most methods assume that the Markov decision process is ergodic and that the initial state distribution does not factor into the optimal policy (Aslanides et al., 2017). In practice however, deep reinforcement-learning methods can be rather sensitive to the initial state distribution (Rajeswaran et al., 2017).

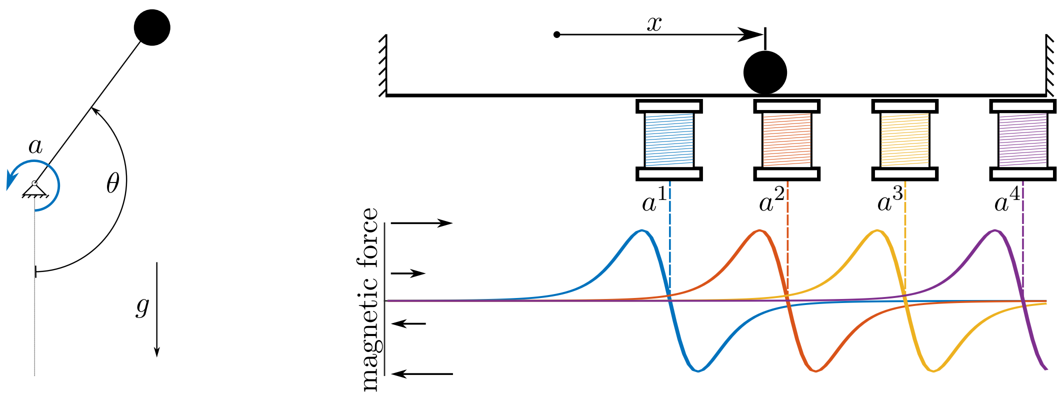
Unfortunately, we do not know to what extent the temporal difference error is caused by the stochasticity of the environment dynamics and to what extent it is caused by function approximation errors. We will empirically investigate the use of importance sampling in Section 8.4.

4. Experimental Benchmarks

In this section, we discuss two relatively simple control tasks that are considered in this paper, so that an understanding of their properties can be used in the following sections. The relative simplicity of these tasks enables a thorough analysis. We test our findings on more challenging benchmarks in Section 8.5.

We perform our tests on two simulated control benchmarks: a pendulum swing-up task and a magnetic manipulation problem. Both were previously discussed by Alibekov et al. (2018). Although both represent dynamical systems with a two dimensional state-space, it will be shown in Section 6 that they are quite different when it comes to the optimal experience selection strategy. Here, a high level description of these benchmarks is presented, with the full mathematical description given in Appendix 9.3.

The first task is the classic under-actuated pendulum swing-up problem, shown in Figure 2a. The pendulum starts out hanging down under gravity. The goal is to balance the pendulum in the upright position. The motor is torque limited such that a swing to one



(a) Pendulum task (b) Magnetic manipulation task

Figure 2: The two benchmark problems considered in this paper. In the pendulum task, an underactuated pendulum needs to be swung up and balanced in the upright position by controlling the torque applied by a motor. In the magnetic manipulation (magman) task, a steel ball (top) needs to be positioned by controlling the currents through four electromagnets. The magnetic forces exerted on the ball are shown at the bottom of the figure and can be seen to be a nonlinear function of the position. The forces scale linearly with the actions $a^1, ..., a^4$, which represent the squared currents through the magnets.

side is needed to build up momentum before swinging towards the upright position in the opposite direction. Once the pendulum is upright it needs to stabilize around this unstable equilibrium point. The state of the problem $s_{\mathcal{E}}$ consists of *normalized* versions of the angle θ and angular velocity $\dot{\theta}$ of the pendulum. The action space is a normalized version of the voltage applied to the motor that applies a torque to the pendulum. A reward is given at every time-step, based on the absolute distance of the state from the reference state of being upright with no rotational velocity.

The second benchmark is a magnetic manipulation (*magman*) task, in which the goal is to accurately position a steel ball on a 1-D track by dynamically changing a magnetic field. The relative magnitude and direction of the force that each magnet exerts on the ball is shown in Figure 2b. This force is linearly dependent on the actions, which represent the squared currents through the electromagnet coils. Normalized versions of the position x and velocity $\dot{x}$ form the state-space of the problem. A reward is given at every time-step, based on the absolute distance of the state from the reference state of having the ball at the fixed desired position.

In experiments where the buffer capacity $\mathcal{C}$ is limited, we take $\mathcal{C} = 10^4$ experiences, unless stated otherwise. All our experiments have episodes which last four seconds. Unless stated otherwise, a sampling frequency of 50 Hz is used, which means the buffer can store 50 episodes of experience tuples $\langle s_i, a_i, s'_i, r_i \rangle$.

Since we are especially interested in physical control problems where sustained exhaustive exploration is infeasible, the amount of exploration is reduced over time from its maximum at episode 1, to a minimum level from episode 500 onwards in all our experiments. At the minimum level, the amplitude of the exploration noise we add to the neural network policy is 10% of the amplitude at episode 1. Details of the exploration strategies used are given in Appendix 9.3.

5. Performance Measures and Experience Selection Notation

This section introduces the performance measures used and the notation used to distinguish between the experience selection strategies.

5.1 Performance Measures

When we investigate the performance of the learning methods in Sections 6 and 8, we are interested in the effect that these methods might have on *three* aspects of the learning performance: *the learning stability, the maximum controller performance* and *the learning speed*. We define performance metrics for these aspects, related to the normalized mean reward per episode μ_r. The normalization is performed such that $\mu_r = 0$ is the performance achieved by a random controller, while $\mu_r = 1$ is the performance of the off-line dynamic programming method described in Appendix 9.3. This baseline method is, at least for the noise-free tests, proven to be close to optimal.

The first learning performance aspect we consider is the *stability* of the learning process. As we have discussed in previous work (de Bruin et al., 2015, 2016a), even when a good policy has already been learned, the learning process can become unstable and the performance can drop significantly when the properties of the training data change. We investigate to what extent different experience replay methods can help prevent this instability. We use the mean of μ_r over the last 100 episodes of each learning run, where the learning runs should have converged to good behavior already, as a measure of learning stability. We denote this measure by μ_r^{final}.

Although changing the data distribution might help stability, it could at the same time prevent us from accurately approximating the true optimal policy. Therefore we also report the *maximum performance* achieved per learning trial μ_r^{max}.

Finally, we want to know the effects of the experience selection methods on the *learning speed*. We therefore report the number of episodes before the learning method achieves a normalized mean reward per episode of $\mu_r = 0.8$ and denote this by Rise-time 0.8.

For these performance metrics we report the means and the 95% confidence bounds of those means over 50 trials for each experiment. The confidence bounds are based on bootstrapping (Efron, 1992).

5.2 Experience Selection Strategy Notation

We consider the problem of *experience selection*, which we have defined as the combination of *experience retention* and *experience sampling*. The experience retention strategy determines which experiences are discarded when new experiences are available to a full buffer. The sampling strategy determines which experiences are used in the updates of the reinforcement-learning algorithm. We use the following notation for the complete experience selection strategy: *retention strategy[sampling strategy]*. Our abbreviations for the retention

Notation	Proxy	Explanation
FIFO	age	The oldest experiences are overwritten with new ones.
FULL DB	-	The buffer capacity $\mathcal{C}$ is chosen to be large enough to retain all experiences.

Table 1: Commonly used experience retention strategies for deep reinforcement learning.

Notation	Proxy	Explanation
Uniform	-	Experiences are sampled uniformly at random.
PER	surprise	Experiences are sampled using rank-based stochastic *prioritized experience replay* based on the temporal difference error. See Section 3.3.1.
PER+IS	surprise	Sampling as above, but with weighted *importance sampling* to compensate for the distribution changes caused by the sampling procedure. See Section 3.3.2.

Table 2: Experience sampling strategies from the literature.

and sampling strategies commonly used in deep RL that were introduced in Section 3.3 are given in Tables 1 and 2 respectively. The abbreviations used for the new or uncommonly used methods introduced in Section 7 are given there, in Tables 4 and 5.

6. Analysis of Experience Utility

As previously noted by Schaul et al. (2016); Narasimhan et al. (2015); Pieters and Wiering (2016) and de Bruin et al. (2016a, 2015), when using experience replay, the criterion that determines which experiences are used to train the reinforcement learning agent can have a large impact on the performance of the method. The aim of this section is to investigate what makes an experience useful and how this usefulness depends on several identifiable characteristics of the control problem at hand.

In the following sections, we mention only some relevant aspects of our implementation of the deep reinforcement-learning methods, with more details given in Appendix 9.3.

6.1 The Limitations of a Single Proxy

To motivate the need for understanding how the properties of a control problem influence the applicability of different experience selection strategies, and the need for multiple proxies for the utility of experiences rather than one universal proxy, we compare the performance of the two strategies from the literature that were presented in Section 3.3 on the benchmarks described in Section 4.

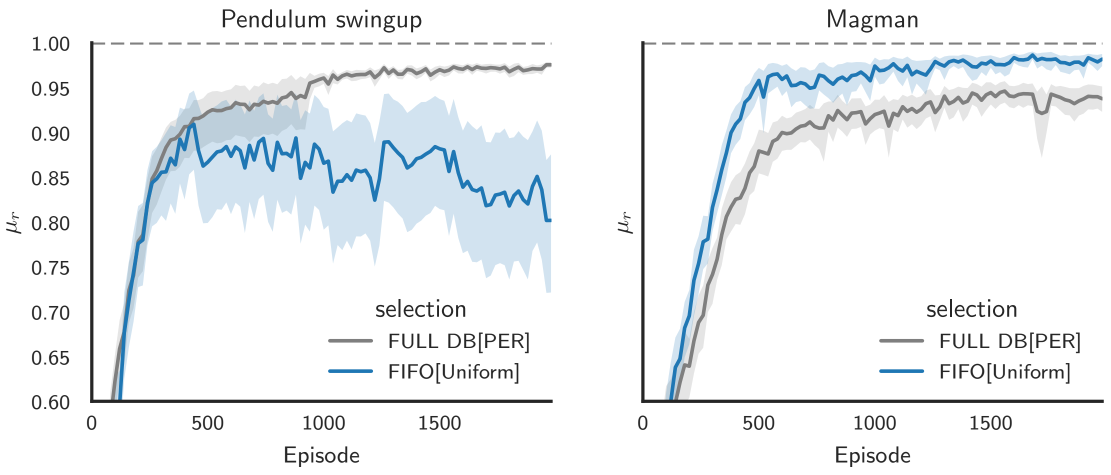


Figure 3: Comparison of the state-of-the-art (FULL DB[PER]) and the default method (FIFO[Uniform]) for experience selection on our two benchmark problems.

The first experience selection strategy tested is FIFO[Uniform]: overwriting the oldest experiences when the buffer is full and sampling uniformly at random from the buffer. We compare this strategy to the state-of-the-art prioritized experience replay method FULL DB[PER] by Schaul et al. (2016). Here, the buffer capacity $\mathcal{C}$ is chosen such that all experiences are retained during the entire learning run ($\mathcal{C} = N = 4 \times 10^5$ for this test).[1] The sampling strategy is the rank-based stochastic prioritized experience replay strategy as described in Section 3.3. The results of the experiments are shown in Figure 3.

Figure 3 shows that FULL DB[PER] method, which samples training batches based on the temporal difference error from a buffer that is large enough to contain all previous experiences, works well for the pendulum swing-up task. The method very reliably finds a near optimal policy. The FIFO[Uniform] method, which keeps only the experiences from the last 50 episodes in memory, performs much worse. As we reported previously (de Bruin et al., 2016a), the performance degrades over time as the amount of exploration is reduced and the experiences in the buffer fail to cover the state-action space sufficiently.

If we look at the result on the magman benchmark in Figure 3, the situation is reversed. Compared to simply sampling uniformly from the most recent experiences, sampling from all previous experiences according to their temporal difference error limits the final performance significantly. As shown in Appendix 9.3, this is not simply a matter of the function approximator capacity, as even much larger networks trained on all available data are outperformed by small networks trained on only recent data. When choosing an experience selection strategy for a reinforcement learning task, it seems therefore important to have some insights into how the characteristics of the task determine the need for specific kinds of experiences during training. We will investigate some of these characteristics below.

1. Schaul et al. (2016) use a FIFO database with a capacity of 10^6 experiences. We here denote this as FULL DB since all our experiments use a smaller number of time-steps in total.

6.2 Generalizability and Sample Diversity

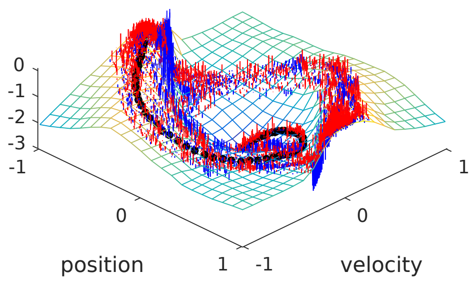
One important aspect of the problem, which at least partly explains the differences in performance for the two methods on the two benchmarks in Figure 3, is the complexity of generalizing the value function and policy across the state and action spaces.

For the pendulum task, learning actor and critic functions that generalize across the entire state and action spaces will be relatively simple as a sufficiently deep neural network can efficiently exploit the symmetry in the value and policy functions (Montufar et al., 2014). Figure 4b shows the learned policy after 100 episodes for a learning run with FIFO[Uniform] experience selection. Due to the thorough initial exploration, the experiences in the buffer cover much of the state-action space. As a result, a policy has been learned that is capable of swinging the pendulum up and stabilizing it in both the clockwise and anticlockwise directions, although the current policy favors one direction over the other.

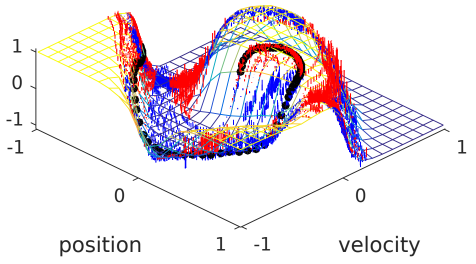
For the next 300 episodes this favored direction does not change and as the amount of exploration is decayed, the experiences in the buffer become less diverse and more centered around this favored trajectory through the state-action space. Even though the information on how to further improve the policy becomes increasingly local, the updates to the network parameters can cause the policy to be changed over the whole state space, as neural networks are global function approximators. This can be seen from Figure 4d, where the updates that further refine the policy for swinging up in the currently preferred direction have removed the previously obtained skill of swinging up in the opposite direction. The policy has suffered from *catastrophic forgetting* (Goodfellow et al., 2013) and has over-fitted to the currently preferred swing up direction.

For the pendulum swing up task, this over-fitting is particularly risky since the preferred swing up direction can and does change during learning, since both directions are equivalent with respect to the reward function. When this happens, the FIFO experience retention method can cause the data distribution in the buffer to change rapidly, which by itself can cause instability. In addition, the updates (4) and (6) now use the critic $Q\left(s, a; \theta_Q\right)$ function in regions of the state-action space that it has not been trained on in a while, resulting in potentially bad gradients. Both of these factors might destabilize the learning process. This can be seen in Figure 4f where, after the preferred swing up direction has rapidly changed a few times, the learning process is destabilized and the policy has deteriorated to the point that it no longer accomplishes the balancing task. By keeping all experiences in memory and ensuring the critic error δ stays low over the entire state-action space, the FULL DB[PER] method largely avoids these learning stability issues. We believe that this accounts for the much better performance for this benchmark shown in Figure 3.

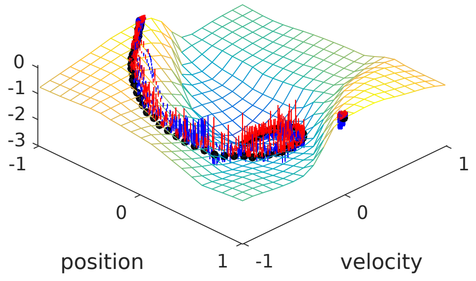
For the magman task, a policy that generalizes over the whole state-space might be harder to find. This is because the effects of the actions, shown as the colored lines in Figure 2b, are strongly nonlinear functions of the (position)-state. The actor and critic functions must therefore be very accurate for the states that are visited under the policy. Requiring the critic to explain all of the experiences that have been collected so far might limit the ability of the function approximators to achieve sufficient accuracy for the relevant states.



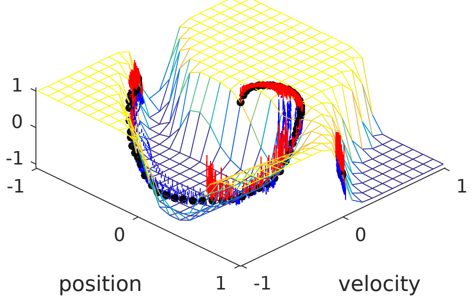
(a) critic, episode 100



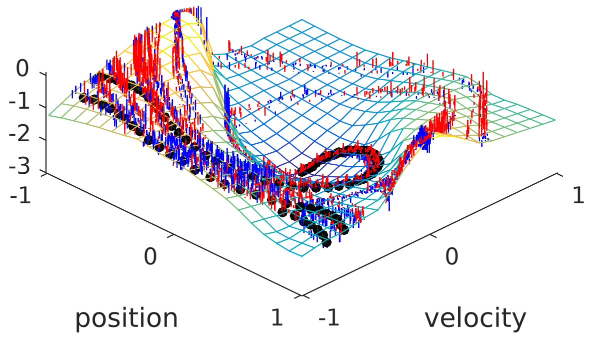
(b) actor, episode 100



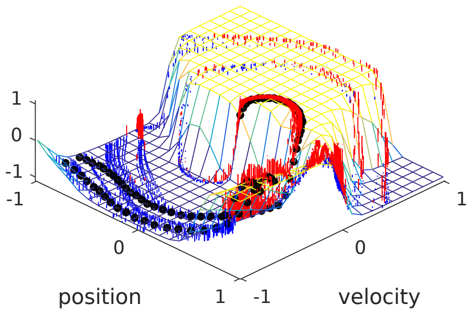
(c) critic, episode 390



(d) actor, episode 390



(e) critic, episode 507



(f) actor, episode 507

Figure 4: The critic $Q\left(s, \pi(s;\theta_\pi);\theta_Q\right)$ and actor $\pi(s;\theta_\pi)$ functions trained on the pendulum swing up task using FIFO[Uniform] experience selection. The surfaces represent the functions. The black dots show the trajectories through the state-action space resulting from deterministically following the current policy. The red and blue lines show respectively the positive and negative 'forces' that shape the surfaces caused by the experiences in the buffer: for the critic these are $\delta(s,a)$ (note $a \neq \pi(s;\theta_\pi)$). For the actor these forces represent $\partial Q\left(s, \pi(s;\theta_\pi);\theta_Q\right)/\partial a$. Animations of these graphs for different experience selection strategies are available at `https://youtu.be/Hli1ky0bgT4`. The episodes are chosen to illustrate the effect of reduced sample diversity described in Section 6.2.

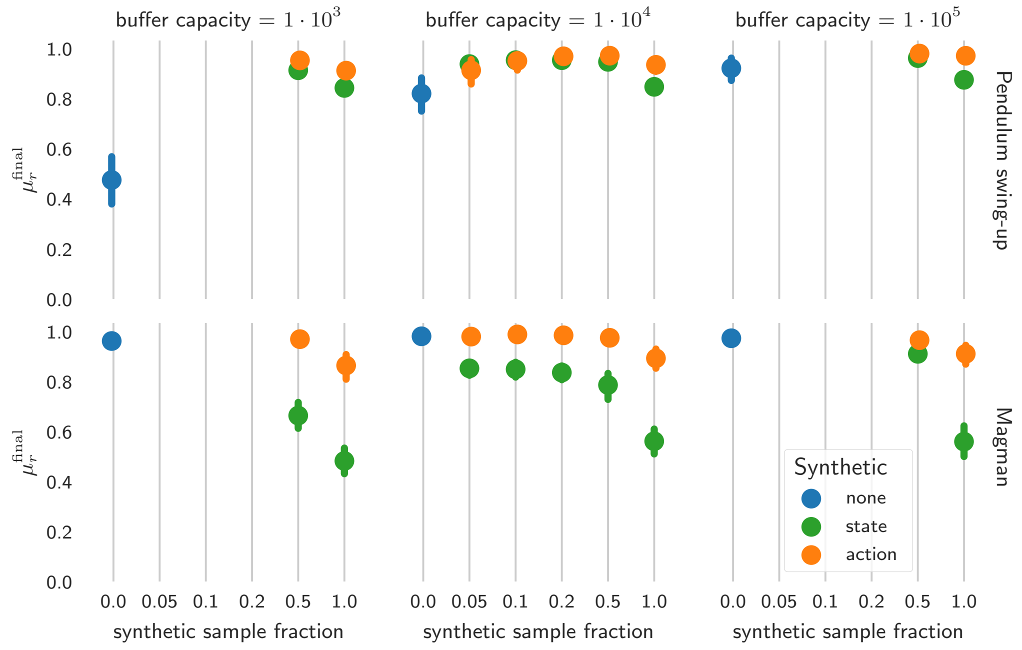
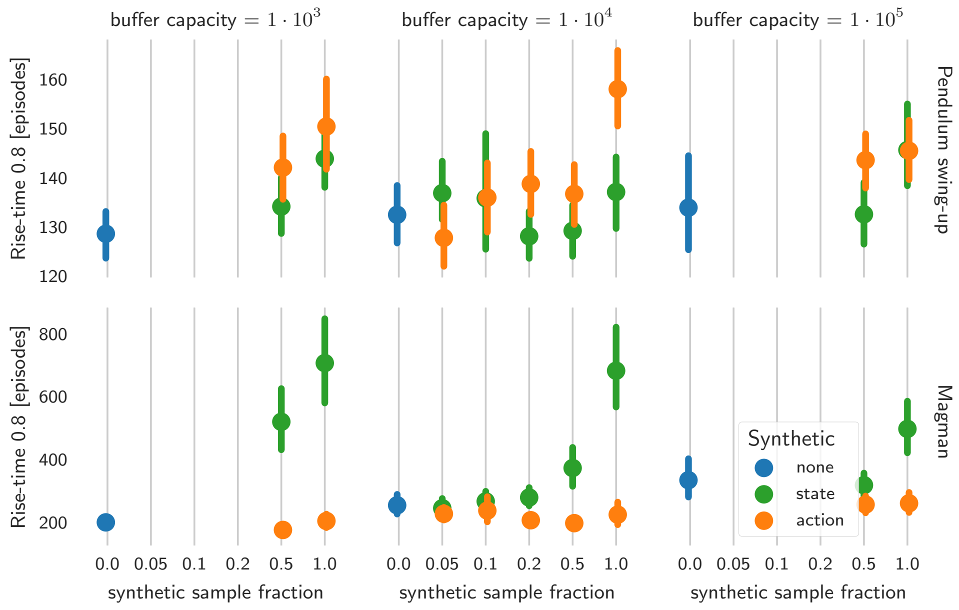


Figure 5: The effect on the mean performance during the last 100 episodes of the learning runs μ_r^{final} of the FIFO[Uniform] method when changing a fraction of the observed experiences with synthetic experiences, for different buffer sizes.

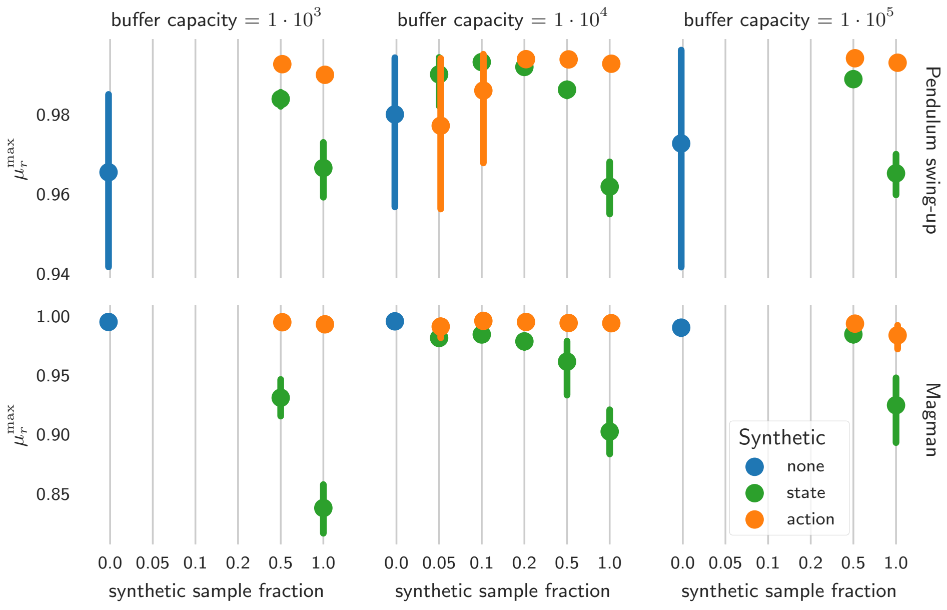
6.2.1 Buffer Size and Synthetic Sample Fraction

To test the hypothesis that the differences in performance observed in Figure 3 revolve around sample diversity, we will artificially alter the sample diversity and investigate how this affects the reinforcement learning performance. We will do so by performing the following experiment. We use the plain FIFO[Uniform] method as a baseline. However, with a certain probability we make a change to an experience $\langle s_i, a_i, s'_i, r_i \rangle$ before it is written to the buffer. We change either the state s_i or the action a_i. The changed states and actions are sampled uniformly at random from the state and action spaces. When the state is re-sampled the action is recalculated as the policy action for the new state including exploration. In both cases, the next state and reward are recalculated to complete the altered experience. To calculate the next state and reward, we use the real system model. This is not possible for most practical problems; it serves here merely to gain a better understanding of the need for sample diversity.

The results of performing this experiment for different probabilities and buffer sizes are given in Figures 5 and 6. Interestingly, for the pendulum swing up task, changing some fraction of the experiences to be more diverse improves the stability of the learning method dramatically, regardless of whether the diversity is in the states or in the actions. The effect is especially noticeable for smaller experience buffers.



(a) Effect on the number of episodes needed to reach $\mu_r = 0.8$.



(b) Effect on the number of maximum controller performance obtained per learning run.

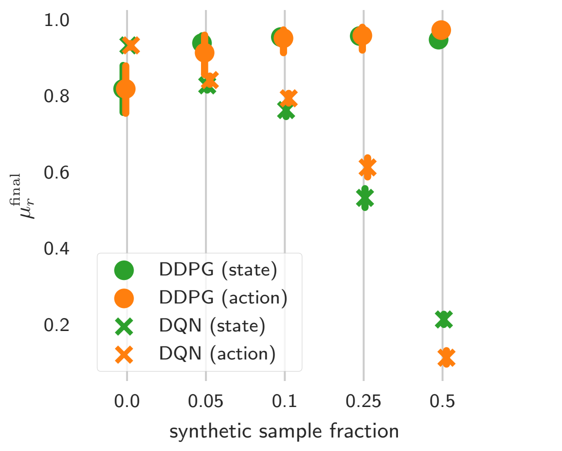
Figure 6: The effects on the learning performance of the FIFO[Uniform] method when replacing a fraction of the observed experiences with synthetic experiences, for different buffer sizes.

For the magman benchmark, as expected, having more diverse states reduces the performance significantly. Having a carefully chosen fraction of more diverse actions in the original states can however improve the stability and learning speed slightly. This can be explained from the fact that even though the effects of the actions are strongly nonlinear in the state-space, they are linear in the action space. Generalizing across the action space might thus be more straightforward and it is helped by having the training data spread out over this domain.

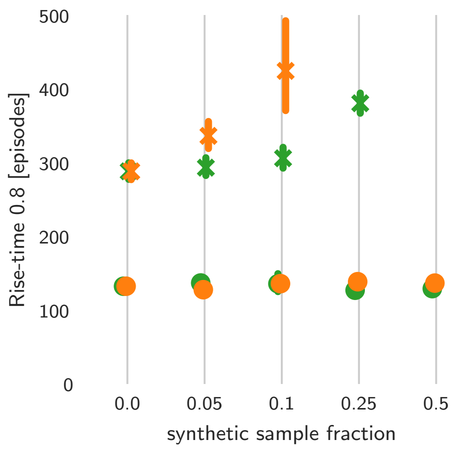
6.3 Reinforcement-Learning Algorithm

The need for experience diversity also depends on the algorithm that is used to learn from those experiences. In the rest of this work we exclusively consider the DDPG actor-critic algorithm, as the explicitly parameterized policy enables continuous actions, which makes it especially suitable for control. An alternative to using continuous actions is to discretize the action space. In this subsection, we compare the need for diverse data of the actor-critic DDPG algorithm (Lillicrap et al., 2016; Silver et al., 2014) to that of the closely related critic-only DQN algorithm (Mnih et al., 2015). The experiments are performed on the pendulum benchmark, where the one dimensional action is divided uniformly into 15 discrete actions. Results for the magman benchmark are omitted as the four dimensional action space makes discretization impractical.

For the actor-critic scheme to work, the critic needs to learn a general dependency of the Q-values on the states and actions. For the DQN critic, this is not the case as the Q-values for different actions are separate. Although the processing of the inputs is shared, the algorithm can learn at least partially independent value predictions for the different



(a) Effect on the mean performance during the last 100 episodes of the learning runs μ_r^{final}.



(b) Effect on the number of episodes needed to reach $\mu_r = 0.8$.

Figure 7: RL algorithm dependent effect of adding synthetic experiences to the FIFO[Uniform] method. Experiments on the pendulum benchmark. The effect on $\mu_r^{\max}$ is given in Figure 22 in Appendix 9.3.

actions. These functions additionally do not need to be correct, as long as the optimal action in a state has a higher value than the sub-optimal actions.

These effects can be seen in Figure 7. The DDPG algorithm can make more efficient use of the state-action space samples by learning a single value prediction, resulting in significantly faster learning than the DQN algorithm. The DDPG algorithm additionally benefits from more diverse samples, with the performance improving for higher fractions of randomly sampled states or actions. The DQN algorithm conversely seems to suffer from a more uniform sampling of the state-action space. This could be because it is now tasked with learning accurate mappings from the states to the state-action values for all actions. While doing so might not help to improve the predictions in the relevant parts of the state-action space, it could increase the time required to learn the function and limit the function approximation capacity available for those parts of the state-space where the values need to be accurate. Note again that learning precise Q-values for all actions over the whole state-space is not needed, as long as the optimal action has the largest Q-value.

Due to the better scalability of policy-gradient methods in continuous control settings, we exclusively consider the DDPG algorithm in the remainder of this work.

6.3.1 Sample Age

In the model-free setting it is not possible to add synthetic experiences to the buffer. Instead, in Section 7 we will introduce ways to select real experiences that have desirable properties and should be remembered for a longer time and replayed more often. This will inevitably mean that some experiences are used more often than others, which could have detrimental effects such as that the learning agent could over-fit to those particular experiences.

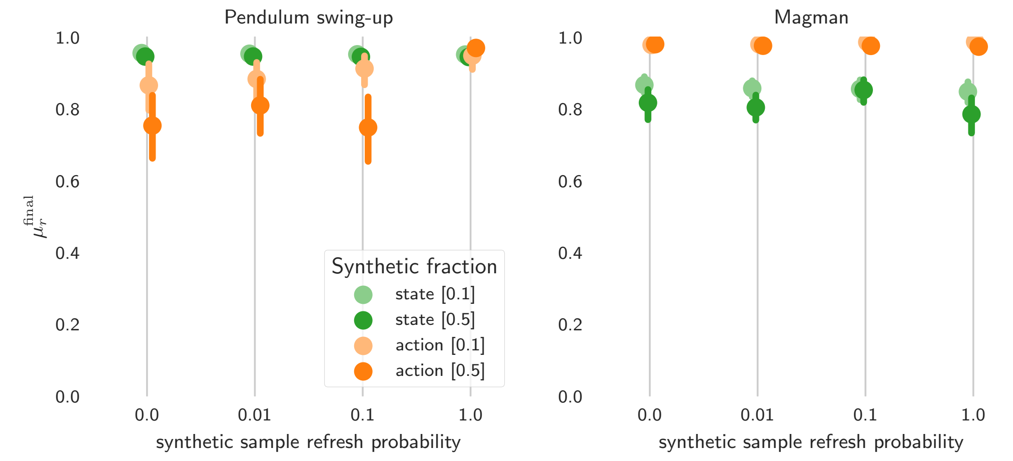


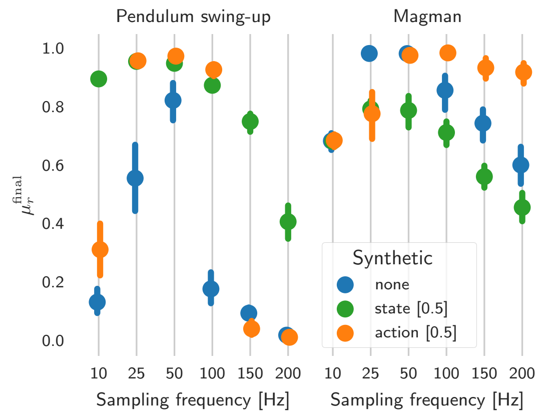
Figure 8: The effects on μ_r^{final} of the FIFO[Uniform] method when changing a fraction of the observed experiences with synthetic experiences, when the synthetic experiences are updated only with a certain probability each time they are overwritten. The effects on $\mu_r^{\max}$ and the rise-time are given in Figure 24 in Appendix 9.3.

To investigate the effects of adding older experiences for diversity, we perform the following experiment. As before, a FIFO buffer is used with a certain fraction of synthetic experiences. However, when a synthetic experience is about to be over-written, we only sample a new synthetic experience with a certain probability. Otherwise, the experience is left unchanged. The result of this experiment is shown in Figure 8. For the pendulum benchmark, old experiences only hurt when they were added to provide diversity in the action space in states that were visited by an older policy. For the magman benchmark the age of the synthetic experiences is not seen to affect the learning performance.

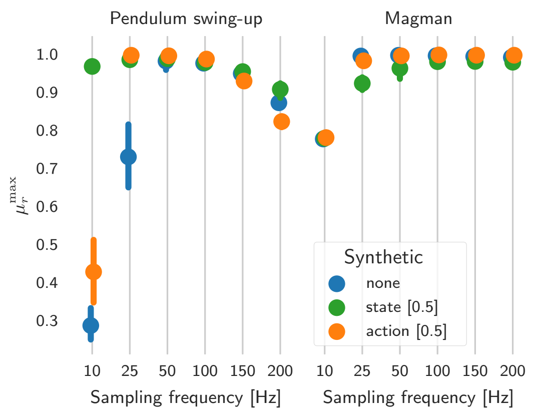
6.4 Sampling Frequency

An important property of control problems that can influence the need for experience diversity is the frequency at which the agent needs to produce control decisions. The sampling frequency of a task is something that is often considered a given property of the environment in reinforcement learning. For control tasks however, a sufficiently high sampling frequency can be crucial for the performance of the controller and for disturbance rejection (Franklin et al., 1998). At the same time, higher sampling frequencies can make reinforcement learning more difficult as the effect of taking an action for a single time-step diminishes for increasing sampling frequencies (Baird, 1994). Since the sampling rate can be an important hyperparameter to choose, we investigate whether changing it changes the diversity demands for the experiences to be replayed.

In Figure 9, the performance of the FIFO[Uniform] method is shown for different sampling frequencies, with and without synthetic samples. The first thing to note is that, as expected, low sampling frequencies limit the controller performance. Interestingly, much of the performance loss on the pendulum at low frequencies can be prevented through in-



(a) Effect on the mean performance during the last 100 episodes of the learning runs μ_r^{final}.



(b) Effect on maximum controller performance per episode $\mu_r^{\max}$.

Figure 9: Sampling frequency dependent effect of adding synthetic experiences to the FIFO[Uniform] method. The effect on the rise time is given in Figure 23 in Appendix 9.3.

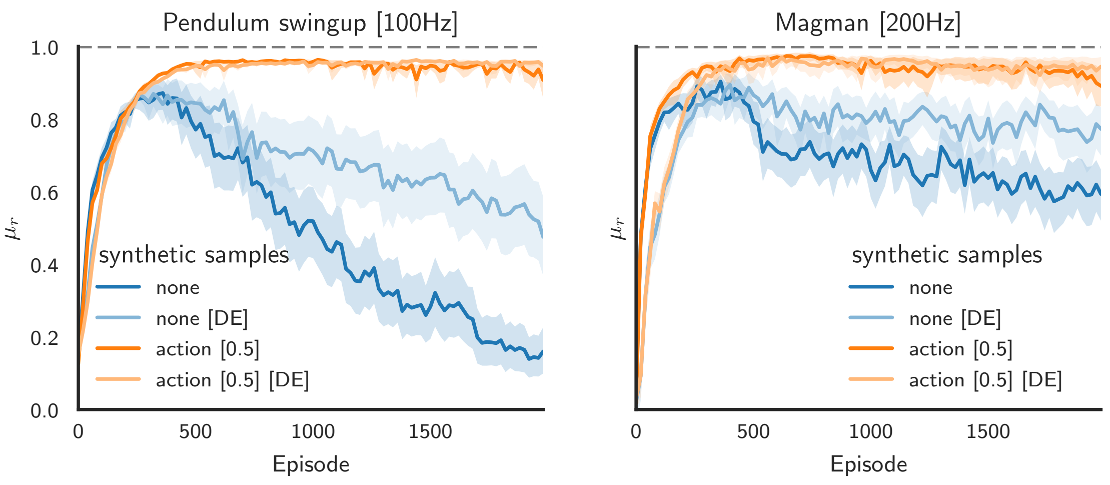


Figure 10: The effect of synthetic actions and stochastically preventing experiences from being written to the buffer [DE] for the FIFO[Uniform] method on the benchmarks with increased sampling frequencies.

creased sample diversity. This indicates that on this benchmark most of the performance loss at the tested control frequencies stems from the learning process rather than the fundamental control limitations. When increasing the sampling frequencies beyond our baseline frequency of 50Hz, sample diversity becomes more important for both stability and performance. For the pendulum swing-up it can be seen that as sampling frequency increases further, increased diversity in the *state-space* becomes more important. For the magman, adding synthetic *action* samples has clear benefits. This is very likely related to the idea that the effects of actions become harder to distinguish for higher sampling frequencies (Baird, 1994; de Bruin et al., 2016b).

There are several possible additional causes for the performance decrease at higher frequencies. The first is that by increasing the sampling frequency, we have increased the number of data points that are obtained and learned from per episode. Yet the amount of information that the data contains has not increased by the same amount. Since the buffer capacity is kept equal, the amount of information that the buffer contains has decreased and the learning rate has effectively increased. To compensate for these specific effects, experiments are performed in which samples are stochastically prevented from being written to the buffer with a probability proportional to the increase in sampling frequency. The results of these experiments are indicated with [DE] (dropped experiences) in Figure 10 and are indeed better, but still worse than the performance for lower sampling frequencies.

The second potential reason for the drop in performance is that we have changed the problem definition by changing the sampling frequency. This is because the forgetting factor γ determines how far into the future we consider the effects of our actions according to:

$$\gamma = e^{-\frac{T_s}{\tau_\gamma}},$$

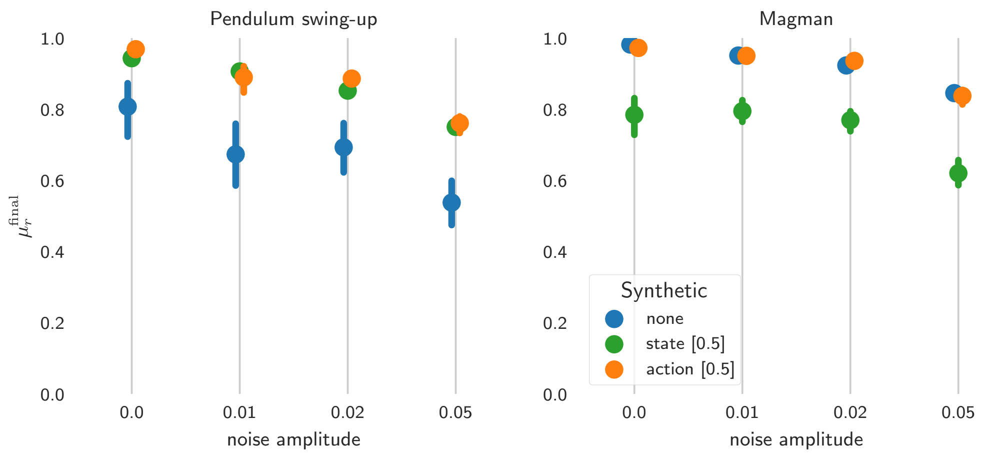


Figure 11: Experiments with altered experiences and sensor and actuator noise. Results are from the last 100 episodes of 50 learning runs. A description of the performance measures is given in Section 5.1.

where T_s is the sampling period in seconds and τ_γ is the lookahead horizon in seconds. To keep the same lookahead horizon, we recalculate γ, which is 0.95 in our other experiments ($T_s = 0.02$), to be $\gamma_{\text{pendulum}} = 0.9747$ ($T_s = 0.01$) and $\gamma_{\text{magman}} = 0.9873$ ($T_s = 0.005$). To keep the scale of the Q functions the same, which prevents larger gradients, the rewards are scaled down. Correcting the lookahead horizon was found to hurt performance on both benchmarks. The likely cause of this is that higher values of γ increase the dependence on the biased estimation of Q over the unbiased immediate reward signal (see Equation (4)). This can cause instability (François-Lavet et al., 2015).

6.5 Noise

The final environment property that we consider is the presence of sensor and actuator noise. So far, the agent has perceived the (normalized) environment state exactly and its (de-normalized) chosen actions have been implemented without change. Now we consider Equations (1) and (2) with $\sigma_s = \sigma_a \in \{0, 0.01, 0.02, 0.05\}$. The results of performing these experiments are shown in Figure 11. The results indicate that the need for data diversity is not dependent on the presence of noise. However, in Section 8.3 it will be shown that the methods used to determine which experiences are useful *can* be affected by noise.

6.6 Summary

This section has presented an investigation into how different aspects of the reinforcement learning problem at hand influence the need for experience diversity. In Table 3 a summary is given of the investigated aspects and the strength of their effect on the need for experience diversity. While this section has used the true environment model to examine the potential

Property	Effect	Explanation
Benchmark	Very high	The need for diverse states and actions largely depends on the ease and importance of generalizing across the state-actions space, which is benchmark dependent.
RL algorithm	Very high	Generalizing across the action space is fundamental to actor-critic algorithms, but not to critic-only algorithms with discrete action spaces.
Sampling frequency	High	The stability of RL algorithms depends heavily on the sampling frequency. Experience diversity can help learning stability. Having diverse actions at higher frequencies might be crucial as the size of their effect on the observed returns diminishes.
Buffer size	Medium	Small buffers can lead to rapidly changing data distributions, which causes unstable learning. Large buffers have more inherent diversity.
Sample age	Low	Although retaining old samples could theoretically be problematic, these problems were not clearly observable in practice.
Noise	None	The presence of noise was not observed to influence the need for experience diversity, although it can influence experience selection strategies, as will be shown in Section 8.3.

Table 3: The dependence of the need for diverse experiences on the investigated environment and reinforcement learning properties.

benefits of diversity, the next section will propose strategies to obtain diverse experiences in ways that are feasible on real problems.

7. New Experience-Selection Strategies

For the reasons discussed in Section 2, we do not consider changing the stream of experiences that an agent observes by either changing the exploration or by generating synthetic experiences. Instead, to be able to replay experiences with desired properties, valuable experiences need to be identified, so that they can be retained in the buffer and replayed from it. In this section we look at how several proxies for the utility of experiences can be used in experience selection methods.

7.1 Experience Retention

Although we showed in Section 6.4 that high sampling rates might warrant dropping experiences, in general we assume that each new experience has at least some utility. Therefore, unless stated otherwise, we will always write newly obtained experiences to the buffer. When the buffer is full, this means that we need some metric that can be used to decide which experiences should be overwritten.

7.1.1 Experience Utility Proxies

A criterion used to manage the contents of an experience replay buffer should be cheap enough to calculate,[2] should be a good proxy for the usefulness of the experiences and should not depend on the learning process in a way that would cause a feedback loop and possibly might destabilize that learning process. We consider three criteria for overwriting experiences.

Age: The default and simplest criterion is age. Since the policy is constantly changing and we are trying to learn its current effects, recent experiences might be more relevant than older ones. This (FIFO) criterion is computationally as cheap as it gets, since determining which experience to overwrite involves simply incrementing a buffer index. For smaller buffers, this does however make the buffer contents quite sensitive to the learning process, as a changing policy can quickly change the distribution of the experiences in the buffer. As seen in Figure 4, this can lead to instability.

Besides FIFO, we also consider reservoir sampling (Vitter, 1985). When the buffer is full, new experiences are added to it with a probability $\mathcal{C}/i$ where i is the index of the current experience. If the experience is written to the buffer, the experience it replaces is chosen uniformly at random. Note that this is the only retention strategy we consider that does not write all new experiences to the buffer. Reservoir sampling ensures that at every stage of learning, each experience observed so far has an equal probability of being in the buffer. As such, initial exploratory samples are kept in memory and the data distribution converges over time. These properties are shared with the FULL DB strategy, without needing the same amount of memory. The method might in some cases even improve the learning stability compared to using a full buffer, as the data distribution converges faster. However, when the buffer is too small this convergence can be premature, resulting in a buffer that does not adequately reflect the policy distribution. This can seriously compromise the learning performance.

Surprise: Another possible criterion is the unexpectedness of the experience, as measured by the temporal difference error δ from (4). The success of the Prioritized Experience Replay (PER) method of Schaul et al. (2016) shows that this can be a good proxy for the utility of experiences. Since the values have to be calculated to update the critic, the computational cost is very small if we accept that the utility values might not be

2. We have discussed the need for experience diversity in Section 6 and we have previously proposed overwriting a buffer in a way that directly optimized for diversity (de Bruin et al., 2016a). However, calculating the experience density in the state-action space is very expensive and therefore prohibits using the method on anything but small-scale problems.

current since they are only updated for experiences that are sampled. The criterion is however strongly linked with the learning process, as we are actively trying to minimize δ. This means that, when the critic is able to accurately predict the long term rewards of the policy in a certain region of the state-action space, these samples can be overwritten. If the predictions of the critic later become worse in this region, there is no way of getting these samples back. An additional problem might be that the error according to (4) will be caused partially by state and actuator noise. Keeping experiences for which the temporal difference error is high might therefore cause the samples saved in the buffer to be more noisy than necessary.

Exploration: We introduce a new criterion based on the observation that problems can occur when the amount of exploration is reduced. On physical systems that are susceptible to damage or wear, or for tasks where adequate performance is required even during training, exploration can be costly. This means that preventing the problems caused by insufficiently diverse experiences observed in Section 6 simply by sustained thorough exploration might not be an option. We therefore view the amount of exploration performed during an experience as a proxy for its usefulness. We take the 1-norm of the deviation from the policy action to be the usefulness metric. In our experiments on the small scale benchmarks we follow the original DDPG paper (Lillicrap et al., 2016) in using an Ornstein-Uhlenbeck noise process added to the output of the policy network. The details of the implementation are given in Appendix 9.3. In the experiments in Section 8.5 a copy of the policy network with noise added to the parameters is used to calculate the exploratory actions (Plappert et al., 2018).

For discrete actions, the cost of taking exploratory actions could be used as a measure of experience utility as well. The inverse of the probability of taking an action could be seen as a measure of the cost of the action. It could also be worth investigating the use of a low-pass filter, as a series of (semi)consecutive exploratory actions would be more likely to result in states that differ from the policy distribution in a meaningful way. These ideas are not tested here, as we only consider continuous actions in the remainder of this work.

Note that the size of the exploration signal is the deviation of the chosen action in a certain state from the policy action for that state. Since the policy evolves over time we could recalculate this measure of deviation from the policy actions per experience at a later time. Although we have investigated using this policy deviation proxy previously (de Bruin et al., 2016b), we found empirically that using the strength of the initial exploration yields better results. This can partly be explained by the fact that recalculating the policy deviation makes the proxy dependent on the learning process and partly by the fact that sequences with more exploration also result in different states being visited.

7.1.2 Stochastic Experience Retention Implementation

For the temporal difference error and exploration-based experience retention methods, keeping some experiences in the buffer indefinitely might lead to over-fitting to these samples.

Notation	Proxy	Explanation
Expl(α)	Exploration	Experiences with the least exploration are stochastically overwritten with new ones.
TDE(α)	Surprise	Experiences with the smallest temporal difference error are stochastically overwritten with new ones.
Resv	Age	The buffer is overwritten such that each experience observed so far has an equal probability of being in the buffer.

Table 4: New and uncommon experience retention strategies considered in this work.

Notation	Proxy	Explanation
Uniform + FIS	-	Experiences are sampled uniformly at random, FIS (Section 7.2) is used to account for the distribution changes caused by the *retention* policy.
PER+FIS	Surprise	Experiences are sampled using rank based stochastic prioritized experience replay based on the temporal difference error. Full importance sampling is used to account for the distribution changes caused by both the retention and sampling policies.

Table 5: New experience sampling strategies considered in this work.

Additionally, although the overwrite metric we choose might provide a decent proxy for the usefulness of experiences, we might still want to be able to scale the extent to which we base the contents of the buffer on this proxy. We therefore use the same stochastic rank-based selection criterion of (7) suggested by Schaul et al. (2016), but now to determine which experience in the buffer is overwritten by a new experience. We denote this as TDE(α) for the temporal difference-based retention strategy and Expl(α) for the exploration-based policy. Here, α is the parameter in (7) which determines how strongly the buffer contents will be based on the chosen utility proxy. A sensitivity analysis of α for both Expl and PER is given in Appendix 9.3. The notation used for the new experience retention strategies is given in Table 4.

7.2 Experience Sampling

For the choice of proxy when *sampling* experiences from the buffer, we consider the available methods from the literature: sampling either uniformly at random [Uniform], using stochastic rank-based prioritized experience replay [PER] and combining this with weighted importance sampling [PER+IS]. Given a buffer that contains useful experiences, these methods have shown to work well. We therefore focus on investigating how the experience retention and experience sampling strategies interact. In this context we introduce a weighted importance sampling method that accounts for the full experience selection strategy.

Importance sampling according to (8) can be used when performing prioritized experience replay from a buffer that contains samples with a distribution that is unbiased with respect to the environment dynamics. When this is not the case, we might need to compensate for the effects of changing the contents of the buffer, potentially in addition to the current change in the sampling probability. The contents of the buffer might be the result of many subsequent retention probability distributions. Instead of keeping track of all of these, we compensate for both the retention and sampling probabilities by using the number of times an experience in the buffer has actually been replayed. When replaying an experience i for the K-th time, we relate the importance-weight to the probability under uniform sampling from a FIFO buffer of sampling an experience X times, where X is at least K: $\Pr(X \geq K|_{\text{FIFO[Uniform]}})$. We refer to this method as Full Importance Sampling (FIS) and calculate the weights according to :

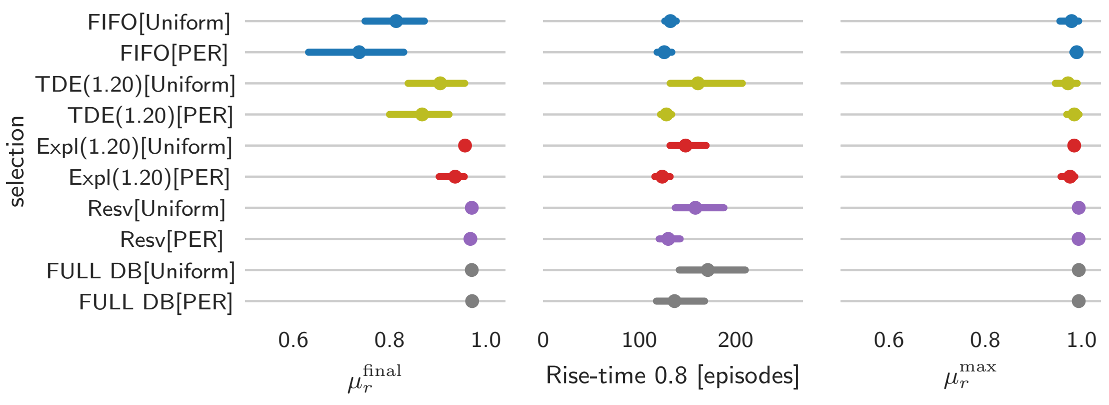
$$\omega_i^{\text{FIS}} = \left(\frac{\Pr(X \geq K|_{\text{FIFO[Uniform]}})}{\left[\sum_{j=1}^{\lceil np \rceil} \Pr(X \geq j|_{\text{FIFO[Uniform]}})\right] / np} \right)^{\beta}.$$

Here, n is the lifetime of an experience for a FIFO retention strategy in the number of batch updates, which is the number of batch updates performed so far when the buffer is not yet full. The probability of sampling an experience during a batch update when sampling uniformly at random is denoted by p. Note that np is the expected number of replays per experience, which following Schaul et al. (2016) we take as 8 by choosing the number of batch updates per episode accordingly. As in Section 3.3.2 we use β to scale between not correcting for the changes and correcting fully. Since the probability of being sampled at least K times is always smaller than one for $K > 0$, we scale the weights such that the sum of the importance weights for the expected np replays under FIFO[Uniform] sampling is the same as when not using the importance weights $(n \cdot p \cdot 1)$. The probability of sampling an experience at least K times under FIFO[Uniform] sampling is calculated using the binomial distribution:

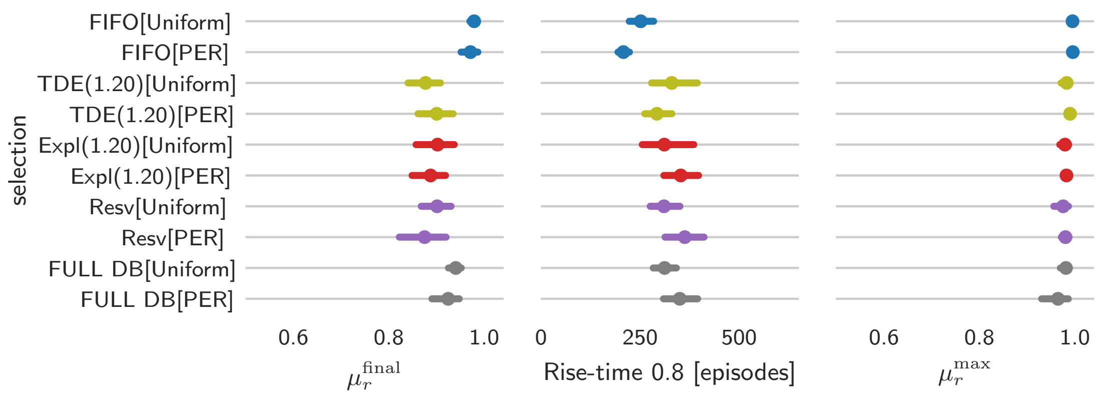
$$\Pr(X \geq K|_{\text{FIFO[Uniform]}}) = 1 - \sum_{k=0}^{K} \binom{n}{k} p^k (1-p)^{n-k}.$$

Correcting fully ($\beta = 1$) for the changed distributions would make the updates as unbiased as those from the unbiased FIFO uniform distribution (Needell et al., 2016). However, since the importance weights of experiences that are repeatedly sampled for stability will quickly go to zero, it might also undo the stabilizing effects that were the intended outcome of changing the distribution in the first place. Additionally, as discussed in Section 3.3.2, the FIFO Uniform distribution is not the only valid distribution. As demonstrated in Section 8.4, it is therefore important to determine whether compensating for the retention strategy is necessary before doing so.

The notation for the selection strategies with this form of importance sampling is given in Table 5.



(a) Swing-up



(b) Magman

Figure 12: Performance of the experience selection methods under the default conditions of moderate sampling frequencies and no state or actuator noise. A description of the performance measures is given in Section 5.1.

8. Experience Selection Results

Using the experience retention and sampling methods discussed in Section 7, we revisit the scenarios discussed in Section 6. We first focus on the methods without importance sampling, which we discuss separately in Section 8.4. Besides the tests on the benchmarks of Section 4, we also show results on six additional benchmarks in Section 8.5. There we also discuss how to choose the size of the experience buffer.

8.1 Basic Configuration

We start by investigating how these methods perform on the benchmarks in their basic configuration, with a sampling rate of 50 Hz and no sensor or actuator noise. The results are given in Figure 12 and show that it is primarily the combination of *retention* method and buffer size that determines the performance. It is again clear that this choice here depends on the benchmark. On the pendulum benchmark, where storing all experiences works well, the Resv method works equally well while storing only 10^4 experiences, which equals 50 of the 3000 episodes. On the magman benchmark, using a small buffer with only recent experiences works better than any other method.

Sampling according to the temporal difference error can be seen to benefit primarily the learning speed on the pendulum. On the magman, PER only speeds up the learning process when sampling from recent experiences. When sampling from diverse experiences, PER will attempt to make the function approximation errors more even across the state-action space, which as discussed before, hurts performance on this benchmark.

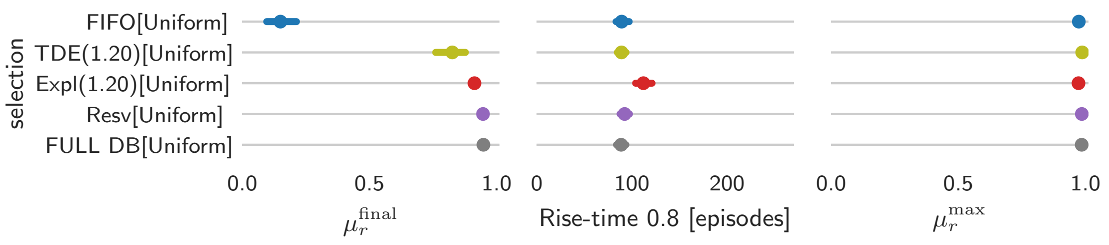
8.2 Effect of the Sampling Frequency

For higher sampling frequencies, the performance of the different experience selection methods is shown in Figure 13. We again see that higher sampling frequencies place different demands on the training data distribution. With the decreasing exploration, retaining the right experiences becomes important. This is most visible on the Magman benchmark where FIFO retention, which resulted in the best performance at the end of training for the base sampling frequency, now performs worst. Retaining all experiences works well on both benchmarks. When not all experiences can be retained, the reservoir retention method is still a good option here, with the exploration-based method a close second.

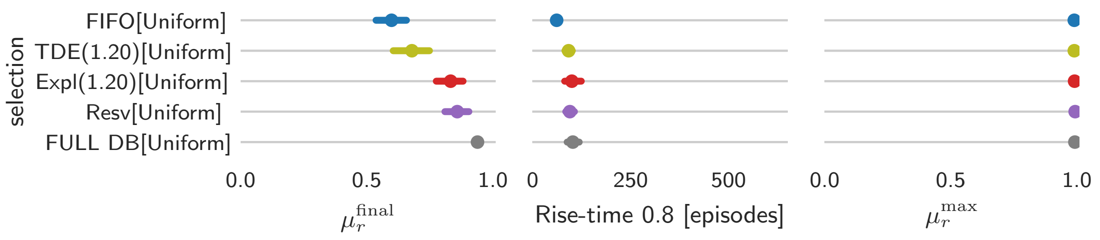
8.3 Sensor and Actuator Noise

We also test the performance of the methods in the presence of noise, similarly to Section 6.5. The main question here is how the noise might affect the methods that use the temporal difference error δ as the usefulness proxy. The concern is that these methods might favor noisy samples, since these samples might cause bigger errors. To test this we perform learning runs on the pendulum task while collecting statistics on all of the experiences in the mini-batches that are sampled for training. The mean absolute values of the noise in the experiences that are sampled are given in Table 6. It can be seen that the temporal difference error-based methods indeed promote noisy samples. The noise is highest for those dimensions that have the largest influence on the value of Q.

In Figure 14 the performance of the different methods on the two benchmarks with noise is given. The tendency to seek out noisy samples in the buffer is now clearly hurting the performance of PER sampling, as the performance with PER is consistently worse than with uniform sampling. For our chosen buffer size the retention strategy is still more influential and interestingly the TDE-based retention method does not seem to suffer as much here. The relative rankings of the retention strategies are similar to those without noise.



(a) Swing-up [100 Hz]

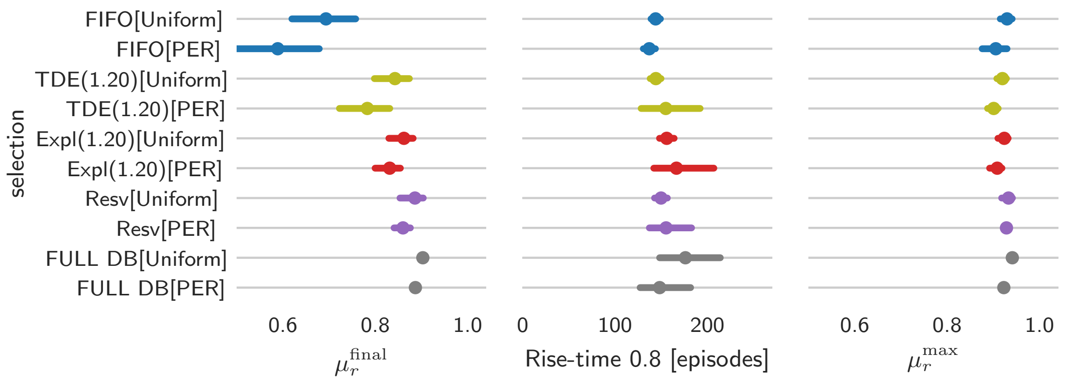


(b) Magman [200 Hz]

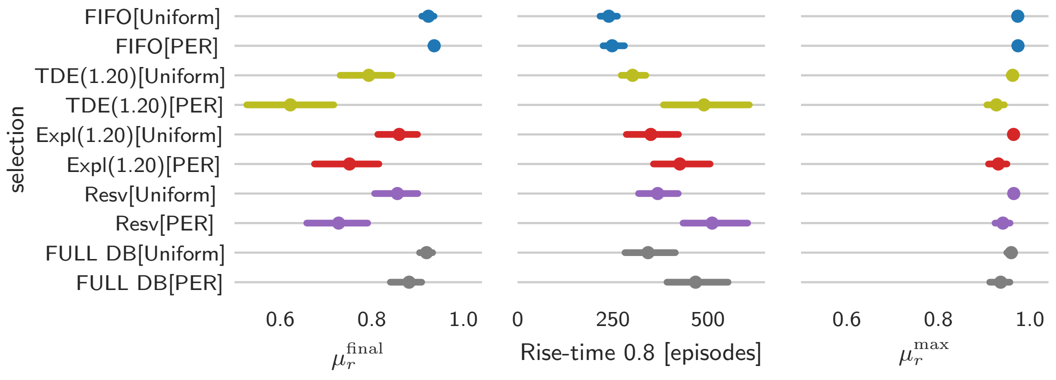
Figure 13: Performance of the experience selection methods with increased sampling frequencies. Results are from 50 learning runs. A description of the performance measures is given in Section 5.1.

	position	velocity	action
Expl(1.0)[Uniform]	$1.584 \cdot 10^{-2}$	$1.582 \cdot 10^{-2}$	$1.594 \cdot 10^{-2}$
Expl(1.0)[PER]	$1.654 \cdot 10^{-2}$	$1.630 \cdot 10^{-2}$	$1.595 \cdot 10^{-2}$
TDE(1.0)[Uniform]	$1.713 \cdot 10^{-2}$	$1.627 \cdot 10^{-2}$	$1.598 \cdot 10^{-2}$
TDE(1.0)[PER]	$1.846 \cdot 10^{-2}$	$1.743 \cdot 10^{-2}$	$1.594 \cdot 10^{-2}$

Table 6: Mean absolute magnitude of the noise per state-action dimension in the mini batches as a function of the experience selection procedure.



(a) Swing-up $[\sigma_s = \sigma_a = 0.02]$



(b) Magman $[\sigma_s = \sigma_a = 0.02]$

Figure 14: Performance of the experience selection methods with with sensor and actuator noise. Results are from 50 learning runs. A description of the performance measures is given in Section 5.1.

8.4 Importance Sampling

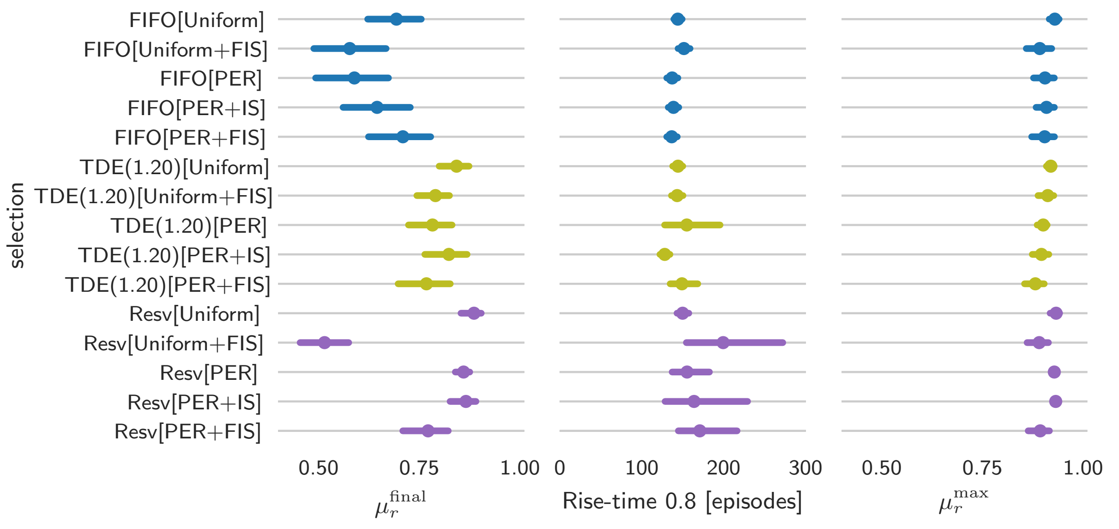
Finally, we investigate the different importance sampling strategies that were discussed in Sections 3.3.2 and 7.2. We do this by using the FIFO, TDE and Resv retention strategies as representative examples. We consider the benchmarks with noise, since as we discussed in Section 3.3.2, the stochasticity in the environment can make importance sampling more relevant. The results are shown in Figure 15. We discuss per retention strategy how the sample distribution is changed and whether the change introduces a bias that should be compensated for through importance sampling.

FIFO: This retention method results in an unbiased sample distribution. When combined with uniform sampling, there is no reason to compensate for the selection method. Doing so anyway (FIFO[Uniform + FIS]) results in downscaling the updates from experiences that happen to have been sampled more often than expected, effectively reducing the batch-size while not improving the distribution. The variance of the updates is therefore increased without reducing bias. This can be seen to hurt performance in Figure 15, especially on the swing-up task where sample diversity is most important. Using PER also hurts performance in the noisy setting as this sampling procedure *does* bias the sample distribution. Using importance sampling to compensate for just the sampling procedure (FIFO[PER+IS]) helps, but the resulting method is not clearly better than uniform sampling.

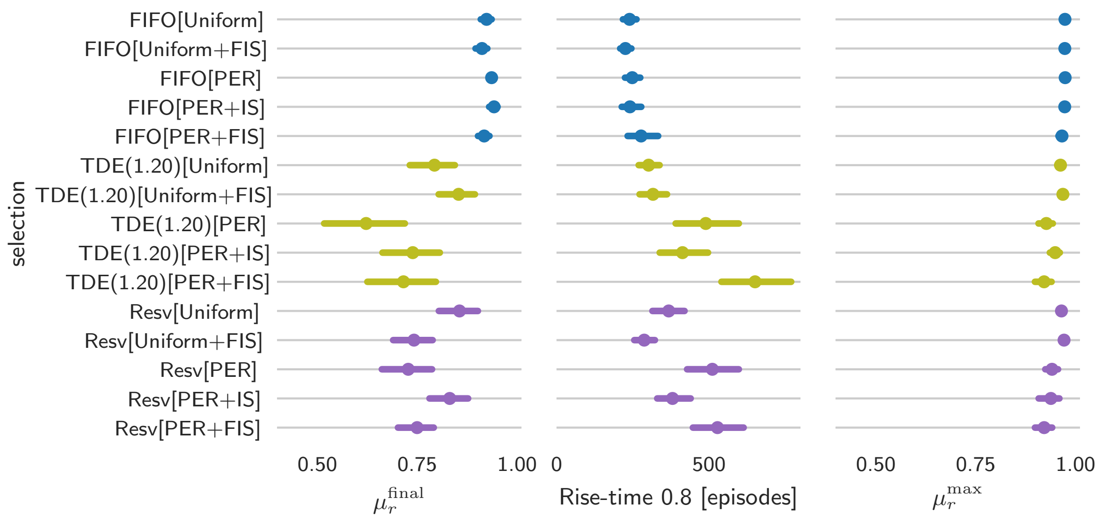
TDE: When the retention strategy is based on the temporal difference error, there is a reason to compensate for the bias in the sample distribution. It can be seen from Figure 15 however, that the full importance sampling scheme improves performance on the magman benchmark, but not on the swing-up task. The likely reason is again that importance sampling indiscriminately compensates for both the unwanted re-sampling of the environment dynamics and reward distributions as well as the beneficial re-sampling of the state-action space distribution. The detrimental effects of compensating for the latter seen to outweigh the beneficial effects of compensating for the former on this benchmark where state-action space diversity has been shown to be so crucial.

Resv: The reservoir retention method is not biased with respect to the reward function or the environment dynamics. Although the resulting distribution is strongly off-policy (assuming the policy has changed during learning), this does not present a problem for a deterministic policy gradient algorithm with Q-learning updates, other than that it might be harder to learn a function that generalizes to a larger part of the state space. When sampling uniformly, we do sample certain experiences, from early in the learning process, far more often than would be expected under a FIFO[Uniform] selection strategy. The FIS method compensates for this by weighing these experiences down, effectively reducing the size of both the buffer and the mini-batches. In Figure 15, this can be seen to severely hurt the performance on the swing-up problem, as well as the learning stability on the magman benchmark.

Interesting to note is that on these two benchmarks, for all three considered retention strategies, using importance sampling to compensate for the changes introduces by PER



(a) Swing-up $[\sigma_s = \sigma_a = 0.02]$



(b) Magman $[\sigma_s = \sigma_a = 0.02]$

Figure 15: Performance of representative experience selection methods with and without importance sampling on the benchmarks with sensor and actuator noise. A description of the performance measures is given in Section 5.1.

only improved the performance significantly when using PER resulted in poorer performance than not using PER. Similarly, using FIS to compensate for the changes introduced in the buffer distribution only improved the performance when those changes should not have been introduced to begin with.

8.5 Additional Benchmarks

The computational and conceptual simplicity of the two benchmarks used so far allowed for comprehensive tests and a good understanding of the characteristics of the benchmarks. However, we also saw that the right experience selection strategy is benchmark dependent. Furthermore, deep reinforcement learning yields most of its advantages over reinforcement learning without simpler function approximation on problems with higher dimensional state and action spaces. To obtain a more complete picture we therefore perform additional tests on 6 benchmarks of varying complexity.

8.5.1 Benchmarks

In the interest of reproducibility, we use the open source RoboSchool (Klimov, 2017) benchmarks together with the openAI baselines (Dhariwal et al., 2017) implementation of DDPG. We have adapted the baselines code to include the experience selection methods considered in this section. Our adapted code is available online.[3]

The baselines version of DDPG uses Gaussian noise added to the parameters of the policy network for exploration (Plappert et al., 2018). In contrast to the other experiments in this work, the strength of the exploration is kept constant during the entire learning run. For the Expl method we still consider the 1-norm of the distance between the exploration policy action and the unperturbed policy action as the utility of the sample.

For the benchmarks listed in Table 7, we compare the default FULL DB[Uniform] selection strategy in the baselines code to the alternative retention strategies considered in this work with uniform sampling. We show the maximum performance for these different retention strategies as a function of the buffer size in Figure 16.

8.5.2 Results

As shown in Figure 16, on these noise-free benchmarks with constant exploration and moderate sampling frequencies, the gains obtained by using the considered non-standard experience selection strategies are limited. However, in spite of the limited number of trials performed due to the computational complexity, trends do emerge on most of the bench-

3. The code is available at https://github.com/timdebruin/baselines-experience-selection.

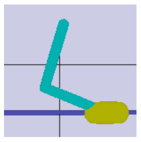
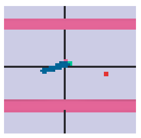
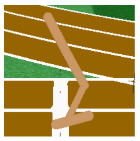
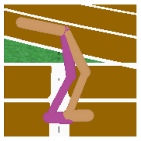
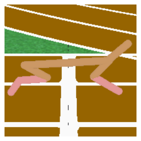
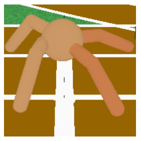
	InvDoublePnd	Reacher	Hopper	Walker2d	HalfCheetah	Ant
						
$\|S\|$	9	9	15	22	26	28
$\|A\|$	1	2	3	6	6	8

Table 7: The RoboSchool benchmarks considered in this section with the dimensionalities of their state and action spaces.

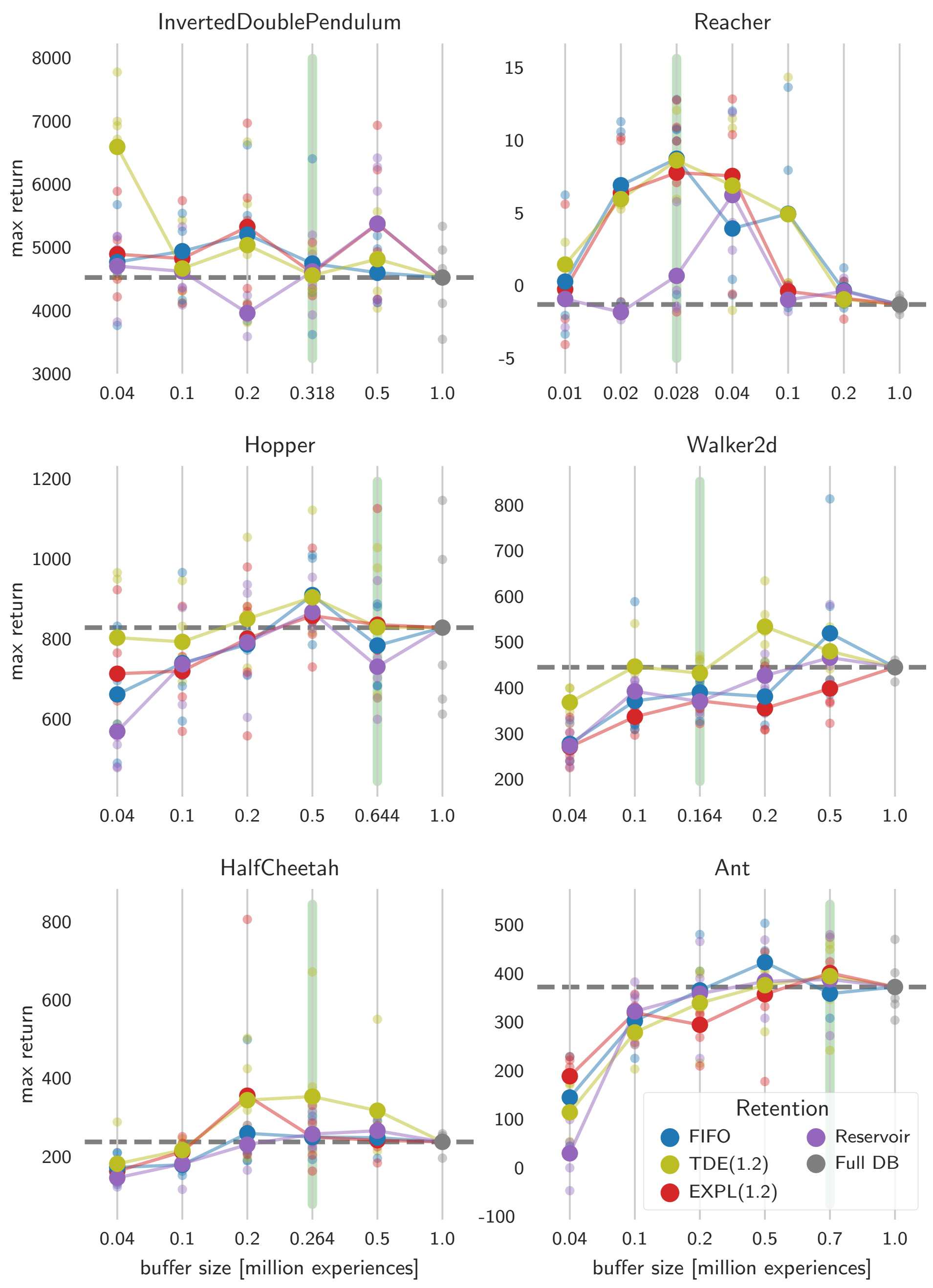


Figure 16: Maximum performance during a training run on the Roboschool benchmarks as a function of the retention strategy and buffer size. Results for the individual runs and their means are shown. In Appendix 9.3, we additionally show the mean (Figure 26) and final (Figure 25) performance. Green lines indicate the rule of thumb buffer sizes of Figure 17.

marks. On all benchmarks, the best performance is seen not when retaining all experiences, but rather when learning from a smaller number of experiences. This is most visible on the reacher task, which involves learning a policy for a 2-DOF arm to move from one random location in its workspace to another random location. For this task, the best performance for all retention strategies is observed when retaining less than a tenth of all experiences.

For these noise-free benchmarks, the temporal difference error is an effective proxy for the utility of the experiences, resulting in the highest or close to the highest maximum performance on all benchmarks.

The exploration-based retention strategy was introduced to prevent problems when reducing exploration and for high sampling frequencies. Since the exploration is not decayed and the sampling frequencies are modest, there is no real benefit when applying this strategy to these benchmarks. However, it also does not seem to hurt performance compared to the age-based retention strategies. The constant exploration on these benchmarks additionally means that the performance of FIFO and Reservoir retention are rather close, although due to premature convergence of the data distribution, reservoir retention does suffer the most when the buffer capacity is too low.

Figures 16, 25 and 26 show that when the right proxy for the utility of experiences is chosen, performance equal to and often exceeding that of retaining all experiences can be obtained while using only a fraction of the memory. This begs the question of how to choose the buffer size.

As it tends to result in more stable learning, retaining as many experiences as possible seems a sensible first choice for the buffer size. We therefore base our suggestion for subsequently tuning the buffer size on the learning curves of the FULL DB[Uniform] method. The complexity of the control task at hand determines the minimal number of environment steps required to learn a good policy, as well as the number of experiences that need to be retained in a buffer for decent learning performance. We propose to use the number of experiences needed to get to 90% of the final performance as a *rough* empirical estimate of the optimal buffer size. We show this rule of thumb in Figure 17 and have indicated the experiments with the proposed buffer sizes in Figure 16 with vertical green lines.

Instead of iteratively optimizing the buffer size over several reinforcement learning trials, extrapolation of the learning curve (Domhan et al., 2015) could also be used to limit the buffer capacity when the remaining learning performance increase is expected to be small. This would allow the method to work immediately for novel tasks.

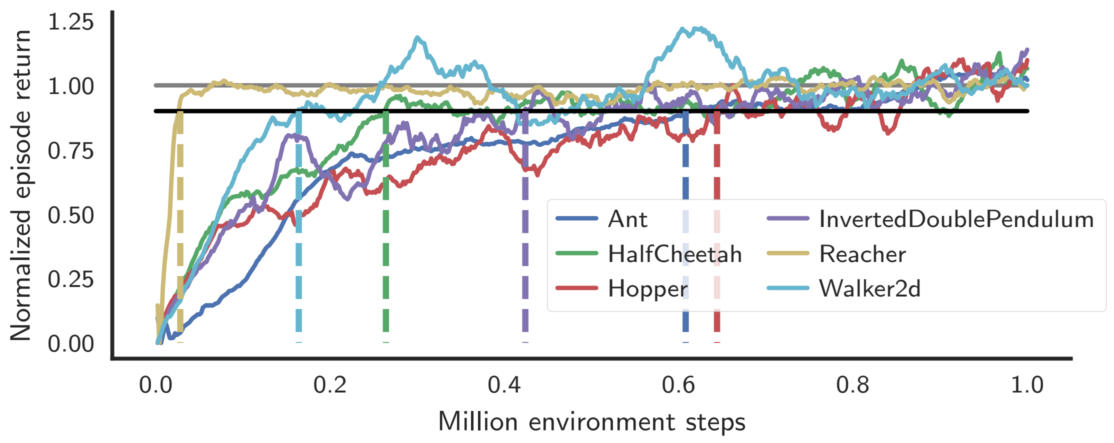


Figure 17: Learning curves of the FULL DB method on the different benchmarks, averaged over 5 trials. The curves are normalized by the final performance (the mean performance over the last $2 \cdot 10^5$ steps). Indicated are the number of steps needed to get to 90% of the final performance.

9. Conclusions and Recommendations

In this work, we have investigated how the characteristics of a control problem determine what experiences should be replayed when using experience replay in deep reinforcement learning.

We first investigated how factors such as the generalizability over the state-action space and the sampling frequency of the control problem influenced the balance between the need for on-policy experiences versus a broader coverage of the state-action space.

We then investigated a number of proxies for the utility of experiences which we used to both decide which experiences to retain in a buffer and how to sample from that buffer. We performed experiments that showed how these methods were affected by noise, increased sampling frequencies and how their performance varied across benchmarks and experience buffer sizes.

Based on these investigations we present a series of recommendations below for the three choices concerning experience selection: how to choose the capacity of the experience replay buffer, which experiences to retain in a buffer that has reached its capacity and how to sample from that buffer. These choices together should ensure that the experiences that are replayed to the reinforcement learning agent facilitate quick and stable learning of a policy with good performance. An example of applying the procedure outlined below on the Magman benchmark is given in Figure 18. Note the proposed methods are especially relevant when faced by issues that might occur in a physical-control setting, such as a need for constrained exploration, high or low sampling frequencies, the presence of noise and hardware limitations that limit the experience buffer size. Section 8.5 shows that the potential gains might be limited for processes where these problems do not occur.

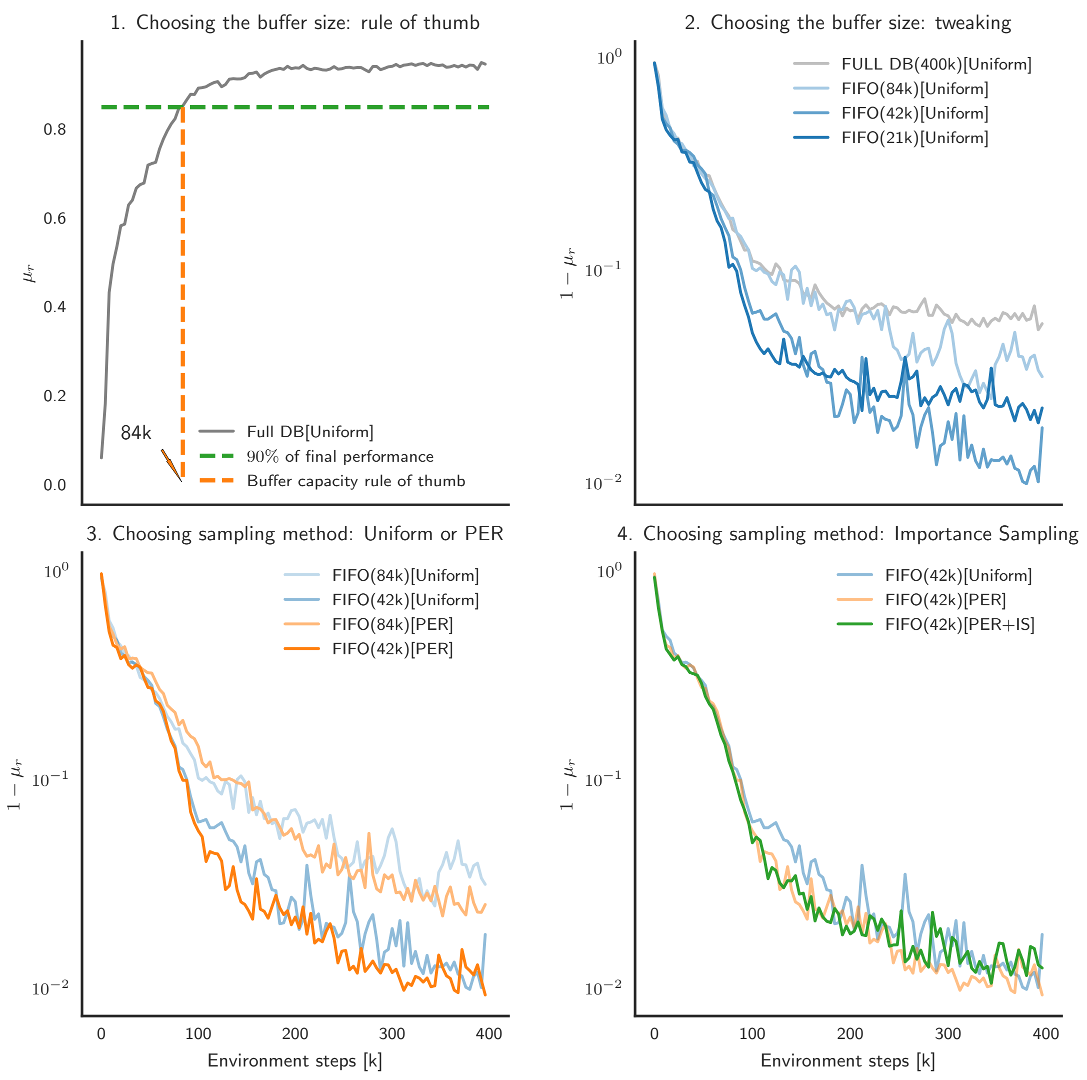


Figure 18: Demonstration of the proposed process for the magman benchmark. 1: Based on the performance of the Full DB[Uniform] method, the rule of thumb indicates a buffer capacity of 84×10^3 experiences. As there are no special circumstances such as high sampling frequencies and the magman requires a very precise policy that does not easily generalize due to the highly nonlinear behavior of the magnets, FIFO retention is used. 2: By exploring around the proposed buffer size, a buffer capacity of 42×10^3 experiences is chosen. 3: Sampling from the buffer based on the temporal difference error can help speed up and stabilize learning, but is very dependent on the experiences that are in the buffer to begin with. 4: Since the benchmark fully deterministic (noise free), importance sampling is not needed in this case and can be seen to undo some of the benefits of PER.

9.1 Choosing the Buffer Capacity

Although it is not the best retention strategy in most of the benchmarks we have considered, retaining as many experiences as possible is a good place to start. This tends to result in more stable learning, even if the eventual performance is not always optimal.

If the learning curve for the FULL DB experiments reaches a level of performance close to the performance after convergence in significantly fewer environment steps than there are experience samples in the buffer, it might be worthwhile reducing the size of the buffer. Our proposed rule of thumb is to to make the buffer size roughly equal to 90% the number of environment steps needed to reach the final performance level.

9.2 Choosing the Experience Utility Proxy

When not all experiences are retained in the buffer, a proxy for the utility of the individual experiences is needed to determine which experiences to retain and which to discard. In this work, we have discussed strategies based on several proxies and shown that the right strategy is problem dependent. Although finding the right one will likely require some experimentation, we discuss here what properties of the control problem at hand make certain strategies more likely or less likely to succeed.

FIFO: Although off-policy reinforcement-learning algorithms can learn from samples obtained by a different policy than the optimal policy that is being learned, the reality of deep reinforcement learning is that a finite amount of shared function approximation capacity is available to explain all of the training data. While simply using larger networks might help, we show in Appendix 9.3 that learning only from more recent data (which corresponds more closely to the policy being learned) can work better. A large potential downfall presents itself when the policy suddenly changes in a way that changes the distribution of the states that are visited. As shown in Section 6.2, this can quickly destabilize the learning process. Extra care should be taken when using FIFO retention in combination with decaying exploration. This is especially true for problems where multiple policies are possible that give similar returns but distinct state-space trajectories, such as swinging up a pendulum either clockwise or anti-clockwise.

TDE: The idea behind selecting certain experiences over others is that more can be learned from these samples. The temporal difference error is therefore an interesting proxy, especially during the early stages of the learning process when the error is mostly caused by the fact that the value function has not been accurately learned yet. In the experiments of Schaul et al. (2016) as well as in our own experiments, prioritizing experiences with larger TD errors was observed to improve both the speed of learning as well as the eventual performance in many cases. The downside of using the TD error as an experience utility proxy is that the error can also be caused by sensor and actuator noise, environment stochasticity or function approximation accuracy differences as a result of differences in the state space coverage. We have shown in Section 8.3 how noise can hurt the performance of the algorithm when using this proxy and argued in Section 3.3.2 how this proxy introduces a harmful bias in the presence of environment stochasticity.

Exploration: We introduced an additional proxy based on the observation that, on physical systems, exploration can be costly. By using the strength of the exploration signal as a proxy for the utility of the experience, some of the problems mentioned for the FIFO strategy when reducing exploration can be ameliorated. As shown in Section 6.4 and Section 8.2, sufficient diversity in the action space is most important when the dependency of the value function on the action is relatively small, such as for increased sampling frequencies. The downside of this strategy is that since it focuses on early experiences that are more off-policy, it can take longer for the true value function to be learned. Besides the impact on training speed, the focus on off-policy data can also limit the maximum controller performance.

Reservoir sampling: By using reservoir sampling as a retention strategy, the buffer contains, at all times, samples from all stages of learning. As with the exploration-based policy, this ensures that initial exploratory samples are retained which can significantly improve learning stability on domains where FIFO retention does not work. However, of the methods mentioned here, reservoir sampling is the one most severely impacted by a too small experience buffer, as the data distribution in the buffer will converge prematurely and will not cover the state-action space distribution of the optimal policy well enough.

9.3 Experience Sampling and Importance Sampling

The experiences that are used to learn from are not just determined by the buffer retention strategy, but also by the method of sampling experiences from the buffer. While the retention strategy needs to ensure that a good coverage of the state-action space is maintained in the buffer throughout learning, the sampling strategy can seek out those experiences that can result in the largest immediate improvement to the value function and policy. It can therefore be beneficial to *sample* based on the temporal difference error (as suggested by Schaul et al. 2016), which can improve learning speed and performance, while basing the *retention* strategy on a more stable criterion that either promotes stability or ensures that only samples from the relevant parts of the state-action space are considered by the sampling procedure.

As discussed in Sections 3.3.2 and 8.4, selecting experiences based on the temporal difference error in stochastic environments introduces a bias that should be compensated for through weighted importance sampling in order to make the learning updates valid. While the other experience selection methods in this work change the distribution of the samples, these changed distributions are still valid for an off-policy deterministic gradient algorithm.

Acknowledgments

This work is part of the research programme Deep Learning for Robust Robot Control (DL-Force) with project number 656.000.003, which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO).

Appendix A. Simple Benchmarks

Here, a more detailed mathematical description is given of the pendulum swing-up and magnetic manipulation benchmarks. A high level description of these benchmarks was given in Section 4.

The benchmarks will be described based on their true (unnormalized) physical environment states s_{unnorm} and actions a_{unnorm}. In the main body of this work the components of these states and actions are normalized: $s, s_{\mathcal{E}} \in [-1,1]^n, a, a_{\mathcal{E}} \in [-1,1]^m$. See Figure 1 for a description of the symbols used.

The dynamics of both problems are defined as differential equations, which we use to calculate the next environment state s'_{unnorm} as a function of the current state s_{unnorm} and action a_{unnorm} using the (fourth order) Runge-Kutta method. The reward is in both cases given by:

$$r = -(W_1|s'_{\text{unnorm}} - s_{\text{unnorm}_{\text{ref}}}| + W_2|a_{\text{unnorm}}|). \tag{9}$$

In both cases a fixed reference state $s_{\text{unnorm}_{\text{ref}}}$ is used.

A.1 Pendulum Swing-Up

For the pendulum swing-up task, the state s_{unnorm} is given by the angle $\theta \in [-\pi, \pi]$ and angular velocity $\dot{\theta}$ of a pendulum, which starts out hanging down under gravity $s^0_{\text{unnorm}} = [\theta\ \dot{\theta}]^T = [0\ 0]^T$. For the normalization of the velocities, $\theta_{\min} = -30\,\text{rad}\,\text{s}^{-1}$ and $\theta_{\max} = 30\,\text{rad}\,\text{s}^{-1}$ are used. The action space is one dimensional: it is the voltage applied to a motor that exerts torque on the pendulum $a_{\text{unnorm}} \in [-3, 3]$ V. The angular acceleration of the pendulum is given by:

$$\ddot{\theta} = \frac{-Mgl\sin(\theta) - (b + K^2/R)\dot{\theta} + (K/R)a_{\text{unnorm}}}{J}.$$

Where $J = 9.41 \times 10^{-4}\,\text{kg}\,\text{m}^2$, $M = 5.5 \times 10^{-2}\,\text{kg}$, $g = 9.81\,\text{m}\,\text{s}^{-2}$, $l = 4.2 \times 10^{-2}\,\text{m}$, $b = 3 \times 10^{-6}\,\text{kg}\,\text{m}^2\,\text{s}^{-1}$, $K = 5.36 \times 10^{-2}\,\text{kg}\,\text{m}^2\,\text{s}^{-2}\,\text{A}^{-1}$ and $R = 9.5\,\text{V}\,\text{A}^{-1}$ are respectively the pendulum inertia, the pendulum mass, the acceleration due to gravity, pendulum length, viscous damping coefficient, the torque constant and the rotor resistance (Alibekov et al., 2018). For this task $W_1 = [50\ 1]$ and $W_2 = 10$ and $s_{\text{unnorm}_{\text{ref}}} = [-\pi\ 0]^T = [\pi\ 0]^T$. The absolute value of the state is used in (9).

A.2 Magnetic Manipulation

In the magnetic manipulation problem, the action space represents the squared currents through four electromagnets under the track; $a^j_{\text{unnorm}} \in [0,\ 0.6]A^2$ for $j = 1, 2, 3, 4$. The state of the problem is defined as the position $x \in [-0.035,\ 0.105]$ m of the ball relative to the center of the first magnet and the velocity $\dot{x}$ $\text{m}\,\text{s}^{-1}$ of the ball: $s_{\text{unnorm}} = [x\ \dot{x}]^T$. For the normalization of the velocities, $\dot{x}_{\min} = -0.4\,\text{m}\,\text{s}^{-1}$ and $\dot{x}_{\max} = 0.4\,\text{m}\,\text{s}^{-1}$ are used. When the position of the ball exceeds the bounds, the position is set to the bound and the velocity is set to $0.01\,\text{m}\,\text{s}^{-1}$ away from the wall. An additional reward of -1 is given for the time-step at which the collision occurred. The acceleration of the ball is given by:

$$\ddot{x} = -\frac{b}{m}\dot{x} + \frac{1}{m}\sum_{j=1}^{4} g\left(x, j\right) a^{j}_{\text{unnorm}},$$

with

$$g(x, j) = \frac{-c_1 \, (x - 0.025j)}{\left((x - 0.025j)^2 + c_2\right)^3}.$$

Here, $g(x, j)$ is the nonlinear magnetic force equation, $m = 3.200 \times 10^{-2}\,\mathrm{kg}$ the ball mass, and $b = 1.613 \times 10^{-2}\,\mathrm{N\,s\,m^{-1}}$ the viscous friction of the ball on the rail. The parameters c_1 and c_2 were empirically determined from experiments on a physical setup to be $c_1 = 5.520 \times 10^{-10}\,\mathrm{N\,m^5\,A^{-1}}$ and $c_2 = 1.750 \times 10^{-4}\,\mathrm{m^2}$ (Alibekov et al., 2018).

For the magnetic manipulation problem we take $W_1 = [100\ 5]$, $W_2 = [0\ 0\ 0\ 0]$, $s^0_{\text{unnorm}} = [0\ 0]^T$ and $s_{\text{unnorm}_{\text{ref}}} = [0.035\ 0]^T$ in (9).

Appendix B. Implementation Details

This appendix discusses the chosen hyperparameters of the methods discussed in Section 3, that were used to obtain the results in this paper. Only those hyperparameters that were not explicitly mentioned in the earlier sections of this work are mentioned here.

B.1 Neural Networks

This appendix describes the architecture and training procedure of the used neural networks.

Swing-up and Magman

To perform the experiments in this work, the DDPG method of Lillicrap et al. (2016) was reimplemented in Torch (Collobert et al., 2011). For all experiments except for the control experiment in Appendix 9.3, the actor and critic networks had the following configuration:

The actor is a fully connected network with two hidden layers, each with 50 units. The hidden layers have rectified linear activation functions. The output layer has hyperbolic tangent nonlinearities to map to the normalized action space.

The critic is a fully connected network with three hidden layers. The layers have rectified linear activation functions and 50, 50 and 20 units respectively. The state is the input to the first hidden layer, while the action is concatenated with the output of the first hidden layer and used as input to the second hidden layer. The output layer is linear.

To train the networks, the ADAM optimization algorithm is used (Kingma and Ba, 2015). We use a batch size of 16 to calculate the gradients. For all experiments we use 0.9 and 0.999 as the exponential decay rates of the first and second order moment estimates respectively. The step-sizes used are 10^{-4} for the actor and 10^{-3} for the critic. We additionally use $\mathcal{L}_2$ regularization on the critic weights of 5×10^{-3}.

For the DQN experiments, a critic network similar to the DDPG critic was used. The critic-only differs in the fact that instead of having actions as an input, the output size is increased to the number of discrete actions considered. The parameters θ^- of the target critic are updated to equal the online parameters θ every 200 batch updates.

Roboschool Benchmarks

For the experiments on the Roboschool benchmarks, we use a slightly modified version of the DDPG implementation in the openAI baselines (Dhariwal et al., 2017) repository. We have adapted the baselines code to include the experience selection methods considered in this section. Our adapted code is available online.[4] We here summarize the relevant differences from the implementation used on the simple benchmarks.

The actor and critic networks have two hidden layers with 64 units each. Layer normalization (Ba et al., 2016) is used in both networks after both hidden layers. The multiplier of the $\mathcal{L}_2$ regularization on the weights of the critic with is 1×10^{-2}. A batch size of 64 is used, with a sample reuse of 32. Training is performed every 100 environment steps, rather than after completed episodes.

B.2 Exploration

Swing-Up and Magman

We use an Ornstein-Uhlenbeck noise process (Uhlenbeck and Ornstein, 1930) as advocated by Lillicrap et al. (2016). The dynamics of the noise process are given by

$$u(k+1) = u(k) + \theta\mathcal{N}(0,1) - \sigma u(k).$$

The equation models the velocity of a Brownian particle with friction. We use $\theta = 5.14$, $\sigma = 0.3$. Using this temporally correlated noise allows for more effective exploration in domains such as the pendulum swing-up. It also reduces the amount of damage on physical systems relative to uncorrelated noise (Koryakovskiy et al., 2017). For high frequencies, uncorrelated noise is unlikely to result in more than some small oscillations around the downward equilibrium position.

The noise signal is clipped between -1 and 1 after which it is added to the policy action and clipped again to get the normalized version of the control action a.

For the DQN experiments, epsilon greedy exploration was used with the probability of taking an action uniformly at random decaying linearly from $\epsilon = 0.7$ to $\epsilon = 0.01$ over the first 500 episodes.

Roboschool Benchmarks

For easy comparison to other work, we use the exploration strategy included in the baselines code. This means that for the Roboschool benchmarks we do not decay the strength of the exploration signal over time. Compared to our other benchmarks, the second difference is that the noise is added in the parameter space of the policy rather than directly in the action space (Plappert et al., 2018). The amplitude of the noise on the parameters is scaled such that the standard deviation of the exploration signal in action-space is 0.2.

Appendix C. Baseline Controller

In this work we use the fuzzy Q-iteration algorithm of Buşoniu et al. (2010) as a baseline. This algorithm uses full knowledge of the system dynamics and reward function to compute

4. The code is available at https://github.com/timdebruin/baselines-experience-selection.

a controller that has a proven bound on its sub-optimality for the deterministic (noise-free) case.

For the tests with sensor and actuator noise, the same controller as in the noise-free setting is used. To make the performance normalization (Section 5.1) fair, the performance of the controller is taken as the mean of 50 repetitions of taking the maximum obtained mean reward per episode over 1000 episodes with different realizations of the noise:

$$r_{\text{baseline with noise}} = \frac{1}{50}\sum_{i=1}^{50} \max(r^{\text{mean}}_{\text{episode i,1}}, \ldots, r^{\text{mean}}_{\text{episode i,1000}}).$$

Note that although this equalizes the chances of getting a favorable realization of the sensor and actuator noise sequences, it does not compensate for the fact that the fuzzy Q-iteration algorithm is unsuitable for noisy environments. Since the DDPG method used in this work can adjust the learned policy to the presence of noise in the environment, it outperforms the baseline in some situations. This is not an issue since we are interested in the relative performance of different experience selection strategies and only use the baseline as a reference point.

Appendix D. Additional Sensitivity Analyses and Figures

This section contains additional analyses and figures that were left out of the main body of the paper for brevity.

D.1 Performance on the Magman Benchmark as a Function of Network Size

In the main body of the paper, a number of experiments are shown in which the performance of the magman benchmark is better with a small FIFO experience buffer than it is when retaining all experiences. As we use relatively small neural networks on the magman benchmark, it could be expected that at least part of the reason that training on all experiences results in poorer performance is that the function approximator simply does not have enough capacity to accurately cover the state-action space. We therefore compare the performance of the networks used on the magman benchmark in the main body of this work to that of the original DDPG architecture, which has more than 40 times as many parameters. Table 8 compares the network architectures and the number of parameters of

Architecture	hidden layer units	parameters swing-up	parameters magman
Small-critic	[50, 50, 20]	3791	3941
Small-actor	[50, 50]	2751	2904
DDPG original-critic	[400, 300]	122101	123001
DDPG original-actor	[400, 300]	121801	122704

Table 8: The architectures of the networks compared in this section, with the number of parameters.

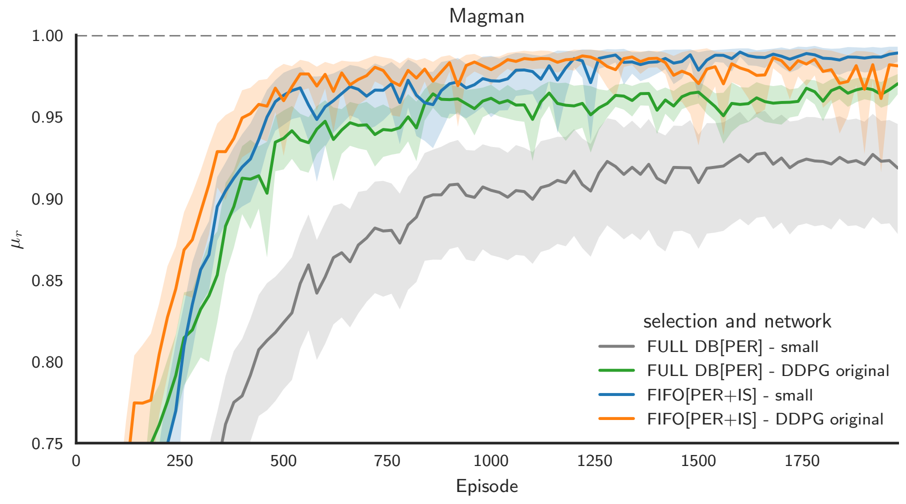


Figure 19: Influence of network size on the performance of the magman benchmark, when retaining all 4×10^5 experiences (FULL DB) versus retaining only the last 10^4 experiences (FIFO). The small policy network used for most of the experiments on the magman has 2904 parameters, while the original DDPG network has 122704 parameters on the magman benchmark. In both cases the critic networks had slightly more parameters.

these architectures. It can be seen from Figure 19 that, while the larger network is able to learn more successfully from the FULL DB buffer, it is outperformed by both the small and the large network using the FIFO buffer. The eventual performance is best for our smaller network trained on a small buffer, although learning is somewhat faster with the larger network.

D.2 Sensitivity Analysis α

In both the PER sampling as well as the TDE and Expl retention methods, the parameter α (7) determines how strongly the used experience utility proxy influences the selection method. Here, we show the sensitivity of both PER (Figure 21) and Expl (Figure 20) with respect to this parameter.

In Figure 20 it can be seen that on the Pendulum benchmark, where Expl retention has already been shown to aid stability, increasing α helps to improve the final performance more. This increased stability comes at the cost of somewhat reduced maximum performance. With PER sampling it does not seem to hurt the learning speed. On the Magman benchmark, where FIFO retention works better than Expl retention, increasing α (and thus relying more on the wrong proxy for the benchmark) hurts performance. Interesting to see is that compared to uniform sampling, PER speeds up the learning for low values of α, while it hurts for large values of α. This demonstrates again the need to choose both parts of experience selection with care.

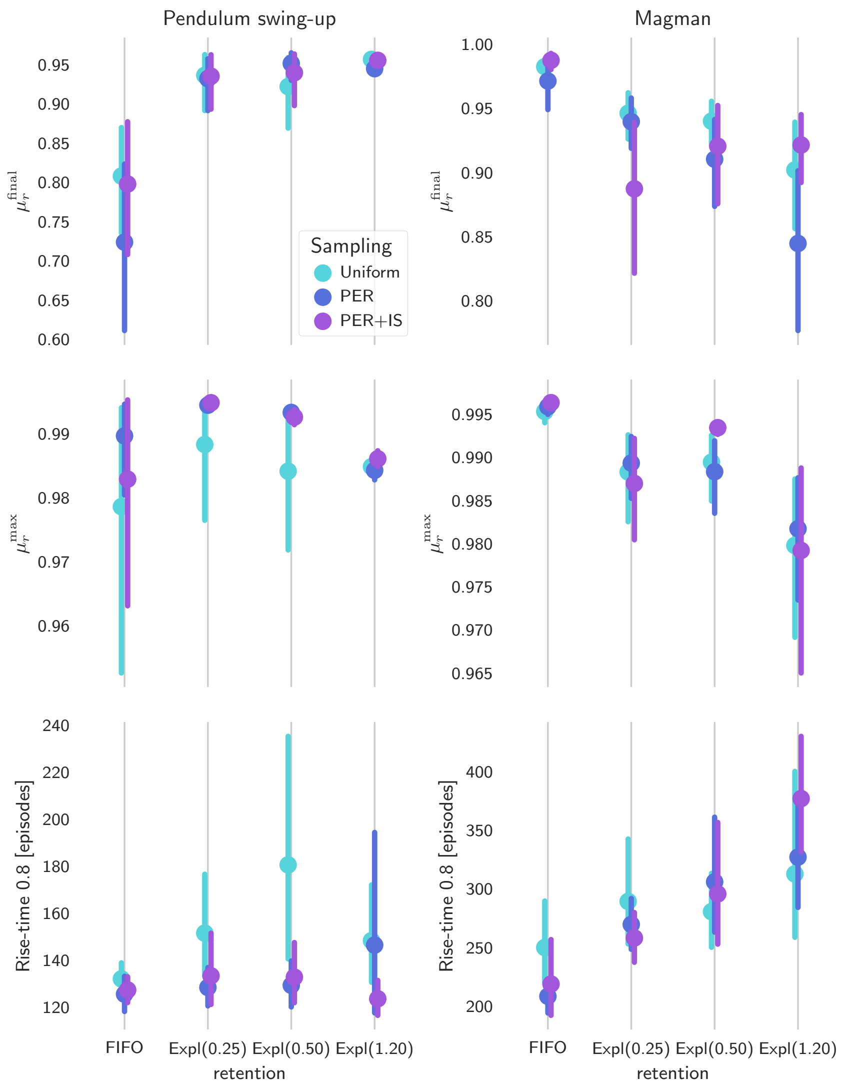


Figure 20: Influence of α in the Expl algorithm for different sampling strategies.

In Figure 21 it can again be seen that the benefits of PER are mostly to the speed of learning. Improvements to the maximum and final performance are possible when α is chosen correctly, but depend mostly on the contents of the buffer that PER is sampling from.

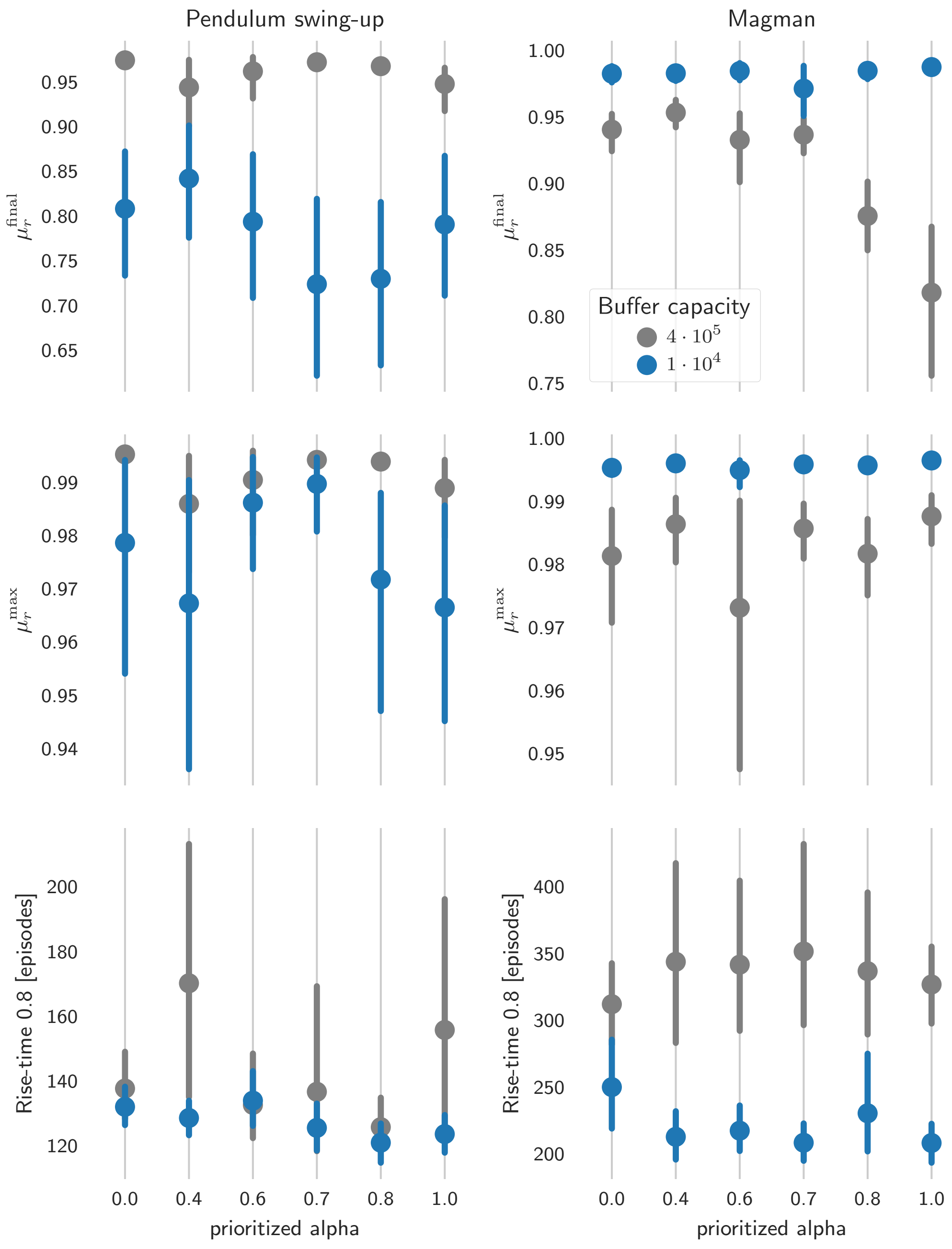


Figure 21: Influence of α in the PER algorithm for the Full DB strategy (buffer capacity $= 4 \times 10^5$) and FIFO retention (buffer capacity 1×10^{-4}).

D.3 Additional Figures Related to the Main Body

This subsection contains several figures that were left out of the main text of this work for brevity. They show the same experiments as Figures 6, 8, 16, according to the remaining performance criteria.

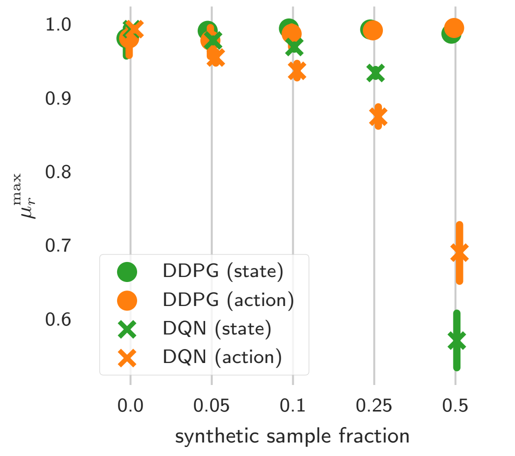


Figure 22: RL algorithm dependent effect of adding synthetic experiences to the FIFO[Uniform] method on the maximum performance per episode $\mu_r^{\max}$ on the pendulum swing-up benchmark. The effect on the final performance and the rise-time is given in Figure 7.

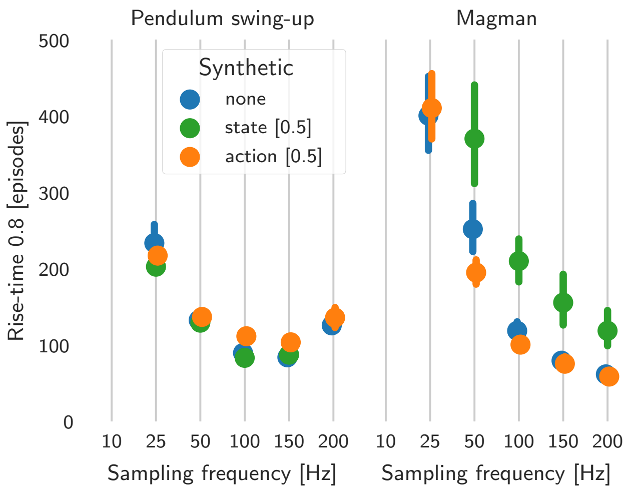
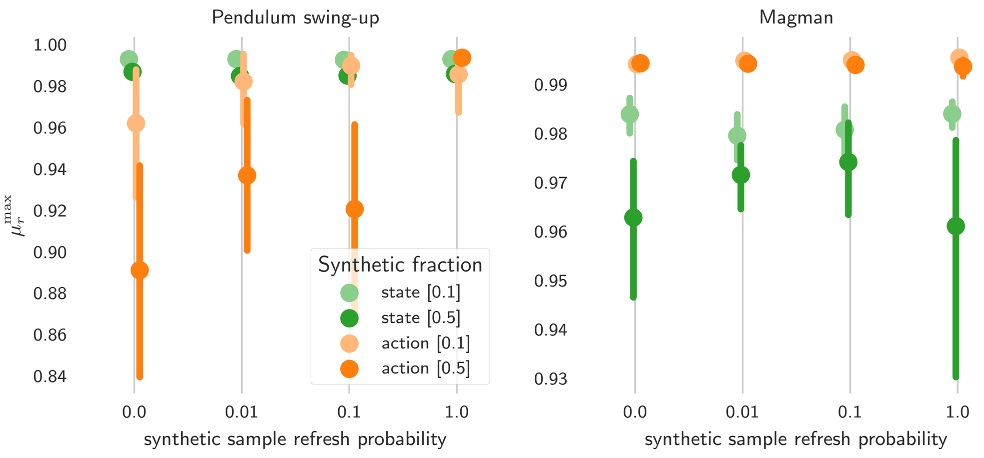
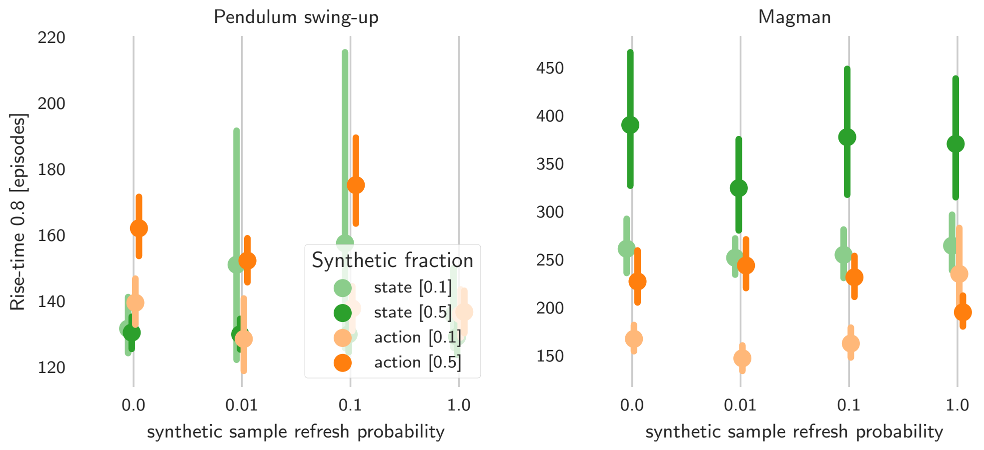


Figure 23: Sampling frequency dependent effect on the learning speed of adding synthetic experiences to the FIFO[Uniform] method. The effect on the final and maximum performance is given in Figure 9.



(a) Effect on $\mu_r^{\max}$.



(b) Effect on Rise − time0.8.

Figure 24: The effects on the performance of the FIFO[Uniform] method when changing a fraction of the observed experiences with synthetic experiences, when the synthetic experiences are updated only with a certain probability each time they are overwritten. The effects on μ_r^{final} is shown in Figure 8.

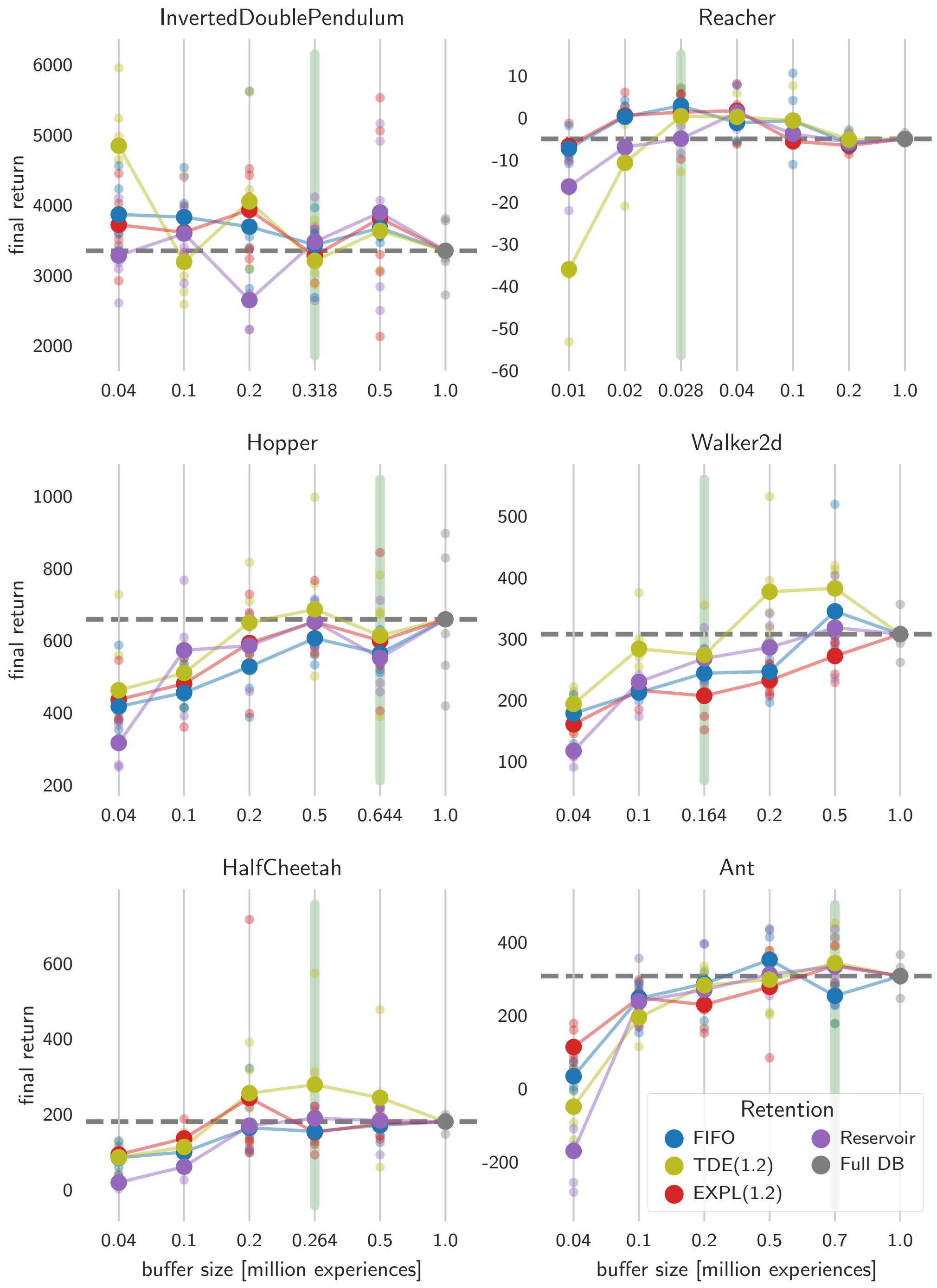


Figure 25: Mean performance during the last 2×10^5 training steps of a 1×10^6 step training run on the Roboschool benchmarks as a function of the retention strategy and buffer size. Results for the individual runs and their means are shown.

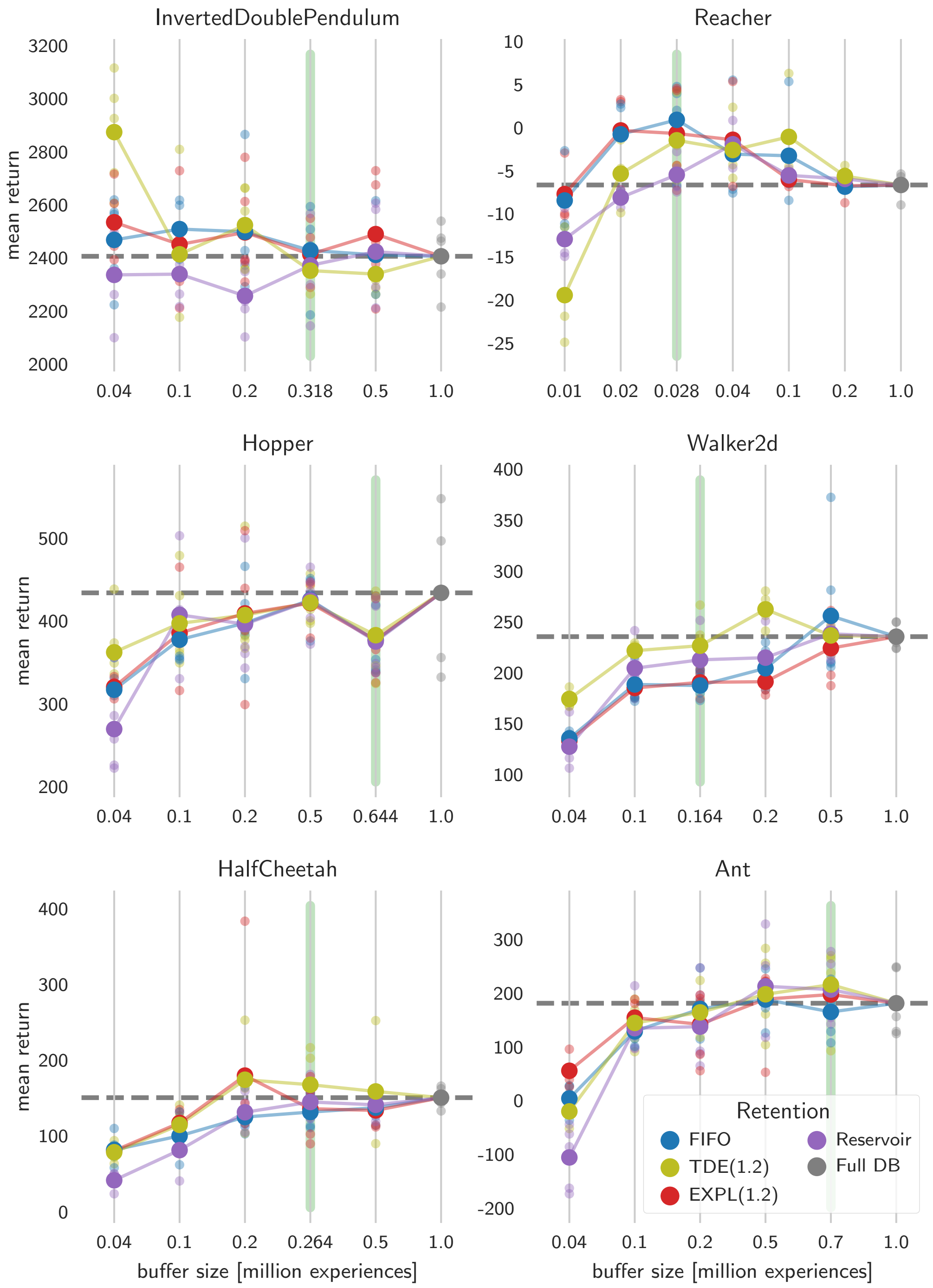


Figure 26: Mean performance during the whole training run on the Roboschool benchmarks as a function of the retention strategy and buffer size. Results for the individual runs and their means are shown.

References

Eduard Alibekov, Jiri Kubalík, and Robert Babuška. Policy derivation methods for critic-only reinforcement learning in continuous action spaces. *Engineering Applications of Artificial Intelligence*, 69:178–187, 2018.

David Andre, Nir Friedman, and Ronald Parr. Generalized prioritized sweeping. In *Advances In Neural Information Processing Systems (NIPS)*, pages 1001–1007. MIT Press, 1997.

John Aslanides, Jan Leike, and Marcus Hutter. Universal reinforcement learning algorithms: Survey and experiments. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1403–1410, 2017.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016.

Leemon C Baird. Reinforcement learning in continuous time: Advantage updating. In *World Congress on Computational Intelligence (WCCI)*, volume 4, pages 2448–2453, 1994.

Bikramjit Banerjee and Jing Peng. Performance bounded reinforcement learning in strategic interactions. In *AAAI National Conference on Artificial Intelligence (AAAI)*, volume 4, pages 2–7, 2004.

Samuel Barrett, Matt Taylor, and Peter Stone. Transfer learning for reinforcement learning on a physical robot. Adaptive Learning Agents Workshop, International Conference on Autonomous Agents and Multiagent Systems (AAMAS - ALA), 2010.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1471–1479, 2016.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *International Conference on Machine Learning (ICML)*, pages 41–48, 2009.

Lucian Buşoniu, Damien Ernst, Robert Babuška, and Bart De Schutter. Approximate dynamic programming with a fuzzy parameterization. *Automatica*, 46(5):804–814, 2010.

Wouter Caarls and Erik Schuitema. Parallel online temporal difference learning for motor control. *IEEE Transactions on Neural Networks and Learning Systems*, 27(7):1457–1468, 2016.

Nuttapong Chentanez, Andrew G Barto, and Satinder P Singh. Intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1281–1288, 2004.

Kamil Ciosek and Shimon Whiteson. OFFER: off-environment reinforcement learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2017.

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. Torch7: A Matlab-like environment for machine learning. BigLearn Workshop, Advances in Neural Information Processing Systems (NIPS - BLWS), 2011.

Tim de Bruin, Jens Kober, Karl Tuyls, and Robert Babuška. The importance of experience replay database composition in deep reinforcement learning. Deep Reinforcement Learning Workshop, Advances in Neural Information Processing Systems (NIPS - DRLWS), 2015.

Tim de Bruin, Jens Kober, Karl Tuyls, and Robert Babuška. Improved deep reinforcement learning for robotics through distribution-based experience retention. In *International Conference on Intelligent Robots and Systems (IROS)*, 2016a.

Tim de Bruin, Jens Kober, Karl Tuyls, and Robert Babuška. Off policy experience retention for deep actor critic learning. Deep Reinforcement Learning Workshop, Advances in Neural Information Processing Systems (NIPS - DRLWS), 2016b.

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. OpenAI Baselines. `https://github.com/openai/baselines`, 2017.

Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 15, pages 3460–3468, 2015.

Bradley Efron. Bootstrap methods: Another look at the jackknife. In *Breakthroughs in Statistics*, pages 569–593. Springer, 1992.

Vincent François-Lavet, Raphael Fonteneau, and Damien Ernst. How to discount deep reinforcement learning: Towards new dynamic strategies. arXiv preprint arXiv:1512.02011, 2015.

Gene F Franklin, David J Powell, and Michael L Workman. *Digital Control of Dynamic Systems*, volume 3. Addison-Wesley Menlo Park, 1998.

Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(771-780):1612, 1999.

Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2672–2680, 2014.

Ian J Goodfellow, Mehdi Mirza, Xiao Da, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgeting in gradient-based neural networks. arXiv preprint arXiv:1312.6211, 2013.

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep Q-learning with model-based acceleration. arXiv preprint arXiv:1603.00748, 2016.

Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E Turner, and Sergey Levine. Q-prop: sample-efficient policy gradient with an off-policy critic. In *International Conference on Learning Representations (ICLR)*, 2017.

Geoffrey E Hinton. To recognize shapes, first learn to generate images. *Progress in Brain Research*, 165:535–547, 2007.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. VIME: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1109–1117, 2016.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference for Learning Representations (ICLR)*, 2015.

Oleg Klimov. OpenAI Roboschool. `https://github.com/openai/roboschool`, 2017.

Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: a survey. *International Journal of Robotics Research (IJRR)*, 32(11):1238–1274, 2013.

Ivan Koryakovskiy, Heike Vallery, Robert Babuška, and Wouter Caarls. Evaluation of physical damage associated with action selection strategies in reinforcement learning. In *IFAC World Congress*, 2017.

Leonid Kuvayev and Richard S Sutton. Model-based reinforcement learning with an approximate, learned model. Yale Workshop on Adaptive Learning Systems, 1996.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

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 *International Conference on Learning Representations (ICLR)*, 2016.

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

Zachary C Lipton, Jianfeng Gao, Lihong Li, Xiujun Li, Faisal Ahmed, and Li Deng. Efficient exploration for dialogue policy learning with BBQ networks & replay buffer spiking. arXiv preprint arXiv:1608.05081, 2016.

Ilya Loshchilov and Frank Hutter. Online batch selection for faster training of neural networks. arXiv preprint arXiv:1511.06343, 2015.

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

Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller, editors. *Neural Networks: Tricks of the Trade*. Lecture Notes in Computer Science (LNCS). Springer, 2nd edition, 2012.

Guido F Montufar, Razvan Pascanu, Kyunghyun Cho, and Yoshua Bengio. On the number of linear regions of deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2924–2932, 2014.

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

Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. Language understanding for text-based games using deep reinforcement learning. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

Deanna Needell, Nathan Srebro, and Rachel Ward. Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. *Mathematical Programming*, 155 (1-2):549–573, 2016.

Brendan O'Donoghue, Remi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. Combining policy gradient and Q-learning. In *International Conference on Learning Representations (ICLR)*, 2017.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped DQN. In *Advances In Neural Information Processing Systems (NIPS)*, pages 4026–4034, 2016.

Mathijs Pieters and Marco A Wiering. Q-learning with experience replay in a dynamic environment. In *Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2016.

Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y. Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. In *International Conference on Learning Representations (ICLR)*, 2018.

Aravind Rajeswaran, Kendall Lowrey, Emanuel V Todorov, and Sham M Kakade. Towards generalization and simplicity in continuous control. In *Advances In Neural Information Processing Systems (NIPS)*, pages 6550–6561, 2017.

Andrei A Rusu, Matej Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. arXiv preprint arXiv:1610.04286, 2016.

Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences*, 3(6):233–242, 1999.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations (ICLR)*, 2016.

Jürgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *From Animals to Animats: International Conference on Simulation of Adaptive Behavior (SAB)*, 1991.

Young-Woo Seo and Byoung-Tak Zhang. Learning user's preferences by analyzing web-browsing behaviors. In *International Conference on Autonomous Agents (ICAA)*, pages 381–387, 2000.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International Conference on Machine Learning (ICML)*, 2014.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

Burrhus F Skinner. Reinforcement today. *American Psychologist*, 13(3):94, 1958.

Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991.

Aviv Tamar, Yinlam Chow, Mohammad Ghavamzadeh, and Shie Mannor. Sequential decision making with coherent risk. *IEEE Transactions on Automatic Control*, 2016.

George E Uhlenbeck and Leonard S Ornstein. On the theory of the Brownian motion. *Physical Review*, 36(5):823, 1930.

Leslie G Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

Jeffrey S Vitter. Random sampling with a reservoir. *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. In *International Conference on Learning Representations (ICLR)*, 2017.