DECENTRALIZED CROSS-ENTROPY METHOD FOR MODEL-BASED REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Cross-Entropy Method (CEM) is a popular approach to planning in model-based reinforcement learning. It has so far always taken a *centralized* approach where the sampling distribution is updated *centrally* based on the result of a top-k operation applied to *all samples*. We show that such a *centralized* approach makes CEM vulnerable to local optima and impair its sample efficiency, even in a one-dimensional multi-modal optimization task. In this paper, we propose **Decent**ralized **CEM (DecentCEM)** where an ensemble of CEM instances run independently from one another and each performs a local improvement of its own sampling distribution. In the exemplar optimization task, the proposed decentralized approach DecentCEM finds the global optimum much more consistently than the existing CEM approaches that use either a single Gaussian distribution or a mixture of Gaussians. Further, we extend the decentralized approach to sequential decision-making problems where we show in 13 continuous control benchmark environments that it matches or outperforms the state-of-the-art CEM algorithms in most cases, under the same budget of the total number of samples for planning.

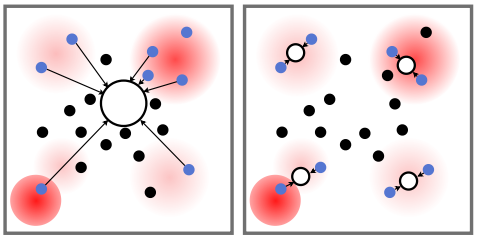
1 INTRODUCTION

Model-based reinforcement learning (MBRL) uses a model as a proxy of the environment for planning actions in multiple steps. This paper studies planning in MBRL with a specific focus on the Cross-Entropy Method (CEM) (De Boer et al., 2005), which is popular in MBRL due to its ease of use and strong empirical performance (Chua et al., 2018; Hafner et al., 2019; Wang & Ba, 2020; Zhang et al., 2021; Yang et al., 2020). CEM is a stochastic, derivative-free optimization method. It uses a sampling distribution to generate imaginary trajectories of environment-agent interactions with the model. These trajectories are then ranked based on their returns computed from the rewards given by the model. The sampling distribution is updated to increase the likelihood of producing the top-k trajectories with higher returns. These steps are iterated and eventually yield an improved distribution over the action sequences to guide the action execution in the real environment.

Despite the strong empirical performance of CEM for planning, it is prone to two problems: (1) lower sample efficiency as the dimensionality of solution space increases, and (2) the Gaussian distribution that is commonly used for sampling may cause the optimization to get stuck in local optima of multi-modal solution spaces commonly seen in real-world problems. Previous works addressing these problems either add gradient-based updates of the samples to optimize the parameters of CEM, or adopt more expressive sampling distributions, such as using Gaussian Mixture Model (Okada & Taniguchi, 2020) or masked auto-regressive neural network (Hakhamaneshi et al., 2020). Nevertheless, all CEM implementations to date are limited to a *centralized* formulation where the ranking step involves *all samples*. As analyzed below and in Section 3, such a centralized design makes CEM vulnerable to local optima and impairs its sample efficiency.

We propose **Decent**ralized **CEM (DecentCEM)** to address the above problems. Rather than ranking *all samples*, as in the *centralized* design, our method distribute the sampling budget across an ensemble of CEM instances. These instances run independently from one another, and each performs a local improvement of its own sampling distribution based on the ranking of its generated samples. The best action is then aggregated by taking an $\arg\max$ among the solution of the instances. It recovers the conventional CEM when the number of instance is one.

We hypothesize that by shifting to this *decentralized* design, CEM can be less susceptible to premature convergence caused by the *centralized* ranking step. As illustrated in Fig. 1, the *centralized* sampling distribution exhibits a bias toward the sub-optimal solutions near top right, due to the *global* top-k ranking. This bias would occur regardless of the family of distributions used. In comparison, a decentralized approach could maintain enough diversity thanks to its *local* top-k ranking in each sampling instance.



(a) Centralized CEM (b) Decentralized CEM

Figure 1: Illustration of CEM approaches in optimization. Shades of red indicate relative value of the 2D optimization landscape: brighter is better. Optimal solutions are near bottom left corner of the solution space. Blue dots ● are top-k samples, and black dots ● are other samples. Open dots ○ represent the sampling distributions with size of dots indicating number of generated samples.

Through a detailed analysis (Section 3) using a one-dimensional multi-modal optimization problem, we show that DecentCEM finds the global optimum much more consistently than centralized CEM approaches that use either a single Gaussian distribution or a mixture of Gaussians. We further apply DecentCEM to sequential decision making problems and use neural network to parameterize the sampling distribution in each CEM instance. Empirical results in commonly used continuous control benchmarks show that DecentCEM effectively improves the sample efficiency over the baseline CEM methods when using the same total number of samples for planning.

2 Preliminaries

We consider a Markov Decision Process (MDP) specified by (S,A,R,P,γ,d_0,T). $S \subset \mathcal{R}^{d_s}$ is the state space, $A \subset \mathcal{R}^{d_a}$ is the action space. $R : S \times A \to \mathbb{R}$ is the reward function that maps a state and action pair to a real-valued reward. $P(s'|s,a) : S \times A \times S \to [0,1]$ is the transition probability from a state and action pair s, a to the next state s'. $\gamma \in [0,1]$ is the discount factor. d_0 denotes the distribution of the initial state s_0. At time step t, the agent receives a state s_t [1] and takes an action a_t according to a policy $\pi(\cdot|s)$ that maps the state to a probability distribution over the action space. The environment transitions to the next state $s_{t+1} \sim P(\cdot|s_t, a_t)$ and gives a reward $r_t = R(s_t, a_t)$ to the agent [2]. The return $G_t = \sum_{i=0}^{T} \gamma^i r_{t+i}$, is the sum of discounted reward within an episode length of T. The agent aims to find a policy π that maximizes the expected return. We denote the learned model in MBRL as $f_\omega(\cdot|s,a)$, which is parameterized by ω and approximates $P(\cdot|s,a)$.

2.1 Planning with the Cross Entropy Method

Planning in MBRL is about leveraging the model to find the best action in terms of its return. Model-Predictive-Control (MPC) performs online planning at each time step up to a horizon to find the optimal action sequence:

$$\pi_{\text{MPC}}(s_t) = \underset{a_{t:t+H-1}}{\arg\max}\, \mathbb{E}[\Sigma_{i=0}^{H-1} \gamma^i r(s_{t+i}, a_{t+i}) + \gamma^H V(s_H)] \tag{1}$$

where H is the planning horizon, $a_{t:t+H-1}$ denotes the action sequence from time step t to $t+H-1$, and $V(s_H)$ is the terminal value function at the end of the planning horizon. The first action in this sequence is executed and the rest are discarded. The agent then re-plans at the next time step.

The Cross-Entropy Method (CEM) is a gradient-free optimization method that can be used for solving Eq. (1). The workflow is shown in Fig. 2. CEM planning starts by generating N samples $\{\tau_j\}_{j=1}^N = \{(\hat{a}_{j,0}, \hat{a}_{j,1}, \cdots, \hat{a}_{j,H-1})\}_{j=1}^N$ from an initial sampling distribution $g_\phi(\tau)$ parameterized by ϕ, where each sample τ_j is an action sequence from the current time step up to the planning horizon H. The domain of $g_\phi(\tau)$ has a dimension of $d_\tau = d_a H$.

[1] We assume full observability, i.e., we assume that the agent has access to the state.

[2] We assume that the agent receives the true reward. This makes the problem easier but is unfair to MBRL methods that do not assume this. Thus, we will limit comparison to methods that also make this assumption.

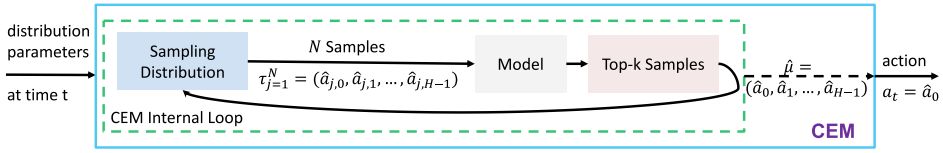


Figure 2: Cross Entropy Method (CEM) for Planning in MBRL

Using a model f, CEM generates imaginary rollouts based on the action sequence $\{\tau_j\}$ (in the case of a stochastic model) and estimate the associated value $v(\tau_j) = \mathbb{E}[\Sigma_{i=0}^{H-1}\gamma^i r(s_{j,i}, a_{j,i})]$ where $s_{j,0}$ is the current state s and $s_{j,i+1} \sim f(s_{j,i}, a_{j,i})$. The terminal value $\gamma^H V(s_{j,H})$ is omitted here following convention in the CEM planning literature but the MPC performance can be further improved if paired with an accurate value predictor (Bertsekas, 2005; Lowrey et al., 2019). The sampling distribution is then updated by fitting to the current top-k samples in terms of their value estimates $v(\tau_j)$, using the Maximum Likelihood Estimation (MLE) which solves:

$$\phi' = \arg\max_{\phi} \sum_{j=1}^{N} \mathbb{1}(v(\tau_j) \geq v_{\text{th}}) \log g_\phi(\tau_j) \tag{2}$$

where v_{th} is the threshold equal to the value of the k-th best sample and $\mathbb{1}(\cdot)$ is the indicator function. In practice, the update to the distribution parameters are smoothed by $\phi^{l+1} = \alpha\phi' + (1-\alpha)\phi^l$ where $\alpha \in [0, 1]$ is a smoothing parameter that balances between the solution to Eq. (2) and the parameter at the current internal iteration l.

CEM repeats this process of sampling and distribution update in an inner-loop, until it reaches the stopping condition: either a maximum number of iterations or the covariance of the distribution reaches a lower threshold. The output of CEM is an action sequence, typically set as the expectation[3] of the most recent sampling distribution for uni-modal distributions such as Gaussians $\hat{\mu} = \mathbb{E}(g_\phi) = (\hat{a}_0, \hat{a}_1, \cdots, \hat{a}_{H-1})$.

2.2 Choices of Sampling Distributions in CEM

A common choice of the sampling distribution in CEM is a multivariate Gaussian distribution under which Eq.(2) has a straight-forward analytical solution. But the uni-modal nature of Gaussian makes it inadequate in solving multi-modal optimization that often occur in MBRL. To increase the capacity of the distribution, a Gaussian Mixture Model (GMM) can be used (Okada & Taniguchi, 2020). We denote such an approach as *CEM-GMM*. Going forward, we use *CEM* to refer to the vanilla version that uses a Gaussian distribution. Computationally, the major difference between *CEM* and *CEM-GMM* is that distribution update in *CEM-GMM* involves solving for more parameters in Eq. (2). Detailed steps can be found in Okada & Taniguchi (2020).

3 Decentralized CEM

In this section, we first introduce the formulation of the proposed *decentralized* approach called the **Decent**ralized **CEM (DecentCEM)**. Then we illustrate the intuition behind the proposed approach using a one-dimensional synthetic multi-modal optimization example where we show the issues of the existing CEM methods and how they can be addressed by DecentCEM.

3.1 Formulation of DecentCEM

DecentCEM is composed of an ensemble of multiple CEM instances indexed by i, each having its own sampling distributions $g(\phi_i)$. They can be described by set of distribution parameters $\Phi = \{\phi_i\}_{i=1}^{M}$. Each instance i manages its own sampling and distribution update by the steps described in Section 2.1, independently from other instances. After the stopping condition is reached for all

[3]Other options are discussed in Appendix A.2

instances, the final sampling distribution is taken as the best distribution in the set Φ in terms of its top-k values:

$$\phi_{\text{DecentCEM}} = \arg\max_{\phi_i \in \Phi} V_{\phi_i} \approx \arg\max_{\phi_i \in \Phi} \sum_{j=1}^{\frac{N}{M}} \mathbb{1}(v(\tau_{i,j}) \geq v_{\text{th}}) v(\tau_{i,j}) \tag{3}$$

where $V(\phi_i)$ denotes the value of the sampling distribution g_{ϕ_i}, approximated by the top-$\frac{k}{M}$ values of the trajectories $\{\tau_{i,j}\}_{j=1}^{\frac{N}{M}}$ sampled from it. v_{th} is the threshold equal to the value of the $\frac{k}{M}$-th best sample. Note that the number of samples and elites are evenly split among the M instances. The key difference from the centralized approach is that the top-k sample sets are decentralized and managed by each instance independently whereas the centralized approach only keeps one set of top-k samples regardless of the distribution family used. When $M = 1$, it recovers the conventional CEM method.

3.2 Motivational Example

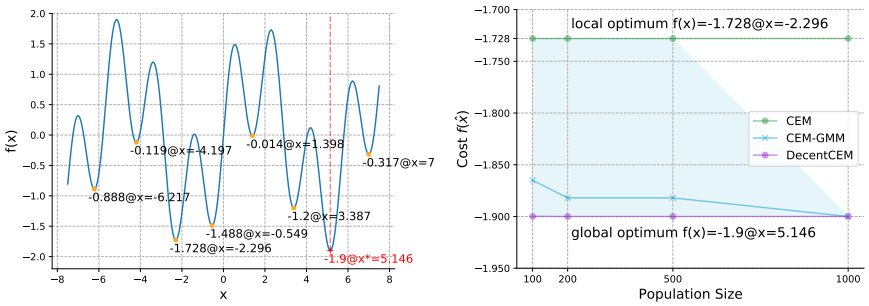


Figure 3: Left: The objective function in a 1D optimization task. Right: Comparison of our proposed *DecentCEM* method to *CEM* and *CEM-GMM*, wherein the line and the shaded region denote the mean and the min/max cost from 10 independent runs. $\hat{x}$: resulting solution of each method.

Consider a one dimensional multi-modal optimization problem shown in Fig.3 (Left). There are eight local optima, including one global optimum $f(x^*) = -1.9$ where $x^* = 5.146$. This objective function mimics the RL value landscape that has many local optima, as shown by Wang & Ba (2020). This optimization problem is "easy" in the sense that a grid search over the domain can get us a solution close to the global optimum. However, only our proposed *DecentCEM* method successfully converges to the global optimum consistently under varying population size (i.e., number of samples) and random runs, as shown in Fig.3 (Right)[4].

Both *CEM-GMM* and the proposed *DecentCEM* are equipped with multiple sampling distributions. The fact that *CEM-GMM* is outperformed by *DecentCEM* may appear surprising. To gain some insights, we illustrate in Fig. 4 how the sampling distribution evolves during the iterative update (more details in Fig. 9 in Appendix). *CEM* updated the unimodal distribution toward the local optimum despite seeing the global optimum. *CEM-GMM* appears to have a similar issue. During MLE on the top-k samples, it moved most distribution components towards the same local optimum which quickly lead to mode collapse. On the contrary, *DecentCEM* successfully escaped the local optima thanks to its independent distribution update over *decentralized* top-k samples and was able to maintain a decent diversity among the distributions.

GMM suits density estimation problems like distribution-based clustering where the samples are drawn from a *fixed true* distribution that can be represented by multi-modal Gaussians. However, in CEM for optimization, exploration is coupled with density estimation: the sampling distribution in CEM is *not fixed* but rather gets updated iteratively toward the top-k samples. And the "true" distribution in optimization puts uniform non-zero densities to the global optima and zero densities

[4]For a fair comparison, hyperparameter search has been conducted on all three methods for each population size (Appendix A).

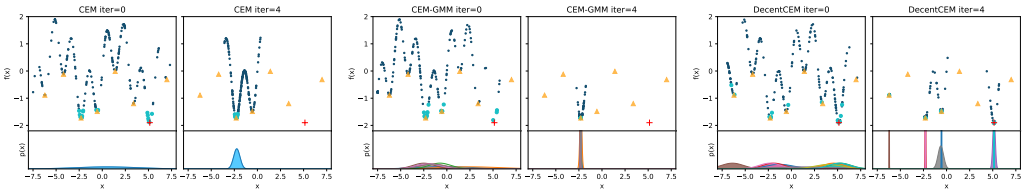


Figure 4: How the sampling distributions evolve in the 1D optimization task, *after* the specified iteration. Symbols include samples ●, elites ●, local optima ●, global +. 2nd row in each figure shows the weighted p.d.f of individual distribution. Population size: 200.

everywhere else. When there is a unique global optimum, it degenerates into a Dirac measure that assigns the entire density to the optimum. Density estimation of such a distribution only needs one Gaussian but the exploration is challenging. In other words, the conditions for GMM to work well are not necessarily met when used as the sampling distribution in *CEM*. *CEM-GMM* is subject to mode collapse during the iterative top-k greedification, causing premature convergence, as observed in Fig 4. In comparison, our proposed decentralized approach takes care of the exploration aspect by running multiple CEM instances independently, each performing its own local improvement. This is shown to be effective from this optimization example and the benchmark results in Section 6. *CEM-GMM* only consistently converge to the global optimum when we increase the population size to the maximum 1,000 which causes expensive computations. Our proposed *DecentCEM* runs more than 100 times faster than *CEM-GMM* at this population size, shown in Table A.3 in Appendix.

4 DECENTCEM FOR PLANNING IN MBRL

In this section, we develop two instantiations of DecentCEM for planning in MBRL where the sampling distributions are parameterized by policy networks.

4.1 CEM PLANNING WITH A POLICY NETWORK

In MBRL, CEM is applied to every state separately to solve the optimization problem stated in Eq. (1). The sampling distribution is typically initialized to a fixed distribution at the beginning of every episode (Okada & Taniguchi, 2020; Pinneri et al., 2020), or more frequently at every time step (Hafner et al., 2019). Such initialization schemes are sample inefficient since there is no mechanism that allows the information of the high-value region in the value space of one state to generalize to nearby states. Also, the information is discarded after the initialization. It is hence difficult to scale the approach to higher dimensional solution spaces, present in many continuous control environments. Wang & Ba (2020) proposed to use a policy network in CEM planning that helped to mitigate the issues above. They developed two methods: *POPLIN-A* that plans in the action space, and *POPLIN-P* that plans in the parameter space of the policy network. In *POPLIN-A*, the policy network is used to learn to output the mean of a Gaussian sampling distribution of actions. In *POPLIN-P*, the policy network parameters serve as the initialization of the mean of the sampling distribution of parameters. The improved policy network can then be used to generate an action. They show that when compared to the vanilla method of using a fixed sampling distribution in the action space, both modes of CEM planning with such a learned distribution perform better. The same principle of combining a policy network with CEM can be applied to the DecentCEM approach as well, which we will describe next.

4.2 DECENTCEM PLANNING WITH AN ENSEMBLE OF POLICY NETWORKS

For better sample efficiency in MBRL setting, we extend DecentCEM to use an ensemble of policy networks to learn the sampling distributions in the CEM instances. Similar to the *POPLIN* paper, we develop two instantiations of *DecentCEM*, namely *DecentCEM-A* and *DecentCEM-P*. The architecture of the proposed algorithm is illustrated in Fig. 5.

DecentCEM-A plans in the action space. It consists of an ensemble of policy networks followed by CEM instances. Each policy network takes the current state s_t as input, outputs

the parameters θ_i of the sampling distribution for CEM instance i. There is no fundamental difference from the DecentCEM formulation in Section 3.1 except that the initialization of sampling distributions is learned by the policy networks rather than a fixed distribution.

The second instantiation *DecentCEM-P* plans in the parameter space of the policy network. The initial sampling distribution is a Gaussian distribution over the policy parameter space with the mean at the current parameter values. In the $\arg\max$ operation in Eq. (3), the sample $\tau_{i,j}$ denotes the parameters of the policy network. Its value is obtained by computing the value of the action sequence generated from the policy network with the parameters $\tau_{i,j}$.

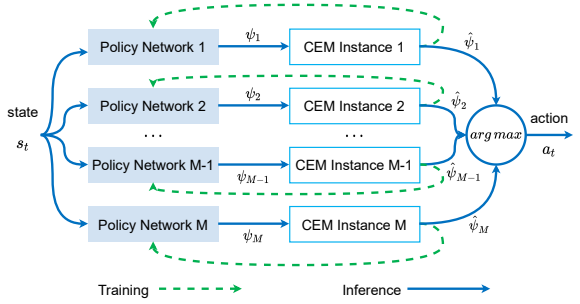


Figure 5: DecentCEM planning architecture. $\psi_i = \phi_i$ for planning in action space and $\psi_i = \theta_i$ for planning in policy network parameter space.

The ensemble of policy networks in both instantiations *DecentCEM-A* and *DecentCEM-P* are initialized with random weights, which is empirically found to be adequate to ensure that the output of the networks do not collapse into the same distribution (Sec.6.3 and Appendix F).

4.3 Training the Policy Network in DecentCEM

When planning in action space, the policy networks are trained by behavior cloning, similar to the scheme in *POPLIN* (Wang & Ba, 2020). Denote the first action in the planned action sequence at time step t by the i-th CEM instance as $\hat{a}_{t,i}$, the i-th policy network is trained to mimic $\hat{a}_{t,i}$ and the training objective is $\min_{\theta_i} \mathbb{E}_{s_t,\hat{a}_{t,i}\sim D_i} \|a_{\theta_i}(s_t) - \hat{a}_{t,i}\|^2$ where D_i denotes the replay buffer with the state and action pairs $(s_t, \hat{a}_{t,i})$. $a_{\theta_i}(s_t)$ is the action prediction at state s_t from the policy network parameterized by θ_i.

While the above training scheme can be applied to both planning in action space and parameter space, we follow the *setting parameter average* (AVG) (Wang & Ba, 2020) training scheme when planning in parameter space. The parameter is updated as $\theta_i = \theta_i + \frac{1}{|D_i|} \sum_{\delta_i \in D_i} \delta_i$ where $D_i = \{\delta_i\}$ is a dataset of policy network parameter updates planned from the i-th CEM instance previously. It is more effective than behavior cloning based on the experimental result reported by Wang & Ba (2020) and our own preliminary experiments.

Note that each policy network in the ensemble is trained independently from the data observed by its corresponding CEM instance rather than from the aggregated result after taking the $\arg\max$. This allows for enough diversity among the instances. More importantly, it increases the size of the training dataset for the policy networks compared to the approach taken in *POPLIN*. For example, with an ensemble of M instances, there would be M training data samples available from one real environment interaction, compared to the one data sample in *POPLIN-A/P*. As a result, *DecentCEM* is able to use larger policy networks than is otherwise possible, shown in Sec. 6.3 and Appendix F.

5 Related Work

There are three main approaches to planning for MBRL (Wang et al., 2019). One approach is Dyna (Sutton, 1990), where the model is simply used for data augmentation and the policy is learned just like in model-free RL. A second approach learns the policy by taking the gradient of the learning objective w.r.t the policy parameters and backpropagating through the model (Heess et al., 2015; Amos et al., 2021). In the third approach, the policy are learned by sampling multi-step actions. The model is used to generate imaginary experiences to estimate the return of the sampled action sequences. We limit our scope to the third approach to planning, which is more flexible in that it does not assume a differentiable model and can deal with long planning horizons.

Vanilla CEM planning in action space with a single Gaussian distribution has been adopted as the planning method for both simulated and real-world robot control (Chua et al., 2018; Finn & Levine, 2017; Ebert et al., 2018; Hafner et al., 2019; Yang et al., 2020; Zhang et al., 2021). Among previous attempts to improve the performance of CEM-based planning, we see two types of approaches. The first type includes CEM in a hybrid of CEM+X where "X" is some other component or algorithm. POPLIN (Wang & Ba, 2020) is a prominent example where "X" is a policy network that learns a state conditioned distribution that initializes the subsequent CEM process. This addition of the policy network allows the CEM to search in the network parameter space which is shown to have a smoother landscape and better exploration. Another common choice of "X" is gradient-based adjustment of the samples drawn in CEM. GradCEM (Bharadhwaj et al., 2020) adjusts the samples in each iteration of CEM by taking gradient ascent of the return estimate w.r.t the actions. The benefit that this method brings is not significant on benchmark control tasks. CEM-RL (Pourchot & Sigaud, 2019) also combines gradient steps with CEM but the samples are in the parameter space of the actor network. To improve computational efficiency, Lee et al. (2020) proposes an asynchronous version of CEM-RL where each CEM instance updates the sampling distribution asynchronously without waiting for other instances to finish. The downside with both version of CEM-RL methods are that they rely on model-free RL algorithms.

The second type of approach aims at improving CEM itself. Amos & Yarats (2020) proposes a fully-differentiable version of CEM called *DCEM*. The key is to make the top-k selection in CEM differentiable such that the entire CEM module can be trained in an end-to-end fashion. Despite cutting down the number of samples needed in CEM, this method does not beat the vanilla CEM in benchmark test. *GACEM* (Hakhamaneshi et al., 2020) increase the capacity of the sampling distribution by replacing the Gaussian distribution with an auto-regressive neural network. This change allows CEM to perform search in multi-modal solution space but it is only verified in toy examples and its computation seems too high to be scaled to MBRL tasks. Another method that increases the capacity of the sampling distribution is *PaETS* (Okada & Taniguchi, 2020) that uses a GMM with CEM. It is the approach that we followed for our *CEM-GMM* implementation. It is not clear how well it performs in benchmark tasks since their environment setup is modified to have a range of actions 5 times larger than the original. Also the running time results in the optimization task in Sec.3.2 shows that it is computationally heavier than the *CEM* and *DecentCEM* methods, limiting its use in complex environments. Overall, this second type of approach did not outperform vanilla CEM, a situation that motivated our move to a decentralized formulation.

6 Experiments

We evaluate the proposed *DecentCEM* methods in simulated environments with continuous action space. The experimental evaluation is mainly setup to understand if *DecentCEM* improves the performance and sample efficiency over conventional CEM approaches.

6.1 Benchmark Setup

We benchmark the algorithms in several continuous-action control environments in OpenAI Gym.

Environments We run the benchmark in a set of 13 environments commonly used in the MBRL literature: Pendulum, InvertedPendulum, Cartpole, Acrobot, FixedSwimmer[5], Reacher, Hopper, Walker2D, HalfCheetah, PETS-Reacher3D, PETS-HalfCheetah, PETS-Pusher, Ant. The three environments prefixed by "PETS" are proposed by Chua et al. (2018). Note that MBRL algorithms often make different assumptions about the dynamics model or the reward function. Their benchmark environments are often modified from the original OpenAI gym environments such that the respective algorithm is runnable. Whenever possible, we inherit the same environment setup from that of the respective baseline methods. This is so that the comparison against the baselines is fair. More details on the environments and their reward functions are in Appendix B.

Algorithms The baseline algorithms are *PETS* (Chua et al., 2018) and *POPLIN* (Wang & Ba, 2020). *PETS* uses CEM with a single Gaussian distribution for planning. The *POPLIN* algorithm combines a single policy network with CEM. As described in Sec.4.1, *POPLIN* comes with two modes:

[5]a modified version of the original Gym Swimmer environment where the velocity sensor on the neck is moved to the head. This fix was proposed by Wang & Ba (2020)

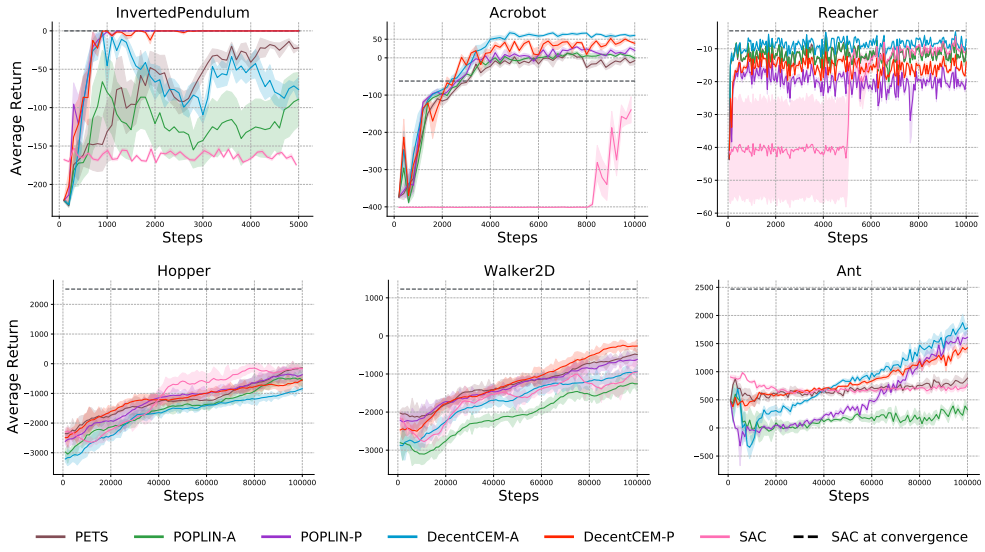


Figure 6: The learning curves of the proposed *DecentCEM* methods and the baseline methods on continuous control environments. The line and shaded region shows the mean and standard error of evaluation results from 5 training runs using different random seeds. Each run is evaluated per training episode in an environment independent from training and reports average return of 5 episodes.

POPLIN-A and *POPLIN-P* with the suffix "A" denotes planning in action space and "P" for the network parameter space. We reuse the default hyperparameters for these algorithms from the original papers if not mentioned specifically. The detailed hyperparameters are listed in the Appendix D.2. For our proposed methods, we include two variations *DecentCEM-A* and *DecentCEM-P* as described in Sec. 4.2 where the suffix carries the same meaning as in *POPLIN-A/P*. All MBRL algorithms studied in this benchmark uses the same ensemble networks proposed by Chua et al. (2018) for the dynamics model learning. We also include a Model-Free RL baseline SAC (Haarnoja et al., 2018) and show its finite-time and asymptotic result.

Evaluation Protocol The learning curve shows the mean and standard error of the test performance out of 5 independent training runs. The test performance is an average return of 5 episodes of the evaluation environment, evaluated at every training episode. At the beginning of each training run, the evaluation environment is initialized with a fixed random seed such that the evaluation environments are consistent across different methods and multiple runs to make it a fair comparison. All experiments were conducted using Tesla V100-PCIE-16GB GPUs.

6.2 Results

The learning curves of the algorithms are shown in Fig. 6 for InvertedPendulum, Acrobot, Reacher, Hopper, Walker2D and Ant, sorted by the difficulty of task. The full results for all environments are included in Appendix E.

We can observe two main patterns from the results. One pattern is that in most environments, the *DecentCEM* methods either match or outperform their counterpart that takes a centralized approach. In fact *DecentCEM* can be seen as a generalization of *POPLIN* by adding a dimension of policy ensemble size, with size one recovering *POPLIN*. It allows for fine-tuning CEM for individual domain. We also included negative results shown in Walker2D where neither *DecentCEM* modes outperform the baselines. Also all model-based methods underperform the model-free method SAC, suggesting the difficulty of model learning. The other pattern is that using policy networks to learn the initial sampling distribution in general helps improving the performance of CEM with both centralized and decentralized formulation. This is expected as discussed in Sec.4.1 since the policy network allows the sampling distribution to "resume" from high-value region seen before and to generalize to similar states.

6.3 ABLATION STUDY

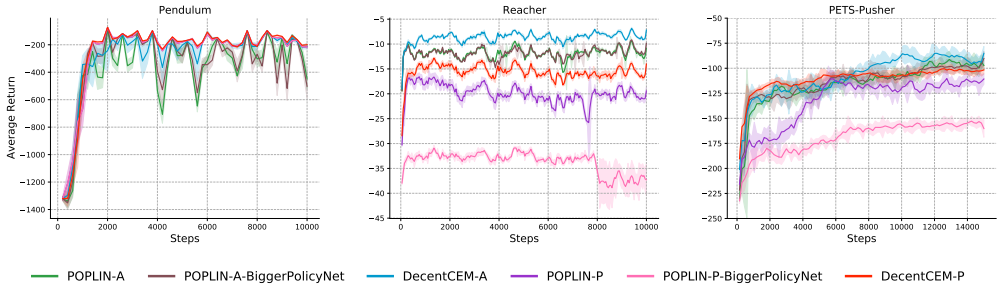


Figure 7: Ablation study on the policy network size where *POPLIN-A&P* have a bigger policy network equivalent in the total number of neural network weights to their *DecentCEM* counterparts. For better visual clarity, curves are smoothed with a sliding window of size 10.

A natural question to ask about the *DecentCEM-A/P* methods is whether the increased performance is from the larger number of neural network parameters. We add two varations of the *POPLIN* baselines where a bigger policy network is used. The number of the network parameters is equivalent to that of the ensemble of policy networks in the proposed *DecentCEM*. We show the comparison using three environments in Fig. 7: Pendulum(1), Reacher(2) and PETS-Pusher(7) (with action dimension in parenthesis). In both action space planning and parameter space planning, using a bigger policy network in *POPLIN* either does not help or can significantly impair the performance (see the *POPLIN-P* results in both reacher and PETS-Pusher). This is expected since unlike our DecentCEM methods, the training data in *POPLIN* do not scale with the size of the policy network, as explained in Sec. 4.3.

Figure 8 (Left) shows the cumulative selection ratio of each CEM instance during training of *DecentCEM-A* with an ensemble size of 5. It suggests that the random initialization of the policy network is sufficient to avoid mode collapse. We also plot the action statistics of the instances in Figure 8 (Right). The line and shaded area represent the mean and max/min action of the instances, respectively. For visual clarity, we show a time segment toward the end of the training rather than all the 10k steps. DecentCEM has maintained enough diversity in the instances even toward the end of the training. *DecentCEM-P* is excluded from both plots since it shows a similar trend as *DecentCEM-A*. More ablations results are included in Appendix F.

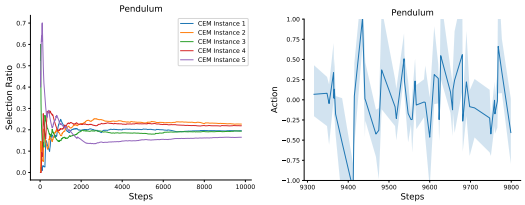


Figure 8: Ablation of ensemble diversity

7 CONCLUSION

In this paper, we study CEM planning in the context of continuous-action MBRL. We propose a novel *decentralized* formulation of CEM named *DecentCEM*, which generalizes CEM to run multiple independent instances and recovers the conventional CEM when the number of instances is one. We illustrate the intuition and the strengths of the proposed DecentCEM approach in a motivational one-dimensional optimization task and show how it fundamentally differs from the CEM approach that uses a Gaussian or GMM. We extend the proposed approach to MBRL by instantiating two decentralized CEM methods that combine with policy networks. We show the efficacy of the proposed methods in benchmark control tasks and ablations studies.

REPRODUCIBILITY STATEMENT

We have included the implementation details in Appendix D and the source code in the supplementary materials.

REFERENCES

Brandon Amos and Denis Yarats. The differentiable cross-entropy method. In *International Conference on Machine Learning*, pp. 291–302. PMLR, 2020.

Brandon Amos, Samuel Stanton, Denis Yarats, and Andrew Gordon Wilson. On the model-based stochastic value gradient for continuous reinforcement learning. In *Learning for Dynamics and Control*, pp. 6–20. PMLR, 2021.

Dimitri P Bertsekas. Dynamic programming and optimal control 3rd edition, volume i. *Belmont, MA: Athena Scientific*, 2005.

Homanga Bharadhwaj, Kevin Xie, and Florian Shkurti. Model-predictive control via cross-entropy and gradient-based optimization. In *Learning for Dynamics and Control*, pp. 277–286. PMLR, 2020.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, pp. 4759–4770, 2018.

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

Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2786–2793. IEEE, 2017.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 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.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pp. 2555–2565. PMLR, 2019.

Kourosh Hakhamaneshi, Keertana Settaluri, Pieter Abbeel, and Vladimir Stojanovic. Gacem: Generalized autoregressive cross entropy method for multi-modal black box constraint satisfaction. *arXiv preprint arXiv:2002.07236*, 2020.

Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. *Advances in Neural Information Processing Systems*, 28:2944–2952, 2015.

Kyunghyun Lee, Byeong-Uk Lee, Ukcheol Shin, and In So Kweon. An efficient asynchronous method for integrating evolutionary and gradient-based policy search. *Advances in Neural Information Processing Systems*, 33, 2020.

Kendall Lowrey, Aravind Rajeswaran, Sham Kakade, Emanuel Todorov, and Igor Mordatch. Plan online, learn offline: Efficient learning and exploration via model-based control. In *International Conference on Learning Representations, ICLR*, 2019.

Masashi Okada and Tadahiro Taniguchi. Variational inference mpc for bayesian model-based reinforcement learning. In *Conference on Robot Learning*, pp. 258–272. PMLR, 2020.

Cristina Pinneri, Shambhuraj Sawant, Sebastian Blaes, Jan Achterhold, Joerg Stueckler, Michal Rolinek, and Georg Martius. Sample-efficient cross-entropy method for real-time planning. *arXiv preprint arXiv:2008.06389*, 2020.

Aloïs Pourchot and Olivier Sigaud. CEM-RL: combining evolutionary and gradient-based methods for policy search. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9*, 2019.

Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *In Proceedings of the Seventh International Conference on Machine Learning*, pp. 216–224. Morgan Kaufmann, 1990.

Tingwu Wang and Jimmy Ba. Exploring model-based planning with policy networks. In *8th International Conference on Learning Representations, ICLR, Addis Ababa, Ethiopia, April 26-30*, 2020.

Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, Shunshi Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *CoRR*, abs/1907.02057, 2019.

Yuxiang Yang, Ken Caluwaerts, Atil Iscen, Tingnan Zhang, Jie Tan, and Vikas Sindhwani. Data efficient reinforcement learning for legged robots. In *Conference on Robot Learning*, pp. 1–10. PMLR, 2020.

Baohe Zhang, Raghu Rajan, Luis Pineda, Nathan Lambert, André Biedenkapp, Kurtland Chua, Frank Hutter, and Roberto Calandra. On the importance of hyperparameter optimization for model-based reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 4015–4023. PMLR, 2021.

A DETAILS OF THE MOTIVATIONAL EXAMPLE

A.1 SETUP AND RUNNING TIME

The hyperparameters for all three methods *CEM*, *CEM-GMM* and *DecentCEM* are summarized in Table A.1 and the best performing hyperparameters for each method are shown in Table A.2. The running time are included in Table A.3.

Table A.1: Hyperparameters

Algorithm	Parameter	Value
Shared Parameters	Total Population Size	100, 200, 500, 1000
	Elite Ratio	0.1
	α: Smoothing Ratio	0.1
	ϵ: Minimum Variance Threshold	1e-3
	Maximum Number of Iterations	100
CEM-GMM	M: Number of Mixture Components	3,5,8,10
	κ: Weights of Entropy Regularizer	0.25, 0.5
	r: Return Mode	's': mean of the mixture component sampled based on their weights 'm': mean of the component that achieves the minimum cost
DecentCEM	E: Number of instances in the Ensemble	3,5,8,10

Table A.2: Best Hyper-Parameter

	Total Population Size			
	100	200	500	1000
CEM-GMM	$M = 10$ $\kappa = 0.25$, $r =$'m'	$M = 8$ $\kappa = 0.5$ $r =$'m'	$M = 8$ $\kappa = 0.25$ $r =$'m'	$M = 8$ $\kappa = 0.5$ $r =$'s'
DecentCEM	$E = 10$	$E = 10$	$E = 10$	$E = 8$

Table A.3: Total Time of 10 Runs (in seconds)

	Total Population Size			
	100	200	500	1000
CEM	0.079	0.093	0.165	0.318
CEM-GMM	7.322	11.500	24.431	59.844
DecentCEM	0.407	0.420	0.506	0.545

A.2 OUTPUT OF CEM APPROACHES

In terms of the output of CEM approaches, there exist different options in the literature. The most common option is to return the sample in the domain that corresponds to the highest probability density in the final sampling distribution. It is the mean in the case of Gaussian and the mode with the highest probability density in the case of GMM. One can also draw a sample from the final sampling distribution (Okada & Taniguchi, 2020) and return it. Another option is to return the best sample observed (Pinneri et al., 2020). The best option among the three may be application dependent. It has been observed that in many applications, the sequence of sampling distributions numerically converges to a deterministic one (De Boer et al., 2005), in which case the first two options are identical.

B BENCHMARK ENVIRONMENT DETAILS

In this section, we go over the details of the benchmark environments used in the experiments.

Table B.1 lists the environments along with their properties, including the dimensionality of the observation and action spaces and the maximum episode length. Table B.2 provides a list of the reward function for each environment. Whenever possible, we reuse the original implementations from the literature as noted in Table B.1 so as to avoid confusion. The environments that start with "PETS" are from the PETS paper (Chua et al., 2018), which is one of the baseline methods. Most other environments are from Wang et al. (2019) where the dynamics are the same as the OpenAI gym version and the reward function in Table B.1 is exposed to the agent. For more details of the environments, the readers are referred to the original paper.

Note that FixedSwimmer is a modifed version of the original Gym Swimmer environment where the velocity sensor on the neck is moved to the head. This fix was originally proposed by Wang & Ba (2020). For the Pendulum environment, we use the OpenAI Gym version. The modified version in (Wang et al., 2019) uses a different reward function which we have found to be incorrect.

Table B.1: The setup of the environments. The number in the bracket in the "Environment Name" column denotes the source of this environment: [1] refers to the benchmark paper from Wang et al. (2019); [2] denotes PETS (Chua et al., 2018).

Environment Name	Observation Dim	Action Dim	Maximum Episode Length
Pendulum	3	1	200
InvertedPendulum [1]	4	1	100
Cartpole [1]	4	1	200
Acrobot [1]	6	1	200
FixedSwimmer [1]	9	2	1000
Reacher [1]	11	2	50
Hopper [1]	11	3	1000
Walker2d [1]	17	6	1000
HalfCheetah [1]	17	6	1000
PETS-Reacher3D [2]	17	7	150
PETS-HalfCheetah [2]	18	6	1000
PETS-Pusher [2]	20	7	150
Ant [1]	27	8	1000

Table B.2: Reward Functions. $\mathbf{d}_t$ denotes the vector between the end effector to the target position. z_t denotes the height of the robot. $\|\mathbf{v}\|_1$ and $\|\mathbf{v}\|_2$ denote the 1-norm and 2-norm of vector $\mathbf{v}$, respectively. In PETS-Pusher, $\mathbf{d}_{1,t}$ is the vector between the object position and the goal and $\mathbf{d}_{2,t}$ denotes the vector between the object position and the end effector.

Environment Name	Reward Function
Pendulum	$\theta_t^2 + 0.1\dot{\theta_t}^2 + 0.001a_t^2$
InvertedPendulum	$-\theta_t^2$
Cartpole	$cos\theta_t - 0.01x_t^2$
Acrobot	$-cos\theta_{1,t} - cos(\theta_{1,t} + \theta_{2,t})$
FixedSwimmer	$\dot{x}_t - 0.0001\|\mathbf{a}_t\|_2^2$
Reacher	$-\|\mathbf{d}_t\| - \|\mathbf{a}_t\|_2^2$
Hopper	$\dot{x}_t - 0.1\|\mathbf{a}_t\|_2^2 - 3(z_t - 1.3)^2$
Walker2d	$\dot{x}_t - 0.1\|\mathbf{a}_t\|_2^2 - 3(z_t - 1.3)^2$
HalfCheetah	$\dot{x}_t - 0.1\|\mathbf{a}_t\|_2^2$
PETS-Reacher3D	$-\|\mathbf{d}_t\|_2^2 - 0.01\|\mathbf{a}_t\|_2^2$
PETS-HalfCheetah	$\dot{x}_t - 0.1\|\mathbf{a}_t\|_2^2$
PETS-Pusher	$-1.25\|\mathbf{d}_{1,t}\|_1 - 0.5\|\mathbf{d}_{2,t}\|_1 - 0.1\|\mathbf{a_t}\|_2^2$
Ant	$\dot{x}_t - 0.1\|\mathbf{a}_t\|_2^2 - 3(z_t - 0.57)^2$

C ALGORITHMS

In this section, we give the pseudo-code of the proposed algorithms DecentCEM-A and DecentCEM-P in Algorithm 1 and 2 respectively. We only show the training phase. The algorithm at inference time is simply the same process without the data saving and network update. For the internal process of CEM, we refer the readers to De Boer et al. (2005); Wang & Ba (2020).

Algorithm 1: DecentCEM-A Training

Initialize the policy networks p_i with $\theta_i, i = 1, 2, \cdots, M$ where M is the ensemble size. Planning horizon H. Initialize the dynamics network f_ω parameterized by ω. Empty Datasets D_m and D_p
// Episode 1, warmup phase
Rollout using a random policy, fill the dataset D_m with the transition data $\{(s_t, a_t, s_{t+1})\}$
Update ω using D_m by Mean-Squared Loss // Train the dynamics network with D_m
// Episode 2 onwards
repeat
  $t = 0$, $D_p = \{\}$ // Each episode, reset time and dataset
  repeat
    foreach *policy network p_i in the ensemble* **do**
      generate reference mean of action sequence distribution μ_i using p_i and the model f_ω.
      // Apply CEM to refine the action distribution.
      // $\hat{\mu}_i$, v_i are the mean action sequence of the refined distribution and its expected value
      $\hat{\mu}_i, v_i = \text{CEM}(\mu_i)$
      $\hat{a}_{t,i} = \hat{\mu}_i[0]$
    end
    $a_t = \arg\max_{\hat{a}_{t,i}} v_i$ // Pick best distribution
    $s_{t+1} = step(a_t)$ // Execute the first action in the mean sequence
    Append the transition (s_t, a_t, s_{t+1}) to D_m
    Append the data $\{(s_t, \hat{a}_{t,i})\}_{i=1}^M$ to D_p
    $t = t + 1$ // Update time step
  until *Either reached the maximum episode length or terminal state*
  Update the model parameter ω using dataset D_m
  Update the policy network weights $\{\theta_i\}_{i=1}^M$ using dataset D_p by the behavior cloning objective
until *bored*

Algorithm 2: DecentCEM-P Training

Initialize the policy networks p_i with $\theta_i, i = 1, 2, \cdots, M$ where M is the ensemble size. Planning horizon H. Initialize the dynamics network f_ω parameterized by ω. Empty Datasets D_m and D_p
// Episode 1, warmup phase
Rollout using a random policy, fill the dataset D_m with the transition data $\{(s_t, a_t, s_{t+1})\}$
Update ω using D_m by Mean-Squared Loss // Train the dynamics network with D_m
// Episode 2 onwards
repeat
  $t = 0$, $D_p = \{\}$ // Each episode, reset time and dataset
  repeat
    foreach *policy network p_i in the ensemble* **do**
      // Apply CEM to refine the distribution of the neural network weight.
      // $\hat{\mu}_i$, v_i are the mean of the refined weight distribution sequence and its expected value
      $\hat{\mu}_i, v_i = \text{CEM}(\theta_i)$
      $\delta_i = \hat{\mu}_i[0]$ // Keep the weight at the first step and discard the rest
    end
    $\theta_t = \arg\max_{\theta_i + \delta_i} v_i$ // Pick the best distribution of weight sequence
    $a_t = p_{\theta_t}(s_t)$
    $s_{t+1} = step(a_t)$ // Execute the action returned by the policy network p_{θ_t}
    Append the transition (s_t, a_t, s_{t+1}) to D_m
    Append the data $\{\delta_i\}_{i=1}^M$ to D_p
    $t = t + 1$ // Update time step
  until *either reached the maximum episode length or terminal state*
  Update the model parameter ω using dataset D_m
  Update the policy network weights $\{\theta_i\}_{i=1}^M$ using dataset D_p by the AVG training objective
until *bored*

D IMPLEMENTATION DETAILS

D.1 REPRODUCIBILITY

Our implementation is fully reproducible by identifying the sources of randomness and controlling the random seeds as summarized in Table D.1. The seeds are set once at the beginning of the experiments.

Table D.1: Random Seed. The set {1,2,3,4,5} refers to the seeds for five runs. Note that we control the random seed for the environments since there is a random number generator in openai gym environments independent from other sources

Source of randomness	Random Seed
deep learning framework (tensorflow in our case)	{1,2,3,4,5}
numpy	
python random module	
the training environment	1234
the evaluation environment	0

D.2 HYPERPARAMETERS

This section includes the details of the key hyperparameters used in the proposed *DecentCEM* algorithm (Table D.5) and the baseline algorithms *PETS* (Table D.3), *POPLIN* (Table D.4) and *SAC*[6] (Table D.2). For the neural network architecture for the dynamics model, the *DecentCEM* methods exactly follow the original one in *PETS* and *POPLIN* for a fair comparison, which is an ensemble of fully connected networks.

Table D.2: Hyperparameters of SAC

Parameter	Value
Actor learning rate	0.0001
Critic learning rate	0.0001
Actor network architecture	[dim(observation), 64, 64, 2× dim(action)]
Critic network architecture	[dim(observation)+dim(action), 64, 64, 1]

Table D.3: Hyperparameters of PETS

Parameter	Value
Model learning rate	0.001
Warmup episodes	1
Planning Horizon	30
CEM population size	500 (except in PETS-reacher3D: 400)
CEM proportion of elites	10%
CEM initial distribution variance	0.25
CEM max number of internal iterations	5

[6]Our SAC implementation used network architectures that are similar to the policy network in our method. The results of our implementation either matches or surpasses the ones reported in *PETS*, *POPLIN* and the benchmark by Wang et al. (2019)

Table D.4: Hyperparameters of POPLINA and POPLINP

Parameter	Value
Model learning rate	0.001
Warmup episodes	1
Planning Horizon	30
CEM population size	500 (except in PETS-reacher3D: 400)
CEM proportion of elites	10%
CEM initial distribution variance	0.25
CEM max number of internal iterations	5
Policy network architecture (A)	[dim(observation), 64, 64, dim(action)]
Policy network architecture (P)	[dim(observation), 32, dim(action)]
Policy network learning rate	0.001
Policy network activation function	tanh

Table D.5: Hyperparameters of the proposed DecentCEM-A/P

Parameter	Value
Model learning rate	0.001
Warmup episodes	1
Planning Horizon	30
Ensemble Size	5
CEM population size in each instance	100 (except in PETS-reacher3D: 80)
CEM proportion of elites	10%
CEM initial distribution variance	0.25
CEM max number of internal iterations	5
Policy network architecture (A)	[dim(observation), 64, 64, dim(action)]
Policy network architecture (P)	[dim(observation), 32, dim(action)]
Policy network learning rate	0.001
Policy network activation function	tanh

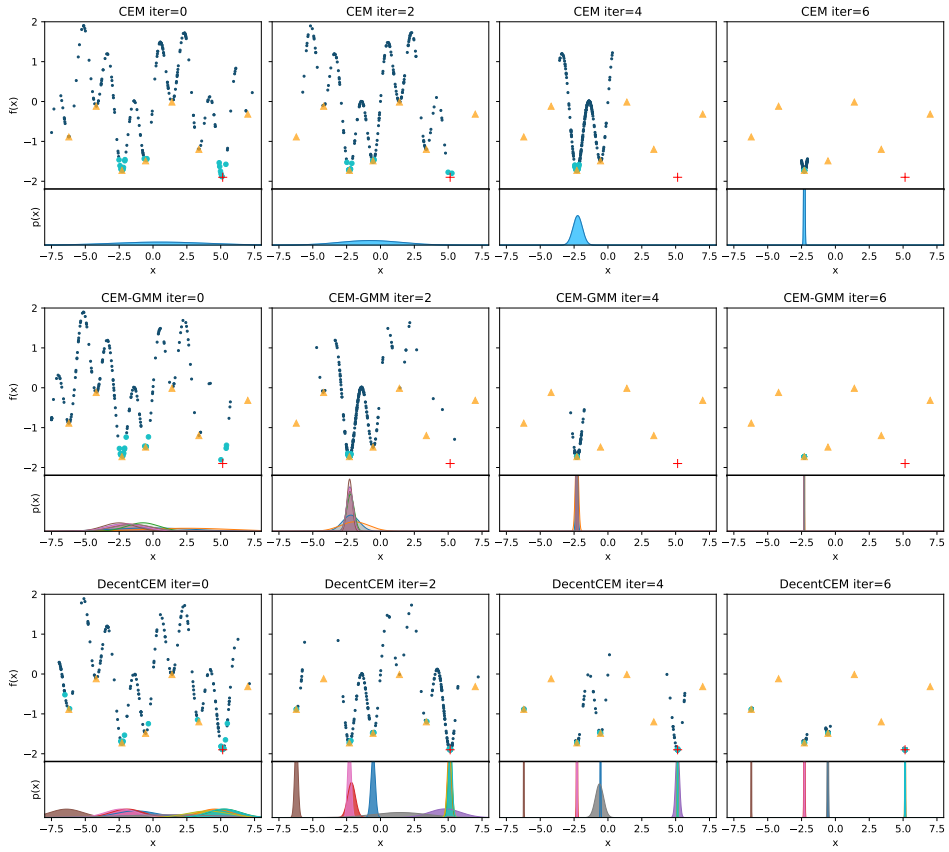


Figure 9: The iterative sampling process in the 1D optimization task.

E Full Results

E.1 Detailed visualization of the iterative updates in the 1D optimization task

Figure 9 is a version of Figure 4 with more iterations. It shows the iterative sampling process of CEM methods in the 1D optimization task and how the sampling distribution evolve over time.

E.2 Full Learning Curves

In Figure 10, we report the learning curves in all 13 environments that we listed in Appendix B.

The algorithms used in the benchmark are: PETS, POPLIN-A, POPLIN-P and the two modes of the proposed method DecentCEM-A and DecentCEM-P. We also included the interim and asymptotic performance of a model-free algorithm SAC as a reference.

The learning curves in some environments can be noisy. We apply smoothing with 1D uniform filter to the results of the following environments for easier interpretation: Cartpole, FixedSwimmer, Hopper, Walker2d, HalfCheetah, PETS-Reacher3D, PETS-Pusher. The window size of the filter was 10 for all but Cartpole, where 30 was used due to its high noise level for all algorithms.

Note that the performance of the baseline methods may be different from the results reported in their original paper. Specifically, in the paper by Wang & Ba (2020), PETS, POPLIN-A and POPLIN-P have been evaluated in a number of environments that we use for the benchmark. Our benchmark results may not be consistent with theirs due to differences in the implementation and evaluation protocol. For example, our results of PETS, POPLIN-A and POPLIN-P in the Acrobot environment are all better than the results in Wang & Ba (2020). We have identified a bug in the POPLIN code base that causes the evaluation results to be on a wrong timescale that is much slower than what it

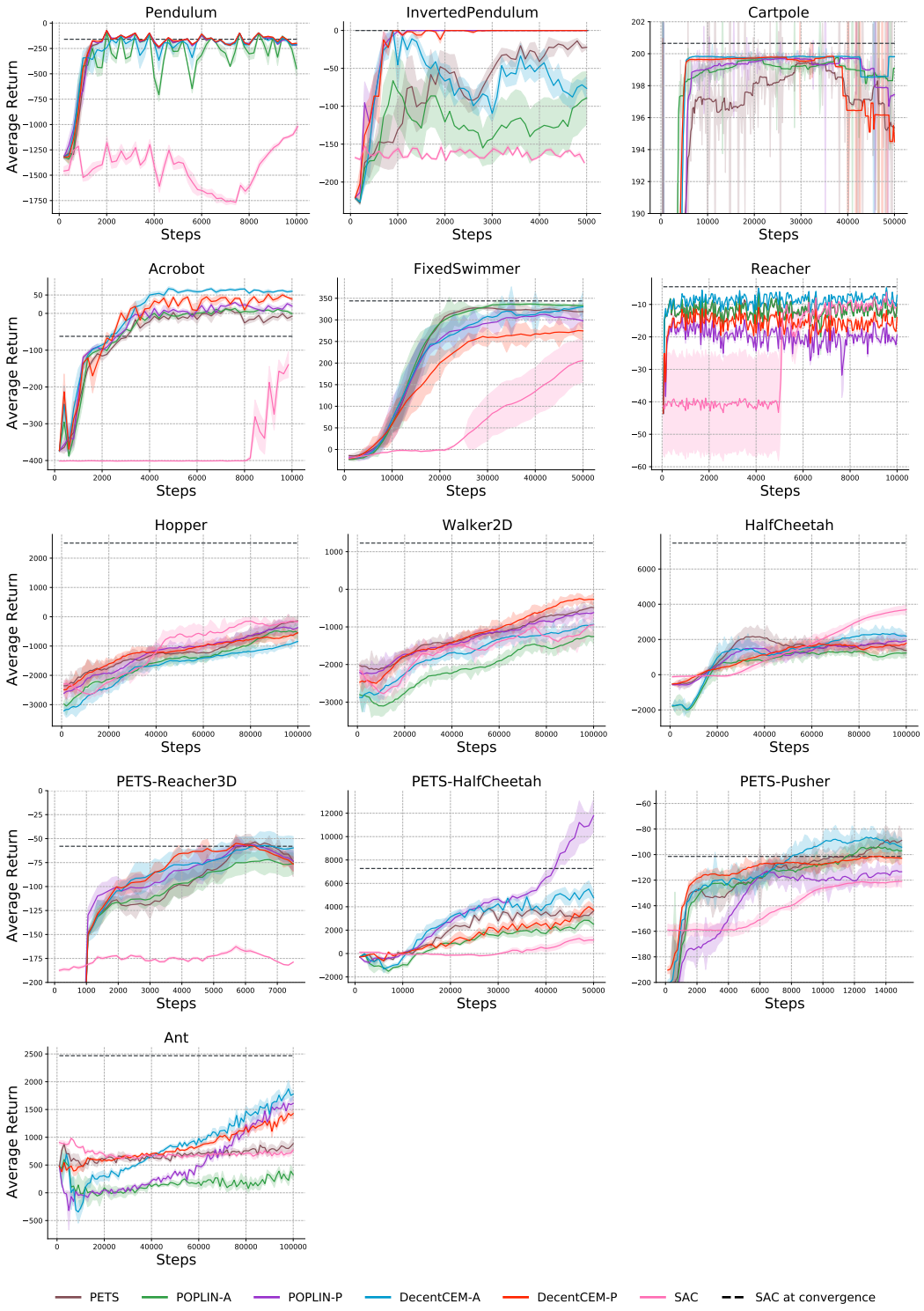
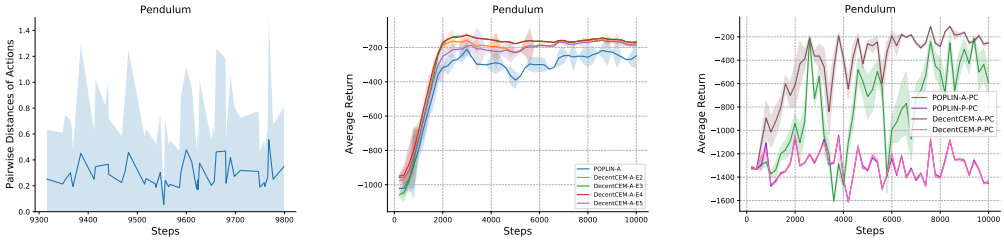


Figure 10: The learning curves of the proposed *DecentCEM* methods and the baseline methods on continuous control environments. The line and shaded region shows the mean and standard error of evaluation results from 5 training runs using different random seeds. Each run is evaluated in an environment independent from training and reports average return of 5 episodes at every training episode. To ensure that the evaluation environments are the same across different methods and multiple runs, we set a fixed random seed in the evaluation environment of each task.



(a) Action Distance (b) Ensemble Size (c) Policy Control

Figure 11: More Ablation **(a)** Pairwise distance between the actions of 5 instances in *DecentCEM-A* during training **(b)** Ensemble size ablation: E2 denotes an ensemble size of 2 **(c)** Policy control performance where the policy network is directly used for control without CEM policy improvement

actually is. Hence the results of our implementation look far better, reaching a return of 0 at about 4k steps as opposed to 20k steps reported in Wang & Ba (2020).

E.3 Analysis

Let's group the environments into two categories based on how well the decentralized methods perform in them:

1. Pendulum, InvertedPendulum, Acrobot, Cartpole, Reacher, Walker2D, HalfCheetah, PETS-Pusher, PETS-Reacher3D, Ant,
2. FixedSwimmer, Hopper, PETS-HalfCheetah

The first category is where the best performing method is one of the proposed *DecentCEM* algorithms: *DecentCEM-A* or *DecentCEM-P*. In environments where the baseline *POPLIN-A* or *POPLIN-P* could reach near-optimal performance such as pendulum and invertedPendulum, applying the ensemble method would yield similar performance as before. In Acrobot, Reacher, Walker2D and PETS-Pusher, applying the decentralized approach increases the performance in both action space planning ("A") and parameter space planning ("P"). In Pendulum, InvertedPendulum, Cartpole, HalfCheetah, PETS-Reacher3D and Ant, ensemble helps in the action space planning but either has no impact or negative impact on the parameter space planning.

The second category is where it is better not to use a decentralized approach with multiple instances (note that by using one instance in the decentralized methods, we can recover one of *PETS*, *POPLIN-A*, *POPLIN-P*). In Hopper, the issue might lie in the MBRL approach in general since all MBRL baselines performed worse than the model-free baseline SAC. One possible issue is that the true dynamics is difficult to approximate with our model learning approach. Another possibility is that it's necessary to learn the variance of the sampling distribution, which none of these MBRL approaches do. To be clear, the variance is adapted online by CEM but it is not learned. In FixedSwimmer, the ensemble approach performs worse in both action space and parameter space planning. One potential reason for the result is that the optimization landscape is complex and needs more samples to estimate the expected return than what was used in each instance of the ensemble. In this case, it is better not to distribute the population into several instances. An alternative is to increase the total population size, with a downside of increasing computation. PETS-HalfCheetah is slightly different in that the ensemble does improve the performance significantly when used for action space planning. However, *POPLIN-P* performs significantly better than all other algorithms. This suggests that the parameter space planning has been able to successfully find a high return region using a single Gaussian distribution. In this case, distributing the population size would not be able to trade the estimation accuracy for better global search.

F More Ablation

Figure 11 (a) is an additional plot for the ensemble diversity ablation. It shows the statistics of the pairwise distance between the output actions by the CEM instances at each time step. The same as

in Fig. 8 (b), we only show a time window toward the end of training for visual clarity and the line and shaded region represent the mean and min/max distances.

Figure 11 (b) shows the performance of the *DecentCEM-A* algorithm under different ensemble size (where we keep the population size of each instance the same). Adjusting the ensemble size has an impact on the performance of *DecentCEM-A*. An ensemble size of 3 or 4 could improve the result reported in Fig. 10 to reach a near-optimal performance. We did not show the results for the parameter space planning mode "P" since all variations achieve near-optimal performance and their curves overlap. We fixed an ensemble size of 5 throughout our benchmark but tuning the ensemble size for each environment could improve the performance further.

We also study the performance of policy control where the policy network is directly used for control without the CEM step, denoted by the extra suffix "-PC". The result is shown in Fig.11 (c). Without the policy improvement from CEM, all algorithms perform worse than their counter-part of using CEM. *POPLIN-P-PC* and *DecentCEM-P-PC* both get stuck in local optima and do not perform very well. This makes sense since the premise of planning in parameter space is that CEM can search more efficiently there. The policy network is not designed to be used directly as a policy. Interestingly, *DecentCEM-A-PC* achieves a high performance from about 6k steps (30 episodes) of training. The ensemble of policy networks adds more robustness to control than using a single one.

G Overhead of the Ensemble

The sample efficiency is not impaired when going from one policy network to the multiple policy networks used in *DecentCEM-A* and *DecentCEM-P*. This is because that the generation of the training data only involve taking imaginary rollouts with the model, rather than interacting with the real environment, as discussed in Section 4.3.

In terms of the population size (number of samples drawn in CEM), the *DecentCEM* based methods do not impose additional cost. We show in both the motivational example (Sec. 3.2) and the benchmark experiments (Appendix E) that the proposed methods work better than CEM under the same total population size.

The additional computational cost is reasonable in *DecentCEM* compared to *POPLIN*. Each branch of policy network and CEM instance runs independently from the others, allowing for a parallel implementation. The instances have to be synchronized ($\arg\max$) but its additional cost is minimal. One caveat with our current implementation though is that it is serial, which slows down the speed. This is a limitation that calls for future work of a parallel implementation.