Guiding Evolutionary Strategies with Off-Policy Actor-Critic

Yunhao Tang
Columbia University
New York, NY, USA
yt2541@columbia.edu

ABSTRACT

Evolutionary strategies (ES) and off-policy learning algorithms are two major workhorses of Reinforcement learning (RL): ES adopt a simple blackbox approach to optimization but it can be slightly more sample inefficient; off-policy learning is by design more sample efficient but the updates can be unstable. Motivated by their trade-offs, we propose CEM-ACER, a combination of Cross-entropy method, a standard ES algorithm, and Actor-critic with experience replay (ACER), an off-policy actor-critic algorithm. Our proposal relies on a key insight: off-policy algorithms provide a natural mechanism to efficiently evolve parameter populations as part of an ES algorithm. Across a wide range of benchmark control tasks, we show that CEM-ACER balances the strengths of CEM and ACER, leading to an algorithm that consistently outperforms its individual building blocks, as well as other competitive baseline algorithms.

KEYWORDS

Reinforcement Learning, Evolutionary Strategies, Off-policy Learning

1 INTRODUCTION

Reinforcement learning (RL) has proved to be a powerful paradigm for general sequential decision making, through its successful applications to numerous simulated and real life domains [20, 24, 33]. Conventional knowledge tends to perceive near on-policy and off-policy algorithms as two complementary approaches for solving challenging RL problems: near on-policy methods [23, 31, 32] construct parameter updates based on trajectories sampled under the current policy iterate. This usually leads to more stable updates at the cost of lower sample efficiency, because the samples are discarded immediately after the on-policy updates; on the other hand, off-policy methods [21, 41] construct updates based on samples generated by arbitrary behavior policies, e.g. past policies. This allows for extensive sample re-use and greatly improves sample efficiency. However, the algorithm might suffer from unstable training due to the fundamental instability of off-policy learning [34, 40]. The above comparison presents a clear trade-off between sample efficiency and learning stability, both critical for RL applications. The trade-off motivates a careful combination of on-policy and off-policy methods to obtain a more efficient middle ground.

We seek a principled combination of on-policy and off-policy algorithms for RL. First, we identify the recently revived Evolution strategies (ES) as a special variant of near on-policy methods [29]. ES construct updates based on local perturbations of the current policy parameter, which preserves the spirit of near on-policy methods. Though ES methods typically discard most of the structure about the underlying problem (e.g. the Markov decision process (MDP) assumption) compared to Policy gradients (PG) based algorithms [23, 31, 32, 35], in practice their performance is competitive to state-of-the-art PG algorithms [22, 29]. Furthermore, many ES methods allow for more flexible updates other than gradient descents (as discussed in Section 3), which potentially entails more efficient combination with off-policy methods.

Main idea. We propose CEM-ACER, a combination of Cross-entropy method (CEM) [5], and Actor-critic with experience replay (ACER) [41]. Our proposal relies on a key insight: off-policy methods provide a natural and principled mechanism to evolve the sampled policy parameters generated by CEM. Off-policy updates allow the policy parameters to aggressively *jump* forward in the parameter space without requiring additional samples from the environment. Meanwhile, the evolutionary mechanism of CEM automatically eliminates solutions which suffer from the instability of off-policy updates, in order to safeguard the performance of the final policy. As a result, the aggregated algorithm achieves both the sample efficiency of off-policy algorithms and the stability of ES algorithms.

Our paper proceeds as follows. In Section 2 and Section 3, we introduce related work, along with background on CEM and ACER. In Section 4, we expand on the idea of CEM-ACER: we provide both intuitive arguments as to why CEM-ACER works, as well as some theoretical guarantees. In Section 5, we show with comprehensive experiments that CEM-ACER outperforms baseline algorithms across a wide range of benchmark tasks. This corroborates that our proposed technique combines the strength of its building blocks while offsetting their drawbacks.

2 RELATED WORK.

Combining on-policy and off-policy updates. There are prior attempts at combining on-policy updates with off-policy information, e.g. through memory [28], imitation learning [26] and exploiting the connection between on-policy PG and Q-learning [25, 30]. To directly combine gradient-based updates, Gruslys et al. [12], Wang et al. [41] derive a general form of PG estimator which interpolates between on-policy and off-policy gradient estimators. This unified estimator achieves a principled trade-off between bias and variance. Gu et al. [13] also derives an interpolated estimator specialized to continuous control. Complementary to the aforementioned work, we study how to incorporate off-policy information through off-policy gradient updates into the evolutionary step of ES methods.

Combining ES and Off-policy updates. Though in spirit similar to near on-policy PG based algorithms, ES are not typically

Proc. of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021), U. Endriss, A. Nowé, F. Dignum, A. Lomuscio (eds.), May 3–7, 2021, Online.

categorized as on-policy algorithms due to their strong connections to blackbox optimization. The idea of combining ES with off-policy updates is not new: Khadka and Tumer [17] apply off-policy Q-learning [21] to speed up genetic algorithms [6]; Pourchot and Sigaud [27] alternate the updates of off-policy Q-learning [11] with CEM; more recently, Khadka et al. [16] improve upon [17] with diversified exploration and better resource allocation. ES have also been applied to entail better exploration, e.g. to generate diverse off-policy samples for a concurrent off-policy learner [4]. Complementary to prior work, we combine CEM with off-policy actor-critic updates. Compared to continuous Q-learning [16, 17, 27], our method provides a more direct guarantee on monotonic improvement and directly handles both discrete/continuous action domains. We expand on the details in Section 4.

3 BACKGROUND

RL is formulated under the standard framework of MDP. At each time step $t \geq 0$, an agent is in a state $s_t \in \mathcal{S}$. When an action $a_t \in \mathcal{A}$ is taken, the agent receives an instant reward $r_t \in \mathbb{R}$ and transitions to a next state $s_{t+1} \sim p(\cdot|s_t, a_t)$. Define a policy π as a conditional distribution given state s_t over actions $a_t \sim \pi(\cdot|s_t)$. Given a discount factor $\gamma \in (0, 1]$, the expected cumulative rewards under policy π is given by

$$J(\pi) = \mathbb{E}_\pi[\sum_{t\geq 0} \gamma^t r_t]. \tag{1}$$

The objective of RL is to search for the optimal policy $\pi^* = \arg\max J(\pi)$. For convenience, we introduce notations for value function $V^\pi(s)$, action value function $Q^\pi(s, a)$ and advantage function $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$ under policy π. To tractably optimize π, it is conventional to parameterize the policy π_θ with parameter θ. We aim to iteratively update θ to improve $J(\pi_\theta)$. The mainstream policy-based model-free updates are generally categorized into either ES or PG based algorithms, which we introduce below.

3.1 Evolutionary Strategies

ES are a class of zeroth-order optimization algorithms for general purpose blackbox optimization, where we search for a solution x to maximize the blackbox function $f(x)$. When applying ES methods to RL, we flatten $J(\pi_\theta)$ into such a blackbox problem where the solutions are policy parameters $x \equiv \theta$ and blackbox functions are the RL returns $f(x) \equiv J(\pi_\theta)$. In its most general form, ES maintain a distribution over solutions and iteratively update the distribution parameters. In general, different ES methods differ in how to maintain the distribution and how to update the distribution parameters [5, 15, 29]. Here we focus on CEM [5].

The CEM maintains a Gaussian distribution over solutions $x \sim \mathcal{N}(\mu, \Sigma)$ with parameter (μ, Σ). At each iteration t, K solutions are sampled from the current distribution $x_i \sim \mathcal{N}(\mu_t, \Sigma_t), 1 \leq i \leq K$. The fitness of each solution x_i is calculated f_i based on the function evaluation (e.g. for maximization, $f_i = f(x_i)$). The top K_e elite solutions $\{x_i^*\}_{i=1}^{K_e}$ are used for updating the distribution parameters

$$\mu_{t+1} = \sum_{i=1}^{K_e} \lambda_i x_i^*, \quad \Sigma_{t+1} = \sum_{i=1}^{K_e} \lambda_i (x_i^* - \mu_t)(x_i^* - \mu_t)^T + \epsilon\mathbb{I}, \tag{2}$$

where the weights can be set as $\lambda_i = 1/K_e$ to assign equal importance to the top K_e solutions. CEM updates (Eqn (2)) can be viewed as re-weighing samples and fitting a new Gaussian to the re-weighted samples (by assigning zero weights to the bottom $K - K_e$ samples and λ_i weights to top K_e samples). The mean updates (Eqn (2)) will shift the distribution center towards a more promising region of the optimization landscape, while the covariance matrix aligns the sampling direction. The diagonal matrix $\epsilon\mathbb{I}$ with small $\epsilon > 0$ properly conditions the full covariance matrix while maintaining a miminal exploration in all directions to prevent premature convergence to local optima. At termination, the algorithm returns the mean parameter μ as the final solution.

3.2 Policy Gradient

For gradient based optimization of the RL objective (Eqn (1)), we aim to construct gradient estimators $\hat{g}_\theta \approx \nabla_\theta J(\pi_\theta)$ and iteratively update the policy $\theta \leftarrow \theta + \alpha\hat{g}_\theta$ with some learning rate $\alpha > 0$. Here the estimators $\hat{g}_\theta$ are called policy gradient estimators. The vanilla policy gradient takes the following form [35]

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim\rho^{\pi_\theta}, a\sim\pi_\theta(\cdot|s)}[Q^{\pi_\theta}(s, a)\nabla_\theta \log \pi_\theta(a|s)], \tag{3}$$

where ρ^{π_θ} is the state visitation distribution induced under π_θ. Because the expectations are over the current policy π_θ, the unbiased sample estimate of Eqn. (3) is called the on-policy gradient estimator. Due to the on-policy nature, a strict implementation of Eqn. (3) would discard all the samples after performing only one gradient update. This is not efficient in practice. Below we introduce an off-policy gradient estimator which allows for re-using off-policy samples.

3.3 Actor-Critic with Experience Replay

ACER [41] propose an off-policy estimator for the policy gradient (Eqn (3)). Assume a behavior policy $\mu(\cdot|s)$ that generates all samples, and let π be the target policy. The off-policy gradient estimator takes the following form

$$\begin{aligned}\hat{g}_\theta^{\text{off}} &= \mathbb{E}_{s\sim\rho^\mu, a\sim\mu(\cdot|s)}[\rho Q_\theta^\pi(s, a)\nabla_\theta \log \pi_\theta(a|s)] \\ &= \mathbb{E}_{s\sim\rho^\mu}\big[\mathbb{E}_{a\sim\mu(\cdot|s)}[\bar{\rho}(s, a)Q^{\pi_\theta}(s, a)\nabla_\theta \log \pi_\theta(a|s)] \\ &\quad + \mathbb{E}_{a\sim\pi(\cdot|s)}[\rho_+(s, a)Q^{\pi_\theta}(s, a)\nabla_\theta \log \pi_\theta(a|s)]\big],\end{aligned} \tag{4}$$

where $\bar{\rho}(s, a) = \min\{c, \pi_\theta(s, a)/\mu(a|s)\}$ is the clipped likelihood ratio between policies for some constant $c > 0$. And $\rho_+(s, a) = [1 - c/\rho(s, a)]_+$ where $[x]_+$ is the positive part of x. Note that the off-policy gradient (Eqn (4)) differs from the on-policy gradient (Eqn (3)) by the state distribution $s \sim \rho^\mu \neq \rho^{\pi_\theta}$. Degris et al. [7] argue using Eqn. (4) as an alternative to Eqn. (3) is justified because the gradient update preserves the globally optimal solution under tabular parameterizations of the policy. The first line of (Eqn (4)) shows that the mismatch between policies $\pi_\theta(\cdot|s)$ and $\mu(\cdot|s)$ are adjusted by the importance sampling ratio $\rho(s, a)$. The second line of (Eqn (4)) can be interpreted as a more carefully designed importance sampling scheme, where the first term controls the variance by clipping $\rho(s, a)$ to $\bar{\rho}(s, a)$; the clipping introduces bias, and the second term corrects for the bias.

In practical implementations, the algorithm parameterizes an action value function $Q_\phi(s, a)$ and a value function $V_\psi(s)$. The

action value function in the first term of (Eqn (4)) is replaced by a recursive off-policy estimate $Q^{\text{ret}}(s,a)$ using retrace [41]

$$Q^{\text{ret}}(s,a) \leftarrow r + \gamma\bar{\rho}(s',a')[Q^{\text{ret}}(s',a') - Q_\phi(s',a')] + V_\psi(s'). \quad (5)$$

The action value function estimate in the second term of (Eqn (4)) is replaced by the critic $Q_\phi(s,a) \approx Q^{\pi_\theta}(s,a)$. This produces the final ACER policy gradient estimator

$$\begin{aligned}\hat{g}_\theta^{\text{acer}} = \mathbb{E}_{s\sim\rho^\mu}\big[&\mathbb{E}_{a\sim\mu(\cdot|s)}[\bar{\rho}(s,a)Q^{\text{ret}}(s,a)\nabla_\theta \log\pi_\theta(a|s)] \\ &+ \mathbb{E}_{a\sim\pi(\cdot|s)}[\rho_+(s,a)Q_\phi(s,a)\nabla_\theta\log\pi_\theta(a|s)]\big]. \quad (6)\end{aligned}$$

Both critics are trained to approximate the on-policy action value function $Q_\phi(s,a) \approx Q^{\pi_\theta}$ and value function $V_\psi(s) \approx V^{\pi_\theta}(s)$. The algorithm also maintains a replay buffer which stores the historical transitions of the policy $D = \{\{s_i, a_i, r_i, s_i'\}\}$. As a result, the behavior policy $\mu(\cdot|s)$ and its visitation distribution ρ^μ are implicitly defined by sampling data from the replay buffer.

4 GUIDING ES WITH OFF-POLICY ACTOR-CRITIC

4.1 Motivations

As discussed before, near on-policy algorithms such as ES and off-policy algorithms have a clear trade-off between sample efficiency and stability. To get the best of both worlds, we make the following interpretation: off-policy algorithms serve as a principled mechanism for evolving the sampled policy parameters in ES. The evolved parameters can then be aggregated into a new population distribution using a conventional ES update.

4.2 Algorithm

We first introduce an algorithm called CEM-ACER. In later sections, we will discuss the intuitions behind the algorithm. The algorithm maintains a Gaussian distribution over policy parameters $\mathcal{N}(\mu,\Sigma)$ as well as a replay buffer $\mathcal{D}$ to store all historical transitions. At each iteration, first sample K sampled policy parameters from the current distribution $\theta_i \sim \mathcal{N}(\mu_t,\Sigma_t), 1 \le i \le K$. Then $K_{\text{off}} \le K$ parameters are updated using an ACER subroutine (**Algorithm 2**) with gradients constructed from a replay buffer $\mathcal{D}$. For convenience, we assume to carry out ACER updates on $\theta_i, 1 \le i \le K_{\text{off}}$, and the resulting updated parameters are denoted $\theta_i', 1 \le i \le K_{\text{off}}$. Finally, we pool all the updated parameters $\theta_i', 1 \le i \le K_{\text{off}}$ and originally sampled parameters $\theta_i, K_{\text{off}}+1 \le i \le K$ together, evaluate their fitness, and perform a CEM update according to (Eqn (2)): $\mu_t \to \mu_{t+1}, \Sigma_t \to \Sigma_{t+1}$. The algorithmic proecedure is summarized in **Algorithm 1**.

When evaluating the fitness of each policy paraneter θ_i, we roll out the corresponding policy π_{θ_i} to generate a trajectory of length T, which produces T transition tuples $\{s_t, a_t, r_t, s_{t+1}'\}_{t=0}^{T-1}$. These transition tuples are stored into the replay buffer $\mathcal{D}$. The Monte-Carlo estimate of return $\hat{R}_i = \sum_{t=0}^{T-1} r_t$ is used as the fitness function of parameter θ_i.

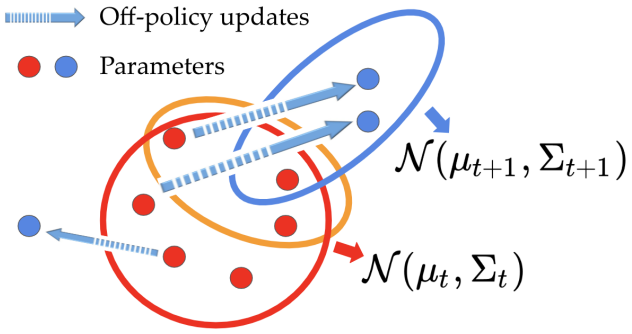


Figure 1: Simplified illustration of the algorithmic procedure of CEM-ACER. Dots represent parameters (e.g. weights of policy networks). The red dots represent parameters at iteration t. The blue arrows show how each parameter gets updated via off-policy subroutines (e.g. ACER), resulting in new parameters as blue dots at iteration $t+1$. The population of red dots form the Gaussian distribution $\mathcal{N}(\mu_t,\Sigma_t)$; the orange contour shows the Gaussian distribution at iteration $t+1$, by performing pure CEM updates on the red dots (i.e. fitting new a Gaussian using high-performing parameters). The blue contour shows the Gaussian distribution at iteration $t+1$, fitted with parameters obtained via off-policy updates. The plot graphically shows that CEM-ACER allows for more aggressive updates $(\mu_t,\Sigma_t) \to (\mu_{t+1},\Sigma_{t+1})$ than CEM, which potentially leads to more efficient learning, as corroborated by the experiment results.

Algorithm 1 CEM-ACER

1: Input: initial distribution parameter μ_0, Σ_0.
2: Initialize iteration counter $t = 0$ and replay buffer $\mathcal{D} = \{\}$
3: **while** forever **do**
4: Sample K sampled policy parameters $\theta_i \sim \mathcal{N}(\mu_t,\Sigma_t)$.
5: Perform off-policy actor-critic updates on $\theta_i, 1 \le i \le K_{\text{off}}$ and return $\theta_i' = \text{ACER}(\theta_i, \mathcal{D})$.
6: Evaluate the fitness of all parameters $\theta_i', 1 \le i \le K_{\text{off}}$ and $\theta_i, K_{\text{off}}+1 \le i \le K$ each using a single rollout $f_i = \hat{R}_i$. The rollout trajectories are stored in $\mathcal{D}$.
7: Use CEM to update μ_{t+1}, Σ_{t+1} according to (Eqn (2)).
8: $t \leftarrow t+1$.
9: **end while**

We briefly introduce the details of the ACER subroutine (**Algorithm 2**). ACER starts with any parameter θ and a replay buffer $\mathcal{D}$. First draw sample tuples from the buffer $(s_i, a_i, r_i, s_i') \sim \mathcal{D}$, then construct off-policy gradient $\hat{g}_\theta^{\text{acer}}$ based on (Eqn (6)). The parameter is updated using T_{off} gradient steps $\theta_i \leftarrow \theta_i + \alpha\hat{g}_\theta^{\text{acer}}$. In the end, the subroutine returns the final θ_i'.

4.3 How does CEM-ACER achieve sample efficiency and learning stability?

We pictorially contrast CEM-ACER with the conventional CEM in Figure 1. The red circle shows the contour of the Gaussian distribution over policy parameters $\mathcal{N}(\mu_t,\Sigma_t)$ at iteration t while the red dots show its samples. We assume that the objective landscape is such that samples at the upper right corner have larger fitness. The

orange contour shows the resulting Gaussian distribution updated via a vanilla CEM. The CEM-ACER randomly selects several sampled policy parameters and update them with off-policy gradients, which transport the red samples into blue samples (the updates are illustrated as blue arrows). As a result of the instability of off-policy updates, it is possible that certain sampled policy parameters have even worse fitness after the update (e.g. the blue sample on the left). The CEM step of CEM-ACER will aggregate only the high-performing sampled policy parameters from the off-policy updates (the two blue samples on the right), which produces $\mathcal{N}(\mu_{t+1}, \Sigma_{t+1})$ shown as the blue contour. This offers a natural to partially address the instability issues induced by off-policy training [34, 40]. In addition, we see that the off-policy updates entail the sampled policy parameters to *jump* over large distances in the parameter space, allowing for more aggressive updates per iteration. On the contrary, CEM only carries out very local update (the orange contour is very close to the red contour, while the blue contour is far away) hence is much less sample efficient.

Algorithm 2 Off-policy Subroutine: ACER

1: Input: parameter θ, replay buffer $\mathcal{D}$
2: Initialize iteration counter $t = 0$
3: **while** $t \leq T_{\text{off}}$ **do**
4: Sample tuples (s_i, a_i, r_i, s_i') from the replay buffer $\mathcal{D}$ and construct off-policy actor-critic gradient $\hat{g}_\theta^{\text{acer}}$ as in (Eqn (6)).
5: Update with gradient $\theta \leftarrow \theta + \alpha \hat{g}_\theta^{\text{acer}}$.
6: $t \leftarrow t + 1$.
7: **end while**
8: Return: θ

Approximate Monotonic Improvement. The CEM-ACER algorithm alternates between CEM updates and ACER off-policy gradient updates. Recall that μ_t is the mean policy parameter of the population distribution at iteration t. We compare the performance of two consecutive policy iterates $J(\pi_{\mu_{t+1}})$ and $J(\pi_{\mu_t})$. The following theorem formalizes the intuition from the last section, and derives the improvement for the policy iterate (see Appendix B for proof)

Theorem 4.1. *Consider a version of the algorithm with a continuum of sampled particles, i.e. $K \to \infty$. Let $\eta_{off} = \frac{K_{off}}{K} \in (0, 1]$ be the proportion of particles being updated via the off-policy subroutine. Assume that the off-policy subroutine provides improvements such that $J(\pi_{\theta_i'}) \geq J(\pi_{\theta_i}) + \delta$ for some improvement lower bound δ. For simplicity, we assume CEM to always fit Gaussian $\mathcal{N}(\mu, \sigma^2 \mathbb{I})$ with a fixed standard deviation σ, and we assume that the distribution over the top $\frac{K_e}{K}$ percent of the elite samples could be perfectly fitted by the distriution $\mathcal{N}(\mu, \sigma^2 \mathbb{I})$ by finding a proper μ. The following holds*

$$J(\pi_{\mu_{t+1}}) \geq J(\pi_{\mu_t}) + \eta_{off}\delta - \sigma^2 M + o(\sigma^2), \tag{7}$$

for some MDP dependent constant $M > 0$. Here $o(x)$ stands for functions of x such that $\lim_{x \to 0} \frac{o(x)}{x} = 0$.

The above theorem assumes that the off-policy subroutine improves the individual policy parameter $J(\pi_{\theta_i'}) \geq J(\pi_{\theta_i}) + \delta$ with guarantee δ. Under some restrictive assumptions on the degree of off-policyness of the data, one could derive more explicit forms of δ following techniques in [31] and more recently [38]. Contrasting Eqn (7) with pure CEM where $\delta = 0$: though the theoretical derivations of δ tend to be conservative, leading to $\delta < 0$ [31, 38], in practice the improvements are generally non-negative $\delta \geq 0$. As a result, CEM-ACER entails enlarged improvements compared to pure ES approaches, with the help of off-policy methods. It is worth noting that other off-policy updates, e.g. Q-learning which minimizes surrogate loss functions such as Bellman errors. This does not directly translate into an improvement in the RL return objective. On the other hand, off-policy actor-critic approximately optimizes $J(\pi_\theta)$ with direct gradient descents and leads to a direct improvement.

Note that a major limitation of the above theorem is that we also assume the solution population (after being updated by off-policy algorithms) can be fit with a Gaussian distribution $\mathcal{N}(\mu, \sigma^2 \mathbb{I})$. This might not be true when the off-policy updates take update parameters to be far away from each other. The resulting population is not uni-modal and cannot be perfectly fit be a Gaussian distribution. Next, we validate the empirical performance of CEM-ACER with extensive experiments.

5 EXPERIMENTS

We aim to address the following questions in the experiments: **(1)** Does CEM-ACER provide performance gains over baseline RL algorithms on benchmark control tasks? Specifically, does CEM-ACER improve upon its building blocks CEM and ACER? **(2)** How sensitive are CEM-ACER to certain important hyper-parameters compared to CEM and ACER?

To address **(1)**, we evaluate CEM-ACER over three sets of benchmark tasks: control tasks with discrete/continuous action space, and with partial observability. To analyze and validate performance gains of CEM-ACER, we compare with the building blocks: CEM and ACER with only off-policy updates. We also compare with ACER with both on/off-policy updates and A2C [23] to assess the importance of on-policy updates. To address **(2)** we carry out ablation study on the hyper-parameters of CEM-ACER.

Implementation Details. Baseline algorithms are all implemented with OpenAI baselines [8] and Tensorflow [1]. The benchmark tasks are simulated in OpenAI gym [3], DeepMind control suites [39] or Roboschool [19]. These tasks include robotics locomotion, simple manipulation and simulated video games as illustrated in Figure 6. We directly implement CEM-ACER on top of the code base of ACER and CEM to ensure meaningful comparison. All gradient based optimizations are carried out with Adam optimizer [18]. In benchmark evaluations below, we set $K = 10$ sampled policy parameters per iteration for CEM-ACER and CEM, and $K_{\text{off}} = 5$ for CEM-ACER. The elite parameter is set to be $K_e = K/2 = 5$. Additional hyper-parameters are specified below and in Appendix A.

5.1 Discrete Action Space

Setup. Though many RL tasks of robotic interest have continuous action space. We convert continuous action spaces into discrete ones by discretizing. To be concrete, given a continuous action

space such as $\mathcal{A} = [-1, 1]^m$, we discretize each dimension into K evenly spaced atomic actions. This results in an action space with K^m joint actions. Despite the explosion in joint action space, prior works have found such simple discretization scheme useful in achieving stable learning of locomotion and manipulation tasks [2, 36]. The policy $\pi_\theta(a|s)$ is parametrized as a categorical distribution and we set $K = 5$. CEM-ACER and ACER are implemented with the best learning rate $\in \{7 \cdot 10^{-3}, 7 \cdot 10^{-4}\}$ and A2C is implemented with the best learning rate $\in \{3 \cdot 10^{-4}, 3 \cdot 10^{-5}\}$.

Results. In Figure 2, we show the learning curves of baseline algorithms across benchmark tasks, where each colored curve corresponds to a different baseline. We make several observations: **(1)** CEM consistently performs poorly. We speculate this is because the underlying policy $\pi_\theta(a|s)$ is stochastic, whose action level noise confounds the parameter level noise of ES methods. This can be shown to lead to higher variance in the parameter updates and worse learning performance [37]; **(2)** ACER with on/off-policy updates are more stable than ACER with only off-policy updates. We speculate that off-policy gradients accumulate bias in the parameter updates, which get periodically mediated by on-policy gradients. However, when on-policy gradients are not available this generally leads to performance bottleneck. This is compatible with results from [9, 10], where they show (off-policy) Q-learning is more stable when on-policy samples are available; **(3)** Most importantly, CEM-ACER significantly outperforms its building components ACER and CEM across almost all tasks. This implies that CEM-ACER combines the benefits of these two algorithms while offsetting their respective drawbacks. Specifically, CEM-ACER makes much more rapid progress in the parameter space than CEM thanks to interwined off-policy updates. On the other hand, unlike ACER with on/off-policy updates, CEM-ACER only carries out off-policy updates. The instability of off-policy updates are alleviated through the proper weighting (Eqn (2)) in the CEM updates - indeed, if a particle solution is stuck at bad performance, CEM updates will eliminate such solutions by assigning them zero weights.

5.2 Continuous Action Space

Setup. We also consider the case where the policy directly outputs continuous actions for the control tasks. Here, the policy $\pi_\theta(a|s) = \mathcal{N}(\mu_\theta(s), \sigma^2)$ is parameterized as a Gaussian distribution with mean $\mu_\theta(s)$ and state-independent variance σ^2. We note that the ACER implementation details differ from the discrete case: indeed, Wang et al. [41] introduce multiple additional techniques to stabilize the training of the continuous policy and require further hyper-parameter optimization. We review such techniques in the Appendix. CEM-ACER and ACER are implemented with the best learning rate $\in 7 \cdot \{10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$ and A2C is implemented with the best learning rate $\in \{3 \cdot 10^{-4}, 3 \cdot 10^{-5}\}$.

Results. We show the learning curves in Figure 3 across multiple continuous control tasks. We make several observations: **(1)** ACER is not quite stable in continuous contexts. Importantly, we note that despite our efforts to stabilize ACER according to the recipe in [41], ACER easily becomes unstable under large learning rates - yet with small learning rates it learns extremely slowly; **(2)** Despite the instability of the underlying ACER subroutine, CEM-ACER is more robust during learning. In this case, CEM-ACER leverages the stabilizing effects of CEM (i.e. automatically eliminating worse-performing parameters) to overcome the unstable learning of ACER. In practice, we find that CEM-ACER performs better with a larger learning rate $\approx 10^{-3}$ in contrast to ACER, where the working learning rate $\approx 10^{-5}$.

5.3 Partially Observable Tasks

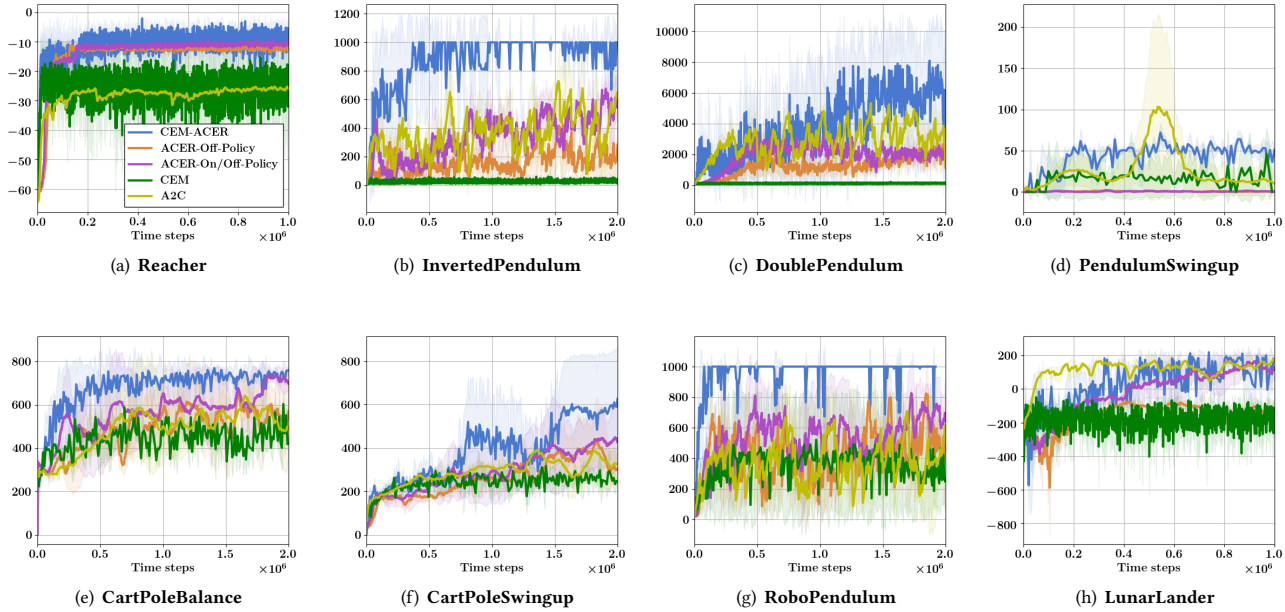
Setup. We modify the original full-observable locomotion tasks [3] into partially observable tasks. The state space of the original tasks contains both generalized position and velocities of robot joints. To make the system partially observable, we remove the generalized velocities from the observation space. To succeed at the task, the agent needs to infer the full state e.g. velocities through a sequence of observations. The policy is parameterized as a LSTM policy $\pi_\theta(a|s, h)$ which takes in the current observation s and a hidden state h to compute the distribution over actions [8, 14]. The hidden state h is also updated based on the new observation s via the LSTM cell. Here the action space the same as Section 5.1. We leave details to the Appendix.

Results. In Figure 4, we show the learning curves of baseline algorithms across partially observable benchmark tasks. We make the following observations from the results: **(1)** Comparing the corresponding fully-observable tasks in Figure 2, algorithms in general learn much more slowly on the partially observable tasks (e.g. compare Figure 2(c) and Figure 4(c). This shows that partially observable tasks are indeed much more challenging than their full observable counterparts - in order to learn, the agent needs to implicitly infer full state information through the recurrent policy. This usually takes place very slowly due to the long range dependencies in the recurrent network [14]; **(2)** Despite slower learning rates, CEM-ACER significantly outperforms the other benchmark algorithms, particularly the individual ACER and CEM. Though CEM by itself does not perform well on partially observable tasks, when augmented with off-policy updates in ACER, the search procedure takes place much more quickly as observed in Figure 4.

5.4 Ablation Study

We carry out ablation study on how various hyper-parameters impact the learning performance. In particular, we study the particle parameter population size K and number of off-policy gradient updates per particle in each iteration T_{off}. We carry out ablation studies by evaluating different hyper-parameter settings on two tasks: Inverted-Pendulum and Double-Pendulum, with a similar setup in Section 5.1.

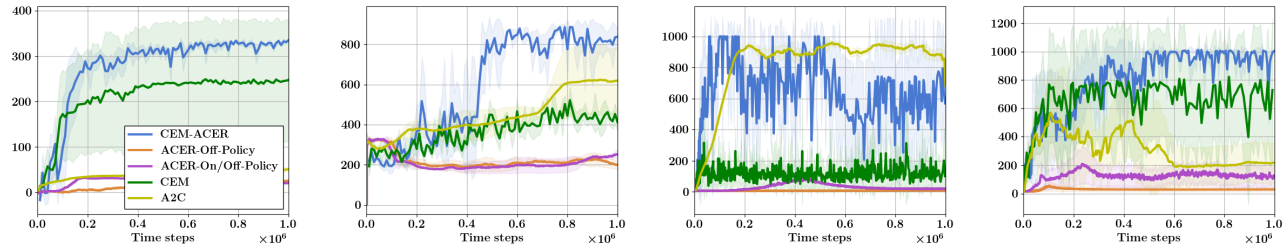
Particle parameter population size. To assess the effect of the population size K, we compare CEM-ACER (solid lines) with vanilla CEM (dashed lines). In general, the dependence of CEM-ACER on K is task-dependent: For Double-Pendulum, the final performance peaks at a population size of $K \approx 10$. We speculate that with K too small, the evolution does not entail sufficient exploration, which causes the algorithm to get stuck at local optima; on the other hand, having very large K tends to consume many more time steps per iteration, leading to worse sample efficiency. For CEM, in our



(a) **Reacher** (b) **InvertedPendulum** (c) **DoublePendulum** (d) **PendulumSwingup**

(e) **CartPoleBalance** (f) **CartPoleSwingup** (g) **RoboPendulum** (h) **LunarLander**

Figure 2: Learning curves on benchmark tasks with discrete action space. Each curve corresponds to a different algorithm (blue: CEM-ACER; purple: ACER with both on/off-policy updates; orange: ACER with both off-policy updates; green: CEM; yellow: A2C). Each curve is averaged cross five random seeds. All tasks are trained for $1 \cdot 10^6 \sim 2 \cdot 10^6$ time steps.



(a) **Swimmer** (b) **CartPole** (c) **InvertedPendulum** (d) **Hopper**

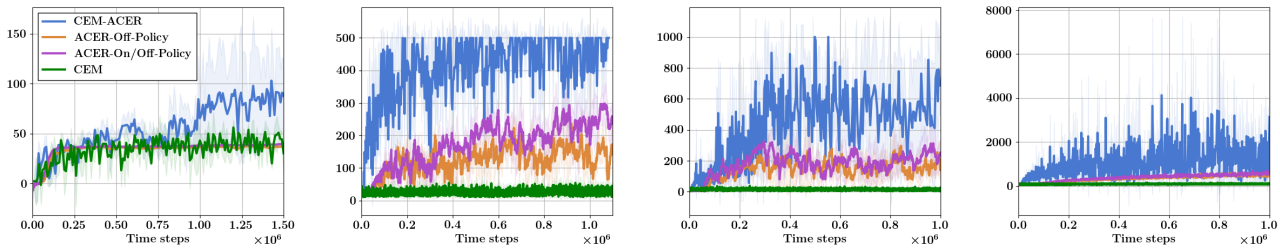
Figure 3: Learning curves on continuous control tasks. Each curve corresponds to a different algorithm (blue: CEM-ACER; purple: ACER with both on/off-policy updates; orange: ACER with both off-policy updates). Each curve is averaged cross five random seeds. All tasks are trained for $1 \cdot 10^6 \sim 2 \cdot 10^6$ time steps.

experiments this does not make too much difference because the algorithm does not learn well.

Number of off-policy gradient updates. For off-policy algorithms, the number of off-policy updates reflect the intensity of sample reuse. To evaluate the impact of the number of off-policy updates per iteration T_{off}, we compare against ACER with only off-policy updates. We see from Figure (5)(b) that for ACER, the dependence on T_{off} is not monotonic: in particular, for InvertedPendulum, the best performance is achieved by setting $T_{\text{off}} \approx 10$. The intuition is that small T_{off} leads to insufficient sample reuse, yet large T_{off} magnifies the instability of off-policy learning. However, for CEM-ACER, increasing T_{off} does not impose such significant instability penalty. We speculate that the evolutionary mechanism guards against unstable off-policy updates and guarantees that the final performance is more robust.

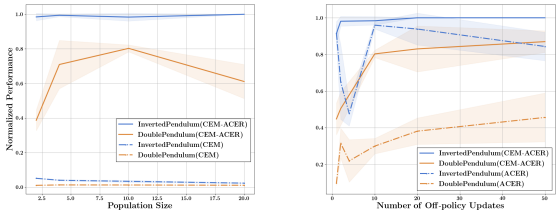
6 CONCLUSION

We have proposed CEM-ACER, a combination of CEM with off-policy actor-critic algorithm ACER. As shown through extensive experiments, CEM-ACER retains both the sample efficiency of off-policy updates in ACER and training stability of on-policy updates as in CEM. Our interpretation of off-policy algorithms as natural mechanisms for evolving sampled policy parameters in ES methods



(a) **Swimmer** (b) **CartPole** (c) **InvertedPendulum** (d) **DoublePendulum**

Figure 4: Learning curves on partially observable benchmark tasks. Each curve corresponds to a different algorithm (blue: CEM-ACER; purple: ACER with both on/off-policy updates; orange: ACER with both off-policy updates). Each curve is averaged cross five random seeds. All tasks are trained for $1 \cdot 10^6 \sim 2 \cdot 10^6$ time steps.



(a) **Population Size** (b) **Number of Updates**

Figure 5: Ablation study on hyper-parameters K and T_{off}.

can be applied to other off-policy settings. We leave as future work its extension to e.g. model-based RL.

A FURTHER EXPERIMENT DETAILS

We provide additional experiment details below. The details consist of the **algorithm setup**, where we introduce implementation details for all baseline algorithms, and **task setup**, where we review and complete the details for benchmark tasks and policy parameterization.

A.1 Algorithm Setup

ACER.. ACER implements a generic gradient estimator as in (Eqn (6)). The clipping constant $c = 10$ for discrete and $c = 5$ for continuous action space. Both the policy π_θ and action value function $Q_\phi(s,a)$ are fully-connected networks with 2 hidden layers with 64 units and tanh non-linear activations. The discount factor for retrace is $\gamma = 0.99$. The value function $V_\psi(s)$ is not explicitly parametrized for discrete action space, because we can approximate $V_\psi(s) = \sum_a \pi_\theta(a|s) Q_\phi(s,a)$. For continuous action space, the value function has 2 hidden layers with 64 units and tanh non-linear activation functions.

The critic functions $Q_\phi(s,a)$ are trained to minimize the square errors against the retrace targets. Wang et al. [41] show that this speeds up the convergence of trained critics.

For continuous action space, there are a few implementation details that deviate from the discrete case. The action value function is parameterized as a Stochastic Dueling Network (SDN) where we only parameterize $V_\phi(s)$ and $A_\phi(s,a)$ as neural networks with similar architecture as above, and produce the action value function as below

$$Q_\phi(s,a) = V_\phi(s) + A_\phi(s,a) - \frac{1}{n}\sum_{i=1}^{n} A_\phi(s,a'),\ a' \sim \pi_\theta(\cdot|s), \quad (8)$$

where $n = 5$. This parameterization decomposes the action value function naturally into a value function $V_\phi(s)$ and an advantage function $A_\phi(s,a)$. In addition, in continuous case we also replace the clipping ratio $\bar{\rho}(s,a) = \min\{c, \frac{\pi_\theta(a|s)}{\mu(a|s)}\}$ in (Eqn (6)) by $\bar{\rho}_d(s,a) = \min\{c, (\frac{\pi_\theta(a|s)}{\mu(a|s)})^{1/d}\}$ where d is set to be the dimension of the action space. This technique is used for smoothing the density ratio in high dimensional action space.

In our benchmark comparison in Section 5.1 to Section 5.3, we use $T_{\text{off}} = 4$ off-policy updates per iteration. The full ACER (ACER with on/off-policy updates in Section 5) also generate one single on-policy gradient update from the collected samples to update the parameter, in practice, this usually makes the learning more stable. In each iteration, we collect $N = 320$ time steps from the environment and save into the buffer. These are default hyper-parameters of the baseline code [8].

CEM.. In our implementation, CEM shares the same policy parameterization as ACER but just without all the critic functions. CEM sets the elite size $K_e = \frac{1}{2}K$ and equal weights across all elite samples. The covariance matrix has a damping parameter ϵ in (Eqn (2)) to maintain the property condition of the matrix. This damping parameter ϵ starts with 10^{-3} and exponentially anneal to 10^{-5}. We borrow the open source implementation from https://github.com/apourchot/CEM-RL.

CEM-ACER.. In our implementation, CEM-ACER by construction shares all the parameterization and implementation techniques as ACER and CEM. In particular, CEM-ACER maintains a distribution over policy parameters $\theta \in \mathcal{N}(\mu, \Sigma)$ for $\pi_\theta(a|s)$, while a single parameter for the critic $Q_\phi(s,a), V_\psi(s)$. Each time a particle policy parameter $\theta_i \sim \mathcal{N}(\mu, \Sigma)$ gets trained using ACER, it gets updated using gradients constructed from the common critic, while the critic is also updated using gradients generated from the policy.

An obvious alternative is to also maintain a distribution over the critic parameter ϕ, ψ and aggregate the critic parameter in an ES manner. However, in practice we find using a single critic parameter works much more stably.

For benchmark evaluation from Section 5.1 to Section 5.3, we set the population size $K = 10$. In each iteration, we collect data from all $K = 10$ sampled policy parameters. Then $K_{\text{off}} = \frac{1}{2}K = 5$ policy parameters get updated using the ACER off-policy gradients. For each of the particle policy parameter, we carry out $\approx KT_{\text{off}}$ updates. This is to ensure that we have similar rate of sample reuse as the original ACER.

A.2 Task Setup.

Most tasks are visually displayed in Figure 6. Please consult the corresponding code base for detailed descriptions of these tasks.

Partially observable tasks. For tasks which are partially observable, i.e. tasks where the observations do not reflect the underlying true states of the control system, we resort to the recurrent policy [14]. Both the policy $\pi_\theta(a|s)$ and critics $Q_\phi(s, a)$ are parameterized as LSTM networks, which maintain a hidden state h per time step. The hidden state h is 256 dimensional and initialized to zeros at the beginning of each episode during execution. At each time step, the hidden state h is updated based on the latest observation s using the LSTM cell $h_t = \text{LSTM}(h_{t-1}, s_t)$. Conceptually, the hidden state h_t summarizes past information and serves as a sufficient statistics to compute the policy and action value function.

In practice, we sample a segment of trajectories from the replay buffer and adopt truncated backprop through time as implemented in [8].

B PROOF OF THEOREM 1

We start with characterizing the return objective $J(\pi_{\mu_t})$. At iteration t, we have a Gaussian distribution $\mathcal{N}(\mu_t, \sigma^2\mathbb{I})$. Let $M = \max_x |\text{Tr}(\nabla_x^2 J(x))|$ be an upper bound on the absolute values of the trace of the Hessian matrix $\nabla_x^2 J(x)$. Then

$$J(\mu_t) \leq \mathbb{E}_{x\sim\mathcal{N}(\mu_t,\sigma^2\mathbb{I})}[J(x)] + \sigma^2 M + o(\sigma^2),$$

and similarly,

$$J(\mu_t) \geq \mathbb{E}_{x\sim\mathcal{N}(\mu_t,\sigma^2\mathbb{I})}[J(x)] - \sigma^2 M + o(\sigma^2).$$

Since we consider a continuum of sampled particles $K \to \infty$, and by construction, η_{off} of all particles get updated via off-policy learning, yielding an improvement of δ. Formally, let θ_i, θ_i' be the updates before and after the off-policy subroutine step, then for any $\theta \sim \mathcal{N}(\mu_t, \sigma^2\mathbb{I})$, with probability η_{off}, $J(\pi_{\theta_i'}) \geq J(\pi_{\theta_i}) + \delta$ and otherwise $J(\pi_{\theta_i'}) = J(\pi_{\theta_i})$. Because by assumption the top $\frac{K_e}{K}$ percent of the elite samples, which are used for fitting $\mathcal{N}(\mu_{t+1}, \sigma^2\mathbb{I})$ could be perfectly fitted by a parametric distribution of the form $\mathcal{N}(\mu, \sigma^2\mathbb{I})$ for some μ, we have

$$\begin{aligned}
J(\pi_{\mu_{t+1}}) &\geq \mathbb{E}_{x\sim\mathcal{N}(\mu_{t+1},\sigma^2\mathbb{I})}[J(x)] - \sigma^2 M + o(\sigma^2) \\
&\geq \mathbb{E}_{\theta'}[J(\theta')] - \sigma^2 M + o(\sigma^2) \\
&\geq \mathbb{E}_{\theta}[J(\theta)] + \eta_{\text{off}}\delta - \sigma^2 M + o(\sigma^2) \\
&= \mathbb{E}_{x\sim\mathcal{N}(\mu_t,\sigma^2\mathbb{I})}[J(x)] + \eta_{\text{off}}\delta - \sigma^2 M + o(\sigma^2) \\
&\geq J(\pi_{\mu_t}) + \eta_{\text{off}}\delta - 2\sigma^2 M + o(\sigma^2).
\end{aligned}$$

The expectations $\mathbb{E}_\theta[\cdot]$ and $\mathbb{E}_{\theta'}[\cdot]$ denotes expectations over the population distribution over the updated parameter θ' and initial parameter θ respectively. In the first and last inequality, we invoked the upper and lower bounds defined previously. In the second inequality, we invoked the assumption that the top $\frac{K_e}{K}$ percent of the elite samples of θ' follows the distribution of $\mathcal{N}(\mu_{t+1}, \sigma^2\mathbb{I})$ and that the selection principle induces the inequality. The third inequality is a result of the monotonic improvement for the updated parameter $\theta \to \theta'$ with $J(\pi_{\theta'}) \geq J(\pi_\theta) + \delta$ with probability η_{off}. This concludes the proof.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.

[2] Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. 2018. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177* (2018).

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[4] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. 2018. Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms. *arXiv preprint arXiv:1802.05054* (2018).

[5] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. 2005. A tutorial on the cross-entropy method. *Annals of operations research* 134, 1 (2005), 19–67.

[6] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197.

[7] Thomas Degris, Martha White, and Richard S Sutton. 2012. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839* (2012).

[8] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. 2017. Openai baselines. *GitHub, GitHub repository* (2017).

[9] Justin Fu, Aviral Kumar, Matthew Soh, and Sergey Levine. 2019. Diagnosing Bottlenecks in Deep Q-learning Algorithms. *arXiv preprint arXiv:1902.10250* (2019).

[10] Scott Fujimoto, David Meger, and Doina Precup. 2018. Off-Policy Deep Reinforcement Learning without Exploration. *arXiv preprint arXiv:1812.02900* (2018).

[11] Scott Fujimoto, Herke van Hoof, and David Meger. 2018. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477* (2018).

[12] Audrunas Gruslys, Will Dabney, Mohammad Gheshlaghi Azar, Bilal Piot, Marc Bellemare, and Remi Munos. 2017. The reactor: A fast and sample-efficient actor-critic agent for reinforcement learning. *arXiv preprint arXiv:1704.04651* (2017).

[13] Shixiang Shane Gu, Timothy Lillicrap, Richard E Turner, Zoubin Ghahramani, Bernhard Schölkopf, and Sergey Levine. 2017. Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. In *Advances in neural information processing systems*. 3846–3855.

[14] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[15] Christian Igel, Nikolaus Hansen, and Stefan Roth. 2007. Covariance matrix adaptation for multi-objective optimization. *Evolutionary computation* 15, 1 (2007), 1–28.

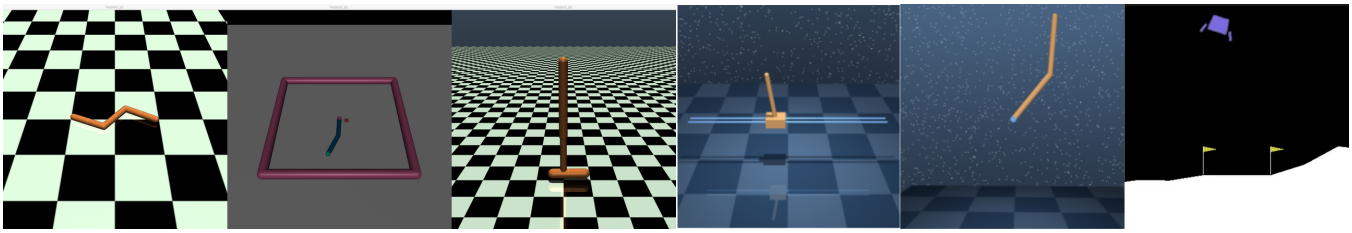


Figure 6: Illustration of Benchmark tasks. Benchmark tasks include robotics locomotion, simple manipulation as well as simulated video games. The state space $\mathcal{S}$ consists of sensor inputs and the action space $\mathcal{A}$ consists of actuators (e.g. torque or position control) applied to the system.

[16] Shauharda Khadka, Somdeb Majumdar, Santiago Miret, Evren Tumer, Tarek Nassar, Zach Dwiel, Yinyin Liu, and Kagan Tumer. 2019. Collaborative evolutionary reinforcement learning. *arXiv preprint arXiv:1905.00976* (2019).

[17] Shauharda Khadka and Kagan Tumer. 2018. Evolution-Guided Policy Gradient in Reinforcement Learning. In *Advances in Neural Information Processing Systems*. 1188–1200.

[18] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[19] Oleg Klimov and J Schulman. 2017. Roboschool.

[20] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. 2016. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research* 17, 1 (2016), 1334–1373.

[21] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[22] Horia Mania, Aurelia Guy, and Benjamin Recht. 2018. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055* (2018).

[23] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*. 1928–1937.

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

[25] Brendan O'Donoghue, Remi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. 2016. Combining policy gradient and Q-learning. *arXiv preprint arXiv:1611.01626* (2016).

[26] Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. 2018. Self-imitation learning. *arXiv preprint arXiv:1806.05635* (2018).

[27] Aloïs Pourchot and Olivier Sigaud. 2018. CEM-RL: Combining evolutionary and gradient-based methods for policy search. *arXiv preprint arXiv:1810.01222* (2018).

[28] Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adria Puigdomenech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. 2017. Neural episodic control. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2827–2836.

[29] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. 2017. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864* (2017).

[30] John Schulman, Xi Chen, and Pieter Abbeel. 2017. Equivalence between policy gradients and soft q-learning. *arXiv preprint arXiv:1704.06440* (2017).

[31] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. 2015. Trust region policy optimization. In *International Conference on Machine Learning*. 1889–1897.

[32] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

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

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

[35] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*. 1057–1063.

[36] Yunhao Tang and Shipra Agrawal. 2019. Discretizing Continuous Action Space for On-Policy Optimization. *arXiv preprint arXiv:1901.10500* (2019).

[37] Yunhao Tang, Krzysztof Choromanski, and Alp Kucukelbir. 2019. Variance Reduction for Evolution Strategies via Structured Control Variates. *arXiv preprint arXiv:1906.08868* (2019).

[38] Yunhao Tang, Michal Valko, and Rémi Munos. 2020. Taylor expansion policy optimization. *arXiv preprint arXiv:2003.06259* (2020).

[39] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. 2018. Deepmind control suite. *arXiv preprint arXiv:1801.00690* (2018).

[40] Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. 2018. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648* (2018).

[41] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. 2016. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224* (2016).