Action Noise in Off-Policy Deep Reinforcement Learning: Impact on Exploration and Performance

Anonymous authors
Paper under double-blind review

Abstract

Many Deep Reinforcement Learning (D-RL) algorithms rely on simple forms of exploration such as the additive action noise often used in continuous control domains. Typically, the scaling factor of this action noise is chosen as a hyper-parameter and is kept constant during training. In this paper, we focus on action noise in off-policy deep reinforcement learning for continuous control. We analyze how the learned policy is impacted by the noise type, noise scale, and impact scaling factor reduction schedule. We consider the two most prominent types of action noise, Gaussian and Ornstein-Uhlenbeck noise, and perform a vast experimental campaign by systematically varying the noise type and scale parameter, and by measuring variables of interest like the expected return of the policy and the state-space coverage during exploration. For the latter, we propose a novel state-space coverage measure $X_{\mathcal{U}_{\text{rel}}}$ that is more robust to boundary artifacts than previously-proposed measures. Larger noise scales generally increase state-space coverage. However, we found that increasing the space coverage using a larger noise scale is often not beneficial. On the contrary, reducing the noise scale over the training process reduces the variance and generally improves the learning performance. We conclude that the best noise type and scale are environment dependent, and based on our observations derive heuristic rules for guiding the choice of the action noise as a starting point for further optimization. https://github.com/[anonymized]

1 Introduction

In (deep) reinforcement learning an agent aims to learn a policy to act optimally based on data it collects by interacting with the environment. In order to learn a well performing policy, data (i.e. state-action-reward sequences) of sufficiently good behavior need to be collected. A simple and very common method to discover better data is to induce variation in the data collection by adding noise to the action selection process. Through this variation, the agent will try a wide range of action sequences and eventually discover useful information.

Action Noise In off-policy reinforcement learning algorithms applied to continuous control domains, a go-to approach is to add a randomly-sampled *action noise* to the action chosen by the policy. Typically the action noise is sampled from a Gaussian distribution or an Ornstein-Uhlenbeck process, either because algorithms are proposed using these noise types (Fujimoto et al., 2018; Lillicrap et al., 2016), or because these two types are provided by reinforcement learning implementations (Liang et al., 2018; Raffin et al., 2021a; Fujita et al., 2021; Seno and Imai, 2021). While adding action noise is simple, widely used, and surprisingly effective, the impact of action noise type or scale does not feature very prominently in the reinforcement learning literature. However, the action noise can have a huge impact on the learning performance as the following example shows.

A motivating example: Consider the case of the *Mountain-Car* (Moore, 1990) environment. In this task, a car starts in a valley between mountains on the left and right and does not have sufficient power to simply drive up the mountain. It needs repetitive swings to increase its potential and kinetic energy to finally make it up to the top of the mountain on the right side. The actions apply a force to the car and incur a cost that is

A motivating example: Mountain-Car

noise Type	Gaussian	Ornstein-Uhlenbeck
Scale	0.6	0.5
Return	-30.2	-30.4
+-	0.1	1.3

Table 1: Untrained random policies, Gaussian ($\sigma = 0.6$) and Ornstein-Uhlenbeck ($\sigma = 0.5$) achieve similar returns and appear interchangeable.

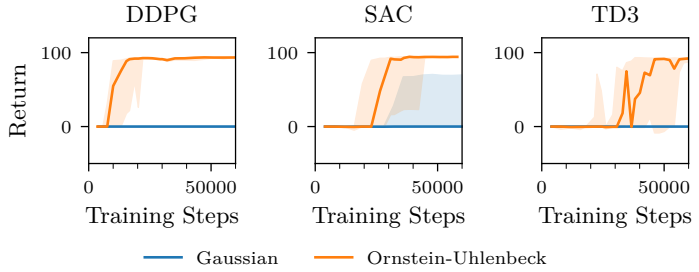


Figure 1: Training with the action noises (Table 1) shows the impact of noise type; Ornstein-Uhlenbeck solves the task, but Gaussian does not. Other algorithm parameters are taken from the tuned parameters found by Raffin (2020). The lines indicate the medians, the shaded areas the quartiles of ten independent runs.

quadratic to the amount of force, while reaching the goal yields a final reward of 100. This parameterization implies a local optimum: not performing any action and achieving a return of zero.

Driving the environment with purely random policies based on the two noise types (Gaussian, $\sigma = 0.6$, Ornstein-Uhlenbeck $\sigma = 0.5$, see Table 1), yields similar returns. However, when we apply the algorithms DDPG, TD3 and SAC (Lillicrap et al., 2016; Fujimoto et al., 2018; Haarnoja et al., 2019) to this task, the resulting learning curves (Figure 1) very clearly depict the huge impact the noise configuration has. While returns of the purely random noise-only policies were similar, we achieve substantially different learning results. Learning either fails (Gaussian) or leads to success (Ornstein-Uhlenbeck). This shows the huge importance of the action noise configuration. See Section A for further details.

Relation to ε-greedy

A very common strategy in Q-learning algorithms applied to discrete control is to select a random action with a certain probability ε. In this *epsilon-greedy* strategy, the probability ε is often chosen higher in the beginning of the training process and reduced to a smaller value over course of the training. Although very common in Q-learning, a comparable strategy has not received a lot of attention for action noise in continuous control. The most prominent algorithms using action noise, namely DDPG (Lillicrap et al., 2016) and TD3 (Fujimoto et al., 2018), do not mention changing the noise over the training process. Another prominent algorithm, SAC (Haarnoja et al., 2019), adapts the noise to an entropy target. The entropy target, however, is kept constant over the training process. In many cases the optimal policy would be deterministic, but the agent has to behave with similar average action-entropy no matter whether the optimal policy has been found or not.

Another indication that reducing the randomness over the training process has received little attention is that only very few reinforcement learning implementations, e.g., RLlib (Liang et al., 2018), implement reducing the impact of action noise over the training progress. Some libraries, like `coach` (Caspi et al., 2017), only implement a form of continuous epsilon greedy: sampling the action noise from a uniform distribution with probability ε. The majority of available implementations, including `stable-baselines` (Raffin et al., 2021a), `PFRL` (Fujita et al., 2021), `acme` (Hoffman et al., 2020), and `d3rlpy` (Seno and Imai, 2021), do not implement any strategies to reduce the impact of action noise over the training progress.

Contributions

In this paper we analyze the impact of Gaussian and Ornstein-Uhlenbeck noise on the learning process of DDPG, TD3, SAC and a deterministic SAC variant. Evaluation is performed on multiple popular environments (Table C.1): Mountain-Car (Brockman et al., 2016) environment from the OpenAI Gym, Inverted-Pendulum-Swingup, Reacher, Hopper, Walker2D and Half-Cheetah environments implemented using PyBullet (Coumans and Bai, 2016; Ellenberger, 2018).

- We investigate the relation between exploratory state-space coverage X, returns collected by the exploratory policy R and learned policy performance P.
- We propose to assess the state-space coverage using our novel measure $\mathrm{X}_{\mathcal{U}_{\mathrm{rel}}}$ that is more robust to approximation artifacts on bounded spaces compared to previously proposed measures.
- We perform a vast experimental study and investigate the question whether one of the two noise types is generally preferable *(Q1)*, whether a specific scale should be used *(Q2)*, whether there is any benefit to reducing the scale over the training progress (linearly, logistically) compared to keeping it constant *(Q3)*, and which of the parameters noise type, noise scale and scheduler is most important *(Q4)*.
- We provide a set of heuristics derived from our results to guide the selection of initial action noise configurations.

Findings We found that the noise configuration, noise type and noise scale, have an important impact and can be necessary for learning (e.g. Mountain-Car) or can break learning (e.g. Hopper). Larger noise scales tend to increase state-space coverage, but for the majority of our investigated environments increasing the state-space coverage is not beneficial. *We recommend to select and tune action noise based on the reward and dynamics structure on a per-environment basis.*

We found that across noise configurations, decaying the impact of action noise tends to work better than keeping the impact constant, in both reducing the variance across seeds and improving the learned policy performance and can thus make the algorithms more robust to the action noise hyper-parameters scale and type. *We recommend to reduce the action noise scaling factor over the training time.*

We found that for all environments investigated in this study noise scale σ is the most important parameter, and some environments (e.g. Mountain-Car) benefit from larger noise scales, while other environments require very small scales (e.g. Walker2D). *We recommend to assess an environment's action noise scale preference first.*

2 Related Work

By combining Deep Learning with Reinforcement Learning in their DQN method, Mnih et al. (2015) achieved substantial improvements on the Atari Games RL benchmarks (Bellemare et al., 2013) and sparked lasting interest in *Deep Reinforcement learning* (D-RL).

Robotic environments: In robotics, the interest in Deep Reinforcement Learning has also been rising and common benchmarks are provided by the OpenAI Gym (Brockman et al., 2016), which includes control classics such as the Mountain-Car environment (Moore, 1990) as well as more complicated (robotics) tasks based on the Mujoco simulator (Todorov et al., 2012). Another common benchmark is the DM Control Suite (Tassa et al., 2018), also based on Mujoco. While Mujoco has seen widespread adoption it was, until recently, not freely available. A second popular simulation engine, that has been freely available, is the Bullet simulation engine (Coumans and Bai, 2016) and very similar benchmark environments are also available for the Bullet engine (Coumans and Bai, 2016; Ellenberger, 2018).

Continuous Control: While the Atari games feature large and (approximately) continuous observation spaces, their action spaces are discrete and relatively small, making Q-learning a viable option. In contrast, typical robotics tasks require *continuous action spaces*, implying uncountably many different actions.

A tabular Q-learning approach or a discrete Q-function output for each action are therefore not possible and maximizing the action over a learned function approximator for $Q(s, a)$ is computationally expensive (although not impossible as Kalashnikov et al. (2018) have shown). Therefore, in continuous action spaces, *policy search* is employed, to directly optimize a function approximator *policy*, mapping from state to best performing action (Williams, 1992). To still reap the benefits of reduced sample complexity of TD-methods, policy search is often combined with learning a value function, a *critic*, leading to an *actor-critic* approach (Sutton et al., 1999).

On- and Off-policy: Current state of the art D-RL algorithms consist of *on-policy* methods, such as TRPO (Schulman et al., 2015) or PPO (Schulman et al., 2017), and *off-policy* methods, such as DDPG (Lillicrap et al., 2016), TD3 (Fujimoto et al., 2018) and SAC (Haarnoja et al., 2019). While the on-policy methods optimize the next iteration of the policy with respect to the data collected by the current iteration, off-policy methods are, apart from stability issues and requirements on the samples, able to improve policy performance based on data collected by *any arbitrary* policy and thus can also re-use older samples.

To improve the policy, variation (*exploration*) in the collected data is necessary. The most common form of exploration is based on randomness: in on-policy methods this comes from a *stochastic policy* (TRPO, PPO), while in the off-policy case it is possible to use a stochastic policy (SAC) or, to use a *deterministic policy* (Silver et al., 2014) with added *action noise* (DDPG, TD3). Since off-policy algorithms can learn from data collected by other policies, it is also possible to combine stochastic policies (e.g. SAC) with action noise.

State-Space Coverage: Often, the reward is associated with reaching certain areas in the state-space. Thus, in many cases, *exploration* is related to *state-space coverage*. An intuitive method to calculate state space coverage is based on binning the state-space and counting the percentage of non-empty bins. Since this requires exponentially many points as the dimensionality increases, other measures are necessary. Zhan et al. (2019) propose to measure state coverage by drawing a bounding box around the collected data and measuring the means of the side-lengths, or by measuring the sum of the eigenvalues of the estimated covariance matrix of the collected data. However, so far, there is no common and widely adopted approach.

Methods of Exploration: The architecture for the stochastic policy in SAC (Haarnoja et al., 2019) consists of a neural network parameterizing a Gaussian distribution, which is used to sample actions and estimate action-likelihoods. A similar stochastic policy architecture is also used in TRPO (Schulman et al., 2015) and PPO (Schulman et al., 2017). While this is the most commonly used type of distribution, more complicated parameterized stochastic policy distributions based on normalizing flows have been proposed (Mazoure et al., 2020; Ward et al., 2019). In case of action noise, the noise processes are not limited to uncorrelated Gaussian (e.g. TD3) and temporally correlated Ornstein-Uhlenbeck noise (e.g. DDPG): a whole family of action noise types is available under the name of colored noise, which has been successfully used to improve the Cross-Entropy-Method (Pinneri et al., 2020). A quite different type of random exploration are the parameter space exploration methods (Mania et al., 2018; Plappert et al., 2017), where noise is not applied to the resulting action, but instead, the parameters of the policy are varied. As a somewhat intermediate method, state dependent exploration (Raffin et al., 2021b) has been proposed, where action noise is deterministically generated by a function based on the state. Here, the function parameters are changed randomly for each episode, leading to different deterministic "action noise" for each episode. Presumably among the most intricate methods to generate exploration are the methods that train a policy to achieve exploratory behavior by rewarding exploratory actions (Burda et al., 2019; Tang et al., 2017; Mutti et al., 2020; Hong et al., 2018; Pong et al., 2020).

It is however, not clear yet, which exploration method is most beneficial, and when a more complicated method is actually worth the additional computational cost and complexity. In this work we aim to reduce this gap, by investigating the most widely used baseline method in more detail: exploration by action noise.

3 Methods

In this section, we describe the action noise types, the schedulers to reduce the scaling factor of the action noise over time and the evaluation process in more detail. We briefly list the analyzed benchmark environments and their most important properties. We chose environments of increasing complexity that model widely used benchmark tasks. We list the used algorithms and then describe how we gather evaluation data and how it is aggregated. Last, we describe the methods we use for analyzing state-space coverage.

3.1 Noise types: Gaussian and Ornstein-Uhlenbeck

The action noise ε_{a_t} is added to the action drawn from the policy:

$$a_t = \underset{a_{\min}, a_{\max}}{\text{clip}} \left[\tilde{a}_t \quad + \quad \beta \left(\underset{-1;1}{\text{clip}}[\varepsilon_{a_t}] \cdot \frac{a_{\max} - a_{\min}}{2} + \frac{a_{\max} + a_{\min}}{2} \right) \right] \tag{1}$$

where $\tilde{a}_t \sim \pi_\theta(\cdot|s_t)$ for stochastic policies or $\tilde{a}_t = \pi_\theta(s_t)$ for deterministic policies. We introduce an additional impact scaling factor β, which is typically kept constant at the value one. In Section 3.2 we describe how we change β over time to create a noise scheduler. The action noise ε_{a_t} is drawn from either a Gaussian distribution or an Ornstein-Uhlenbeck (OU) process. The noise distributions are factorized, i.e. noise samples are drawn independently for each action dimension. For the generation of action noise samples, the action space is assumed to be limited to $[-1, 1]$ but then rescaled to the actual limits defined by the environment.

Gaussian noise is temporally uncorrelated and is typically applied on symmetric action spaces (Hill et al., 2018; Raffin, 2020) with commonly used values of $\mu = 0$ and $\sigma = 0.1$ with $\Sigma = \boldsymbol{I} \cdot \sigma$. Action noise is sampled according to

$$\varepsilon_{a_t} \sim \mathcal{N}(\mu, \Sigma) \tag{2}$$

Ornstein-Uhlenbeck noise is sampled from the following temporal process, with each action dimension calculated independently of the other dimensions:

$$\varepsilon_{a_t} = \varepsilon_{a_{t-1}} + \theta(\mu - \varepsilon_{a_{t-1}}) \cdot \mathrm{d}t + \sigma\sqrt{(}\mathrm{d}t) \cdot \epsilon_t \tag{3}$$

$$\varepsilon_{a_0} = \mathbf{0} \qquad \epsilon_t \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{4}$$

The parameters we use for Ornstein-Uhlenbeck noise are taken from a widely used RL-algorithm implementation (Hill et al., 2018): $\theta = 0.15$, $\mathrm{d}t = 0.01$, $\mu = \mathbf{0}$, $\sigma = 0.1 \cdot \boldsymbol{I}$.

Due to the huge number of possible combinations of environments, algorithms, noise type, noise scale and the necessary repetition with different seeds, we had to limit the number of investigated scales. We set out with two noise scales σ encountered in pre-tuned hyper-parameterization (Raffin, 2020), $0.1, 0.5$, and continued with a linear increase, $0.9, 1.3, 1.7$. Much smaller noise scales vanish in the variations induced by learning and much larger scales lead to Bernoulli trials of the min-max actions without much difference.

Because the action noise is clipped to $[-1, 1]$ before being scaled to the actual action limits, a very large scale, such as 1.7, implies a larger percentage of on-the-boundary action noise samples and is thus more similar to bang-bang control actions. This is interesting because bang-bang control has been found surprisingly effective in many RL benchmarks (Seyde et al., 2021).

3.2 Scheduling strategies to reduce action noise

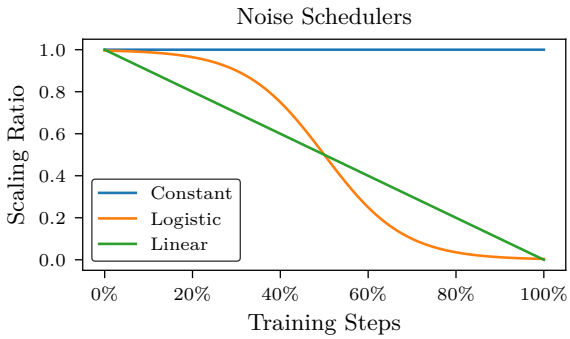


Figure 2: Action noise is used for exploration. The agent should favor exploration in the beginning but later favor exploitation. Similar to ϵ-greedy strategies in discrete-action Q-learning, the logistic and linear schedulers reduce the impact of noise (scaling ratio, β in (1)) over the course of the training progress.

In (1) we introduce the action noise scaling-ratio β. In this work we compare a constant-, linear- and logistic-scheduler for the value of β. The effective scaling of the action noise by the noise schedulers is illustrated in Figure 2. The noise types are described in more detail in Section 3.1.

Changing the σ (see (3) and (2)) instead of β could result in a different shape of the distribution, for example when values are clipped, or when the σ indirectly affects the result as in the Ornstein-Uhlenbeck process. To keep the action noise distribution shape constant, the action noise schedulers do not change the σ parameter of the noise process but instead scale down the resulting sampled action noise values by changing the β parameter: this means that the effective range of the action noise, before scaling and adjusting to the environment limits, changes over time from $[-1, 1]$, the maximum range, to 0 for the linear and logistic schedulers.

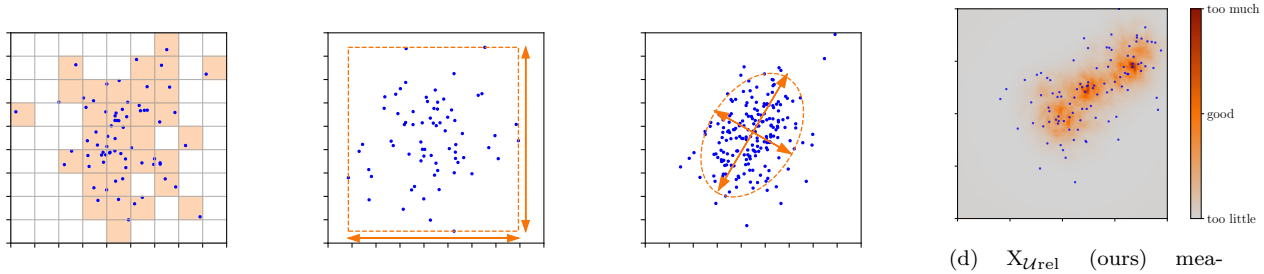
3.3 Environments

For evaluation we use various environments of increasing complexity: Mountain-Car, Inverted-Pendulum-Swingup, Reacher, Hopper, Walker2D, Half-Cheetah. Observation dimensions range from 2 to 26, and action dimensions range from 1 to 6. See Table C.1 for details, including a rough sketch of the reward. The table indicates whether the reward is sparse or dense with respect to a goal state, goal region, or a change of the distance to the goal region. Many environments feature linear or quadratic (energy) penalties on the actions (e.g. Hopper). Penalties on the state can be sparse (such as joint limits), or dense (such as force or required power induced by joint states). See Brockman et al. (2016), Coumans and Bai (2016), and Ellenberger (2018) for further details.

3.4 Performed experiments

We evaluate the effects of action noise on the popular and widely-used algorithms: TD3 (Fujimoto et al., 2018), DDPG (Lillicrap et al., 2016), SAC (Haarnoja et al., 2019), and a deterministic version of SAC (DetSAC, Algorithm B.1). Originally SAC was proposed with only exploration from its stochastic policy. However, since SAC is an off-policy algorithm, it is possible to add additional action noise, a common solution for environments such as the Mountain Car. The stochastic policy in SAC typically is a parameterized Gaussian and combining the action noise with the stochasticity of actions sampled from the stochastic policy could impact the results. Thus, we also compare to our DetSAC version, where action noise is added to the mean action of the DetSAC policy (Algorithm B.1).

We use the implementations provided by Raffin et al. (2021a), following the hyper-parameterizations provided by Raffin (2020), but adapting the action noise settings.



(a) X_{bin} divides the state-space into bins and measures the ratio of non-empty bins.

(b) X_{BBM} measures the spread by the mean of the side-lengths of the bounding box.

(c) X_{NN} measures the spread of the data by the sum of the eigenvalues of the covariance of the data.

(d) $X_{\mathcal{U}\text{rel}}$ (ours) measures the symmetric KL-divergence between a prior over the state space and the collected state-space data.

Figure 3: Illustrations of the state-space coverage measures. $X_{\mathcal{U}\text{rel}}$ scales to high dimensions (unlike X_{bin}) and is not susceptible to boundary artifacts (unlike X_{BBM} and X_{NN}).

The experiments consist of testing 6 environments, 4 algorithms, 5 noise scales, 3 schedulers and 2 noise types. Each experiment is repeated with 20 different seeds, amounting to 14400 experiments in total. On a single node, `AMD Ryzen 2950X` equipped with four `GeForce RTX 2070 SUPER, 8 GB`, running about twenty experiments in parallel this would amount to a runtime of approximately 244 node-days.

Table G.1 lists the number of independent runs performed for each experimental configuration.

3.5 Measuring Performance

For each experiment (i.e. single seed), we divide the learning process into 100 segments and evaluate the exploration and learned policy performance once for each of those segments. At the end of each segment, we perform evaluation rollouts for 100 episodes or 10000 steps, whichever is reached first, using only complete episodes. This is done for both the deterministic *exploitation* policy as well as the *exploratory* (action noise) policy. The two resulting datasets of evaluation rollouts are used to calculate state-space coverages and returns. These evaluation rollouts, both exploring and exploiting, are *not* used for training. We take the mean over these 100 measurements to aggregate them into a single value. This is equivalent to measuring the area under the learning curves.

The learning algorithm uses a noisy (exploratory) policy to collect data and exploratory return and state-space coverage could be assessed based on the replay buffer data. However, to get statistically more robust estimates of the quality of the exploratory policy (returns and state-space coverage), we perform the above mentioned exploratory evaluation rollouts and use these rollouts for assessing state-space coverage and exploratory returns instead of the data in the replay buffer.

3.6 State-Space Coverage

We assess exploration in terms of state-space coverage. We assume that the environment states $s \in \mathcal{R}^d$ have finite upper and lower limits: $\text{low} \leq s \leq \text{high}$, $\text{low}, \text{high} \in \mathcal{R}^d$. We investigate four measures: $X_{\text{bin}}, X_{\mathcal{U}\text{rel}}, X_{\text{BBM}}, X_{\text{NN}}$, which are illustrated in Figure 3.

The most intuitive measure for state-space coverage is a histogram-based approach X_{bin}, which divides the state space into equally many bins along each dimension and measures the ratio of non-empty bins to the total number of bins:

$$X_{\text{bin}} = \frac{\#\text{ of non-empty bins}}{\#\text{ number of bins}} \tag{5}$$

The number of bins, as the product of divisions along each dimension, grows exponentially with the dimensionality. This means that either the number of bins has to be chosen very low, or, if there are more bins than data points, the ratio has to be adjusted. We chose to limit the number of bins. For a sample of size m

and dimensionality d the divisions k along each dimension are chosen to allow for at least c points per bin

$$k = \left\lfloor \left(\frac{m}{c}\right)^{\frac{1}{d}} \right\rfloor \tag{6}$$

However, for high-dimensional data, the number of bins becomes very small and the measure easily reaches 100% and becomes meaningless, or, the required number of data points becomes prohibitively large very quickly. Thus, alternatives are necessary.

Zhan et al. (2019) proposed two state-space coverage measures that also work well in high-dimensional spaces: the *bounding box mean* X_{BBM}, and the *nuclear norm* X_{NN}. X_{BBM} measures the spread of the data by a d dimensional bounding box around the collected data $D = \{\ldots, \boldsymbol{s}^{(j)}, \ldots\}$ and measuring the mean of the side lengths of this bounding box:

$$X_{\text{BBM}} = \frac{1}{d} \sum_{i}^{d} \left[\max_j s_i^{(j)} - \min_j s_i^{(j)} \right] \tag{7}$$

X_{NN}, the nuclear norm estimates the covariance matrix C of the data and measures data spread by the trace, the sum of the eigenvalues of the estimated covariance:

$$X_{\text{NN}}(D) := \text{trace}\left(C(D)\right) \tag{8}$$

As we will see, extreme values or values close to the state-space boundaries can lead to over-estimation of the state-space coverage by these two measures. We therefore propose a measure more closely related to X_{bin} but more suitable to higher dimensions: $\text{X}_{\mathcal{U}\text{rel}}(D)$. The Uniform-relative-entropy measure $\text{X}_{\mathcal{U}\text{rel}}$ assesses the uniformity of the collected data, by measuring the state-space coverage as the symmetric divergence between a uniform prior over the state space U and the data distribution Q_D:

$$\text{X}_{\mathcal{U}\text{rel}}(D) = -D_{\text{KL}}\big(U||Q_D\big) - D_{\text{KL}}\big(Q_D||U\big) \tag{9}$$

The inspiration for this measure comes from the observation that the exploration reward for count-based methods without task reward would be maximized by a uniform distribution. We assume that for robotics tasks reasonable bounds on the state space can be found. In a bounded state space, the uniform distribution is the least presumptive (maximum-entropy) distribution. The addition of the $D_{\text{KL}}\big(U||Q_D\big)$ term helps to reduce under-estimation of the divergence in areas with low density in Q_D. Note that Q_D is only available through estimation, and the support for Q_D is never zero as the density estimate never goes to zero. To estimate the relative uniform entropy we evaluated two divergence estimators, a kNN-based (k-Nearest-Neighbor) estimator and a Nearest-Neighbor-Ratio (NNR) estimator (Noshad et al., 2017). A kNN estimator is susceptible to over- / under-estimation artifacts close to the boundaries (support) of the state space. In contrast, the NNR estimator does not suffer from these artifacts. If not specified explicitly, $\text{X}_{\mathcal{U}\text{rel}}$ refers to the NNR-based variant.

$\text{X}_{\mathcal{U}\text{rel}}$ can be estimated using a *kNN density estimate* $\hat{q}_k(s)$, as described in (Bishop, 2006), where V_d denotes the unit volume of a d-dimensional sphere, $R_k(x)$ is the Euclidean distance to the k-th neighbor of x, and n is the total number of samples in $\mathcal{D}$:

$$V_d = \frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)} \tag{10}$$

$$\hat{q}_k(x) = \frac{k}{n} \frac{1}{V_d R_k(x)^d} = \frac{k}{nV_d} \frac{1}{R_k(x)^d} \tag{11}$$

where Γ denotes the gamma function.

Alternatively, $\text{X}_{\mathcal{U}\text{rel}}$ can be *estimated using NNR*, an f-divergence estimator, based on the ratio of the nearest neighbors around a query point.

For the general case of estimating $D_{\text{KL}}\big(P||Q\big)$, we take samples from $X \sim Q$ and $Y \sim P$. Let $\mathcal{R}_k(Y_i)$ denote the set of the k-nearest neighbors of Y_i in the set $Z := X \cup Y$. N_i is the number of points from $X \cap \mathcal{R}_k(Y_i)$,

M_i is the number of points from $Y \cap \mathcal{R}_k(Y_i)$, M is the number of points in Y and N is the number of points in X, $\eta = \frac{M}{N}$. The NNR estimator assumes lower C_L and upper bounds C_U on the densities P and Q. Assuming all points of a sample of size n are concentrated around a single point, we limit the density to $C_L = \frac{1}{n}$ respectively $C_U = \frac{n}{1}$.

$$D_{\mathrm{KL}}(P||Q) \approx \hat{D}_g(X, Y) \tag{12}$$

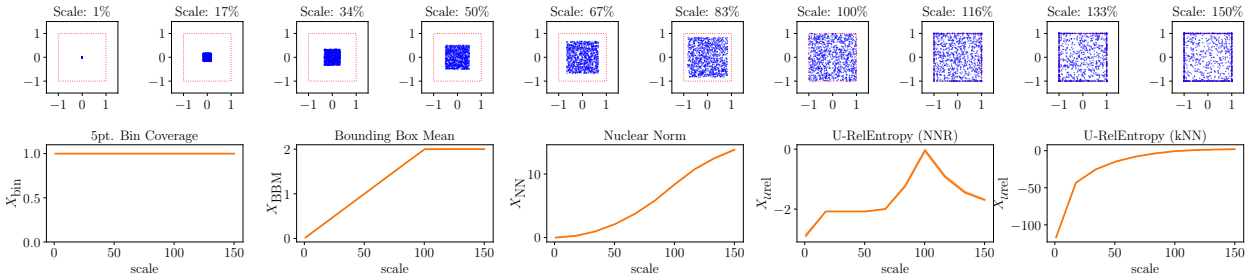
$$\hat{D}_g(X, Y) := \max\left(\frac{1}{M}\sum_{i=1}^{M} \hat{g}\left(\frac{\eta N_i}{M_i + 1}\right), 0\right) \tag{13}$$

$$\text{where } \hat{g}(x) := \max\left(g(x), g(C_L/C_U)\right) \tag{14}$$

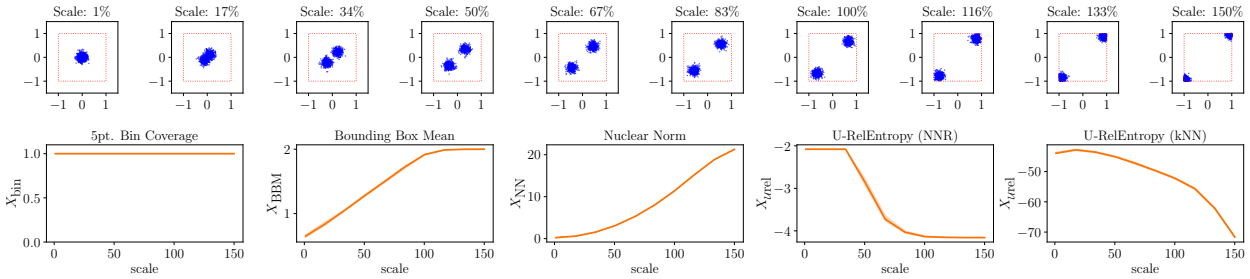
$$g(\rho) := -\log \rho \tag{15}$$

3.6.1 Evaluation of Measures on Synthetic Data

To compare the different exploration measures, we assumed a $d = 25$ dimensional state space, generated data from two different types of distributions, and compared the exploration measures on these data. The experiments were repeated 10 times, and the mean and min-max values are plotted in Figure 4. While the data are d-dimensional, they come from factorial distributions, similarly distributed along each dimension. Thus, we can gain intuition about the distribution from scatter plots of the first vs. second dimension. This is depicted at the top of each of the two parts. The bottom part of each comparison shows the different exploration measures, where the scale parameter is depicted on the x axis and the exploration measure on the y axis.



(a) Growing Uniform distribution: evaluation of the state-space coverage measures on synthetic data – for larger scale values more points are clipped to the state-space boundaries, leading to an expected decrease in state-space coverage for scales larger than 100%. This behavior is only captured by $\mathrm{X}_{\mathcal{U}\mathrm{rel}}$ (NNR).



(b) Growing Distance of Modes of 2-Mixture of Truncated Normal: evaluation of the state-space coverage measures on synthetic data. For larger scale values, the location of the mixture components is closer to the boundary – leading to an expected reduction in coverage for larger scale values. $X_{\mathrm{bin}}, X_{\mathrm{BBM}}, X_{\mathrm{NN}}$ fail to capture this behavior.

Figure 4: state-space coverage measures may not accurately represent the real coverage. Each comparison (a-b) shows the different exploration measures $X_{\mathrm{bin}}, X_{\mathrm{BBM}}, X_{\mathrm{NN}}$ and $\mathrm{X}_{\mathcal{U}\mathrm{rel}}$ (ours) on synthetic 25 dimensional data. X_{bin} becomes constant and X_{BBM} and X_{NN} suffer from boundary artifacts. The different data generating distributions depend on a *scale* parameter. The distributions are factorial and similarly distributed along each dimension. The scatter plots in (a-b) depict first vs. second dimension.

(a) Growing Uniform: Figure 4(a) depicts data generated by a uniform distribution, centered around the middle of the state space, with minimal and maximal values growing relatively to the full state space according to the *scale* parameter from 1% to 150%. Since in the latter case, many points would lie outside the allowed state space; these values are clipped to the state-space boundaries. This loosely corresponds to an undirectedly exploring agent that overshoots and hits the state-space limits, sliding along the state-space boundaries. Note how the estimation (kNN vs. NNR) has a great impact on the $\mathrm{X}_{\mathcal{U}\mathrm{rel}}$ measure's performance here: We would expect a maximum around a scale of 100% and smaller values before and after (due to clipping). Here the $\mathrm{X}_{\mathcal{U}\mathrm{rel}}$ (NNR) measure most closely follows this expectation. The ground-truth value of the divergence would follow a similar shape. However, since the densities are limited for the NNR estimator, the ground-truth divergence would show more extreme values.

(b) Bi-Modal Truncated Normal moving locations: Figure 4(b) shows a mixture of two truncated Gaussian distributions, with equal standard deviations but located further and further apart (depending on the scale parameter). In this case, the state-space coverage should increase until both distributions are sufficiently far apart, should then stay the same, and finally begin to drop because the proximity to the state-space-boundary limits the points to an ever smaller volume. The inspiration for this example distribution is an agent setting off in two opposite directions and getting stuck at these two opposing limits. While somewhat contrived and more extreme than the inspiring example, it highlights difficulties in the exploration measures. Both the bounding-box mean X_{BBM} and the nuclear norm X_{NN} completely fail to account for vastly unexplored areas between the extreme points.

Since the $\mathrm{X}_{\mathcal{U}\mathrm{rel}}$ NNR measure is clipped (by definition of NNR) the measure reaches its limits when the density ratios become extreme, which presumably happens for very small and large scale parameters in this setting. The X_{Urel} kNN approximator is better able to capture the extreme divergence values, however, as pointed out before, this comes at the cost of under-estimating the divergence for points close to the support boundary.

The experiments on synthetic data showed that the histogram based measure is not useful in high-dimensional spaces. The alternatives X_{BBM} and X_{NN} are susceptible to artifacts on bounded support. This susceptibility to boundary artifacts is also present in the kNN-based $\mathrm{X}_{\mathcal{U}\mathrm{rel}}$ estimator, because of these results we employ the NNR-estimator based $\mathrm{X}_{\mathcal{U}\mathrm{rel}}$ in the rest of this paper and refer to it as $\mathrm{X}_{\mathcal{U}\mathrm{rel}}$.

4 Results: What action noise to use?

In this section we analyze the data collected in the experiments described in Section 3.4. We first look at the experiments performed under a constant scale scheduler since this is the most common case in the literature. In this setting we will look at two aspects: first, is one of the two action noise types generally superior to the other *(Q1)*? And secondly, is there a generally preferable action noise scale *(Q2)*? Then, we will compare across constant, linear and logistic schedulers to see if reducing the noise impact over the training process is a reasonable thing to do *(Q3)*. Finally we compare the relative importance of the scheduler, noise type and scale *(Q4)*. See Section D for a brief description of the statistical methods used in this paper.

4.1 (Q1) Which action noise type to use? (and what are the impacts)

To compare the impact of the action noise *type*, we look at the constant $\beta = 1$ case, group the aggregated performance and exploration results (see Section 3.5) by the factors algorithm, environment, and action noise scale and standardize the results to control for their influence. These standardized results are then combined for each noise type. Figure 5 illustrates the results. The comparisons are performed by Welch-t-test, symmetric p-values are listed.

Figure 5 (c) shows that *Ornstein-Uhlenbeck noise leads to increased state-space coverage* under the exploratory policy X as measured by $\mathrm{X}_{\mathcal{U}\mathrm{rel}}$. For completeness Figure 5 (d) shows the state-space coverage of the evaluation policy. Here Ornstein-Uhlenbeck increases coverage which might indicate slightly longer trajectories for policies trained under Ornstein-Uhlenbeck noise, however whether this is preferable or not is task dependant. Exploration likely incurs additional costs, e.g. through action penalties, but also by moving

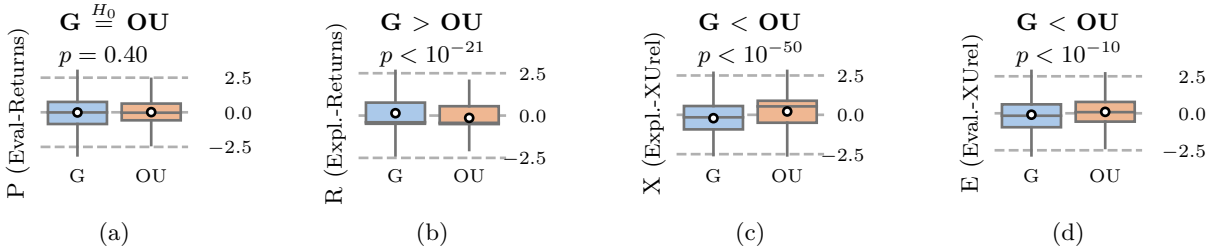


Figure 5: Comparison of standardized measures (P, R, X, E), for Gaussian (G) and Ornstein-Uhlenbeck (OU) noise types, (a-d). Values are standardized to control for and combine algorithm, environment and noise scale: *(a)* For learned performance P, measured by evaluation returns, neither of the two noise types is significantly better. *(b)* For Returns collected under the exploration policy R, Gaussian noise collects data with slightly better returns ($p < 10^{-21}$). (c) For State-space coverage of the exploratory policy X Ornstein-Uhlenbeck performs better. (d) The State-space coverage of evaluation rollouts E is slightly larger for Ornstein-Uhlenbeck noise without significantly affecting the evaluation returns P. Overall neither of the two noise types is superior.

Environment	P	p_P	d_{P}	R	p_R	d_{R}	X	p_X	d_{X}	E	p_E	d_{E}
Half-Cheetah	-	0.89	-	G	0.002	0.22	OU	0.004	0.21	-	0.20	-
Hopper	OU	$<10^{-3}$	0.27	G	$<10^{-4}$	0.29	G	$<10^{-8}$	0.41	-	0.71	-
Inverted-Pendulum-Swingup	-	0.38	-	G	$<10^{-51}$	1.15	OU	$<10^{-56}$	1.22	G	0.002	0.22
Mountain-Car	OU	$<10^{-10}$	0.47	OU	$<10^{-19}$	0.66	OU	$<10^{-5}$	0.34	OU	$<10^{-21}$	0.71
Reacher	G	$<10^{-30}$	0.87	G	$<10^{-26}$	0.80	OU	$<10^{-40}$	1.01	OU	$<10^{-29}$	0.84
Walker2D	-	0.039	-	G	0.010	0.18	OU	$<10^{-9}$	0.46	-	0.28	-

Table 2: Per environment the noise type is important: Comparison of Evaluation Returns P, Exploratory-X$_{\mathcal{U}\text{rel}}$ X, Exploratory Returns R, and Evaluation-X$_{\mathcal{U}\text{rel}}$ E. Values are standardized to control for and aggregate over algorithm, and noise scale. The results are compared using a Welch-t-test. Significantly better noise type for each environment and measure is reported ($p < 0.01$), as well as two-tailed p-values $p_{(\cdot)}$ and Cohen-d effect size $d_{(\cdot)}$. While overall neither of the two noise types leads to significantly better performance P (see Figure 5), per environment noise type difference is significant.

the agent away from high-reward-trajectories. Since Ornstein-Uhlenbeck noise is temporally correlated, it is more efficient in covering more state-space but also in moving the agent away from high-reward trajectories. Thus *exploratory returns R are larger for Gaussian noise* and conversely smaller for Ornstein-Uhlenbeck noise, see Figure 5 (b). The learning process is able to offset some differences in the data as shown in Figure 5 (a): the significant differences in exploratory returns R and exploratory state-space coverage X *do not* translate into significantly-different performance *across environments*. When viewed on a per-environment basis, Table 2 (column P) shows that, *the preferable noise type depends on the environment*: Ornstein-Uhlenbeck is preferable for Hopper and Mountain-Car, but Gaussian for the Reacher environment. Table 2 (column X) shows that Ornstein-Uhlenbeck leads to larger state-space coverage, as before, and Gaussian noise leads to larger exploratory returns (column R). The only exceptions to this are the Hopper environment, where the Ornstein-Uhlenbeck is more likely to topple the agent and the Mountain-Car environment, where the returns are very closely related to increasing the state-space coverage and thus exhibits an improvement of R by Ornstein-Uhlenbeck noise.

These results show that the *noise type is important* and significantly impacts the performance for *some environments*. Neither of the two noise types leads to better performance, evaluation return P, *in general*. However *Ornstein-Uhlenbeck* generally *increases state-space coverage*. This is likely due to the effect, that in many cases the environment acts as an integrator over the actions.

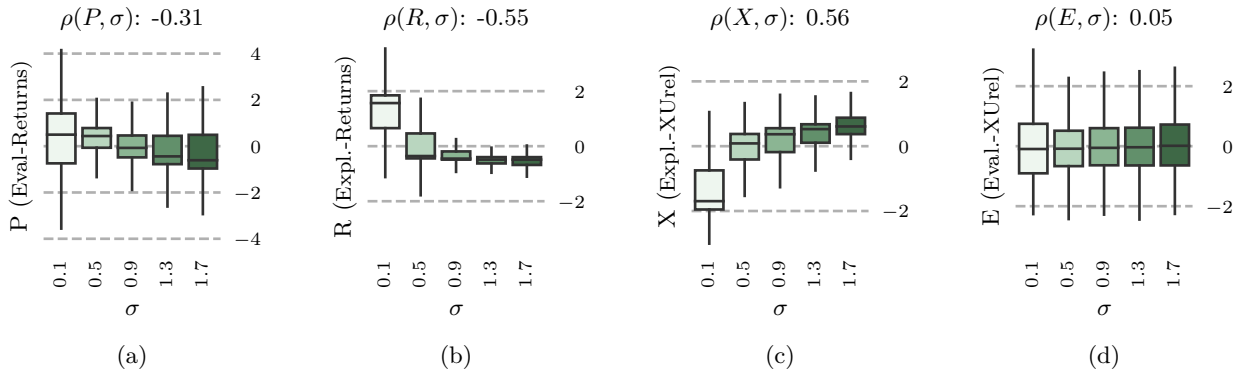


Figure 6: Across environments larger noise scales σ are effective in increasing state-space coverage (c), but reduce exploratory returns (b). Measures (P, X, R, E) are standardized to control for and aggregate over algorithm, environment and noise type. (a) Evaluation Performance P is negatively correlated with action noise scale ($\rho = -0.31$). (b) Larger noise scales correlate with smaller exploratory returns R. (c) Increasing the noise scale σ increases exploratory state space coverage X. (d) State-space coverage of evaluation rollouts E: the learned trajectories appear unaffected by larger noise scale.

Environment	$\rho(P, R)$	$\rho(P, X)$	$\rho(P, \sigma_{\text{scale}})$	$\rho(R, X)$	$\rho(R, \sigma_{\text{scale}})$	$\rho(X, \sigma_{\text{scale}})$
All	0.57	-0.03	-0.31	-0.30	-0.55	0.56
Half-Cheetah	0.22	-0.28	-0.35	-0.64	-0.74	0.75
Hopper	0.69	0.15	-0.87	0.27	-0.74	-0.17
Inverted-Pendulum-Swingup	-0.15	0.23	0.27	-0.88	-0.83	0.77
Mountain-Car	0.94	0.87	0.58	0.76	0.37	0.75
Reacher	0.84	-0.88	-0.56	-0.96	-0.84	0.69
Walker2D	0.76	-0.44	-0.81	-0.52	-0.82	0.63

Table 3: Data quality, measured by exploratory returns R, does not completely determine performance, measured by evaluation returns P. ρ denotes Spearman correlation coefficients. Generally R is positively, but surprisingly not always strongly, correlated with P. For some environments, exploratory state-space coverage X is beneficial, while generally it is associated with decreased evaluation performance P. Across environments and noise types, increasing the noise scale increases exploratory state-space coverage X but reduces exploratory returns R.

4.2 (Q2) Which action noise scale to use?

To analyze the impact of action noise scale, we look at the constant ($\beta = 1$) case, and control for the impact of the factors algorithm, environment and noise type: by grouping the results according to these factors and standardizing the results. Then results for the same noise scale are combined.

An interesting observation shown in Figure 6 (c) is that state-space coverage of the exploratory policy X correlates positively with action noise scale σ (ρ Spearman correlation coefficients). The takeaway from this is that instead of changing the noise type, one might *increase state-space coverage by increasing* σ. This however leads to a reduction in the exploratory returns R, see Figure 6 (c), ($\rho(R, \sigma) = -0.55$). Subsequently, larger noise scales σ are associated with decreased learned performance, i.e. smaller evaluation returns P, Figure 6 (a), when viewed across environments. Note that for very small noises ($\sigma = 0.1$) the variance of the results P becomes very large. It appears that, in many cases, less noise is actually better, but too little noise often does not work well. A good default for σ appears to be > 0.1 but < 0.9. The scale σ does not appear to have a strong effect on the evaluation state-space coverage E, Figure 6 (d). When viewed separately for each environment (Table 3), the association between X and σ is consistent. The only exception is the Hopper task, where a large noise is more likely to topple the agent, making it fail earlier, thereby reducing state-space coverage. The association between $\rho(R, \sigma)$ is consistently negative, with the exception of the Mountain-Car where more state-space coverage directly translates to higher returns, because the

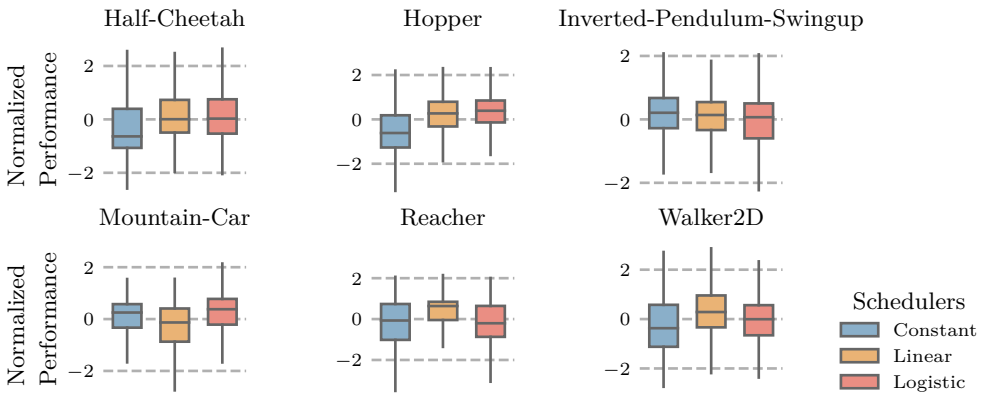


Figure 7: In the majority of cases action noise schedulers improve performance. The figure shows the comparison of the learned policy performance, measured by evaluation returns P, for each environment and scheduler. Data is standardized to control for influence of algorithm, environment, noise scale σ and noise type. In the majority of cases the linear and logistic schedulers perform better than or comparably to the constant scheduler.

	var(P)			P		
Scheduler	< Constant	< Linear	< Logistic	> Constant	> Linear	> Logistic
Constant	0	0	0	0	1	1
Linear	4	0	1	4	0	2
Logistic	4	1	0	4	1	0

Table 4: In the majority of cases, using a scheduler reduces variance of the performance (evaluation returns) var(P), and improves expected performance P. The evaluation returns P are standardized to control for the influence of algorithm, noise scale σ and noise type. Levene's tests are used to assess difference in variance var(P) and a multiple-comparison Games-Howell test indicates superior performance P. The table shows the number of environments on which each scheduler (row) is significantly better than the other schedulers (column). See Table F.1 for full per-environment results.

environment is underactuated and energy needs to be injected into the system. Offline-RL findings indicate that it is easier to learn from expert data than from data of mixed-quality (Fu et al., 2020). As such, we would expect a very strong correlation between exploratory returns R as a measure of data quality and evaluation returns P as a measure of learned performance. Indeed, $\rho(P, R)$ shows that overall exploratory returns R and evaluation returns P are mostly positively correlated. However, the correlation is not always very strong and can even be negative. This is interesting, because this means that *exploratory returns are not the only determining factor* for learned performance. For example, in the Inverted-Pendulum-Swingup, $\rho(P, R)$ is slightly negative while $\rho(P, X)$ is positive. The results indicate that, the noise scale σ has to be chosen to achieve a trade-off between either increasing state-space coverage X or returns R as required for each specific environment.

4.3 (Q3) Should we scale down the noise over the training process?

The previous sections indicated that there is no unique solution for the best noise type and that this choice is dependent on the environment. The analysis of the noise scale showed an overall preference for smaller noise scales, but also showed that, in contrast, some environments require more noise to be solved successfully. In this section we analyze schedulers that reduce the influence of action noise (β) over the training progress.

Figure 7 shows the performance for each environment and each scheduler. The data is normalized by environment and algorithm before aggregation. The general tendency observed across environments is that, when the environment reacts negatively to larger action noise scale (Half-Cheetah, Hopper, Reacher, Walker2D; as shown in Table 3), *reducing the noise impact* β over time consistently *improves performance*. The *re-*

Envname	Spearman Correlation $\rho(P,X)$	$\rho(P,\sigma)$	$\rho(P,R)$	η^2 Effect Size $\eta^2_{\text{Scheduler}}$	η^2_{Type}	η^2_{σ}
All	-0.503	-0.120	0.770	0.005	0.000	**0.084**
Mountain-Car	0.662	0.442	0.959	0.032	0.060	**0.261**
Inverted-Pendulum-Swingup	-0.003	0.123	0.163	0.005	0.009	**0.115**
Reacher	-0.872	-0.382	0.803	0.048	**0.181**	**0.181**
Hopper	-0.349	-0.599	0.651	0.045	0.022	**0.660**
Walker2D	-0.658	-0.494	0.677	0.014	0.017	**0.607**
Half-Cheetah	-0.581	-0.259	0.745	0.007	0.002	**0.148**

Table 5: Spearman correlation coefficients and ANOVA η^2 effect sizes on P for: scheduler, noise type and noise scale σ. action noise scale σ is associated with the largest effect size for evaluation returns P. Results are shown across all environments (standardized and controlled for environment and algorithm, first row), and per environment (standardized and controlled for algorithm). Generally, exploratory returns R and evaluation performance P are positively associated, while generally larger state-space coverage X appears to impact performance P negatively.

verse effect appears to be *less important*: for environments benefiting from larger noise scales, the constant scheduler does not consistently outperform the linear and logistic schedulers.

Table 4 shows summarized results indicating the number of environments where scheduler (1), indicated by row, is better than scheduler (2), indicated by column, in terms of variance $\text{var}(P)$ and mean performance P. See Table F.1 for full results on the pairwise comparisons. Performance differences are assessed by a Games-Howell multiple comparisons test, while variance is compared using Levene's test.

The tests underlying Table 4 show that the differences observed in Figure 7 are indeed significant. Furthermore, the schedulers (linear, logistic) reduce variance $\text{var}(P)$ compared to the constant case in four out of six cases. Keeping the impact β constant has no beneficial effect on variance in any environment. This indicates that *using a scheduler* to reduce action noise impact increases *consistency in terms of learned performance*.

4.4 (Q4) How important are the different parameters?

In the previous sections we looked at each noise configuration parameter independently, first for the constant β case (Q1, Q2), secondly for scheduled reduction of β (Q3). However, the question remains whether all the parameters are equally important. We standardize results to control for environment and algorithm, and compare across all noise types, noise scales σ and all three schedulers.

Table 5 shows Spearman correlation coefficients $\rho(P,X), \rho(P,\sigma), \rho(P,R)$ across all three schedulers (compare to Table 3 which showed correlations for the constant $\beta = 1$ case only). Across environments the schedulers *reduce* correlation $\rho(P,\sigma)$ between learned performance (measured by evaluation returns P) and noise scale σ: from $\rho(P,\sigma) = -0.31$ in the constant scheduler case to $\rho(P,\sigma) = -0.12$ when compared across all three types of schedulers. This is a further indication that using a scheduler increases robustness to σ. The correlations between $\rho(P,R)$ are increased to 0.77 vs. 0.57, presumably because reducing β makes the exploratory policy *more on-policy* and thus P and R become more similar. Interestingly, the schedulers also increase the negative correlation $\rho(P,X)$ between the performance and the exploratory state-space coverage, from -0.03 in the constant case to -0.50 when viewed across all schedulers. This could be driven by the environments reacting positively to reduced state-space coverage, which under the schedulers achieve more runs high in R but low in X, and thus a stronger negative correlation.

The three columns on the right in Table 5 show η^2 effect sizes of a three-way ANOVA on the evaluation returns P: $\eta^2_{\text{Scheduler}}$, η^2_{Type}, η^2_{σ}. The η^2 effect sizes measure the percentage-of-total variance explained by each factor. Only in the Reacher environment, action noise *type* is very important. Surprisingly, in *all cases* the most important factor is *action noise scale*, while the requirement for a large or small action noise scale varies for each environment.

Envname	Scheduler	σ	Type	Horizon	Recommendation
All	lin	0.1/0.5	OU		
Mountain-Car	log	1.7	OU	L	large σ, OU, sched
Inverted-Pendulum-Swingup	con	0.5	Gauss	L	large σ
Reacher	lin	0.1	Gauss	-	small σ, Gauss, sched
Hopper	lin/log	0.1	OU	S	small σ, sched, OU
Walker2D	lin	0.1	OU	S	small σ, OU, sched
Half-Cheetah	lin/log	0.5	Gauss/OU	S	small σ

Table 6: Comparison of best-ranked noise type, scale and scheduler across all environments and for each environment individually. Scheduler, type and scale are investigated separately by standardizing the values to control for environment, algorithm and the other two respective factors. Horizon indicates whether we expect a long (L) or short (S) effective planning horizon. Recommendation indicates action noise configuration choices in order of importance as per Table 5, for options with effect sizes $\eta^2 > 0.01$ (small effect).

5 Discussion & Recommendations

The experiments conducted in this paper showed that the action noise does, depending on the environment, have a *significant* impact on the evaluation performance of the learned policy (Q1). Which action noise type is best unfortunately *depends on the environment*. For the action noise scale (Q2), our results have shown that generally a larger noise scale increases state-space coverage. But since for many environments, learning performance is negatively associated with larger state-space coverage, a large noise scale does not generally have a preferable impact. Similarly, very small scales also appear not to have a preferable impact, as they appear to increase variance of the evaluation performance (Figure 6). However, overall, reducing the action noise scaling factor over time (Q3) mostly has positive effects. Finally we also looked at all factors concurrently (Q4) and found that for most environments noise scale is the most important factor.

It is difficult to draw general conclusions from a limited set of environments and extending the evaluation is limited by the prohibitively large computational costs. However, we would like to provide heuristics derived from our observations that may guide the search for the right action noise. Table 6 shows the best-ranking scheduler, scale and type configurations for each, and across environments. The ranking is based on the count of significantly better comparisons (pairwise Games-Howell test on difference, $p \leq 0.01$, positive test statistic). For each of scheduler, type and scale we standardize to control for the other two factors. Intuitively, the locomotion environments require only a short effective planning horizon: the reward in the environments is based on the distance moved and is relevant as soon as the locomotion pattern is repeated; for example a 30-step horizon is enough for similar locomotion benchmarks (Pinneri et al., 2020). In contrast, the Mountain-Car environment only provides informative reward at the end of a successful episode and thus, the planning horizon needs to be long enough to span a complete successful trajectory (e.g. closer to 100 steps). Similar, the Inverted-Pendulum-Swingup uses a shaped reward that does not account for spurious local optima: to swing up and increase system energy, the distance to the goal has to be increased again. These observations are indicated in the column *Horizon* (Table 6). Finally, the recommendation column interprets the best-ranked results under the observed importance (Q4) reported in Table 5. Given these results, we provide the following intuitions as a starting point for optimizing the action noise parameters (read as: to address this $\triangleright$ do that):

Environment is under-actuated $\triangleright$ increase state-space coverage We found that in the case of the Mountain-Car and the Inverted-Pendulum-Swingup, both of which are underactuated tasks and require a swinging up phase, larger state-space coverages or larger action noise scales appear beneficial (Table 3 and Table 5). Intuitively, under-actuation implies harder-to-reach state-space areas.

Reward is misleading $\triangleright$ increase state-space coverage Actions are penalized in the Mountain Car by an action-energy penalty, which means not performing any action forms a local optimum. In the case of the Inverted-Pendulum-Swingup, the distance to the goal forms a shaped reward. However, when swinging up, increasing the distance to the goal is necessary. Thus, the shaped reward can be misleading and optimizing

for the reward too greedily moves the agent away from taking necessary steps. Optimizing for a spurious local optimum implies not reaching areas of the state space where the actual goal would be found, thus the state-space coverage needs to be increased to find these areas.

Horizon is short ▷ reduce state-space coverage The environments Hopper, Reacher, Walker2D model locomotion tasks with repetitive movement sequences. In the Mountain-Car, positive reward is only achieved at the successful end of the episode, where as in the locomotion tasks positive reward is received after each successful cycle of the locomotion pattern. Thus effectively the required planning horizon is shorter compared to tasks such as the Mountain-Car. Consistently with the previous point, if the effective horizon is shorter, the rewards are shaped more efficiently, we see negative correlations with the state-space coverage and the noise scale: if the planning horizon is shorter, the reward can be optimized more greedily, meaning the state-space coverage can be more focused and thus smaller.

Need more state-space coverage ▷ increase scale Our analysis showed that, to increase state-space coverage, one way is to increase the scale of the action noise. This leads to a higher probability of taking larger actions. In continuous control domains, actions are typically related to position-, velocity- or torque-control. In position-control, larger actions are directly related to more extreme positions in the state space. In velocity control, larger actions lead to moving away from the initial state more quickly. In torque control, larger torques lead to more energy in the system and larger velocities. Currently most policies in D-RL are either uni-modal stochastic policies, or deterministic policies. In both cases, larger action noise leads to a broader selection of actions and, by the aforementioned mechanism, to a broader state-space coverage. Note that while this is the general effect we observed, it is also possible that a too large action can have a detrimental effect, e.g. the Hopper falling, and the premature end of the episode will lead to a reduction of the state-space coverage.

Need more state-space coverage ▷ try Ornstein-Uhlenbeck Depending on the environment dynamics, correlated noise (Ornstein-Uhlenbeck) can increase the state-space coverage: for example, if the environment shows integrative behavior over the actions, temporally uncorrelated noise (Gaussian) leads to more actions that "undo" previous progress and thus less coverage. Thus correlated Ornstein-Uhlenbeck noise helps to increase state-space coverage.

Need less state-space coverage or on-policy data ▷ reduce scale | use scheduler to decrease β If the policy is already sufficiently good, or the reward is shaped well enough, exploration should focus around good trajectories. This can be achieved using a small noise scale σ. However, if the environment requires more exploration to find a reward signal, it makes to sense to use a larger action noise scale σ in the beginning while gradually reducing the impact of the noise (Q3). The collected data then gradually becomes "more on-policy".

In general ▷ use a scheduler We found that using schedulers to reduce the impact of action noise over time, decreases variance of the performance, and thus makes the learning more robust, while also generally increasing the evaluation performance overall. Presumably because, once a trajectory to the goal is found, more fine grained exploration around the trajectory is better able to improve performance.

6 Conclusion

In this paper we present an extensive empirical study on the impact of action noise configurations. We compared the two most prominent action noise types: Gaussian and Ornstein-Uhlenbeck, different scale parameters $(0.1, 0.5, 0.9, 1.3, 1.7)$, proposed a scheduled reduction of the impact β of the action noise over the training progress and proposed the state-space coverage measure $X_{\mathcal{U}\text{rel}}$ to assess the achieved exploration in terms of state-space coverage. We compared DDPG, TD3, SAC, and its deterministic variant detSAC on the benchmarks Mountain-Car, Inverted-Pendulum-Swingup, Reacher, Hopper, Walker2D, and Half-Cheetah.

We found that (Q1) neither of the two noise types (Gaussian, Ornstein-Uhlenbeck) is generally superior across environments, but that the impact of noise type on learned performance can be significant when viewed separately for each environment: the *noise type* needs to be chosen to *fit the environment*. We found that (Q2) increasing action noise scale, across environments, increases state-space coverage but tends to reduce learned performance. Again, whether state-space coverage and performance are positively correlated,

and thus a larger scale is desired, *depends on the environment*. The positive or negative *correlation should guide the selection* of action noise. Reducing the impact (β) of action noise over training time (Q3), improves performance in the majority of cases and decreases variance in performance and thus *increases robustness* to the action noise *choice*. Surprisingly, we found (Q4) that the *most important factor* appears to be the action noise *scale* σ: if less state-space coverage is required, the scale can be reduced. More state-space coverage can be achieved by increasing the action noise scale. This approach is successful even for Gaussian noise on the Mountain-Car. We synthesized our results into a set of *heuristics* on how to choose the action noise based on the properties of the environment. Finally we *recommend a scheduled reduction* of the action noise impact factor β of over the training progress to improve robustness to the action noise configuration.

References

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing Function Approximation Error in Actor-Critic Methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, October 2018.

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 *Proc. 4th Int. Conf. Learning Representations, (ICLR)*, 2016.

Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. RLlib: Abstractions for Distributed Reinforcement Learning. *arXiv:1712.09381 [cs]*, June 2018.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021a.

Yasuhiro Fujita, Prabhat Nagarajan, Toshiki Kataoka, and Takahiro Ishikawa. ChainerRL: A Deep Reinforcement Learning Library. *Journal of Machine Learning Research*, 22(77):1–14, 2021. ISSN 1533-7928.

Takuma Seno and Michita Imai. D3rlpy: An Offline Deep Reinforcement Learning Library. *arXiv:2111.03788 [cs]*, November 2021.

Andrew William Moore. Efficient memory-based learning for robot control. 1990.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic Algorithms and Applications. *arXiv:1812.05905 [cs, stat]*, January 2019.

Antonin Raffin. RL baselines3 zoo. *GitHub repository*, 2020.

Itai Caspi, Gal Leibovich, Shadi Endrawis, and Gal Novik. Reinforcement Learning Coach. Zenodo, December 2017.

Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. Acme: A Research Framework for Distributed Reinforcement Learning. *arXiv:2006.00979 [cs]*, June 2020.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv:1606.01540 [cs]*, June 2016.

Erwin Coumans and Yunfei Bai. PyBullet, a Python module for physics simulation for games, robotics and machine learning. 2016.

Benjamin Ellenberger. PyBullet gymperium. *GitHub repository*, 2018.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 0028-0836, 1476-4687. doi: 10.1038/nature14236.

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

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, October 2012. doi: 10.1109/IROS.2012.6386109.

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind Control Suite. *arXiv:1801.00690 [cs]*, January 2018.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. June 2018.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. Trust Region Policy Optimization. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1889–1897, Lille, France, 2015. JMLR.org.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *CoRR*, abs/1707.06347, 2017.

David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

Zeping Zhan, Batu Aytemiz, and Adam M Smith. Taking the scenic route: Automatic exploration for videogames. In *KEG@ AAAI*, 2019.

Bogdan Mazoure, Thang Doan, Audrey Durand, Joelle Pineau, and R. Devon Hjelm. Leveraging exploration in off-policy algorithms via normalizing flows. In *Conference on Robot Learning*, pages 430–444. PMLR, May 2020.

Patrick Nadeem Ward, Ariella Smofsky, and Avishek Joey Bose. Improving Exploration in Soft-Actor-Critic with Normalizing Flows Policies. *arXiv:1906.02771 [cs, stat]*, June 2019.

Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, Jan Achterhold, Joerg Stueckler, Michal Rolinek, and Georg Martius. Sample-efficient Cross-Entropy Method for Real-time Planning. *arXiv:2008.06389 [cs, stat]*, August 2020.

Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. *arXiv:1803.07055 [cs, math, stat]*, March 2018.

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. *CoRR*, abs/1706.01905, 2017.

Antonin Raffin, Jens Kober, and Freek Stulp. Smooth Exploration for Robotic Reinforcement Learning. *arXiv:2005.05719 [cs, stat]*, June 2021b.

Yuri Burda, Harrison Edwards, Amos J. Storkey, and Oleg Klimov. Exploration by random network distillation. In *7th International Conference on Learning Representations, ICLR*, 2019.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. #Exploration: A Study of Count-Based Exploration for Deep Reinforcement Learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2753–2762. Curran Associates, Inc., 2017.

Mirco Mutti, Lorenzo Pratissoli, and Marcello Restelli. A Policy Gradient Method for Task-Agnostic Exploration. *arXiv:2007.04640 [cs, stat]*, July 2020.

Zhang-Wei Hong, Tzu-Yun Shann, Shih-Yang Su, Yi-Hsiang Chang, Tsu-Jui Fu, and Chun-Yi Lee. Diversity-driven exploration strategy for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 10489–10500, 2018.

Vitchyr Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 7783–7792. PMLR, 2020.

Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable Baselines. *GitHub repository*, 2018.

Tim Seyde, Igor Gilitschenski, Wilko Schwarting, Bartolomeo Stellato, Martin Riedmiller, Markus Wulfmeier, and Daniela Rus. Is Bang-Bang Control All You Need? Solving Continuous Control with Bernoulli Policies. *arXiv:2111.02552 [cs]*, November 2021.

Morteza Noshad, Kevin R. Moon, Salimeh Yasaei Sekeh, and Alfred O. Hero. Direct estimation of information divergence using nearest neighbor ratios. In *2017 IEEE International Symposium on Information Theory (ISIT)*, pages 903–907, June 2017. doi: 10.1109/ISIT.2017.8006659.

Christopher M. Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006.

Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4RL: Datasets for Deep Data-Driven Reinforcement Learning. *arXiv:2004.07219 [cs, stat]*, June 2020.

Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001.

Raphael Vallat. Pingouin: Statistics in Python. *Journal of Open Source Software*, 3(31):1026, November 2018. ISSN 2475-9066. doi: 10.21105/joss.01026.

Derek C. Sauder and Christine E. DeMars. An Updated Recommendation for Multiple Comparisons. *Advances in Methods and Practices in Psychological Science*, 2(1):26–44, March 2019. ISSN 2515-2459. doi: 10.1177/2515245918808784.

Morton B Brown and Alan B Forsythe. Robust Tests for the Equality of Variances. page 5, 2022.

Appendices

A A motivating example

The action is generated as $\tilde{a}_t \sim \pi_\theta(s_t)$, $a_t = \tilde{a}_t + \varepsilon_{a_t}$, where ε_{a_t} denotes the action noise. We calibrate the noise scale to achieve similar returns for both noise types. To calibrate the action noise scale, we assume a constant-zero-action policy upon which the action noise is added and effectively use $a_t = \varepsilon_{a_t}$ as the action sequence. We find that a scale of about 0.6 for Gaussian action noise and a scale of about 0.5 for Ornstein-Uhlenbeck noise lead to a mean return of about -30. This is shown in Table 1. A successful solution to the Mountain-Car environment yields a positive return $0 < \sum r_t < 100$. We then use these two noise configurations and perform learning with DDPG, SAC and TD3. The resulting learning curves are shown in Figure 1 and very clearly depict the huge impact the noise configuration has: with similar returns of the only policies, we achieve substantially different learning results, either leading to failure or success on the task.

To achieve a swing-up, the actions must not change direction too rapidly but rather need to change direction with the right frequency. Ornstein-Uhlenbeck noise is temporally correlated and thus helps solving the environment successfully with a smaller scale σ. In this environment, the algorithms tend to converge either to the successful solution of the environment by swinging up, or to a passive zero-action solution which incurs no penalty.

B Deterministic SAC

Algorithm B.1 (Deterministic) Soft Actor-Critic

Initialize parameter vectors ψ, $\bar{\psi}$, θ, ϕ.
for each iteration **do**
  for each environment step **do**
    $\mu_t, \sigma_t = f_\phi(s_t)$
    $\varepsilon_t \sim \mathcal{A}$ ▷ $\mathcal{A}$ . . . action noise process
    $a_t = \mu_t + \varepsilon_t$ ▷ DetSAC
    $\pi_\phi(\cdot|s_t) = \mathcal{N}(\cdot|\mu_t, \sigma_t)$ ▷ SAC
    $a'_t \sim \pi_\phi(a_t|s_t)$
    $a_t = a_t + \varepsilon_t$
    $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
  end for
  for each gradient step **do**
    . . . *original SAC update* (Haarnoja et al., 2019)
  end for
end for

C Benchmark Environments

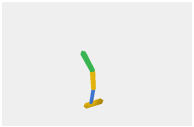
Environment	Illustration	$\dim(\mathcal{O})$	$\dim(\mathcal{A})$	Reward
Mountain-Car		2	1	$\mathbf{1}(s_t, s_G) \quad -\lvert a_t\rvert_2^2$
Inverted-Pendulum-Swingup		5	1	$\lvert\varphi(s_t) - \varphi_G\rvert_1$
Reacher		9	2	$\nabla^-\lvert s_t - s_G\rvert_2 \quad -\lvert\varphi(s_t)\rvert_2^2 \; - \mathbf{1}(\varphi(s_t), \varphi_{\text{limit}}) \quad -\lvert a_t\rvert_1$
Hopper		15	3	$\nabla^-\lvert s_t - s_G\rvert_1 \quad -\lvert\varphi(s_t)\rvert_2^2 \; - \mathbf{1}(\varphi(s_t), \varphi_{\text{limit}}) \quad -\lvert a_t\rvert_1$
Walker2D		22	6	$\nabla^-\lvert s_t - s_G\rvert_1 \quad -\lvert\varphi(s_t)\rvert_2^2 \; - \mathbf{1}(\varphi(s_t), \varphi_{\text{limit}}) \quad -\lvert a_t\rvert_1$
Half-Cheetah		26	6	$\nabla^-\lvert s_t - s_G\rvert_1 \quad -\lvert\varphi(s_t)\rvert_2^2 \; - \mathbf{1}(\varphi(s_t), \varphi_{\text{limit}}) \quad -\lvert a_t\rvert_1$

Table C.1: Benchmarks environments used in our evaluation in increasing order of complexity. $|\mathcal{O}|$ denotes Observation space dimensions. $|\mathcal{A}|$ denotes Action space dimensions. Explanation of Reward components: $\mathbf{1}(b, c)$ indicator function (sparse reward or penalty) of b w.r.t. to the set c; $|b|_n$ n-norm of b; $\varphi(s_t)$ angular component of state; $\nabla^- b$ finite-difference reduction of b between time-steps; $\varphi_{\max}$ joint limit; s_G goal state; $|\varphi(s_t)|_2^2$ denotes an angular-power-penalty. Factors in the reward are omitted. Distances e.g. $|s_t - s_G|_n$ may refer to a subspace of the vector s_t. Section 3.3

D Statistical methods

We use statistical methods implemented in (Jones et al., 2001; Vallat, 2018) as well as our own implementations.

Welch t-test : does not assume equal variance. Reporting two-tailed p-value. Significant for one-tailed when $\frac{p}{2} < \alpha$.

Games-Howell test Performing multiple comparisons with a t-test increases the risk of Type I errors. To control for Type I errors, the Games-Howell test, a multiple-comparison test applicable to heteroscedastic cases, should be used (Sauder and DeMars, 2019).

The test statistic t is distributed according to Tukey's studentized range q and the test statistic t is defined as

$$t = \frac{\bar{x}_i - \bar{x}_j}{\sigma} \tag{16}$$

$$\sigma = \sqrt{\left(\frac{s_i^2}{n_i} + \frac{s_j^2}{n_j}\right)} \tag{17}$$

$$df = \frac{\left(\frac{s_i^2}{n_i} + \frac{s_j^2}{n_j}\right)^2}{\frac{\left(\frac{s_i^2}{n_i}\right)^2}{n_i - 1} + \frac{\left(\frac{s_j^2}{n_j}\right)^2}{n_j - 1}} \tag{18}$$

$$\tag{19}$$

The p-value is then calculated for k sample-groups as

$$q_{t \cdot \sqrt{2}, k, df} \tag{20}$$

ANOVA We perform a *balanced* N-way ANOVA, i.e. with N independent factors, each with multiple levels (categorical values). Since the study design is balanced this is equivalent to a type-I ANOVA in which the order of terms does not matter (because the design is balanced).

Eta squared η^2 The effect size eta squared η^2 denotes the relative variance explained by a factor to the total variance observed: $\eta^2 = \frac{SS_{C(x)}}{SS_{\text{Total}}}$

	DF	Sum of Squares	F	PR(>F)
C(y)	9.0	4167.583	478.576	0
C(x)	9.0	91.118	10.463	1.7e-15
C(y):C(x)	81.0	81.172	1.036	0.397
Residual	901.0	871.798		
Total		5211.672		

Table D.1: ANOVA example. The partial η^2 for a factor is calculated as the sum of squares, variance explained by that factor, divided by the sum of the variance explained plus the unexplained residual variance.

Effect sizes are interpreted as:

$$\eta^2 \geq 0.01 \quad \text{small effect} \tag{21}$$

$$\eta^2 \geq 0.06 \quad \text{medium effect} \tag{22}$$

$$\eta^2 \geq 0.14 \quad \text{large effect} \tag{23}$$

$$\tag{24}$$

Levene's Test assesses (un)equality of group variances.

$$z_{ij} = |y_{ij} - \tilde{y}_j| \tag{25}$$

$$F = \frac{N-p}{p-1} \frac{\sum_{j=1}^{p} n_j (\tilde{z}_j - \tilde{z})^2}{\sum_{j=1}^{p} \sum_{i=1}^{n_j} (z_{ij} - \tilde{z}_j)^2} \tag{26}$$

$$d_1 = p - 1 \tag{27}$$

$$d_2 = N - p \tag{28}$$

$$\tilde{z}_j = \frac{1}{n_j} \sum_{i=1}^{n_j} z_{ij} \tag{29}$$

$$\tilde{z} = \frac{1}{N} \sum_{j=1}^{p} \sum_{i=1}^{n_j} z_{ij} \tag{30}$$

where p is the number of groups, n_j is the size of group j and N is the total number of observations. $\tilde{y}_j$ is the median of group j, z_{ij} denotes sample i in group j. The F statistic follows the F-distribution with degrees of freedom d_1, d_2.

This variant of Levene's test, $\tilde{y}_j$ median instead of mean, is also called Brown-Forsythe test (Brown and Forsythe, 2022) and is more robust to non-normal distributions.

Cohen-d effect size : Cohen-d is illustrated in Figure D.1 and measures the distance of the means of two sample groups normalized to the pooled variance:

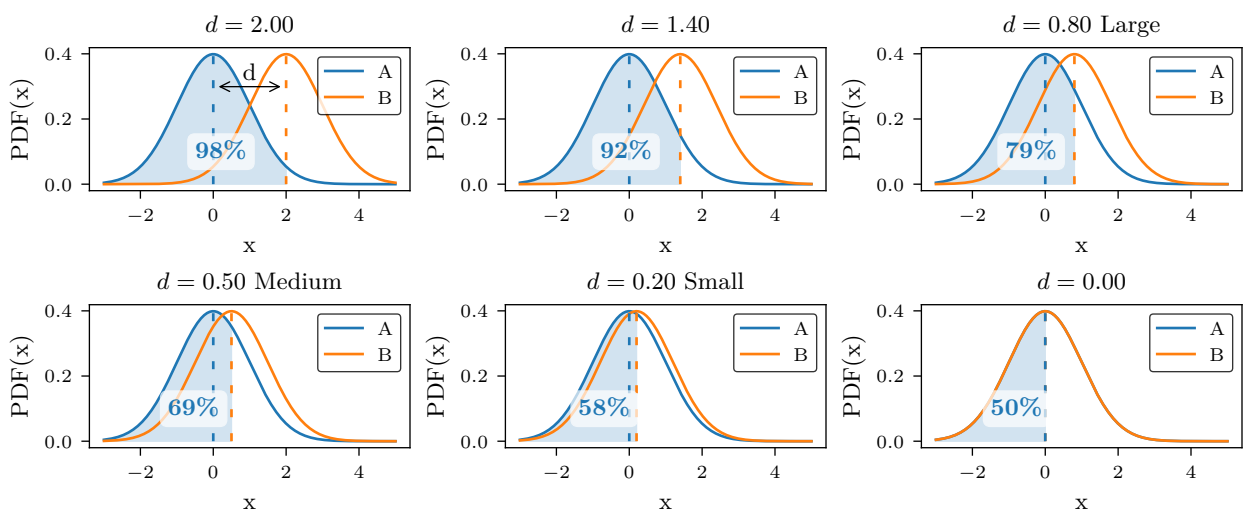


Figure D.1: Illustration of Cohen-d effect size: the Cohen-d measures the standardized difference between the means of two groups, equivalent to a z-score. Effect sizes $d \geq 0.2$ are called small, $d \geq 0.5$ medium, $d \geq 0.8$ large effects. Under equal-variance Gaussian assumption this can be interpreted as n-percent of group A below the mean of group B. Illustrated as the shaded area.

$$\text{Effect size} = \frac{[\text{Mean Group A}] - [\text{Mean Group B}]}{\text{Pooled Std Deviation}} \tag{31}$$

$$d = \frac{\bar{x}_1 - \bar{x}_2}{s} \tag{32}$$

$$s = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}} \tag{33}$$

E (Q1) Noise Type difference per environment

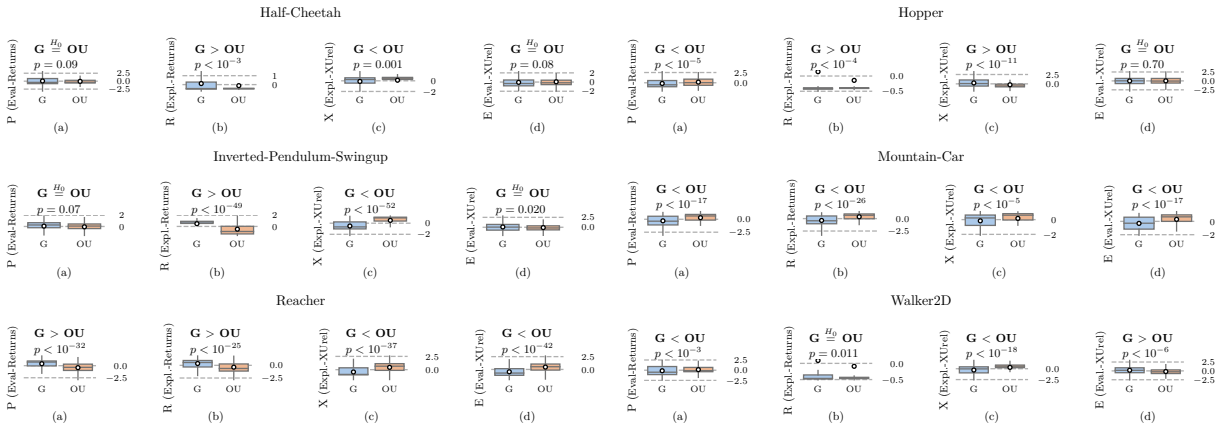


Figure E.1: Separate results for constant scheduler; comparison of noise type on P, R, X, E.

F Impact of Scheduler on Variance and Learned Performance

		var(P)			P		
		< Constant	< Linear	< Logistic	> Constant	> Linear	> Logistic
Scheduler	Envname						
Constant	Half-Cheetah		No	No		No	No
	Hopper		No	No		No	No
	Inverted-Pendulum-Swingup		No	No		No	Yes $p < 10^{-5}$
	Mountain-Car		No	No		Yes $p < 10^{-6}$	No
	Reacher		No	No		No	No
	Walker2D		No	No		No	No
Linear	Half-Cheetah	Yes $p < 10^{-3}$		No	Yes $p < 10^{-6}$		No
	Hopper	Yes $p < 10^{-6}$		No	Yes $p < 10^{-6}$		No
	Inverted-Pendulum-Swingup	No		No	No		No
	Mountain-Car	No		No	No		No
	Reacher	Yes $p < 10^{-29}$		Yes $p < 10^{-15}$	Yes $p < 10^{-6}$		Yes $p < 10^{-6}$
	Walker2D	Yes $p < 10^{-4}$		No	Yes $p < 10^{-6}$		Yes $p < 10^{-6}$
Logistic	Half-Cheetah	Yes $p < 10^{-4}$	No		Yes $p < 10^{-6}$	No	
	Hopper	Yes $p < 10^{-8}$	No		Yes $p < 10^{-6}$	No	
	Inverted-Pendulum-Swingup	No	No		No	No	
	Mountain-Car	No	Yes $p = 0.002$		Yes $p < 10^{-5}$	Yes $p < 10^{-6}$	
	Reacher	Yes $p < 10^{-4}$	No		No	No	
	Walker2D	Yes $p < 10^{-6}$	No		Yes $p < 10^{-4}$	No	
Constant	Sum	0	0	0	0	1	1
Linear	Sum	4	0	1	4	0	2
Logistic	Sum	4	1	0	4	1	0

Table F.1: In the majority of cases, using a scheduler *reduces variance* var(P) of the performance (evaluation returns), and *improves* expected performance P. The rows shows whether "Scheduler" is significantly better than the scheduler indicated in the columns var(P) and P. The evaluation returns P are standardized to control for the influence of algorithm, noise scale σ and noise type. Levene's test is used to assess difference in variance var(P) and a multiple-comparison Games-Howell test indicates superior performance P.

G Performed experiments

			run									
		noise-scale	0.1	0.5	0.9	1.3	1.7	0.1	0.5	0.9	1.3	1.7
		noise_type	gauss	gauss	gauss	gauss	gauss	ou	ou	ou	ou	ou
algorithm	envname	noise-scheduler										
ddpg	HalfCheetahPyBulletEnv-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
	HopperPyBulletEnv-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
	InvertedPendulumSwingupPyBulletEnv-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
	MountainCarContinuous-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
	ReacherPyBulletEnv-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
	Walker2DBulletEnv-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
detsac	HalfCheetahPyBulletEnv-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
	HopperPyBulletEnv-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
	InvertedPendulumSwingupPyBulletEnv-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
	MountainCarContinuous-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
	ReacherPyBulletEnv-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
	Walker2DBulletEnv-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
sac	HalfCheetahPyBulletEnv-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
	HopperPyBulletEnv-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
	InvertedPendulumSwingupPyBulletEnv-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
	MountainCarContinuous-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
	ReacherPyBulletEnv-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
	Walker2DBulletEnv-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
td3	HalfCheetahPyBulletEnv-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
	HopperPyBulletEnv-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
	InvertedPendulumSwingupPyBulletEnv-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
	MountainCarContinuous-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
	ReacherPyBulletEnv-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20
	Walker2DBulletEnv-v0	constant_schedule	20	20	20	20	20	20	20	20	20	20
		linear_schedule	20	20	20	20	20	20	20	20	20	20
		logistic_schedule	20	20	20	20	20	20	20	20	20	20

Table G.1: This table shows the number and configurations of independent learning experiments with different noise settings we use in this paper. Each run is performed from an independently, randomly drawn seed.

H Hyperparameters

Environment	MountainCarContinuous-v0	InvertedPendulumSwingupPyBulletEnv-v0	ReacherPyBulletEnv-v0	HopperPyBulletEnv-v0	Walker2DBulletEnv-v0	HalfCheetahPyBulletEnv-v0
policy	MlpPolicy	MlpPolicy	MlpPolicy	MlpPolicy	MlpPolicy	MlpPolicy
timesteps	60000.0	1000000.0	1000000.0	2000000.0	2000000.0	2000000.0
env_wrapper				TimeFeatureWrapper	TimeFeatureWrapper	TimeFeatureWrapper
gamma	0.99	0.99	0.99	0.99	0.99	0.98
buffer_size	50000.0	1000000.0	1000000.0	1000000.0	1000000.0	1000000.0
learning_starts	0.0	1000.0	1000.0	1000.0	1000.0	10000.0
gradient_steps	1.0	-1.0	-1.0	-1.0	-1.0	-1.0
train_freq	1	(1, episode)	(1, episode)	(1, episode)	(1, episode)	(1, episode)
learning_rate	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003
batch_size	64.0	64.0	64.0	256.0	256.0	256.0
ent_coef	auto	0.01	0.01	0.01	0.01	auto
tau	0.005	0.005	0.005	0.005	0.005	0.01

a SAC/DetSAC Hyperparameters

Environment	MountainCarContinuous-v0	InvertedPendulumSwingupPyBulletEnv-v0	ReacherPyBulletEnv-v0	HopperPyBulletEnv-v0	Walker2DBulletEnv-v0	HalfCheetahPyBulletEnv-v0
policy	MlpPolicy	MlpPolicy	MlpPolicy	MlpPolicy	MlpPolicy	MlpPolicy
timesteps	300000.0	300000.0	300000.0	1000000.0	1000000.0	1000000.0
env_wrapper		TimeFeatureWrapper	TimeFeatureWrapper	TimeFeatureWrapper	TimeFeatureWrapper	TimeFeatureWrapper
gamma	0.99	0.98	0.98	0.98	0.98	0.98
buffer_size	1000000	200000.0	200000.0	200000.0	200000.0	200000.0
learning_starts	100	10000.0	10000.0	10000.0	10000.0	10000.0
gradient_steps	-1	-1.0	-1.0	-1.0	-1.0	-1.0
train_freq	(1, 'episode')	(1, episode)	(1, episode)	(1, episode)	(1, episode)	(1, episode)
learning_rate	0.001	0.001	0.001	0.001	0.001	0.001
policy_kwargs	None	{'net_arch': [400, 300]}	{'net_arch': [400, 300]}	{'net_arch': [400, 300]}	{'net_arch': [400, 300]}	{'net_arch': [400, 300]}

b TD3 Hyperparameters

Environment	MountainCarContinuous-v0	InvertedPendulumSwingupPyBulletEnv-v0	ReacherPyBulletEnv-v0	HopperPyBulletEnv-v0	Walker2DBulletEnv-v0	HalfCheetahPyBulletEnv-v0
policy	MlpPolicy	MlpPolicy	MlpPolicy	MlpPolicy	MlpPolicy	MlpPolicy
timesteps	300000.0	300000.0	300000.0	1000000.0	1000000.0	1000000.0
env_wrapper		TimeFeatureWrapper	TimeFeatureWrapper	TimeFeatureWrapper	TimeFeatureWrapper	TimeFeatureWrapper
gamma	0.99	0.98	0.98	0.98	0.98	0.98
buffer_size	1000000	200000.0	200000.0	1000000.0	1000000.0	200000.0
learning_starts	100	10000.0	10000.0	10000.0	10000.0	10000.0
gradient_steps	-1	-1.0	-1	-1.0	-1.0	-1.0
train_freq	(1, 'episode')	(1, episode)	1	(1, episode)	(1, episode)	(1, episode)
learning_rate	0.001	0.001	0.001	0.0007	0.0007	0.001
policy_kwargs	None	{'net_arch': [400, 300]}	{'net_arch': [400, 300]}	{'net_arch': [400, 300]}	{'net_arch': [400, 300]}	{'net_arch': [400, 300]}
batch_size	100	100	100	256.0	256.0	100

c DDPG Hyperparameters

Table H.1: Hyperparameters for SAC, TD3 and DDPG are taken from (Raffin, 2020) or left at default values defined in (Raffin et al., 2021a).

I Environment Limits

Environment	MountainCarContinuous-v0	InvertedPendulumSwingupPyBulletEnv-v0	ReacherPyBulletEnv-v0	HopperPyBulletEnv-v0	Walker2DBulletEnv-v0	HalfCheetahPyBulletEnv-v0
$s^{(0)}$	-1.2000 . . . 0.6000	-1.0993 . . . 1.0931	-0.2700 . . . 0.2700	-1.2433 . . . 0.8614	-1.2316 . . . 0.1270	-0.6542 . . . 0.5536
$s^{(1)}$	-0.0700 . . . 0.0700	-6.1276 . . . 6.0216	-0.2700 . . . 0.2700	-0.0000 . . . 0.0000	-0.0000 . . . 0.0000	-0.0000 . . . 0.0000
$s^{(2)}$		-1.0000 . . . 1.0000	-0.4799 . . . 0.4798	-1.0000 . . . 1.0000	-1.0000 . . . 1.0000	-1.0000 . . . 1.0000
$s^{(3)}$		-1.0000 . . . 1.0000	-0.4799 . . . 0.4795	-5.0000 . . . 3.4373	-3.5129 . . . 1.8573	-1.8748 . . . 2.0801
$s^{(4)}$		-21.9001 . . . 21.2146	-1.0000 . . . 1.0000	-0.0000 . . . 0.0000	-0.0000 . . . 0.0000	-0.0000 . . . 0.0000
$s^{(5)}$			-1.0000 . . . 1.0000	-5.0000 . . . 1.6368	-3.6400 . . . 0.7000	-1.9558 . . . 1.3548
$s^{(6)}$			-10.0000 . . . 10.0000	-3.1416 . . . 3.1416	-3.1416 . . . 0.0000	-3.1416 . . . 0.0000
$s^{(7)}$			-1.2745 . . . 1.2701	-1.5708 . . . 1.5342	-1.5708 . . . 1.0625	-1.5708 . . . 1.0959
$s^{(8)}$			-10.0000 . . . 10.0000	-1.3921 . . . 2.1682	-2.2274 . . . 1.5482	-5.0000 . . . 1.1894
$s^{(9)}$				-5.0000 . . . 5.0000	-5.0000 . . . 4.6218	-3.6561 . . . 3.8614
$s^{(10)}$				-1.3917 . . . 1.8215	-1.5703 . . . 1.8112	-4.8688 . . . 2.1159
$s^{(11)}$				-5.0000 . . . 5.0000	-4.2228 . . . 4.3884	-5.0000 . . . 3.7174
$s^{(12)}$				-3.2458 . . . 2.1586	-3.7981 . . . 1.5704	-3.7094 . . . 4.0843
$s^{(13)}$				-5.0000 . . . 5.0000	-4.9543 . . . 3.1231	-5.0000 . . . 5.0000
$s^{(14)}$				-0.0000 . . . 1.0000	-2.8370 . . . 1.2062	-2.7263 . . . 1.6122
$s^{(15)}$					-4.9879 . . . 4.1318	-5.0000 . . . 4.6168
$s^{(16)}$					-1.7225 . . . 1.7902	-5.0000 . . . 3.2957
$s^{(17)}$					-3.7233 . . . 4.2734	-5.0000 . . . 5.0000
$s^{(18)}$					-4.0686 . . . 1.5315	-3.9515 . . . 3.3214
$s^{(19)}$					-5.0000 . . . 2.7369	-5.0000 . . . 5.0000
$s^{(20)}$					-0.0000 . . . 1.0000	-0.0000 . . . 1.0000
$s^{(21)}$					-0.0000 . . . 1.0000	-0.0000 . . . 1.0000
$s^{(22)}$						-0.0000 . . . 1.0000
$s^{(23)}$						-0.0000 . . . 1.0000
$s^{(24)}$						-0.0000 . . . 1.0000
$s^{(25)}$						-0.0000 . . . 1.0000

Table I.1: The calculation of $X_{\mathcal{U}\text{rel}}$ requires defined state space limits for each environments. However, some environments define the limits as $(-\infty, \infty)$. In these cases we collected state space samples and defined the limits empirically.