Improving Sample Efficiency in Evolutionary RL Using Off-Policy Ranking

S. R. Eshwar(✉), Shishir Kolathaya, and Gugan Thoppe

Indian Institute of Science, Bangalore, India
{eshwarsr,shishirk,gthoppe}@iisc.ac.in

Abstract. Evolution Strategy (ES) is a potent black-box optimization technique based on natural evolution. A key step in each ES iteration is the ranking of candidate solutions based on some fitness score. In the Reinforcement Learning (RL) context, this step entails evaluating several policies. Presently, this evaluation is done via on-policy approaches: each policy's score is estimated by interacting several times with the environment using that policy. Such ideas lead to wasteful interactions since, once the ranking is done, only the data associated with the top-ranked policies are used for subsequent learning. To improve sample efficiency, we introduce a novel off-policy ranking approach using a local approximation for the fitness function. We demonstrate our idea for two leading ES methods: Augmented Random Search (ARS) and Trust Region Evolution Strategy (TRES). MuJoCo simulations show that, compared to the original methods, our off-policy variants have similar running times for reaching reward thresholds but need only around 70% as much data on average. In fact, in some tasks like HalfCheetah-v3 and Ant-v3, we need just 50% as much data. Notably, our method supports extensive parallelization, enabling our ES variants to be significantly faster than popular non-ES RL methods like TRPO, PPO, and SAC.

Keywords: Reinforcement learning · Evolutionary strategies · Off-policy ranking · ARS · TRES

1 Introduction

In optimization, features of the objective function such as linearity, convexity, or differentiability are often non-existent, unknown, or impossible to detect. An

ESR was supported by the Prime Minister's Research Fellowship (PMRF). SK was supported by the SERB Core Research Grant CRG/2021/008115. GT was supported in part by DST-SERB's Core Research Grant CRG/2021/00833, in part by IISc Start-up grants SG/MHRD-19-0054 and SR/MHRD-19-0040, and in part by the "Pratiksha Trust Young Investigator" award.

E. Kalyvianaki and M. Paolieri (Eds.): VALUETOOLS 2023, LNICST 539, pp. 33–50, 2024.
https://doi.org/10.1007/978-3-031-48885-6_3

Evolution Strategy (ES) [10], due to its derivative-free nature, is a go-to alternative in such scenarios. [17] introduced the first competitive ES method in Reinforcement Learning (RL). However, that technique's efficacy hinges on several intricate concepts and the application of neural networks to parameterize policies. These intricacies encompass converting rewards into rankings and utilizing these rankings for computing updates, partitioning the action space to promote exploration, and adopting policies parameterized by neural networks with virtual batch normalization.

Fortunately, two recent ES methods—the Augmented Random Search (ARS) [13] and Trust Region Evolution Strategy (TRES) [12]—have shown that (deterministic) linear policies and a suitably modified basic random search are often effective enough. In that, they can help to cut down the running times by an order of magnitude. Our work introduces novel off-policy adaptations of ARS and TRES. These adaptations maintain similar computation times as the original techniques but require notably less data (sometimes as little as 50%).

The formal motivation for our work is to explore ES algorithms in RL that minimize agent-environment interactions. Clearly, this would be of significance in practical RL. A vital example is robotics, where collecting samples is very expensive since the process involves active calibration, maintenance, and safety checks for every component, and overlooking these can result in unsafe behaviours.

We now describe how existing ES methods for RL operate and also point out the key step at which they are sample-inefficient. Recall that, in each iteration, a generic ES method i.) obtains a bunch of candidate solutions from some sampling distribution, ii.) ranks them using some fitness score based on the objective function, and iii.) uses the top-ranked ones to update the sampling distribution for use in the next iteration. In the context of RL, a candidate solution is a specific policy, while its fitness is the associated value function. Existing techniques such as ARS and TRES use an on-policy approach to find this fitness score: interact with the environment using every policy whose score needs to be estimated. Since multiple policies need to be ranked in each iteration, current ES approaches end up with significantly high interactions. Notably, most of this data is discarded in each iteration except those related to the top-ranked policies.

Our proposed idea to improve sample efficiency is to replace the above wasteful ranking approach with an off-policy alternative. Our new approach involves two key ideas: the fitness function choice, and the use of a kernel approximation to estimate the same. Their details are as follows.

1. **Fitness Function**: Instead of using the value function $\eta(\tilde{\pi})$ of a candidate policy $\tilde{\pi}$ as its fitness score, we use its approximation $L_\pi(\tilde{\pi})$ defined in terms of a different policy π called the behavior policy [21, (6)]. Indeed, any policy ranking based on a value-function approximation is sub-optimal. However, it is also the key factor enabling our off-policy ranking. The data generated by the *single* policy π can now be used to rank all the candidate policies!
2. **Kernel Approximation**: $L_\pi(\tilde{\pi})$ is estimated in [21] via importance sampling (see (40) in ibid). This idea works only when both π and $\tilde{\pi}$ are stochastic. However, ARS's sample efficiency crucially relies on the candidate policies

being deterministic. To circumvent this deterministic-stochastic gap, we propose to alternatively smooth the deterministic policies using a suitable kernel function. This approach is loosely inspired from [7,8], which extends ideas from discrete contextual bandit settings to the continuous case.

Key Contributions: The main highlights of this work are as follows.

1. We propose novel variants of ARS and TRES; see Algorithms 1 and [3, Algorithm A1]. Their main feature is off-policy ranking: data from one behavior policy is used to identify the best among multiple candidate policies. We emphasize that this is the first usage of off-policy ranking in any ES method. Moreover, our idea is extremely simple to implement and directly applies to deterministic policies. Also, while we use this idea for ARS and TRES, it is extendable to other ES methods.
2. Our simulations on MuJoCo tasks (Sect. 4) show that our variants reach reward thresholds in running times comparable to the original ARS and TRES. Notably, we often need just 50–80% as much data. This is significant when enivornmental interactions are hard or expensive, e.g., robotics.
3. We also do sensitivity to seed and hyperparameter tests similar to [13]. Our results are similar in spirit to what was obtained in ibid. That is, the median trajectory crosses the reward threshold in most environments, confirming that our algorithm is robust to seed and hyperparameter choices. This contrasts behaviors of non-ES methods, e.g., DDPG [11], TRPO [18], PPO [19].

2 Preliminaries

Here, we describe our RL setup and provide an overview of the original ARS and TRES algorithms.

2.1 RL Setup and Important Definitions

The primary goal in RL is to estimate a control policy that empowers an agent to effectively perform a task involving sequential decision making. Formally, our setup is that of an MDP. At each time-step t, the agent notes the current state of its environment as $s_t \in \mathcal{S}$ (state space), and then executes an action $a_t \in \mathcal{A}$ (action space) sampled from its current policy $(\pi_t(a|s_t))_{a\in\mathcal{A}}$ (which is a distribution). The environment then transitions to a new state $s_{t+1} \sim \mathbb{P}(s'|s,a)$, where $\mathbb{P}$ is the transition function of the MDP, and the agent receives an associated scalar reward $r_t = r(s_t, a_t)$. The aforementioned goal then is to find a policy that has the largest average reward, i.e., one that solves

$$\max_{\pi} \eta(\pi), \tag{1}$$

where

$$\eta(\pi) := \lim_{H\to\infty} \frac{1}{H} \mathbb{E}_{\tau\sim P_\pi(\tau)} \left[\hat{\eta}(\pi)\right].$$

In the above relation, $\hat{\eta}(\pi) \equiv \hat{\eta}_H(\pi, \tau) = \sum_{t=0}^{H-1} r(s_t, a_t)$ and $P_\pi(\tau)$ is the likelihood of a trajectory $\tau = \{(s_0, a_0, r_0), (s_1, a_1, r_1), ...\}$ under the policy π.

Note that the sequence of states observed under a fixed policy π forms a Markov chain. If this chain has a stationary distribution d_π and

$$d_\pi(s) = \lim_{H\to\infty} \frac{1}{H} \sum_{t=0}^{H-1} \mathbb{P}_{\tau\sim P_\pi(\tau)}(s_t = s), \tag{2}$$

then $\eta(\pi) = \mathbb{E}_{s\sim d_\pi, a\sim\pi}[r(s, a)]$.

In our later discussions, we will also be using some terms related to the value function such as the state-bias function (V_π), action-bias function (Q_π), and the advantage function (A_π). These are given by

$$V_\pi(s) := \mathbb{E}_{\tau\sim P_\pi(\tau)} \left[\sum_{t=0}^{\infty} (r(s_t, a_t) - \eta(\pi)) | s_0 = s \right]$$

$$Q_\pi(s, a) := \mathbb{E}_{\tau\sim P_\pi(\tau)} \left[\sum_{t=0}^{\infty} (r(s_t, a_t) - \eta(\pi)) | s_0 = s, a_0 = a \right]$$

and $A_\pi(s, a) := Q_\pi(s, a) - V_\pi(s)$, respectively.

2.2 Review of ARS and TRES

ARS belongs to a family of iterative black-box optimization methods called random search [14]. Basic Random Search (BRS) is the simplest member of this family and is also where the origins of ARS lie. For ease of exposition, we first explain BRS's approach to solving (1). Throughout this subsection, we restrict our attention to finite-horizon MDPs where the search space is some parameterized family of policies. Note that this is often the case in practical RL and is also what we deal with in our MuJoCo simulations.

For the above setup, by fixing a large H, the problem in (1) translates to

$$\max_\theta \mathbb{E}_\tau[\hat{\eta}(\pi_\theta)] \equiv \max_\theta \mathbb{E}_\tau[\hat{\eta}_H(\pi_\theta, \tau)]. \tag{3}$$

In general, this objective function need not be smooth. To circumvent this issue, BRS looks at its smooth variant and then uses the idea of a stochastic gradient ascent. Specifically, the alternative objective considered is $\mathbb{E}_\delta \mathbb{E}_\tau[\hat{\eta}(\pi_{\theta+\nu\delta})]$, where ν is a suitably fixed scalar, and δ is a random perturbation made up of i.i.d. standard Gaussian entries. Further, in each iteration, the gradient of this function is estimated via a finite-difference method. That is, N random directions $\delta_1, \ldots, \delta_N$ are first generated in the parameter space, $\hat{\eta}(\pi_{\theta+\nu\delta_k})$ and $\hat{\eta}(\pi_{\theta-\nu\delta_k})$ are then estimated by interacting with the environment using $\pi_{\theta+\nu\delta_k}$ and $\pi_{\theta-\nu\delta_k}$ for each k, and finally $\frac{1}{N}\sum_{k=1}^{N}[\hat{\eta}(\pi_{\theta+\nu\delta_k}) - \hat{\eta}(\pi_{\theta-\nu\delta_k})]\delta_k$ is used as a proxy for the gradient at θ. BRS's update rule, thus, has the form $\theta_{j+1} = \theta_j + \frac{\alpha}{N}\sum_{k=1}^{N}\left[\hat{\eta}(\pi_{\theta_j+\nu\delta_k}) - \hat{\eta}(\pi_{\theta_j-\nu\delta_k})\right]\delta_k$ for some parameter $\alpha > 0$.

Mania et al. [13] developed ARS by making the following changes to BRS. To begin with, they restricted the search space to a class of deterministic and linearly parameterized policies: a policy now is represented by a matrix M and the vector Ms denotes the deterministic action to be taken at state s under that policy. Further, three modifications were made to the update rule of BRS. The first was to scale the gradient estimate by the standard deviation of the $\hat{\eta}$ values; this yields the ARS-V1 algorithm. The second was to normalize the states, given as input to the policies, so that all state vector components are given equal importance; this yields ARS-V2. The final modification was to pick some $b < N$ and use only the b best-performing search directions for estimating the gradient in each iteration. The first and the third step yield the ARS-V1t algorithm, while the combination of all three gives ARS-V2t. The ARS-V2t variant is of most interest to us and its update rule has the form $M_{j+1} = M_j + \frac{\alpha}{b\sigma_R}\sum_{k=1}^{b}[\hat{\eta}(\pi_{j,(k),+}) - \hat{\eta}(\pi_{j,(k),-})]\delta_{(k)}$, where σ_R is the standard deviation of the 2b $\hat{\eta}$ values, $\delta_{(k)}$ denotes the k-th largest direction decided based on the value of $\max\{\hat{\eta}(\pi_{j,k,+}), \hat{\eta}(\pi_{j,k,-})\}$ for different k choices, and $\pi_{j,k,+}(s) = (M_j + \nu\delta_k)\text{diag}(\Sigma_j)^{-1/2}(s - \mu_j)$ and $\pi_{j,k,-}(s) = (M_j - \nu\delta_k)\text{diag}(\Sigma_j)^{-1/2}(s - \mu_j)$ with μ_j and Σ_j being the mean and covariance of the $2bHj$ states encountered from the start of the training.

The reason for focusing on ARS-V2t is that, in MuJoCo tasks, it typically outperforms the other ARS variants and also the previous state-of-the-art approaches such as TRPO [18], PPO [19], DDPG [11], the ES method from [17], and the Natural Gradient method from [16]. This shows that normalization of states, and then ranking and only picking the best directions for updating parameters often helps in improving the sample efficiency.

Nevertheless, in each iteration, ARS uses an on-policy technique to estimate $\hat{\eta}(\pi_{j,k,+})$ and $\hat{\eta}(\pi_{j,k,-})$ for each k so that the b best-performing directions can be identified. Because of this, we claim that ARS still does more interactions with the environment than what is needed. Also, in each iteration, it discards data that do not correspond to the top-ranked policies.

TRES [12] is another state-of-the-art ES method that has been shown to outperform methods such as TRPO, PPO, and the ES method from [17]. While broadly similar, the major differences between TRES and ARS is in the choice of the objective function. Instead of using a simple sum of rewards on a single trajectory as in (3), TRES uses a novel local approximation ([12, (25)]) to the value function that matches the latter up to the first-order. The main advantage of this alternative choice is that it guarantees monotonic improvement in successive policy estimates. Also, this new objective function enables TRES to use the same data from top performing directions to update the parameters multiple times in the same iteration. This is not possible with the objective function used in ARS, and is claimed to be core reason for improving sample efficiency. A detailed review of TRES is given in [3, Appendix B].

Algorithm 1 Off-policy ARS

Setup: State space $\mathbb{R}^n$, Action Space $\mathbb{R}^p$

Hyperparameters: step-size α, number of directions sampled per iteration N, standard deviation of the exploration noise ν, number of top-performing directions to use b, bandwidth to use for kernel approximation h, number of behaviour policy trajectories to run n_b

Initialize: $M_0 = \mathbf{0} \in \mathbb{R}^{p\times n}$, $\mu_0 = \mathbf{0} \in \mathbb{R}^n$, $\Sigma_0 = \mathbf{I}_n \in \mathbb{R}^{n\times n}$ (identity matrix), $j = 0$

while ending condition not satisfied **do**

Sample $\delta_1, \delta_2, \ldots, \delta_N$ in $\mathbb{R}^{p\times n}$ with i.i.d. Standard Gaussian entries

Run n_b trajectories using policy parameterized by M_j, resulting in N_d interactions

Sort the directions δ_k based on $f_{\pi_j}(\delta_k, h)$ scores (using (7)), denote by $\delta_{(k)}$ the k-th largest direction, and by $\pi_{j,(k),+}$ and $\pi_{j,(k),-}$ the corresponding policies

Collect $2b$ rollouts of horizon H and their corresponding return ($\hat{\eta}(\cdot)$) using the $2b$ policies of the b best direction

$$\pi_{j,(k),+}(s) = (M_j + \nu\delta_{(k)})\mathrm{diag}(\Sigma_j)^{-1/2}(s - \mu_j)$$

$$\pi_{j,(k),-}(s) = (M_j - \nu\delta_{(k)})\mathrm{diag}(\Sigma_j)^{-1/2}(s - \mu_j)$$

Make the update step:

$$M_{j+1} = M_j + \frac{\alpha}{b\sigma_R}\sum_{k=1}^{b}[\hat{\eta}(\pi_{j,(k),+}) - \hat{\eta}(\pi_{j,(k),-})]\delta_{(k)},$$

where σ_R is the standard deviation of $2b$ returns used in the update step.

Set μ_{j+1}, Σ_{j+1} to be the mean and covariance of the $2bH(j+1)$ states encountered since the start of the training

$j \leftarrow j + 1$

end while

3 Off-Policy ARS and TRES

Here, we provide a detailed description of our proposed approach to improve upon the wasteful on-policy ranking step in ARS. We use the same idea also to improve upon TRES, but leave these details to [3, Appendix C].

Intuitively, in each iteration of our ARS variant, we plan to identify a suitable deterministic policy, interact with the environment using just this *single* policy, and then use the resultant data to rank the $2N$ deterministic policies $\{\pi_{j,k,+}, \pi_{j,k,-} : 1 \le k \le N\}$. As a first step, we come up with a way to approximate the value function of a deterministic policy $\tilde{\pi}$ in terms of another deterministic policy π. We focus on deterministic policies here since ARS's performance crucially depends on this determinism.

If π and $\tilde{\pi}$ were stochastic in nature, then such an approximation has already been given in [21], which itself is inspired from similar estimates given in [6] and [18]. We now discuss the derivation of this approximation.

Consider the average reward RL setup described in Sect. 2.1. Suppose that, for every stationary policy π, the induced Markov chain is irreducible and aperiodic and, hence, has a stationary distribution d_π satisfying (2). In this framework, [21, Lemma 1] showed that the value functions of the two stochastic policies π and $\tilde{\pi}$ satisfy

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s\sim d_{\tilde{\pi}}, a\sim\tilde{\pi}}\left[A_\pi(s,a)\right]. \tag{4}$$

Given this relation, a natural question to ask is whether $\eta(\tilde{\pi})$ can be estimated using just the data collected by interacting with the environment using π. The answer is no, mainly because the expectation on the RHS is with respect to the states being drawn from $d_{\tilde{\pi}}$. In general, this distribution is unknown a priori and is also hard to estimate unless you interact with the environment with $\tilde{\pi}$ itself.

Inspired by [6] and [18], [21] proposed using

$$L_\pi(\tilde{\pi}) := \eta(\pi) + \mathbb{E}_{s\sim d_\pi, a\sim\tilde{\pi}}\left[A_\pi(s,a)\right] \tag{5}$$

as a proxy for the RHS in (4) to overcome the above issue. Two notable benefits arise from the aforementioned approximation. The first is that $L_\pi(\tilde{\pi})$, since it uses d_π instead of $d_{\tilde{\pi}}$, can be estimated from only environmental interactions involving π. Second, and importantly, [21, Lemmas 2, 3] showed that $|L_\pi(\tilde{\pi}) - \eta(\tilde{\pi})|$ is bounded by the total variation distance between π and $\tilde{\pi}$. Thus, when π and $\tilde{\pi}$ are sufficiently close, an estimate for $L_\pi(\tilde{\pi})$ is also one for $\eta(\tilde{\pi})$. Hence, $L_\pi(\tilde{\pi})$ paves the way for estimating $\eta(\tilde{\pi})$ in an off-policy fashion, i.e., using data from a different policy π. Henceforth, we refer to the policy chosen for interaction (e.g., π above) as the behavior policy and the one whose value needs to be estimated (e.g., $\tilde{\pi}$ above) as the target policy.

We now extend the above idea to the case with deterministic policies, which we emphasize is one of our main contributions. While the idea may look simple on paper, the actual extension is not at all straightforward. The key issue is that, in the case of stochastic policies, the idea of importance sampling and, in particular, the relation

$\mathbb{E}_{s\sim d_\pi, a\sim\tilde{\pi}}\left[A_\pi(s,a)\right] = \mathbb{E}_{s\sim d_\pi, a\sim\pi}\left[\frac{\tilde{\pi}(a|s)}{\pi(a|s)}A_\pi(s,a)\right]$ is used for estimating the second term in (5). However, for deterministic policies, the ratio $\tilde{\pi}(a|s)/\pi(a|s)$ will typically be 0, which means the estimate for the second term will also almost always be zero. Hence, this importance sampling idea for estimating $L_\pi(\tilde{\pi})$ fails for deterministic policies.

The alternative we propose is to modify the definition of $L_\pi(\tilde{\pi})$ so that it becomes useful even for deterministic policies. Specifically, we redefine $L_\pi(\tilde{\pi})$ as

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s\sim d_\pi, a\sim\pi}\left[\frac{K_h(\|a-\tilde{\pi}(s)\|)}{K_h(\|a-\pi(s)\|)}A_\pi(s,a)\right], \tag{6}$$

where $K_h(u) = h^{-1}K(u/h)$ and $K : \mathbb{R} \to \mathbb{R}$ denotes a suitably chosen kernel function satisfying $\int K(u)\mathrm{d}u = 1$ and $\int uK(u)\mathrm{d}u = 0$. This approach is loosely inspired from [7,8] which look at extending policy evaluation and control algorithms from discrete contextual bandit settings to the continuous case.

While there are multiple choices for K, we use $K(u) = e^{-u^2}$ in this work. Substituting this definition in (6) gives

Table 1. Comparison of median number of interactions for ARS, TRES, OP-ARS and OP-TRES on MuJoCo locomotion tasks to achieve prescribed reward thresholds (Th). Intx represents the number of interactions in order of 10^3. % column represents the percentage of data required by our method compared to the original method. The * in Ant-v3 signifies the interleaving of on-policy evaluations. The # signifies 16 seeds were used instead of 8 in Walker2d environment for ARS and OP-ARS. We use the following abbreviations in the table below: SW = Swimmer, HO = Hopper, HC = HalfCheetah, WA = Walker2d, AN = Ant, HU = Humanoid

Env	Th	ARS			OP-ARS		TRES			OP-TRES	
		N	b	Intx	Intx	%	N	b	Intx	Intx	%
SW	325	2	1	520 (580)	**440**	85	4	2	800 (560)	800	100
HO	3120	8	4	883 (1098)	**765**	86	–	–	–	–	–
HC	3430	32	4	4480 (3840)	2400	53	16	8	2720 (2400)	**1275**	46
WA$^{\#}$	4390	40	30	18802 (23151)	**14414**	76	–	–	–	–	–
AN	3580	60	20	10492 (17711)	15071*	143	40	20	10849 (10409)	**6165**	56
HU	6000	350	230	40852 (23594)	**14260**	35	–	–	–	–	–

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s\sim d_\pi}\left[e^{-\|\pi(s)-\tilde{\pi}(s)\|^2/h^2} A_\pi(s,\pi(s))\right]$$

The reason for the above choice of K is that it performs quite well in simulations and also provides a clean intuitive explanation for $L_\pi(\tilde{\pi})$. That is, $L_\pi(\tilde{\pi})$ assigns a higher value to a policy $\tilde{\pi}$ if it takes actions similar to π at all those states s where $A_\pi(s,\pi(s))$ is large. In summary, $L_\pi(\tilde{\pi})$ given above provides us with the desired expression to approximate the value function of a deterministic target policy with only the data from a deterministic behavior policy.

We now discuss incorporating this expression in ARS to improve its sample efficiency. In particular, we now show how we can rank the $2N$ policies $\{\pi_{j,k,+}, \pi_{j,k,-}\}$, generated randomly in each iteration of ARS, in an off-policy fashion. The first thing we need to decide is the choice of the behavior policy for interacting with the environment. Recall from the discussion below (5) that $|\eta(\tilde{\pi}) - L_\pi(\tilde{\pi})|$ is small when π and $\tilde{\pi}$ are sufficiently close. Now, since the policy parameterized by M_j is close to each of the $2N$ policies specified above, it is the natural choice for the behavior policy and, indeed, this is what we use.

In summary, our ranking in each iteration works as follows:

1. Interact with the environment using the behavior policy $\pi_j \equiv \pi_{M_j}$ on n_b number of trajectories. Each trajectory here is presumed to have H many time steps (or less in case of premature termination). Suppose these interactions result in N_d (s_t, a_t, r_t, s_{t+1}) transitions overall.
2. Estimate $Q_{\pi_j}(s_t, a_t)$, $0 \le t \le N_d - 1$, using the definition given in Sect. 2.1.

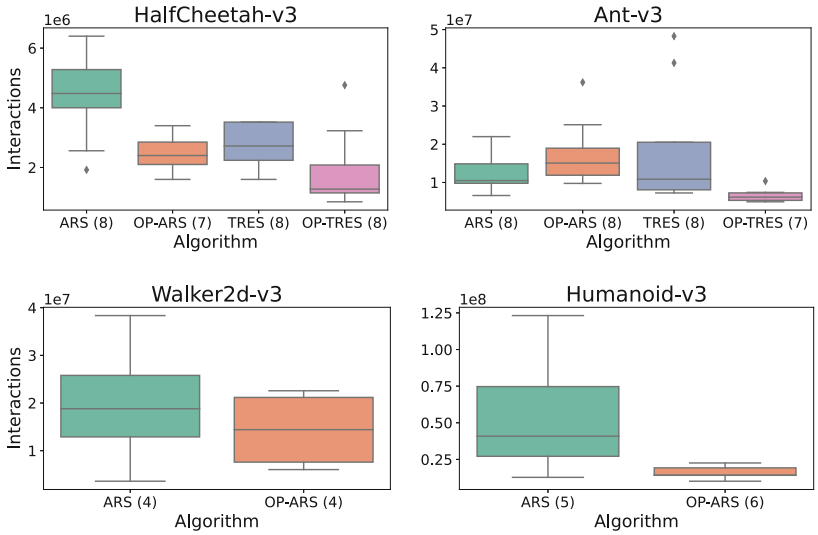


Fig. 1. Box plots of number of interactions required to reach the reward threshold in ARS, TRES, OP-ARS and OP-TRES. The number next to the algorithm's name represents the number of seeds in which the threshold was reached.

3. Estimate

$$f_{\pi_j}(\delta_k, h) = \mathbb{E}[e^{-\|\nu\delta_k s\|^2/h^2} Q_{\pi_j}(s,a)] \approx \frac{1}{N_d} \sum_{t=0}^{N_d-1} [e^{-\|\nu\delta_k s\|^2/h^2} Q_{\pi_j}(s_t, a_t)] \tag{7}$$

for each $1 \leq k \leq N$. Note that $f_{\pi_j}(\delta_k, h)$ is a proxy for the expression in (6). In that, it ignores all the constant terms: those that depend only on the behavior policy π_j.

4. Use the above estimates to rank $\{\pi_{j,k,+}, \pi_{j,k,-}\}$.

Once the b best–performing directions are identified, the rest of our ARS variant more or less proceeds as the original. That is, we come up with better estimates of the value-functions of these top policies in an on-policy fashion and improve upon M_j along the lines discussed in Sect. 2.2. The complete details are given in Algorithm 1. A detailed section discussing the differences in the original ARS from our off-policy variant is given in Appendix A.

We end our discussion here by pointing out that the original ARS used $2N$ interaction trajectories, each of length roughly H, in each iteration. In our variant, we only need $2b + n_b$ many trajectories. When $b < N$, this difference is what leads to the significant reduction in interactions seen in our simulations.

4 Experiments

We compare ARS and TRES with our proposed off-policy variants on benchmark MuJoCo [20] tasks available in OpenAI gym [1]. We show that these variants (henceforth, called OP-ARS and OP-TRES) i.) take significantly less number of interactions to reach rewards thresholds, ii.) are robust to random seeds and hyperparameter choices, and iii.) have very low running times. Separately, in Appendix C, we show that OP-ARS outperforms ARS even on Linear Quadratic Regulator. Finally, we discuss some limitations of our proposed approach.

Expt. 1 (Sample efficiency): Because an ES method is stochastic, the intermediate policies that are learned will be different on each run. Hence, to have a robust comparison, we initiate all random number generators and OpenAI gym environments as a function of a single seed, and then study the performance of our algorithms for eight uniformly random seeds. Further, after every ten iterations of our Algorithm, we compute the value function of the current policy (e.g., the policy parameterized by M_j in Algorithm 1) by averaging the total reward obtained over 100 trajectories.

The sample complexity of our ES methods on each of the eight seeds is the first time the value function of the learned policy is above a certain reward threshold. We use the same thresholds that were used in [13]. The comparison of sample complexity estimates for ARS, TRES, OP-ARS, and OP-TRES is given in Table 1, Figs. 1, and 2.

In Table 1, the 'Env' and 'Th' columns represent the environment names and its corresponding thresholds. The median of sample complexity estimates is provided under the column titled 'Intx' for different N and b choices. In particular, the values in round brackets are for the case where b random directions are chosen and data from all the b directions is used for updating the policy parameter. The ones outside are for the case where N directions are chosen and data from only the top performing b directions is used for parameter update. Finally, the numbers under % denote the percentage of data required by OP-ARS (resp. OP-TRES) to reach the threshold compared to ARS (resp. TRES). Clearly, the median estimates (see bold text in Table 1) for our variants are significantly lower than those of ARS and TRES.

In Table 1, observe that there are two scenarios: either sampling only b directions and using all of them for parameter update is better (see HalfCheetah and Humanoid in ARS, and HalfCheetah and Ant in TRES) or sampling N directions and then picking the top-performing b directions is better (e.g., Swimmer, Hopper, Walker2d, and Ant in ARS). The former scenario corresponds to the case when the overhead of evaluating the additional $N - b$ directions is costlier than the benefit of exploring more directions and using the best performing ones. In both scenarios, by circumventing this overhead issue via off-policy ranking, OP-ARS and OP-TRES significantly cuts down on sample complexity. Note that Humanoid-v3 is considered the most challenging MuJoCo environment.

While Table 1 shows that the median of the timesteps needed for the first crossover happens much earlier in our off-policy variants, it doesn't fully capture

the variance. We address this in Figs. 1 and 2. Figure 1 consists of box plots of the number of interactions required to reach the threshold in ARS, TRES, OP-ARS and OP-TRES. The number of seeds on which each method reached the threshold is mentioned next to the algorithm's name. Clearly, the variance in our approach is either less or comparable to original algorithms. The advantage is particularly significant for OP-TRES in Ant-v3 and for OP-ARS in Humanoid-v3.

In Fig. 2, the overall progress of various algorithms is shown for different seeds. In particular, the horizontal green dotted lines represent the specific reward thresholds, while the different curves correspond to different seeds. The blue and red dots on the curves represent the first timestep where the threshold is reached. Finally, the star represents the median of all the dots marked over various runs. The plots clearly show that our methods reach the threshold significantly faster than the original methods in most cases.

Due to space constraints we have represented the figures only from few environments, the figures of all the remaining environments can be found in [3, Appendix I]. The details about the environment is given in [3, Appendix D] and the implementation details are given in [3, Appendix E].

Expt. 2 (Random Seeds and Hyperparameter choices): The top row of Fig. 3 shows the performance of OP-ARS over 100 random seeds sampled from $[0, 10000]$. We see that the median (thick blue line) crosses the threshold in most of the environments, demonstrating robustness to seeds.

Next, we discuss the sensitivity of OP-ARS to the two new hyperparameters we introduce: h (the bandwidth choice in kernel approximation) and n_b (the number of trajectories generated using the behaviour policy). For this, we first identify multiple choices for h and n_b, wherein the performance of the proposed algorithms is reasonable (see Appendix B). Next, we run our algorithm under all possible combinations of these hyperparameter choices. The performance graphs (refer to the lower row of Fig. 3) align with those observed in the 100-seed test conducted earlier, thus confirming that the algorithm's performance remains largely unaffected by hyperparameter selections. This implies that the specific values of hyperparameters do not significantly impact the process of learning. This is in sharp contrast to behaviors of deep reinforcement learning methodologies which often display high sensitivity to hyperparameters; e.g., in DDPG, slight adjustments often disrupts the learning process [2,4,5]. We don't look at the sensitivity of OP-ARS to other hyperparameters such as ν, α, N, b. Such a study has been done in [13] for ARS and the role of these parameters in both ARS and our off-policy variant are similar.

Expt. 3 (Comparison to non-ES methods): Table 2 shows the quality of the policy obtained after a fixed number of environmental interactions under various RL methods. While the outputs among ES methods are always comparable, there are certain environments, e.g., HalfCheetah and Humanoid, where non-ES methods such as SAC and/or DDPG truly outperform others. The key reason is

Table 2. The maximum average policy return after running the algorithms for 1 million timesteps (2 million for Humanoid-v3). The abbreviations SW, ..., HU have similar expansions as in Table 1.

Algorithm	SW	HO	HC	WA	AN	HU
ARS	358	3308	3147	2481	1246	657
OP-ARS	345	3308	2890	1759	1084	881
TRES	346	–	2713	–	1310	–
OP-TRES	344	–	3559	–	1208	–
TRPO	367	3753	4838	5346	4568	4329
PPO	367	1018	6682	4368	3909	1153
DDPG	42	2775	12025	3928	721	263
A2C	201	1028	2885	395	-38	472
SAC	353	3644	11434	4952	5588	6949

Table 3. Wall clock time required to reach the standard threshold (see Table 1). HU 5% means 5% of the standard threshold for HU. The notation − means that the corresponding thresholds were not reached.

Algorithm	SW	HO	HC	WA	AN	≈ HU				
						5%	10%	15%	25%	100%
ARS	40	48	42	364	322	30	112	454	696	1050
OP-ARS	52	80	110	366	665	34	79	205	400	625
TRES	32	–	37	–	581	–	–	–	–	–
OP-TRES	63	–	37	–	533	–	–	–	–	–
TRPO	267	1020	878	1656	–	357	2412	3789	3691	–
PPO	1658	–	1007	4390	5616	76	5803	24170	17106	–
DDPG	–	5300	349	6434	–	7	–	–	–	–
SAC	14692	6457	672	7704	10754	281	1374	1999	6293	24893

that these non-ES methods have been meticulously refined to extract the maximum information from a provided dataset. However, non-ES methods offer very little scope for parallelization. Accordingly, as Table 3 shows, the wall-clock times recorded to achieve different thresholds are remarkably high for non-ES methods (even when executed on GPUs) compared to our methods (which are executed on CPUs). In summary, our methods maintain comparable run times to ARS and TRES, but provide substantial advantage in terms of sample complexity.

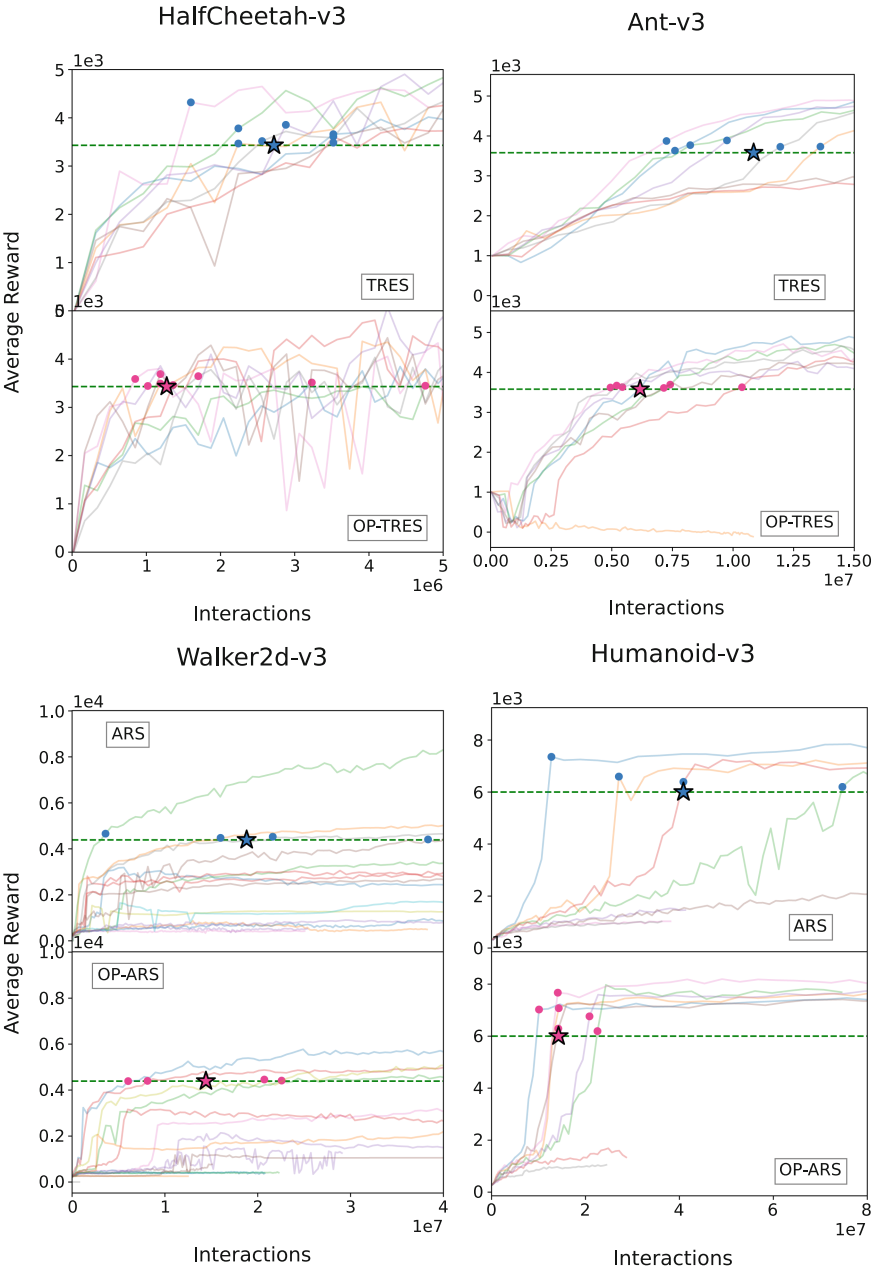


Fig. 2. Figures representing the trajectories of various runs of ARS, OP-ARS, TRES and OP-TRES algorithms where the number of interactions with the environment is plotted against the average reward.

5 Conclusion

This work proposes an off-policy ranking idea for improving sample efficiency in evolutionary RL methods. While traditional off-policy methods are not directly applicable to deterministic policies, we enable it using kernel approximations. Our experiments show that our proposed ARS and TRES variants have roughly the same run time as the original, but reach reward thresholds with only 50-80% as much interactions. We believe our approach is easily extendable to other ES or Random Search methods as well. A promising direction of future work would be to investigate the same theoretically.

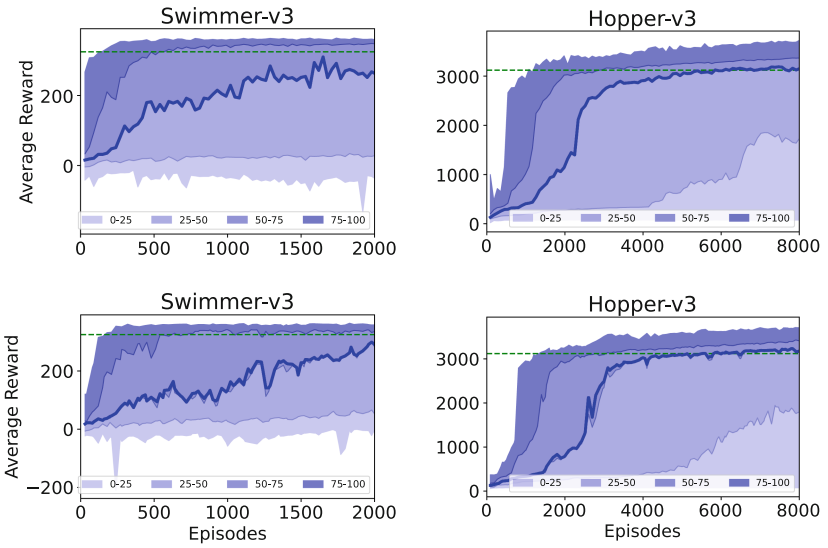


Fig. 3. The figures in the top row represent the evaluation of OP-ARS over 100 seeds. The figures in the bottom row represent the evaluation of OP-ARS's sensitivity to hyperparameters. The average reward is plotted against episodes. The thick blue line represents the median curve, and the shaded region corresponds to the percentiles mentioned in the legend (see bottom of each figure).

Recently, hybrid algorithms such as CEM-RL [15] that mix ES and Deep RL methods have shown to be more sample efficient than ES methods. However, there too all the policies are evaluated in an on-policy fashion. We strongly believe that our off-policy ranking idea can help in these hybrid algorithms as well.

A Differences in the Original ARS from Our Off-Policy Variant

In this section, we mention the key differences between the original ARS and our off-policy variant OP-ARS. Table 4 indicates the exact steps that differ between the algorithms and their implications.

Table 4. Key differences between ARS and our off-policy variant OP-ARS. The step column refers to the step number in Algorithm 1.

Step	ARS	OP-ARS	Implications
6	There is no corresponding step in ARS	Run n_b trajectories	The trajectories are run using policy parameterized by M_j (used as behavior data for off-policy evaluation)
8	Step 5 in ARS; collects **2N** rollouts	Collects only **2b** Rollouts (note that $b < N$)	We collect data required only for update step
7	Step 6 in ARS; sorts δ_k based on returns from the $2N$ trajectories	We sort δ_k based on the fitness function (7) derived using off-policy technique	We generate less data as we find the approximate ranking of δs using off-policy technique even before generating trajectories, unlike in ARS where trajectories are run for all $2N$ policies to find the rankings
9	Step 7 in ARS; uses $r(\pi)$ for the total reward received from policy π	We use $\hat{\eta}(\pi)$ to denote the total reward received from policy π	We use $\hat{\eta}(\cdot)$ to denote the total reward because we use $r(\cdot)$ for one-step reward
7, 8	Step 5, 6 in ARS; first collect data from $2N$ policies, then rank	We first rank; then collect only data from $2b$ policies	We generate less samples as mentioned earlier

B Hyperparameters

In this section, we would like to describe the set of hyperparameters used in our experiments. The ARS and TRES algorithms have a predefined set of hyperparameters, which have been fine-tuned in the corresponding papers. Hence, we use the same hyperparameters in our algorithms for most of the environments. Our off-policy variants OP-ARS and OP-TRES have two new hyperparameters: the number of trajectories to run using behavior policy n_b and the bandwidth in kernel function h. We experiment with varying values of n_b and h, as indicated in Table 5, as part of our hyperparameter exploration process. The most effective hyperparameters, denoted in bold, are employed to generate the outcomes presented in Table 1 and Fig. 1, including the images in the uppermost row of Fig. 3.

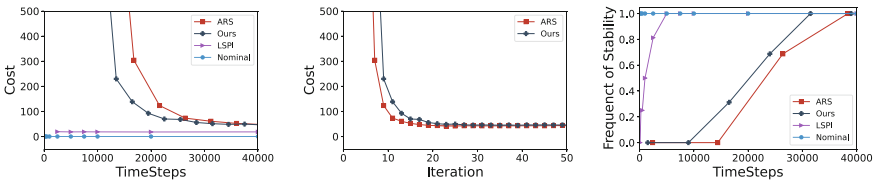
Table 5. Hyperparameter grid used in each environment

Env.(-v3)	OP-ARS				OP-TRES			
	N	b	n_b	h	N	b	n_b	h
Swimmer	2	1	1, **2**	1.0, 0.5, 0.25, **0.1**	4	2	**1**, 2	1.0, 0.5, 0.25, **0.1**
Hopper	8	4	1, **2**	1.0, 0.5, **0.25**, 0.1	–	–	–	–
HalfCheetah	32	4	1, **2**	**1.0**, 0.5, 0.25, 0.1	16	8	**1**, 2	1.0, **0.5**, 0.25, 0.1
Walker2d	40	30	1, **2**	1.0, **0.5**, 0.25, 0.1	–	–	–	–
Ant	60	20	**1**, 2	1.0, 0.5, **0.25**, 0.1	40	20	**1**, 2	1.0, **0.5**, 0.25, 0.1
Humanoid	350	230	1, **2**	1.0, 0.5, **0.25**, 0.1	–	–	–	–

C LQR Experiments

In this section, we showcase the outcomes of our experiments conducted with the LQR environment. As elucidated in Sect. 4.3 of [13], there exist limitations inherent to MuJoCo robotic tasks. Particularly notable is the lack of knowledge concerning the optimal policies within these environments. This uncertainty extends to the comparison between the learned policy of their algorithm and the optimal policy. A viable approach involves applying the algorithms to straightforward, widely recognized environments with well-established optimal policies. In [13], the choice fell upon the Linear Quadratic Regulator (LQR) as the benchmarking environment due to its known dynamics. For a more in-depth understanding of this environment, additional insights can be found in [3, Appendix D.2].

We employ the same framework utilized by [13] to compare our approach with model-based Nominal Control, LSPI [9], and ARS [13]. As demonstrated in the work by [13], the Nominal method exhibits significantly greater sample efficiency than LSPI and ARS by orders of magnitude, underscoring the potential for enhancement. Our experiments corroborate this notion, revealing that our method surpasses ARS in terms of sample efficiency, as indicated in Fig. 4a. Furthermore, Fig. 4b underscores our algorithm's superiority over ARS in terms of stability frequency.



(a) Cost vs TimeSteps (b) Cost vs Iterations (c) Frequency of Stability

Fig. 4. Comparison of sample efficiency and stability of various algorithms on LQR.

References

1. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym. arXiv preprint arXiv:1606.01540 (2016)
2. Duan, Y., Chen, X., Houthooft, R., Schulman, J., Abbeel, P.: Benchmarking deep reinforcement learning for continuous control. In: International Conference on Machine Learning, pp. 1329–1338. PMLR (2016)
3. Eshwar, S., Kolathaya, S., Thoppe, G.: Improving sample efficiency in evolutionary RL using off-policy ranking arXiv:2208.10583 (2023)
4. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International Conference on Machine Learning, pp. 1861–1870. PMLR (2018)
5. Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D.: Deep reinforcement learning that matters. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32 (2018)
6. Kakade, S., Langford, J.: Approximately optimal approximate reinforcement learning. In: In Proc. 19th International Conference on Machine Learning. Citeseer (2002)
7. Kallus, N., Uehara, M.: Doubly robust off-policy value and gradient estimation for deterministic policies. Advances in Neural Information Processing Systems 33 (2020)
8. Kallus, N., Zhou, A.: Policy evaluation and optimization with continuous treatments. In: International Conference on Artificial Intelligence and Statistics, pp. 1243–1251. PMLR (2018)
9. Lagoudakis, M.G., Parr, R.: Model-free least-squares policy iteration. Advances in neural information processing systems **14** (2001)
10. Li, Z., Lin, X., Zhang, Q., Liu, H.: Evolution strategies for continuous optimization: A survey of the state-of-the-art. Swarm Evol. Comput. **56**, 100694 (2020)
11. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. In: ICLR (Poster) (2016)
12. Liu, G., Zhao, L., Yang, F., Bian, J., Qin, T., Yu, N., Liu, T.Y.: Trust region evolution strategies. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 4352–4359 (2019)
13. Mania, H., Guy, A., Recht, B.: Simple random search of static linear policies is competitive for reinforcement learning. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. pp. 1805–1814 (2018)
14. Matyas, J.: Random optimization. Autom. Remote. Control. **26**(2), 246–253 (1965)
15. Pourchot, A., Sigaud, O.: Cem-rl: Combining evolutionary and gradient-based methods for policy search. arXiv preprint arXiv:1810.01222 (2018)
16. Rajeswaran, A., Lowrey, K., Todorov, E., Kakade, S.: Towards generalization and simplicity in continuous control. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. pp. 6553–6564 (2017)
17. Salimans, T., Ho, J., Chen, X., Sidor, S., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:1703.03864 (2017)
18. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: International conference on machine learning. pp. 1889–1897. PMLR (2015)

19. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
20. Todorov, E., Erez, T., Tassa, Y.: Mujoco: A physics engine for model-based control. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 5026–5033. IEEE (2012)
21. Zhang, Y., Ross, K.W.: On-policy deep reinforcement learning for the average-reward criterion. In: International Conference on Machine Learning. pp. 12535–12545. PMLR (2021)