CEM-RL: COMBINING EVOLUTIONARY AND GRADIENT-BASED METHODS FOR POLICY SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep neuroevolution and deep reinforcement learning (deep RL) algorithms are two popular approaches to policy search. The former is widely applicable and rather stable, but suffers from low sample efficiency. By contrast, the latter is more sample efficient, but the most sample efficient variants are also rather unstable and highly sensitive to hyper-parameter setting. So far, these families of methods have mostly been compared as competing tools. However, an emerging approach consists in combining them so as to get the best of both worlds. Two previously existing combinations use either a standard evolutionary algorithm or a goal exploration process together with the DDPG algorithm, a sample efficient off-policy deep RL algorithm. In this paper, we propose a different combination scheme using the simple cross-entropy method (CEM) and TD3, another off-policy deep RL algorithm which improves over DDPG. We evaluate the resulting algorithm, CEM-RL, on a set of benchmarks classically used in deep RL. We show that CEM-RL benefits from several advantages over its competitors and offers a satisfactory trade-off between performance and sample efficiency.

1 INTRODUCTION

Policy search is the problem of finding a policy or controller maximizing some unknown utility function. Recently, research on policy search methods has witnessed a surge of interest due to the combination with deep neural networks, making it possible to find good enough continuous action policies in large domains. From one side, this combination gave rise to the emergence of efficient deep reinforcement learning (deep RL) techniques (Lillicrap et al., 2015; Schulman et al., 2015; 2017). From the other side, evolutionary methods, and particularly deep neuroevolution methods applying evolution strategies (ES) to the parameters of a deep network emerged as a competitive alternative to deep RL due to their higher parallelization capability (Salimans & Kingma, 2016; Conti et al., 2017; Petroski Such et al., 2017).

Both families of techniques have clear distinguishing properties. Evolutionary methods are significantly less sample efficient than deep RL methods because they learn from complete episodes, whereas deep RL methods use elementary steps of the system as samples, and thus exploit more information (Sigaud & Stulp, 2018). In particular, off-policy deep RL algorithms can use a replay buffer to exploit the same samples as many times as useful, greatly improving sample efficiency. Actually, the sample efficiency of ESs can be improved using the "importance mixing" mechanism, but a recent study has shown that the capacity of importance mixing to improve sample efficiency by a factor of ten is still not enough to compete with off-policy deep RL (Pourchot et al., 2018). From the other side, sample efficient off-policy deep RL methods such as the DDPG algorithm (Lillicrap et al., 2015) are known to be unstable and highly sensitive to hyper-parameter setting. Rather than opposing both families as competing solutions to the policy search problem, a richer perspective looks for a way to combine them so as to get the best of both worlds. As covered in Section 2, there are very few attempts in this direction so far.

After some background in Section 3, we propose in Section 4 a new combination method that combines the cross-entropy method (CEM) with TD3, an off-policy deep RL algorithm which improves over DDPG. In Section 5, we investigate experimentally the properties of this CEM-RL method, showing its advantages both over the components taken separately and over a competing approach.

Beyond the results of CEM-RL, the conclusion of this work is that there is still a lot of unexplored potential in new combinations of evolutionary and deep RL methods.

2 RELATED WORK

Policy search is an extremely active research domain. The realization that evolutionary methods are an alternative to continuous action reinforcement learning and that both families share some similarity is not new (Stulp & Sigaud, 2012a;b; 2013) but so far works have mainly focused on comparing them (Salimans et al., 2017; Petroski Such et al., 2017; Conti et al., 2017). Under this perspective, it was shown in (Duan et al., 2016) that, despite its simplicity with respect to most deep RL methods, the Cross-Entropy Method (CEM) was a strong baseline in policy search problems. Here, we focus on works which combine both families of methods.

Synergies between evolution and reinforcement learning have already been investigated under the light of the so-called *Baldwin effect* (Simpson, 1953). This literature is somewhat related to research on meta-learning, where one seeks to evolve an initial policy from which a self-learned reinforcement learning algorithm will perform efficient improvement (Wang et al., 2016; Houthooft et al., 2018; Gupta et al., 2018). The key difference with respect to the method proposed here is that in this literature, the outcome of the RL process is not incorporated back into the genome of the agent, whereas here evolution and reinforcement learning update the same parameters in iterative sequences.

Closer to ours, the work of Colas et al. (2018) sequentially applies a *goal exploration process* (GEP) to fill a replay buffer with purely exploratory trajectories and then applies DDPG to the resulting data. The GEP shares many similarities with evolutionary methods, apart from its focus on diversity rather than on performance of the learned policies. The authors demonstrate on the Continuous Mountain Car and HALF-CHEETAH-V2 benchmarks that their combination, GEP-PG, is more sample-efficient than DDPG, leads to better final solutions and induces less variance during learning. However, due to the sequential nature of the combination, the GEP part does not benefit from the efficient gradient search of the deep RL part.

Another approach related to ours is the work of Maheswaranathan et al. (2018), where the authors introduce optimization problems with an surrogate gradient, i.e. a direction which is correlated with the real gradient. They show that by modifying the covariance matrix of an ES to incorporate the informations contained in the surrogate, an hybrid algorithm can be constructed. They provide a thorough theoretical investigation of their procedure, which they experimentally show capable of outperforming both a standard gradient descent method and a pure ES on several simple benchmarks. They argue that this method could be useful in RL, since surrogate gradients appear in Q-learning and actor-critic methods. However, a practical analysis of those claims remains to be performed. Their approach resembles ours, since they use a gradient method to enhance an ES. But a notable difference is that they use the gradient information to directly change the distribution from which samples are drawn, whereas we use gradient information on the samples themselves, which changes the distribution only indirectly.

The work which is the closest to ours is Khadka & Tumer (2018). The authors introduce an algorithm called ERL (for Evolutionary Reinforcement Learning), which is presented as an efficient combination of a deep RL algorithm, DDPG, and a population-based evolutionary algorithm. It takes the form of a population of actors, which are constantly mutated and selected based on their fitness. In parallel, a single DDPG agent is trained from the samples generated by the population. This single agent is then periodically inserted into the population. When the gradient-based policy improvement mechanism of DDPG is efficient, this individual outperforms its evolutionary siblings, it gets selected into the next generation and draws the whole population towards higher performance. Through their experiments, Khadka & Tumer demonstrate that this setup benefits from an efficient transfer of information between the RL algorithm and the evolutionary algorithm.

However, their combination scheme cannot be applied in the context of an ES. Indeed, in these methods a covariance matrix is used to produce the next generation but, because the additional individual from DDPG is generated in isolation, it may not comply with this covariance matrix. This is unfortunate because ESs are generally the most efficient evolutionary methods, and importance mixing can only be applied in their context to bring further sample efficiency improvement.

By contrast with the works outlined above, the method presented here combines CEM and TD3 in such a way that our algorithm benefits from the gradient-based policy improvement mechanism of TD3, from the better stability of ESs, and may even profit from the additional sample efficiency brought by importance sampling.

3 BACKGROUND

In this section, we provide a quick overview of the evolutionary and deep RL methods used throughout the paper.

3.1 EVOLUTIONARY ALGORITHMS, EVOLUTION STRATEGIES AND EDAS

Evolutionary algorithms manage a limited population of individuals, and generate new individuals randomly in the vicinity of the previous *elite* individuals (Back, 1996). Evolution strategies can be seen as specific evolutionary algorithms where only one individual is retained from one generation to the next, this individual being the mean of the distribution from which new individuals are drawn. More specifically, an optimum individual is computed from the previous samples and the next samples are obtained by adding Gaussian noise to the current optimum. Finally, among ESs, Estimation of Distribution Algorithms (EDAs) are a specific family where the population is represented as a distribution using a covariance matrix $\mathbf{\Sigma}$ (Larrañaga & Lozano, 2001). This covariance matrix defines a multivariate Gaussian function and samples at the next iteration are drawn with a probability proportional to this Gaussian function. Along iterations, the ellipsoid defined by $\mathbf{\Sigma}$ is progressively adjusted to the top part of the hill corresponding to the local optimum $\boldsymbol{\theta}^*$. Various instances of EDAs, such as CEM, CMA-ES, PI^2-CMA, are covered in Stulp & Sigaud (2012a;b; 2013). Here we focus on the first two.

3.2 THE CROSS-ENTROPY METHOD

The Cross-Entropy Method (CEM) is a simple EDA where the number of elite individuals is fixed to a certain value K_e (usually set to half the population). After all individuals of a population are evaluated, the K_e most fit individuals are used to compute the new mean and variance of the population, from which the next generation is sampled after adding some limited extra variance ϵ to prevent premature convergence.

In more details, we denote by μ the mean of the distribution, and Σ its covariance. Each individual x_i is sampled by adding Gaussian noise around μ, according to the current covariance matrix Σ, i.e. $x_i \sim \mathcal{N}(\mu, \Sigma)$. The problem-dependent fitness of these new individuals $(f_i)_{i=1,\dots,\lambda}$ is then computed. The top-performing K_e individuals, $(z_i)_{i=1,\dots,K_e}$ are then used to update the parameters of the distribution as follows:

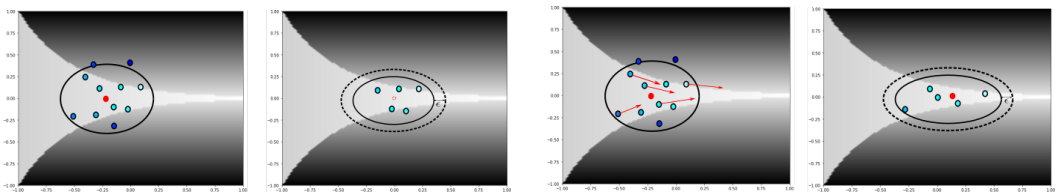
$$\mu_{new} = \sum_{i=1}^{K_e} \lambda_i z_i \tag{1}$$

$$\Sigma_{new} = \sum_{i=1}^{K_e} \lambda_i (z_i - \mu_{old})(z_i - \mu_{old})^T + \epsilon\mathcal{I}, \tag{2}$$

where $(\lambda_i)_{i=1,\dots,K_e}$ are weights given to the individuals. Common choices for those weights are $\lambda_i = \frac{1}{K_e}$ or $\lambda_i = \frac{log(1+K_e)/i}{\sum_{i=1}^{K_e} \log(1+K_e)/i}$. In the former, each individual is given the same importance, whereas the latter gives more importance to better individuals.

In this work, we add some noise in the form of $\epsilon\mathcal{I}$ to the usual covariance update. This prevents early convergence, which is inevitable considering the greediness of CEM. We choose to have an exponentially decaying ϵ, by setting an initial and a final standard deviation, respectively σ_{init} and σ_{end}, initializing ϵ to σ_{init} and updating ϵ at each iteration with $\epsilon = \tau_{cem}\epsilon + (1 - \tau_{cem})\sigma_{end}$.

An iteration on the CEM algorithm is depicted in Figure 1a.



(a) (b)

Figure 1: One iteration of the Cross-Entropy Method (a) and CEM-RL algorithm (b). The gradient landscape is the "narrowing path" taken from Lehman et al. (2017). (a) Left: the current population is represented by the current optimum (red dot) and covariance matrix (ellipse). Individuals are generated from the multivariate Gaussian model and evaluated according to the gradient landscape. Right: preparing the next generation: only the best individuals have been retained, from these individuals a new optimum (dotted red) and covariance matrix are computed, and some extra variance ϵ is added to prevent premature convergence. (b) Same as (a), the only difference being that steps of the gradient from the TD3 critic are applied to half the individuals of the CEM population (chosen randomly). These gradient steps are represented as red vectors. As a result, if the gradient is accurate, the population moves faster towards the optimum.

3.3 CMA-ES

Like CEM, CMA-ES is an EDA where the number of elite individuals is fixed to a certain value K_e. The mean and covariance of the new generation are constructed from those individuals. However this construction is more elaborate than in CEM. The top K_e individuals are ranked according to their performance, and are assigned weights conforming to this ranking. Those weights measure the impact of individuals on the construction of the new mean and covariance. Quantities called "Evolutionary paths" are also used to accumulate the search directions of successive generations. In fact, the updates in CMA-ES are shown to approximate the natural gradient, without explicitly modeling the Fisher information matrix (Arnold et al., 2011).

3.4 IMPORTANCE MIXING

Importance mixing is a mechanism to improve the sample efficiency of ESs. It was initially introduced in Sun et al. (2009) and consisted in reusing some samples from the previous generation into the current one, to avoid the cost of re-evaluating the corresponding policies in the environment. The mechanism was recently extended in Pourchot et al. (2018) to reusing samples from any generation stored into an archive. Empirical results showed that importance sampling can improve sample efficiency by a factor of ten, and that most of these savings just come from using the samples from the previous generation, as performed by the initial algorithm. A pseudo-code of the importance mixing mechanism is given in Appendix B and more details can be found in Pourchot et al. (2018).

3.5 DDPG AND TD3

The DDPG (Lillicrap et al., 2015) and TD3 (Fujimoto et al., 2018) algorithms are two off-policy, actor-critic and sample efficient deep RL algorithms. The DDPG algorithm suffers from instabilities partly due to an overestimation bias in the way it updates the critic, and is known to be difficult to tune given its sensitivity to hyper-parameter settings. The availability of properly tuned code baselines incorporating several advanced mechanisms improves on the latter issue (Dhariwal et al., 2017). The TD3 algorithm rather improves on the former issue, limiting the over-estimation bias by using two critics and taking the lowest estimate of the action value functions in the update mechanisms. Our CEM-RL algorithm uses TD3 rather than DDPG as the former has been shown to consistently outperform the latter (Fujimoto et al., 2018). However, for the sake of comparison with ERL, we also use CEM in combination with DDPG in Section 5.2.2.

4 METHODS

Our method combines CEM and TD3. The combination scheme can be explained as follows: The mean actor of the CEM population, referred to as π_μ, is first initialized with a random actor network. A critic network $\mathcal{Q}^\pi$ is also initialized. At each iteration, a population of actors is sampled by adding Gaussian noise around the current mean π_μ, according to the current covariance matrix $\mathbf{\Sigma}$. Half of the resulting actors are directly evaluated. The corresponding fitness is computed as the cumulative reward obtained during an episode in the environment. The other half of the population follows the direction of the gradient given by the current critic $\mathcal{Q}^\pi$, as performed in DDPG and TD3. This gradient is applied for a fixed number of steps, before the actor gets evaluated too. CEM then takes the top-performing half of the resulting global population to compute its new π_μ and $\mathbf{\Sigma}$.

The steps used to evaluate all actors in the population are fed into the replay buffer. The critic is then trained pro rata to the quantity of new information introduced in the buffer at the current generation. For instance, if the population contains 10 individuals, and if each episode lasts 1000 time steps, then $10{,}000$ new samples are introduced in the replay buffer at each generation. The critic is thus trained for $10{,}000$ mini-batches. This is a common practice in deep RL algorithms, where one mini-batch backward propagation step is done for each step of the actor in the environment. We also choose this number of steps to be the number of gradient steps taken by half of the population at the next iteration. A pseudo-code of CEM-RL is provided in Algorithm 1.

An iteration of CEM-RL is illustrated on a simple 2-dimensional case in Figure 1b. By comparing to Figure 1a, one can see that the strength of this combination between CEM and TD3 resides in the greediness of the gradient steps added to the CEM algorithm. In cases where the direction of the gradient is correlated with an increase in the performance of the actor, CEM feeds from this direction by incorporating the corresponding actor in its computations. By contrast, in cases where the gradient steps decrease performance, the resulting actors are ignored by CEM, which instead focuses on standard samples around π_μ. Those poor samples do not bring new insight on the current distribution of the CEM algorithm, since the gradient steps takes them away from the current distribution. However, since all actors fill the replay buffer, the resulting experience is still fed to the critic and the future learning actors, providing some supplementary exploration.

Similarly to Khadka & Tumer (2018), this approach generates a beneficial flow of information between the deep RL part and the evolutionary part. Indeed, on one hand, good actors found by following the current critic directly improve the evolutionary population. On the other hand, good actors found through evolution fill the replay buffer with generated experiences from which the RL algorithm learns. But, by contrast with Khadka & Tumer (2018), gradient steps are directly applied to several samples, and using the CEM algorithm makes it possible to use importance mixing, as described in Section 3.4.

Given that CMA-ES is generally considered as more sophisticated than CEM, one may wonder why not using CMA-ES instead of CEM into the CEM-RL algorithm. Actually, the key contribution of CMA-ES with respect to CEM consist of the evolutionary path mechanism (see Section 3.3), but this mechanism is incompatible with the CEM-RL approach since it results in some inertia of the covariance matrix which resists to the beneficial effect of using the gradient from the deep RL algorithm.

Note that although we introduced the CEM algorithm in its general form with a full covariance matrix $\mathbf{\Sigma}$, in practice $\mathbf{\Sigma}$ can be too large for computing the updates and sampling new individuals. Indeed, if n denotes the number of parameters, simply sampling from $\mathbf{\Sigma}$ scales at least in $\mathcal{O}(n^{2.3})$, which becomes quickly intractable. Instead, we constrain $\mathbf{\Sigma}$ to be diagonal. This means that in our computations, we replace the update in (2) by

$$\mathbf{\Sigma}_{new} = \sum_{i=1}^{K_e} \lambda_i (z_i - \mu_{old})^2 + \epsilon \mathcal{I}, \tag{3}$$

where the square of the vectors denote the vectors of the square of the coordinates.

Algorithm 1 CEM-RL

Require: *max_steps*, the maximum number of steps in the environment
$\tau_{\text{CEM}}, \sigma_{init}, \sigma_{end}$ and *pop_size*, hyper-parameters of the CEM algorithm
γ, τ, lr_{actor} and lr_{critic}, hyper-parameters of DDPG

Initialize a random actor π_μ, to be the mean of the CEM algorithm
Let $\Sigma = \sigma_{init}\mathcal{I}$ be the covariance matrix of the CEM algorithm
Initialize the critic $\mathcal{Q}^\pi$ and the target critic $\mathcal{Q}^\pi_t$
Initialize an empty cyclic replay buffer $\mathcal{R}$

total_steps, actor_steps = $0, 0$
while *total_steps* $<$ *max_steps*:

  Draw the current population *pop* using $\mathcal{N}(\pi_\mu, \Sigma)$ with importance mixing (see Algorithm 2)
  for $i \leftarrow 1$ to $pop_size/2$:
    Set the current policy π to $pop[i]$
    Initialize a target actor π_t with the weights of π
    Train $\mathcal{Q}^\pi$ for $2 *$ *actor_steps* / *pop_size* mini-batches
    Train π for *actor_steps* mini-batches
    Reintroduce the weights of π in *pop*

  actor_steps = 0
  for $i \leftarrow 1$ to pop_size:
    Set the current policy π to $pop[i]$
    (fitness f, steps s) $\leftarrow$ evaluate(π)
    Fill $\mathcal{R}$ with the collected experiences
    actor_steps = *actor_steps* + s
  total_steps = *total_steps* + *actor_steps*

  Update π_μ and $\mathbf{\Sigma}$ with the top half of the population (see (1) and (2) in Section 3.2)

end while

5 EXPERIMENTAL STUDY

In this section, we study the CEM-RL algorithm to answer the following questions:

- How does the performance of CEM-RL compare to the performance of CEM and TD3 taken separately?
- What does importance mixing bring in terms of sample efficiency in this context?
- How does CEM-RL perform compared to ERL? What are the main factors explaining the difference between both algorithms?

5.1 EXPERIMENTAL SETUP

In order to investigate the above questions, we evaluate the corresponding algorithms in several continuous control tasks simulated with the MUJOCO physics engine: HALF-CHEETAH-V2, HOPPER-V2, SWIMMER-V2, ANT-V2 and WALKER2D-V2 (Brockman et al., 2016).

We implemented CEM-RL with the PYTORCH library [1]. We built our code around the DDPG and TD3 implementations given by the authors of the TD3 algorithm[2]. For the ERL implementation, we used one given by the authors[3].

[1]The code for reproducing the experiments is available and will be disclosed after the review process to preserve anonymous reviewing.

[2]Available here: `https://github.com/sfujim/TD3`.

[3]Available here: `https://github.com/Anonymous991/ERL`.

Unless specified otherwise, each curve represents the average over 10 runs of the corresponding quantity, and the variance displayed corresponds to the 68% confidence interval for the estimation of the mean. The x-axis represents the total number of steps actually performed in the environment, to take into account the potential effect on importance mixing on sample efficiency.

Most of the TD3 and DDPG hyper-parameters were reused from Fujimoto et al. (2018)'s paper. The architectures of the networks is described in Appendix A. The only notable difference is the use of $tanh$ non linearities instead of $ReLU$ in the architecture of the actor. We trained the networks with the Adam optimizer (Kingma & Ba, 2014), with a learning rate of $1e^{-3}$ for both the actor and the critic. The discount rate γ was set to 0.99, and the target weight τ to $5e^{-3}$. The size of the population was fixed to 10 on all benchmarks, and the standard deviations σ_{init}, σ_{end} and the constant τ_{cem} of the CEM algorithm were respectively set to $1e^{-3}$, $1e^{-5}$ and 0.95. Finally, the size of the replay buffer was set to $1e^6$, and the batch size to 100.

5.2 RESULTS

We first compare CEM-RL to TD3 and CEM, then to ERL, and finally we perform additional studies to analyze some factors of this performance.

5.2.1 COMPARISON TO TD3 AND CEM

In this section, we compare CEM-RL to two baselines: the deep RL TD3 algorithm, and the CEM evolution strategy. For TD3, we report the average of the score of the agent over 5 episodes for every 5000 steps performed in the environment. For CEM and CEM-RL, we report after each generation the average of the score of the top-half of the population (the average is thus computed over 5 individuals). From Figure 2, one can see that CEM-RL outperforms CEM and TD3 on HALF-CHEETAH-V2, HOPPER-V2 and WALKER2D-V2. On ANT-V2, CEM-RL outperforms CEM and is on par with TD3. On most benchmarks, CEM-RL also displays less variance than TD3. However, CEM does better than TD3 and CEM-RL on SWIMMER-V2.

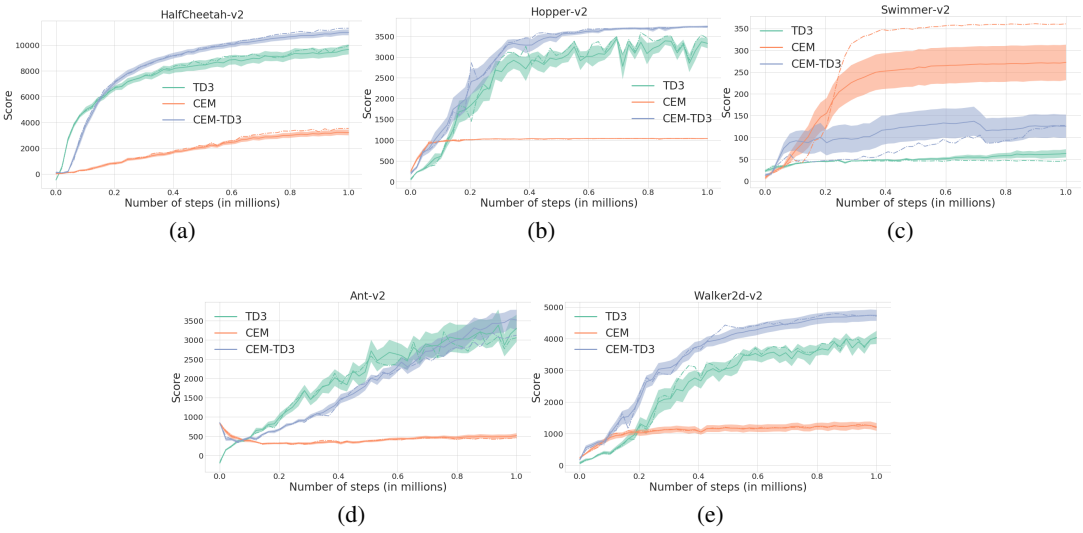
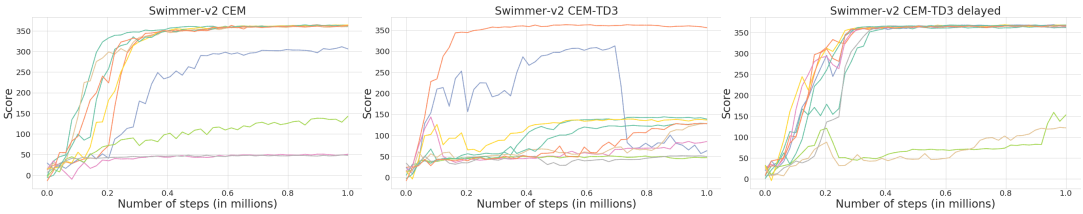


Figure 2: Learning curves of TD3, CEM and CEM-RL on the HALF-CHEETAH-V2, HOPPER-V2, SWIMMER-V2, ANT-V2 and WALKER2D-V2 benchmarks. Dotted curves represent medians.

We can conclude that, in environments where TD3 is already able to provide correct solutions, CEM-RL enhances TD3 by reducing variance in the learning process, and eventually performs better in the end. Having a population of actors stabilizes performances by absorbing instabilities that can appear during learning. This prevents inserting too many poor samples into the replay buffer. The stabilizing effect is further enhanced by the greedy nature of the CEM algorithm, which prevents actors ending in poor regions of the search space from influencing the current parameters of the

sampling distribution. Sampling from the evolution strategy also results in sufficient exploration and performs better than adding simple Gaussian noise to the actions.

However, the greediness of our approach has some drawbacks: on the SWIMMER-V2 benchmark, one can see that CEM outperforms both TD3 and CEM-RL. Looking at the individual learning curves in Figure 3, we can get an intuition as to why CEM-RL failed to learn. We see that around 60% of the time, CEM is able to find a policy with a score of around 350. Another 20% of trials start to learn but display slower convergence rates. Finally, the last 20% gets stuck in a local optimum performing around 50. This explains why the median in Figure 2c displays a final score of 350, whereas the mean curve is stuck around 250 with a large variance. Moreover, most of the the curves that end up in the highest fitness region start learning after 100,000 steps, and up to 200,000 steps, meaning that through random search, this very promising direction is hard to find. However, as can be seen from Figure 2c, TD3 always gets stuck in the 50 fitness local optimum, and relatively quickly compared to the time necessary for CEM to leave this region of the search space. This means that in the beginning of learning, half of the CEM-RL population acts as a burden to the rest of the population by consistently getting stuck into a local optimum. Even if CEM is able to find one or two really good actors from time to time, as long as the rest of the top half of the population is filled with those weights, the convergence of the algorithm to good solutions is compromised. Indeed, one can see from Figure 3b that only one run manages to reach a score of 350. To corroborate those claims, we performed an additional experiment where, instead of letting half the population take gradient steps right away, we waited for 200,000 time steps in the environment before letting them do so. The resulting learning curves are presented in Figure 3c. The new trajectories are similar to those of the CEM algorithm alone, which suggests that delaying the beginning of the deep RL learning indeed helps CEM-RL escape from this local optimum. But since CEM-RL comes with the computational costs of deep RL methods, there is no clear advantage in using CEM-RL rather than CEM on this benchmark. This conclusion could already be established from Khadka & Tumer (2018), where the evolution algorithm alone produces results on par with the ERL algorithm. This makes the SWIMMER-V2 environment particularly interesting, as we are not aware of any deep RL method capable of solving this benchmark quickly and reliably.



(a) (b) (c)

Figure 3: Learning curves of (a) CEM, (b) CEM-RL and (c) CEM-RL with delay on the SWIMMER-V2 benchmark. Each curve represents a run of the algorithm.

In Figure 5 of Appendix C, we display the influence of the importance mixing mechanism over the evolution of performance, for CEM and CEM-RL. Once again, we report after each generation the average of the score of the top-half of the population.

In CEM, importance mixing is implemented as described in (Pourchot et al., 2018). By contrast, the importance mixing algorithm needs adaptation in CEM-RL. Actors which take gradient steps can no longer be regarded as sampled from the current distribution of the CEM algorithm. We thus choose to apply the importance mixing mechanism only to the half of the population which does not receive gradient steps from the critic. In practice, only actors which do not take gradient steps are inserted into the actor archive and can be replaced with samples from previous generations.

From Figure 5, one can see that in the CEM case, using importance mixing can result into significant sample reuse, whilst introducing some small instability. However, this benefit varies greatly from environment to environment. The gain in sample reuse for the CEM-RL algorithm is almost null. A performance drop is seen on SWIMMER-V2, and importance mixing introduces instability in the learning of HOPPER-V2. On HALF-CHEETAH-V2 and WALKER2D-V2, importance mixing seems to provide better results, but the margin is not sufficiently large to assert a clear superiority over

the standard version of CEM-RL. On ANT-V2, importance mixing seems to accelerate learning in the beginning, but final performances are equivalent to those of CEM-RL. Thus importance mixing seems to have a limited impact in CEM-RL.

As a summary, Table 1 gives the final best results of all explored methods. The conclusion is that CEM-RL is generally superior to CEM and TD3.

	CEM-RL	CEM-RL + IM	CEM	TD3
HALF-CHEETAH-V2	10977±2.0%	**11398±1.5%**	3208±6.8%	9651±4.0%
HOPPER-V2	**3732±0.61%**	3614±1.0%	1039±0.48%	3327±3.5%
SWIMMER-V2	125±21%	95±14%	**272±15%**	63±14%
ANT-V2	3520±7.6%	**3597±6.6%**	516±12%	3303±11%
WALKER2D-V2	4732±3.7%	**4772±3.5%**	1206±10%	4033±5.1%

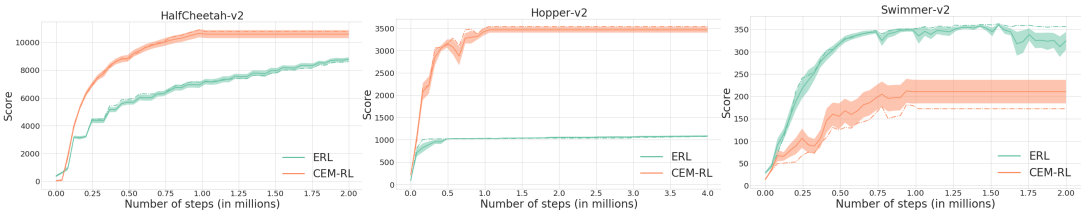
Table 1: Performance of CEM-RL with and without importance mixing, with TD3 and CEM, on the HALF-CHEETAH-V2, HOPPER-V2, SWIMMER-V2, ANT-V2 and WALKER2D-V2 environments. We report means over the top 5 performing individuals after 1 million steps for CEM and CEM-RL, and the mean over 5 episodes for TD3. The displayed variance correspond to the 68% confidence interval over the estimation of the mean over 10 runs.

5.2.2 COMPARISON TO ERL

In Figure 4, we compare ERL and CEM-RL. The ERL method using DDPG, we also use DDPG with CEM-RL instead of TD3 to isolate the effect of the combination scheme from the improvement brought by TD3 itself. Due to limited computational resources, we stop learning with CEM-RL after 1 million steps. However, we let ERL learn for the same number of steps as in Khadka & Tumer, namely 2 millions on HALF-CHEETAH-V2 and SWIMMER-V2, and 4 millions on HOPPER-V2. Besides, for the same reason, we did not perform this comparison for ANT-V2 and WALKER2D-V2 which require respectively 6 and 10 million steps.

Our results show some discrepancy with those of the ERL paper (Khadka & Tumer, 2018). We explain this difference by two factors. First, the authors only average their results over 5 different seeds, whereas we used 10 seeds. Second, the released implementation of ERL could have been slightly different from the one that was used to produce the results[4], raising again the reproducibility issue recently discussed in the reinforcement learning literature (Henderson et al., 2017).

One can see from the curves that CEM-RL outperforms ERL on HALF-CHEETAH-V2 and HOPPER-V2. On HALF-CHEETAH-V2, both methods learn relatively good policies, but CEM-RL reaches better scores and does so faster. On HOPPER-V2, we did not find a single run where ERL escapes from the local optimum at 1000, whereas CEM-RL consistently reached scores of 3500. On SWIMMER-V2 however, ERL outperforms CEM-RL. CEM-RL with DDPG encounters the same problem as described in Section 5.2.1. On the other hand, ERL is able to successfully escape from the local optimum. More work will be necessary to understand this phenomenon in detail.



(a) (b) (c)

Figure 4: Learning curves of ERL and CEM-RL on the HALF-CHEETAH-V2, HOPPER-V2 and SWIMMER-V2 benchmarks. Dotted curves represent medians.

[4]personal communication with the authors

5.2.3 OTHER FACTORS OF CEM-RL PERFORMANCE

In Table 2, we compare the performance of CEM-RL with some variants to better understand the factors of its efficiency.

	CEM-RL	CEM-RL + AN	CEM-RL-DDPG	CEM-RL-RELU
HALF-CHEETAH-V2	**10977±2.0%**	10546±4.0%	10589±386	8261±11%
HOPPER-V2	**3732±0.61%**	3679±1.0%	3465±1.8%	3655±3.7%
SWIMMER-V2	125±21%	98±23%	**210±12%**	117±13%
ANT-V2	3520±7.6%	**3820±6.9%**	1088±15%	2488±15%
WALKER2D-V2	4732±3.7%	4738±2.8%	3757±8.4%	**4772±4.0%**

Table 2: Performance of CEM-RL with and without action noise (AN), with DDPG, and with RELU non-linearities in MUJOCO environments. We report means over the top 5 performing individuals after 1 million steps. Displayed variance corresponds to the 68% confidence interval over the estimation of the mean.

First, in Khadka & Tumer (2018), the authors indicate that one reason for the efficiency of their approach is that the replay buffer of the DDPG algorithm gets filled with two types of noisy experiences. On one hand, the buffer gets filled with noisy interactions of the DDPG actor with the environment. This is usually referred to as *action space noise*. On the other hand, actors with different parameters also fill the buffer, which is more similar to *parameter space noise* (Plappert et al., 2017). In our CEM-RL algorithm, we only use parameter space noise, but it would also be possible to add action space noise. To explore this direction, we let each of the actors taking gradient steps perform a noisy episode in the environment. We report final results after 1 million steps in Table 2. Learning curves are available in Figure 6 of Appendix C. Unlike what the authors suggested, we dot not find any conclusive evidence that action space noise improves performance. In CEM-RL, the CEM part seems to explore enough of the action space on its own.

Second, we replace TD3 with DDPG. Learning curves are displayed in Figure 7 of Appendix C. One can see that on most benchmarks, using TD3 allows for overall better performance than using DDPG, both in terms of quality of learning, convergence speed, and learning stability. This is especially true for hard environments such as WALKER2D-V2 and ANT-V2.

Finally, we explore the impact of the type of non-linearities, and more generally the architecture used in the actor of CEM-RL. Table 2 reports the results of CEM-RL using RELU non-linearities between the linear layers, instead of $\tanh$. The learning curves are available in Figure 8 of Appendix C. Results indicate that on some benchmarks, changing from $\tanh$ to RELU can cause a huge drop in performance. This is particularly obvious in the HALF-CHEETAH-V2 and ANT-V2 benchmarks, where the average performances respectively drop by 25% and 30%. Besides, Figure 9 of Appendix C shows that, for the CEM algorithm on the SWIMMER-V2 benchmark, using RELU causes a 60% performance drop. As previously reported in the literature (Henderson et al., 2017), this study suggests that the architectures of the networks can sometimes have a large impact on performance.

6 CONCLUSION AND FUTURE WORK

We advocated in this paper for combining evolutionary and deep RL methods rather than opposing them. In particular, we have proposed such a combination, the CEM-RL algorithm, and showed that in most cases it was outperforming not only some evolution strategies and some sample efficient off-policy deep RL algorithms, but also another combination, the ERL algorithm. Importantly, despite being mainly an evolutionary method, CEM-RL performs at a state-of-the-art level even when considering sample efficiency.

Beyond these positive performance results, our study raises more fundamental questions. For instance, why does the simple CEM algorithm perform so well on the SWIMMER-V2 benchmark? Besides, our study of the potential benefits from importance mixing did not confirm a clear benefit of using it, neither did the effect of noise on the actions. Understanding the fundamental reasons behind the above findings will require further investigations. Such a deeper study may help understand which properties are critical in the performance and sample efficiency of policy search algorithms, and define even more efficient policy search algorithms in the future. Finally, given the impact of the

neural architecture on our results, we believe that a more systemic search of architectures through techniques such as neural architecture search (Zoph & Le, 2016; Elsken et al., 2018) may provide important progress in performance of deep policy search algorithms.

REFERENCES

L. Arnold, A. Auger, N. Hansen, and Y. Ollivier. Information-geometric optimization algorithms: A unifying picture via invariance principles. Technical report, INRIA Saclay, 2011.

Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

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

Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. GEP-PG: Decoupling exploration and exploitation in deep reinforcement learning algorithms. *arXiv preprint arXiv:1802.05054*, 2018.

Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. *arXiv preprint arXiv:1712.06560*, 2017.

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

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. *arXiv preprint arXiv:1604.06778*, 2016.

T. Elsken, J. Hendrik Metzen, and F. Hutter. Neural Architecture Search: A Survey. *ArXiv e-prints*, August 2018.

Scott Fujimoto, Herke van Hoof, and Dave Meger. Addressing function approximation error in actor-critic methods. *CoRR*, abs/1802.09477, 2018. URL `http://arxiv.org/abs/1802.09477`.

Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. *arXiv preprint arXiv:1802.07245*, 2018.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*, 2017.

Rein Houthooft, Richard Y Chen, Phillip Isola, Bradly C Stadie, Filip Wolski, Jonathan Ho, and Pieter Abbeel. Evolved policy gradients. *arXiv preprint arXiv:1802.04821*, 2018.

Shauharda Khadka and Kagan Tumer. Evolutionary reinforcement learning. *arXiv preprint arXiv:1805.07917*, 2018.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Pedro Larrañaga and Jose A Lozano. *Estimation of distribution algorithms: A new tool for evolutionary computation*, volume 2. Springer Science & Business Media, 2001.

Joel Lehman, Jay Chen, Jeff Clune, and Kenneth O. Stanley. ES is more than just a traditional finite-difference approximator. *arXiv preprint arXiv:1712.06568*, 2017.

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

Niru Maheswaranathan, Luke Metz, George Tucker, and Jascha Sohl-Dickstein. Guided evolutionary strategies: escaping the curse of dimensionality in random search. *arXiv preprint arXiv:1806.10230*, 2018.

Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y. Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

Aloïs Pourchot, Nicolas Perrin, and Olivier Sigaud. Importance mixing: Improving sample reuse in evolutionary policy search methods. *arXiv preprint arXiv:1808.05832*, 2018.

Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv preprint arXiv:1602.07868*, 2016.

Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR, abs/1502.05477*, 2015.

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

Olivier Sigaud and Freek Stulp. Policy search in continuous action domains: an overview. *arXiv preprint arXiv:1803.04706*, 2018.

George Gaylord Simpson. The Baldwin effect. *Evolution*, 7(2):110–117, 1953.

Freek Stulp and Olivier Sigaud. Path integral policy improvement with covariance matrix adaptation. In *Proceedings of the 29th International Conference on Machine Learning*, pp. 1–8, Edinburgh, Scotland, 2012a.

Freek Stulp and Olivier Sigaud. Policy improvement methods: Between black-box optimization and episodic reinforcement learning. Technical report, hal-00738463, 2012b.

Freek Stulp and Olivier Sigaud. Robot skill learning: From reinforcement learning to evolution strategies. *Paladyn Journal of Behavioral Robotics*, 4(1):49–61, august 2013. doi: 10.2478/pjbr-2013-0003.

Yi Sun, Daan Wierstra, Tom Schaul, and Juergen Schmidhuber. Efficient natural evolution strategies. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pp. 539–546. ACM, 2009.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

A ARCHITECTURE OF THE NETWORKS

Our architecture is very similar to the one described in Fujimoto et al. (2018). In particular the size of the layers remains the same. The only differences resides in the non-linearities. We use $\tanh$ operations in the actor between each layer, where Fujimoto et al. use RELU. We use leaky RELU in the critic, where Fujimoto et al. use simple RELU.

Table 3: Architecture of the networks

Actor	Critic
(state dim, 400)	(state dim + action dim, 400)
tanh	leaky ReLU
(400, 300)	(400, 300)
tanh	leaky ReLU
(300, 1)	(300, 1)
tanh	

B IMPORTANCE MIXING ALGORITHM

The importance mixing algorithm aims at generating samples following a distribution with a known probability density function (pdf) p_{new} while reusing previously generated samples from another distribution with pdf p_{old}. At each iteration of the inner loop, one alternatively tries an existing sample from p_{old} and a newly drawn sample drawn from p_{new}. Both samples have a given probability to be accepted. This step is repeated until either all existing samples are tested, in which case the rest of the samples is sampled from p_{new}, or until the number of desired samples is reached, marking the end of the procedure. See Pourchot et al. (2018) for more details.

Algorithm 2 Importance mixing

Require: $p(z, \boldsymbol{\theta}_{new})$: current probability density function (pdf), $p(z, \boldsymbol{\theta}_{old})$: old pdf, g_{old}: old generation

$g_{new} \leftarrow \emptyset$
for $i \leftarrow 1$ to N
  Draw rand1 and rand2 uniformly from $[0, 1]$
  Let z_i be the ith individual of the old generation g_{old}
  if $min(1, \frac{p(z_i, \boldsymbol{\theta}_{new})}{p(z_i, \boldsymbol{\theta}_{old})}) >$ rand1:
    Append z_i to the current generation g_{new}
  Draw z_i' according to the current pdf $p(., \boldsymbol{\theta}_{new})$
  if $max(0, 1 - \frac{p(z_i', \boldsymbol{\theta}_{old})}{p(z_i', \boldsymbol{\theta}_{new})}) >$ rand2:
    Append z_i' to the current generation g_{new}
  size = $|g_{new}|$
  if size $\geq N$: go to 12
if size $> N$: remove a randomly chosen sample
if size $< N$: fill the generation sampling from $p(., \boldsymbol{\theta}_{new})$
return g_{new}

C LEARNING CURVES

The following learning curves have been rejected in Appendix due to space constraints. They are discussed in the main text.

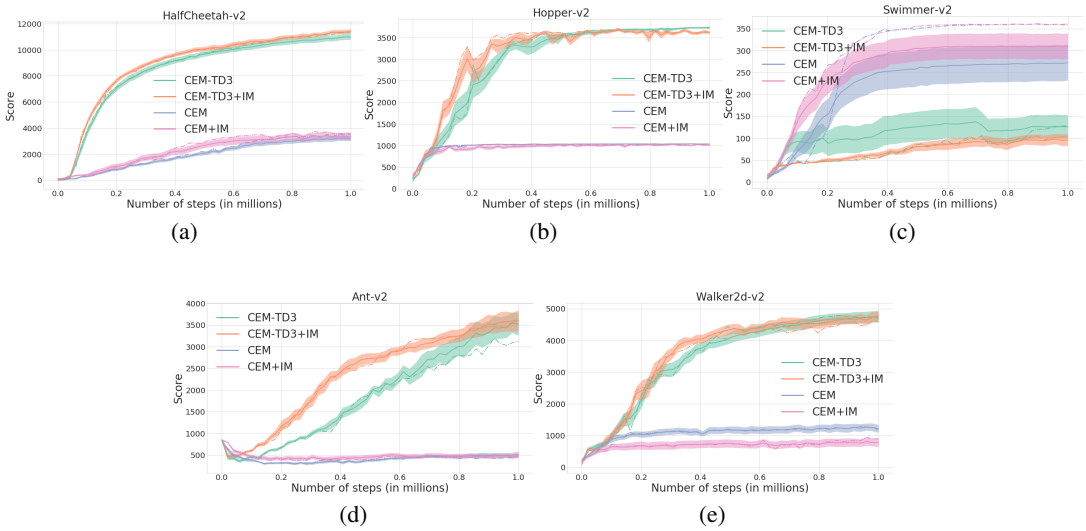


Figure 5: Learning curves of CEM-RL and CEM with and without importance mixing on the HALF-CHEETAH-V2, HOPPER-V2, SWIMMER-V2, ANT-V2 and WALKER2D-V2 benchmarks. Dotted curves represent medians.

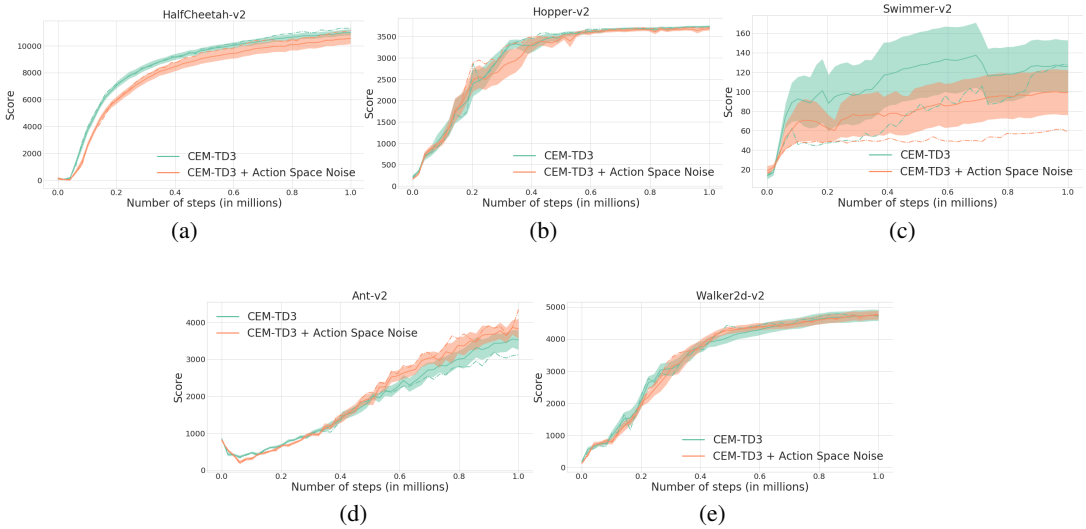


Figure 6: Learning curves of CEM-RL with and without action space noise on the HALF-CHEETAH-V2, HOPPER-V2, SWIMMER-V2, ANT-V2 and WALKER2D-V2 benchmarks. Dotted curves represent medians.

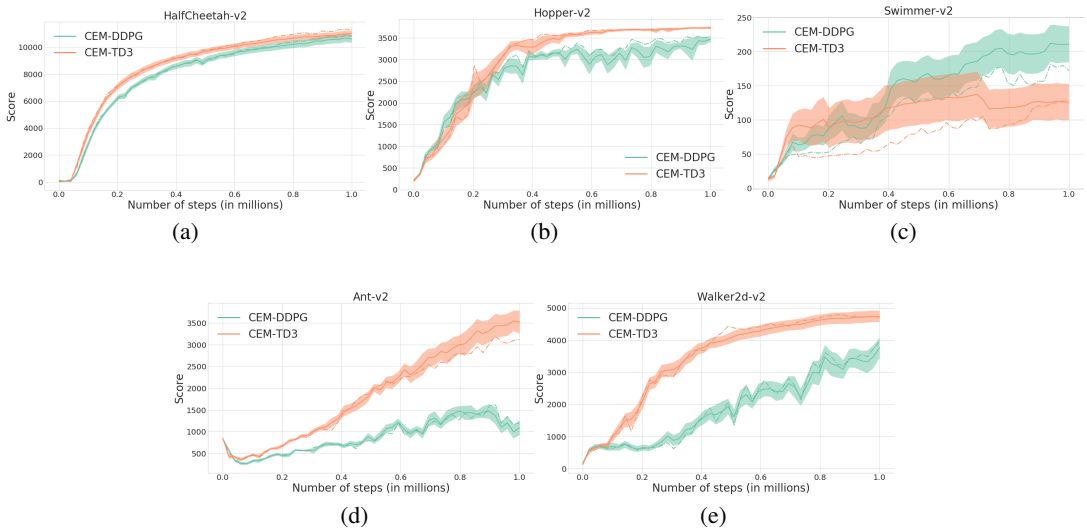


Figure 7: Learning curves of CEM-RL with DDPG and TD3 on the HALF-CHEETAH-V2, HOPPER-V2, SWIMMER-V2, ANT-V2 and WALKER2D-V2 benchmarks. Dotted curves represent medians.

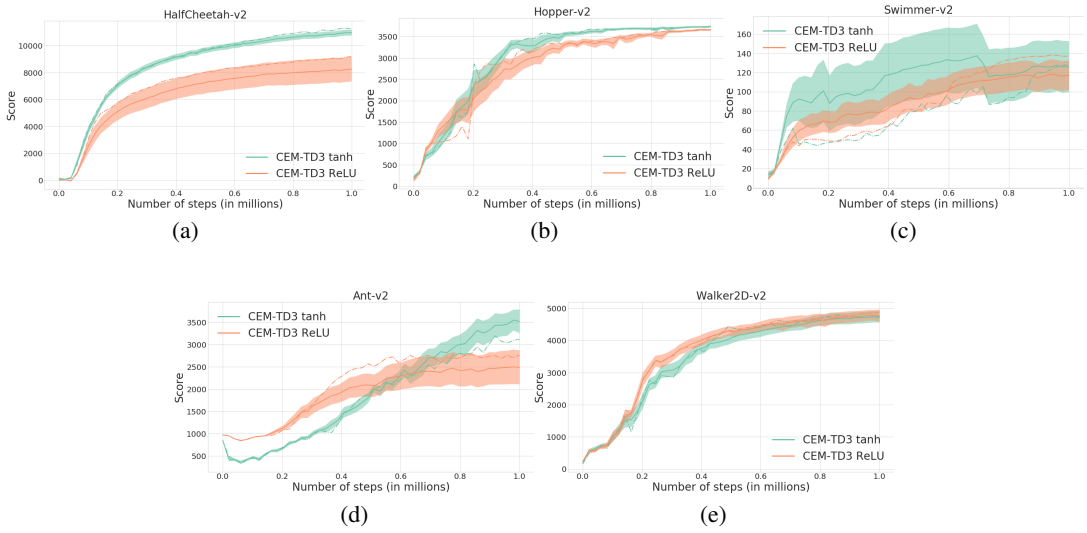
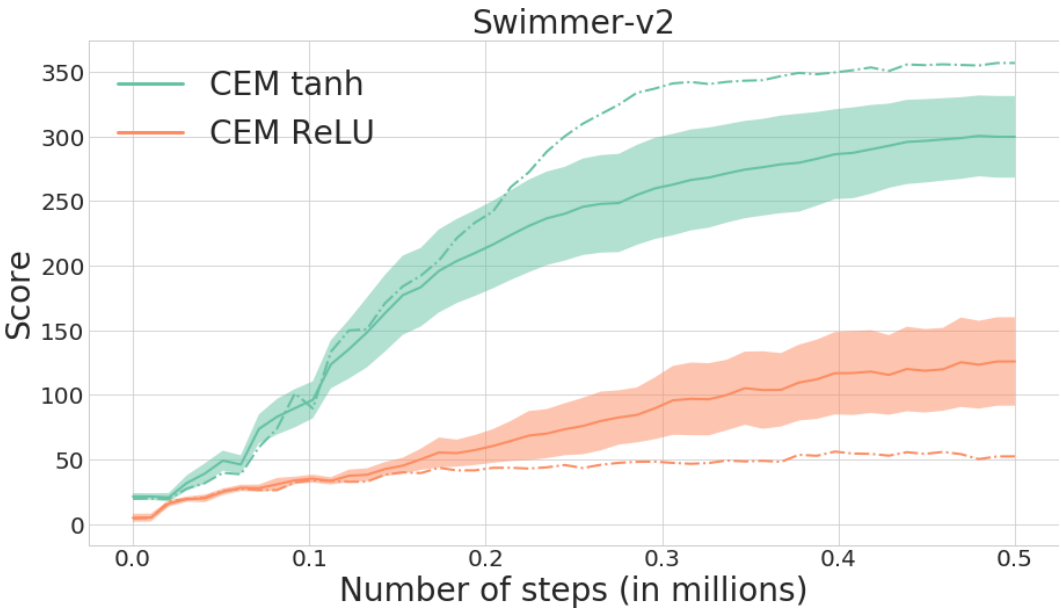


Figure 8: Learning curves of CEM-RL with tanh and RELU as non-linearities in the actors, on the HALF-CHEETAH-V2, HOPPER-V2, SWIMMER-V2, ANT-V2 and WALKER2D-V2 benchmarks. Dotted curves represent medians.



(a)

Figure 9: Learning curves of CEM with tanh and RELU as non-linearities in the actors, on the SWIMMER-V2 benchmark. Dotted curves represent medians.