Exploring Model-based Planning with Policy Networks

Tingwu Wang[1,2] **& Jimmy Ba**[1,2]
[1] Department of Computer Science, University of Toronto [2] Vector Institute
{tingwuwang,jba}@cs.toronto.edu

Abstract

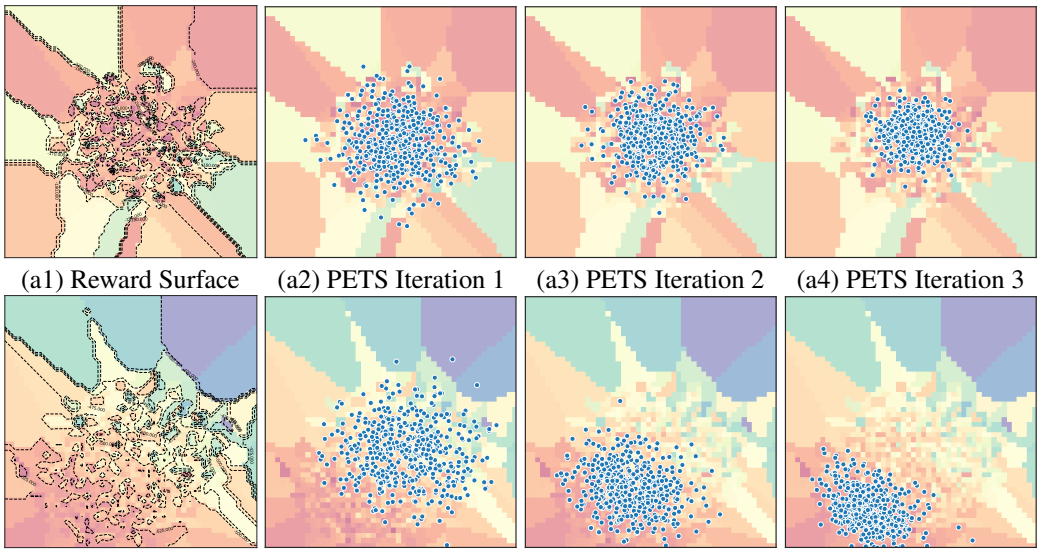
Model-based reinforcement learning (MBRL) with model-predictive control or online planning has shown great potential for locomotion control tasks in terms of both sample efficiency and asymptotic performance. Despite their initial successes, the existing planning methods search from candidate sequences randomly generated in the action space, which is inefficient in complex high-dimensional environments. In this paper, we propose a novel MBRL algorithm, model-based policy planning (POPLIN), that combines policy networks with online planning. More specifically, we formulate action planning at each time-step as an optimization problem using neural networks. We experiment with both optimization w.r.t. the action sequences initialized from the policy network, and also online optimization directly w.r.t. the parameters of the policy network. We show that POPLIN obtains state-of-the-art performance in the MuJoCo benchmarking environments, being about 3x more sample efficient than the state-of-the-art algorithms, such as PETS, TD3 and SAC. To explain the effectiveness of our algorithm, we show that the optimization surface in parameter space is smoother than in action space. Further more, we found the distilled policy network can be effectively applied without the expansive model predictive control during test time for some environments such as Cheetah. Code is released in https://github.com/WilsonWangTHU/POPLIN.

1 Introduction

A model-based reinforcement learning (MBRL) agent learns its internal model of the world, i.e. the dynamics, from repeated interactions with the environment. With the learnt dynamics, a MBRL agent can for example perform online planning, interact with imaginary data, or optimize the controller through dynamics, which provides significantly better sample efficiency [7, 38, 21, 23]. However, MBRL algorithms generally do not scale well with the increasing complexity of the reinforcement learning (RL) tasks in practice. And modelling errors in dynamics that accumulate with time-steps greatly limit the applications of MBRL algorithms. As a result, many latest progresses in RL has been made with model-free reinforcement learning (MFRL) algorithms that are capable of solving complex tasks at the cost of large number of samples [34, 16, 33, 28, 26, 14].

With the success of deep learning, a few recent works have proposed to learn neural network-based dynamics models for MBRL. Among them, random shooting algorithms (RS), which uses model-predictive control (MPC), is shown to have good robustness and scalability [31]. In shooting algorithms, the agent randomly generates action sequences, use the dynamics to predict the future states, and choose the first action from the sequence with the best expected reward. However, RS usually has worse asymptotic performance than model-free controllers [29], and the authors of the the PETS algorithm [5] suggest that the performance of RS is directly affected by the quality of the learnt dynamics. They propose a probabilistic ensemble to capture model uncertainty, which enables PETS algorithm to achieve both better sample efficiency and better asymptotic performance than

Preprint. Under review.



(a1) Reward Surface (a2) PETS Iteration 1 (a3) PETS Iteration 2 (a4) PETS Iteration 3

(b1) Reward Surface (b2) POPLIN Iteration 1 (b3) POPLIN Iteration 2 (b4) POPLIN Iteration 3

Figure 1: We transform each planned candidate action trajectory with PCA into a 2D blue scatter. The top and bottom figures are respectively the visualization of PETS [5] and our algorithm. The red area has higher reward. From left to right, we show how candidate trajectories are updated, across different planning iterations within one time-step. As we can see, while both reward surface is not smooth with respect to action trajectory. POPLIN, using policy networks, has much better search efficiency, while PETS is stuck around its initialization. The details are in section 5.3.

state-of-the-art model-free controllers in environments such as Cheetah [2]. However, PETS is not as effective on environments with higher dimensionality.

In this paper, we explore MBRL algorithms from a different perspective, where we treat the planning at each time-step as an optimization problem. Random search in action space, as what is being done in state-of-the-art MBRL algorithms such as PETS, is insufficient for more complex environments. On the one hand, we are inspired by the success of AlphaGo [35, 36], where a policy network is used to generate proposals for the Monte-Carlo tree search. On the other hand, we are inspired by the recent research into understanding deep neural networks [30, 24, 37]. Deep neural networks, frequently observed in practices, is much less likely to get stuck in sub-optimal points. In Figure 1, we apply principal component analysis (PCA) on the action sequences generated in each planning iteration within one time-step. The reward surface of the action space is not smooth and prone to local-minimas. We argue that optimization in the policy network's parameter space will be more efficient. Furthermore, we note that the state-of-the-art MBRL algorithm with MPC cannot be applied real-time. We therefore experiment with different policy network distillation schemes for fast control without MPC. To sum up, the contribution of this paper is three-fold:

- We apply policy networks to generate proposals for MPC in high dimensional locomotion control problems with unknown dynamics.
- We formulate planning as optimization with neural networks, and propose policy planning in parameter space, which obtain state-of-the-art performance on current bench-marking environments, being about 3x more sample efficient than the previous state-of-the-art algorithm, such as PETS [5], TD3 [9] and SAC [14].
- We also explore policy network distillation from the planned trajectories. We found the distilled policy network alone achieves high performance on environments like Cheetah without the expansive online planning.

2 Related Work

Model-based reinforcement learning (MBRL) has been long studied. Dyna [38, 39] algorithm alternately performs sampling in the real environments and optimize the controllers on the learned

model of the environments. Other pioneering work includes PILCO [7], where the authors model the dynamics using Gaussian Process and directly optimize the surrogate expected reward. Effective as it is to solve simple environments, PILCO heavily suffers the curse of dimensionality. In [21, 23, 22, 4, 43], the authors propose guided policy search (GPS). GPS uses iLQG [25, 42, 40] as the local controller, and distill the knowledge into a policy neural network. In SVG [17], the authors uses stochastic value gradient so that the stochastic policy network can be optimized by back-propagation through the learnt dynamics network with off-policy data. Recently with the progress of model-free algorithms such as [33, 34], in [20, 27] the authors propose modern variants of Dyna, where TRPO [33] is used to optimize the policy network using data generated by the learnt dynamics. Most recently, random shooting methods such as [29, 5] have shown its robustness and effectiveness on benchmarking environments. PETS algorithm [5] is considered by many to be the state-of-the-art MBRL algorithm, which we discuss in detail in section 3. Dynamics is also used to obtain better value estimation to speed up training [11, 8, 3]. Latent dynamics models using VAE [19] are commonly used to solve problems with image input [12, 13, 15, 18].

3 Background

3.1 Reinforcement Learning

In reinforcement learning, the problem of solving the given task is formulated as a infinite-horizon discounted Markov decision process. For the agent, we denote the action space and state space respectively as $\mathcal{A}$ and $\mathcal{S}$. We also denote the reward function and transition function as $r(s_t, a_t)$ and $f(s_{t+1}|s_t, a_t)$, where $s_t \in \mathcal{S}$ and $a_t \in \mathcal{A}$ are the state and action at time-step t. The agent maximizes its expected total reward $J(\pi) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} r(s_t, a_t)]$ with respect to the agent's controller π.

3.2 Random Shooting Algorithm and PETS

Our proposed algorithm is based on the random shooting algorithm [31]. In random shooting algorithms [29, 5], a data-set of $\mathcal{D} = \{(s_t, a_t, s_{t+1})\}$ is collected from previously generated real trajectories. The agent learns an ensemble of neural networks denoted as $f_\phi(s_{t+1}|s_t, a_t)$, with the parameters of the neural networks denoted as ϕ. In planning, the agent randomly generates a population of K candidate action sequences. Each action sequence, denoted as $\mathbf{a} = \{a_0, ..., a_\tau\}$, contains the control signals at every time-steps within the planning horizon τ. The action sequence with the best expected reward given the current dynamics network $f_\phi(s_{t+1}|s_t, a_t)$ is chosen. RS, as a model-predictive control algorithm, only executes the first action signal and re-plan at time-step. In PETS [5], the authors further use cross entropy method (CEM) [6, 1] to re-samples sequences near the best sequences from the last CEM iteration.

4 Model-Based Policy Planning

In this section, we describe two variants of POPLIN: model-based policy planning in action space (**POPLIN-A**) and model-based policy planning in parameter space (**POPLIN-P**). Following the notations defined in section 3.2, we first define the expected planning reward function at time-step i as follows:

$$\mathcal{R}(s_i, \mathbf{a}_i) = \mathbb{E}\left[\sum_{t=i}^{i+\tau} r(s_t, a_t)\right], \text{ where } s_{t+1} \sim f_\phi(s_{t+1}|s_t, a_t). \tag{1}$$

The action sequence $\mathbf{a}_i = \{a_i, a_{i+1}, ..., a_{i+\tau}\}$ is generated by the policy search module, as later described in Section 4.1 and 4.2. The expectation of predicted trajectories $\{s_i, s_{i+1}, ..., s_{i+\tau}\}$ is estimated by creating P particles from the current state. The dynamics model $f_\phi^{k,t}(s_{t+1}|s_t, a_t)$ used by k^{th} particle at time-step t is sampled from deterministic ensemble models or probabilistic ensemble models. To better illustrate, throughout the paper we treat this dynamics as a fixed deterministic model, i.e. $f_\phi^{k,t} \equiv f_\phi$. This does not violate the math in this paper, and we refer readers to PETS [5] for details.

4.1 Model-based Policy Planning in Action Space

In model-based policy planning in action space (POPLIN-A), we use a policy network to generate good initial action distribution. We denote the policy network as $\pi(s_t)$. Once the policy network proposes sequences of actions on the expected trajectories, we add Gaussian noise to the candidate actions and use CEM to fine-tune the mean and standard deviation of the noise distribution.

Similar to defining $\mathbf{a}_i = \{a_i, a_{i+1}, ..., a_{i+\tau}\}$, we denote the noise sequence at time-step t with horizon τ as $\boldsymbol{\delta}_i = \{\delta_i, \delta_{i+1}, ..., \delta_{i+\tau}\}$. We initialize the noise distribution as a Gaussian distribution with mean $\mu_0 = \mathbf{0}$ and covariance $\Sigma_0 = \sigma_0^2 \boldsymbol{I}$, where σ_0^2 is the initial noise variance. In each CEM iteration, we first sort out the sequences with the top ξ expected planning reward, whose noise sequences are denoted as $\{\boldsymbol{\delta}_i^0, \boldsymbol{\delta}_i^1, ..., \boldsymbol{\delta}_i^\xi\}$. Then we estimate the noise distribution of the elite candidates, i. e.,

$$\Sigma' \leftarrow \text{Cov}(\{\boldsymbol{\delta}_i^0, \boldsymbol{\delta}_i^1, ..., \boldsymbol{\delta}_i^\xi\}),\ \mu' \leftarrow \text{Mean}(\{\boldsymbol{\delta}_i^0, \boldsymbol{\delta}_i^1, ..., \boldsymbol{\delta}_i^\xi\}). \tag{2}$$

The elite distribution (μ', Σ') in CEM algorithm is used to update the candidate noise distribution as $\mu = (1-\alpha)\mu + \alpha\mu'$, $\Sigma = (1-\alpha)\Sigma + \alpha\Sigma'$. For every time-step, several CEM iterations are performed by candidate re-sampling and noise distribution updating. We provide detailed algorithm boxes in appendix A.1. We consider the following two schemes to add action noise.

POPLIN-A-Init: In this planning schemes, we use the policy network only to propose the initialization of the action sequences. When planning at time-step i with observed state s_i, we first obtain the initial reference action sequences, denoted as $\hat{\mathbf{a}}_i = \{\hat{a}_i, \hat{a}_{i+1}, ..., \hat{a}_{i+\tau}\}$, by running the initial forward pass with policy network. At each planning time-step t, where $i \leq t \leq i+\tau$, we have

$$\hat{a}_t = \pi(\hat{s}_t),\ \text{where}\ \hat{s}_t = f_\phi(\hat{s}_{t-1}, a_{t-1}),\ \hat{s}_i = s_i \tag{3}$$

Then the expected reward given search noise $\boldsymbol{\delta}_i$ will be:

$$\mathcal{R}(s_i, \boldsymbol{\delta}_i) = \mathbb{E}\left[\sum_{t=i}^{i+\tau} r(s_t, \hat{a}_t + \delta_t)\right],\ \text{where}\ s_{t+1} = f_\phi(s_{t+1}|s_t, \hat{a}_t + \delta_t). \tag{4}$$

POPLIN-A-Replan: POPLIN-A-Replan is a more aggressive planning schemes, which always re-plans the controller according the changed trajectory given the current noise distribution. If we had the perfect dynamics network and the policy network, then we expect re-planning to achieve faster convergence the optimal action distribution. But it increases the risk of divergent behaviors. In this case, the expected reward for each trajectory is

$$\mathcal{R}(s_i, \boldsymbol{\delta}_i) = \mathbb{E}\left[\sum_{t=i}^{i+\tau} r(s_t, \pi(s_t) + \delta_t)\right],\ \text{where}\ s_{t+1} = f_\phi(s_{t+1}|s_t, \pi(s_t) + \delta_t). \tag{5}$$

4.2 Model-based Policy Planning in Parameter Space

While planning in the action space is a natural extension of the original PETS algorithm, we found it provides little performance improvement in complex environments. One potential reason is that POPLIN-A still performs CEM searching in action sequence space, where the conditions of convergence for CEM is usually not met. Let's assume that a robot arm needs to either go left or right to get past the obstacle in the middle. In CEM planning in the action space, the theoretic distribution mean is always going straight, which fails the task.

Indeed, planning in action space is a non-convex optimization whose surface has lots of holes and peaks. Recently, much research progress has been made in understanding why deep neural networks are much less likely to get stuck in sub-optimal points [30, 24, 37]. And we believe that planning in parameter space is essentially using deeper neural networks. Therefore, we propose model-based policy planning in parameter space (POPLIN-P). Instead of adding noise in the action space, POPLIN-P adds noise in the parameter space of the policy network. We denote the parameter vector of policy network as θ, and the parameter noise sequence starting from time-step i as $\boldsymbol{\omega}_i = \{\omega_i, \omega_{i+1}, ..., \omega_{i+\tau}\}$. The expected reward function is now denoted as

$$\mathcal{R}(s_i, \boldsymbol{\omega}_i) = \mathbb{E}\left[\sum_{t=i}^{i+\tau} r\left(s_t, \pi_{\theta+\omega_t}(s_t)\right)\right],\ \text{where}\ s_{t+1} = f_\phi(s_{t+1}|s_t, \pi_{\theta+\omega_t}(s_t)). \tag{6}$$

Similarly, we update the CEM distribution towards the following elite distribution:

$$\Sigma' \leftarrow \text{Cov}(\{\boldsymbol{\omega}_i^0, \boldsymbol{\omega}_i^1, ..., \boldsymbol{\omega}_i^\xi\}),\ \mu' \leftarrow \text{Mean}(\{\boldsymbol{\omega}_i^0, \boldsymbol{\omega}_i^1, ..., \boldsymbol{\omega}_i^\xi\}). \tag{7}$$

We can force the policy network noise within the sequence to be consistent, i.e. $\omega_i = \omega_{i+1} = ... = \omega_{i+\tau}$, which we name as **POPLIN-P-Uni**. This reduces the size of the flattened noise vector from $(\tau+1)|\theta|$ to $|\theta|$, and is more consistent in policy behaviors. The noise can also be separate for each time-step, which we name as **POPLIN-P-Sep**. We benchmark both schemes in section 5.4.

Equivalence to stochastic policy with re-parameterization trick: Stochastic policy network encourages exploration, and increases the robustness against the impact of compounded model errors. POPLIN-P, which inserts exogenous noise into the parameter space, can be regarded as stochastic policy network using re-parameterization trick, which natural combines stochastic policy network with planning.

4.3 Model-predictive Control and Policy Control

MBRL with online re-planning or model-predictive control (MPC) is effective, but at the same time time-consuming. Many previous attempts have tried to distill the planned trajectories into a policy network [21, 23, 4, 43], and control only with policy network. In this paper, we define two settings of using POPLIN: **MPC Control** and **Policy Control**. In MPC control, the agent uses policy network during the online planning and only execute the first action. In policy control, the agent directly executes the signal produced by the policy network given current observation, just like how policy network is used in MFRL algorithms. We show both performance of POPLIN in this paper.

4.4 Policy Distillation Schemes

The agents iterate between interacting with the environments, and distilling the knowledge from planning trajectory into a policy network. We consider several policy distillation schemes here, and discuss their effectiveness in the later experimental section.

Behavior cloning (BC): BC can be applied both to POPLIN-A and POPLIN-P, where we minimize the squared L2 loss as

$$\min_\theta \mathbb{E}_{s,\, a\in\mathcal{D}} ||\pi_\theta(s) - a||^2, \tag{8}$$

where $\mathcal{D}$ is the collection of observation and planned action from real environment. When applying BC to POPLIN-P, we fix the parameter noise of the target network to be zeros.

Generative adversarial network training (GAN) [10]: GAN can be applied to POPLIN-P. We consider the following fact. During MPC control, the agent only needs to cover the best action sequence in its action sequence distribution. Therefore, instead of point-to-point supervised training such as BC, we can train the policy network using GAN:

$$\min_{\pi_\theta} \max_{\psi} \mathbb{E}_{s,\, a\in\mathcal{D}} \log(D_\psi(s,a)) + \mathbb{E}_{s\in\mathcal{D},\, z\sim\mathcal{N}(\mathbf{0},\sigma_0\boldsymbol{I})} \log(1 - D_\psi(s, \pi_{\theta+z}(s))), \tag{9}$$

where a discriminator D parameterized by ψ is used, and we sample the random noise z from the initial CEM distribution $\mathcal{N}(\mathbf{0}, \sigma_0\boldsymbol{I})$.

Setting parameter average (AVG): AVG is also applicable to POPLIN-P. During interaction with real environment, we also record the optimized parameter noise in to the data-set, i. e. $\mathcal{D} = \{(s, \omega)\}$. And we sacrifice the effectiveness of the policy control and only use policy network as a good search initialization. The new parameter is updated as

$$\theta = \theta + \frac{1}{|\mathcal{D}|} \sum_{\omega\in\mathcal{D}} \omega. \tag{10}$$

5 Experiments

In section 5.1, we compare POPLIN with existing algorithms. We also show the policy control performance of POPLIN with different training methods in section 5.2. In section 5.3, we provide explanations and analysis for the effectiveness of our proposed algorithms by exploring and visualizing the reward optimization surface of the planner. In section 5.4, we study the sensitivity of our algorithms with respect to hyper-parameters, and show the performance of different algorithm variants.

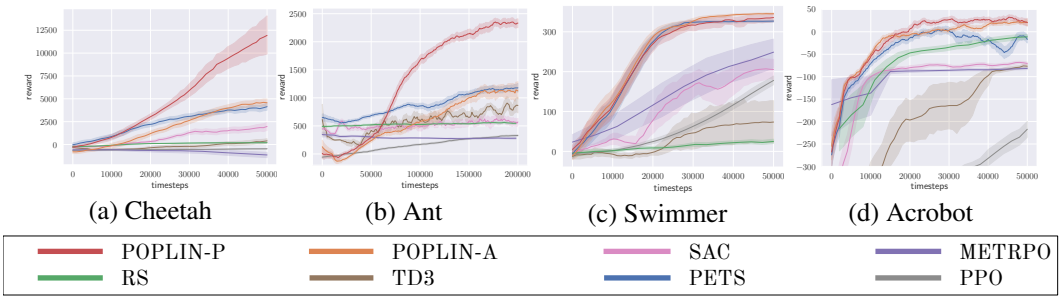


Figure 2: Performance curves of POPLIN-P, POPLIN-A and other state-of-the-art algorithms on different bench-marking environments. 4 random seeds are run for each environment, and the full figures of all 12 MuJoCo environments are summarized in appendix 8.

	Cheetah	Ant	Hopper	Swimmer	Cheetah-v0	Walker2d
POPLIN-P (ours)	**12227.9 ± 5652.8**	**2330.1 ± 320.9**	**2055.2 ± 613.8**	**334.4 ± 34.2**	**4235.0 ± 1133.0**	**597.0 ± 478.8**
POPLIN-A (ours)	4651.1 ± 1088.5	1148.4 ± 438.3	202.5 ± 962.5	**344.9 ± 7.1**	1562.8 ± 1136.7	-105.0 ± 249.8
PETS [5]	4204.5 ± 789.0	1165.5 ± 226.9	114.9 ± 621.0	326.2 ± 12.6	2288.4 ± 1019.0	282.5 ± 501.6
METRPO [20]	-744.8 ± 707.1	282.2 ± 18.0	1272.5 ± 500.9	225.5 ± 104.6	2283.7 ± 900.4	-1609.3 ± 657.5
TD3 [9]	218.9 ± 593.3	870.1 ± 283.8	1816.6 ± 994.8	72.1 ± 130.9	3015.7 ± 969.8	-516.4 ± 812.2
SAC [14]	1745.9 ± 839.2	548.1 ± 146.6	788.3 ± 738.2	204.6 ± 69.3	3459.8 ± 1326.6	164.5 ± 1318.6
Training Time-step	50000	200000	200000	50000	200000	200000
	Reacher3D	Pusher	Pendulum	InvertedPendulum	Acrobot	Cartpole
POPLIN-P (ours)	**-29.0** ± 25.2	**-55.8 ± 23.1**	167.9 ± 45.9	**-0.0 ± 0.0**	**23.2 ± 27.2**	**200.8 ± 0.3**
POPLIN-A (ours)	**-27.7** ± 25.2	**-56.0 ± 24.3**	**178.3 ± 19.3**	**-0.0 ± 0.0**	20.5 ± 20.1	**200.6 ± 1.3**
PETS [5]	-47.7 ± 43.6	**-52.7 ± 23.5**	155.7 ± 79.3	-29.5 ± 37.8	-18.4 ± 46.3	199.6 ± 4.6
METRPO [20]	-43.5 ± 3.7	-98.5 ± 12.6	174.8 ± 6.2	-29.3 ± 29.5	-78.7 ± 5.0	138.5 ± 63.2
TD3 [9]	-331.6 ± 134.6	-216.4 ± 39.6	168.6 ± 12.7	-102.9 ± 101.0	-76.5 ± 10.2	-409.2 ± 928.8
SAC [14]	-161.6 ± 43.7	-227.6 ± 42.2	159.5 ± 12.1	-0.2 ± 0.1	-69.4 ± 7.0	195.5 ± 8.7
Training Time-step	50000	50000	50000	50000	50000	50000

Table 1: The training time-step varies from 50,000 to 200,000 depending on the difficulty of the tasks. The performance is averaged across four random seeds with a window size of 3000 time-steps at the end of the training.

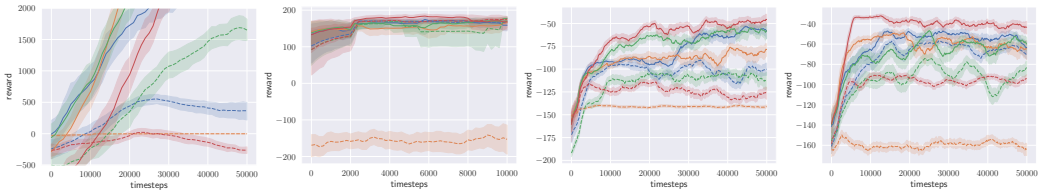
5.1 MuJoCo Benchmarking Performance

In this section, we compare POPLIN with existing reinforcement learning algorithms including PETS [5], GPS [22], RS [31], MBMF [29], TD3 [9] METRPO [20], PPO [34, 16], TRPO [33] and SAC [14], which includes the most recent progress of both model-free and model-based algorithms. We examine the algorithms with 12 environments, which is a wide collection of environments from OpenAI Gym [2] and the environments proposed in PETS [5], which are summarized in appendix A.2. Due to the page limit and to better visualize the results, we put the complete figures and tables in appendix A.3. And in Figure 2 and Table 1, we show the performance of our algorithms and the best performing baselines. The hyper-parameter search is summarized in appendix A.3.1.

As shown in Table 1, POPLIN achieves state-of-the-art performance in almost all of the environments, solving most of the environments with 200,000 or 50,000 time-steps, which is much less than the 1 million time-steps commonly used in MFRL algorithms. POPLIN-A has the best performance in simpler environments such as Pendulum, Cart-pole, Swimmer. But on complex environments such as Ant, Cheetah or Hopper, POPLIN-A does not have obvious performance gain compared with PETS. POPLIN-P on the other hand, has consistent and stable performance among different environments. POPLIN-P is significantly better than all other algorithms in complex environments such as Ant and Cheetah. However, like other model-based algorithms, POPLIN cannot efficient solve environments such as Walker and Humanoid. Although in the given episodes, POPLIN has better sample efficiency, gradually model-free algorithms will have better asymptotic performance. We view this as a bottleneck of our algorithms and leave it to future research.

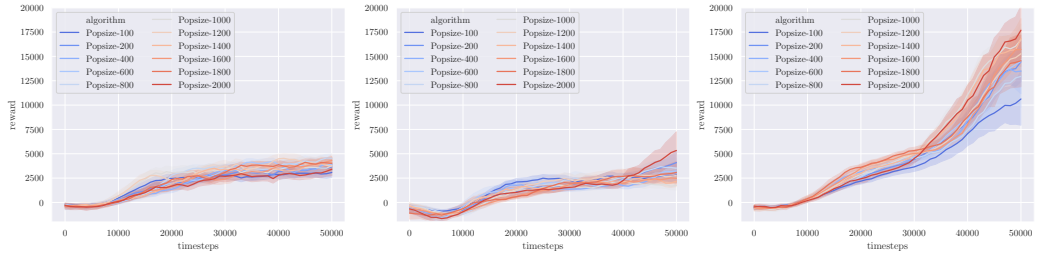
5.2 Policy Control Performance

In this section, we show the performance of POPLIN without MPC. To be more specific, we show the performance with the Cheetah, Pendulum, Pusher and Reacher3D, and we refer readers to



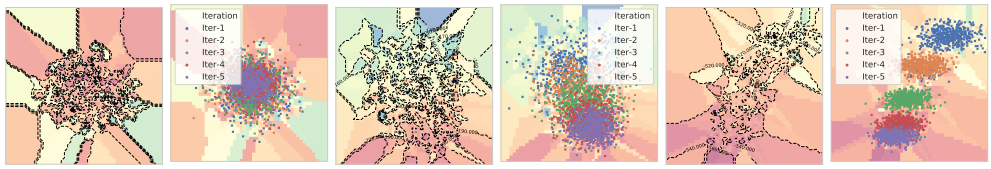
(a) Cheetah (b) Pendulum (c) Pusher (e) Reacher3D

Figure 3: The MPC control and policy control performance of the proposed POPLIN-A, and POPLIN-P with its three training schemes, which are namely behavior cloning (BC), generative adversarial network training (GAN) and setting parameter average (Avg).



(a) PETS (b) POPLIN-A (c) POPLIN-P

Figure 4: The performance of PETS, POPLIN-A, POPLIN-P using different population size of candidates. The variance of the candidates trajectory σ in POPLIN-P is set to 0.1.



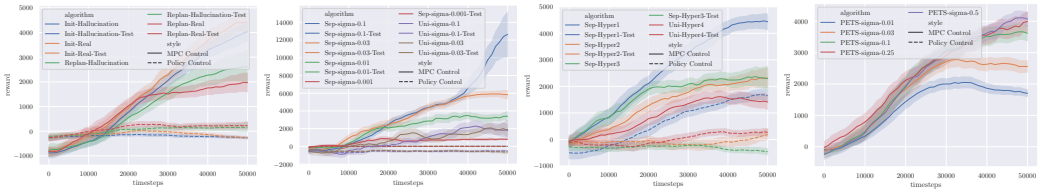
(a) PETS Surface (b) PETS Population (c) POPLIN-A Surface (d) POPLIN-A Population (e) POPLIN-P Surface (f) POPLIN-P Population

Figure 5: The reward optimization surface in the solution space. The expected reward is higher from color blue to color red. We visualize candidates using different colors as defined in the legend. The full results can be seen in appendix A.7.

appendix A.4 for the full results. We note that policy control is not always successful, and in environments such as Ant and Walker2D, the performance is almost random. In simple environments such as Pusher and Reacher3D, POPLIN-A has the best MPC performance, but has worse policy control performance compared with POPLIN-P-BC and POPLIN-P-GAN. At the same time, both POPLIN-P-BC and POPLIN-P-GAN are able to efficiently distill the knowledge from planned trajectory. Which one of POPLIN-P-BC and POPLIN-P-GAN is better depends on the environment tested, and they can be used interchangeably. This indicates that POPLIN-A, which uses a deterministic policy network, is more prone to distillation collapse than POPLIN-P, which can be interpreted as using a stochastic policy network with reparameterization trick. POPLIN-P-Avg, which only use policy network as optimization initialization has good MPC performance, but sacrifices the policy control performance. In general, the performance of policy control lags behind MPC control.

5.3 Search Effectiveness and Reward Surface

In this section, we explore the reasons for the effectiveness of POPLIN. In Figure 4, we show the performance of PETS, POPLIN-A and POPLIN-P with different population sizes. As we can see, PETS and POPLIN-A, which are the two algorithms that add search noise in the action space, cannot increase their performance by having bigger population size. However, POPLIN-P is able to efficiently increase performance with bigger population size. We then visualize the candidates in their



(a) POPLIN-A (b) POPLIN-P-Avg (c) POPLIN-P-GAN (d) PETS

Figure 7: The performance of POPLIN-A, POPLIN-P-BC, POPLIN-P-Avg, POPLIN-P-GAN using different hyper-parameters.

reward or optimization surface in Figure 1. We use PCA (principal component analysis) to transform the action sequences into 2D features. As we can see, the reward surface is not smooth, with lots of local-minima and local-maxima islands. The CEM distribution of PETS algorithm is almost fixed across iterations on this surface, even if there are potentially higher reward regions. POPLIN is able to efficiently search through the jagged reward surface, from the low-reward center to the high reward left-down corner. To further understand why POPLIN is much better at searching through the reward surface, we then plot the figures in the solution space in Figure 5. More specifically, we now perform PCA on the policy parameters for POPLIN-P. As we can see in Figure 5 (c), the reward surface in parameter space is much smoother than the reward surface in action space, which are shown in Figure 5 (a), (b). POPLIN-P can efficiently search through the smoother reward surface in parameter space.

In Figure 6, we also visualize the actions distribution in one episode taken by PETS, POPLIN-A and POPLIN-P using policy networks of different number of hidden layers. We again use PCA to project the actions into 2D feature space. As we can see, POPLIN-P shows a clear pattern of being more multi-model with the use of deeper the network.

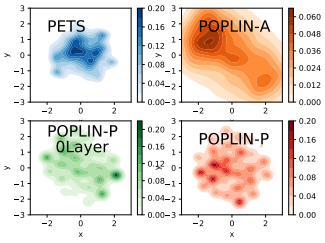


Figure 6: The action distribution in a episode visualized in the projected 2D PCA space.

5.4 Ablation Study

In this section, we study how sensitive our algorithms are with respect to some of the crucial hyper-parameters, for example, the initial variance of the CEM noise distribution. We also show the performance of different algorithm variants. The full ablation study and performance against different random seeds are included in appendix A.5.

In Figure 7 (a), we show the performance of POPLIN-A using different training schemes. We try both training with only the real data samples, which we denote as "Real", and training also with imaginary data the agent plans into the future, which we denote as "Hallucination". In practice, POPLIN-A-Init performs better than POPLIN-A-Replan, which suggests that there can be divergent or overconfident update in POPLIN-A-Replan. And training with or without imaginary does not have big impact on the performance. In Figure7 (b) and (c), we also compare the performance of POPLIN-P-Uni with POPLIN-P-Sep, where we show that POPLIN-P-Sep has much better performance than POPLIN-P-Uni, indicating the search is not efficient enough in the constrained parameter space. For POPLIN-P-Avg, with bigger initial variance of the noise distribution, the agent gets better at planning. However, increasing initial noise variance does not increase the performance of PETS algorithm, as shown in 7 (b), (d). It is worth mentioning that POPLIN-P-GAN is highly sensitive to the entropy penalty we add to the discriminator, with the 3 curves in Figure7 (c) using entropy penalty of 0.003, 0.001 and 0.0001 respectively,

6 Conclusions

In this paper, we explore efficient ways to combine policy networks with model-based planning. We propose POPLIN, which obtains state-of-the-art performance on the MuJoCo benchmarking environments. We study different distillation schemes to provide fast controllers during testing. More importantly, we formulate online planning as optimization using deep neural networks. We believe POPLIN will scale to more complex environments in the future.

References

[1] Zdravko I Botev, Dirk P Kroese, Reuven Y Rubinstein, and Pierre L'Ecuyer. The cross-entropy method for optimization. In *Handbook of statistics*, volume 31, pages 35–59. Elsevier, 2013.

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

[3] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*, pages 8224–8234, 2018.

[4] Yevgen Chebotar, Karol Hausman, Marvin Zhang, Gaurav Sukhatme, Stefan Schaal, and Sergey Levine. Combining model-based and model-free updates for trajectory-centric reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 703–711. JMLR. org, 2017.

[5] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*, 2018.

[6] 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.

[7] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

[8] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018.

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

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[11] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016.

[12] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems*, pages 2450–2462, 2018.

[13] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[14] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[15] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.

[16] Nicolas Heess, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, Ali Eslami, Martin Riedmiller, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

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

[18] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[19] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[20] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.

[21] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.

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

[23] Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning*, pages 1–9, 2013.

[24] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pages 6389–6399, 2018.

[25] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *ICINCO (1)*, pages 222–229, 2004.

[26] 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.

[27] Yuping Luo, Huazhe Xu, Yuanzhi Li, Yuandong Tian, Trevor Darrell, and Tengyu Ma. Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. *ICLR*, 2019.

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[29] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. *arXiv preprint arXiv:1708.02596*, 2017.

[30] Quynh Nguyen and Matthias Hein. The loss surface of deep and wide neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2603–2612. JMLR. org, 2017.

[31] Arthur George Richards. *Robust constrained model predictive control*. PhD thesis, Massachusetts Institute of Technology, 2005.

[32] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[33] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1889–1897, 2015.

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

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

[36] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[37] Daniel Soudry and Elad Hoffer. Exponentially vanishing sub-optimal local minima in multilayer neural networks. *arXiv preprint arXiv:1702.05777*, 2017.

[38] Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning Proceedings 1990*, pages 216–224. Elsevier, 1990.

[39] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991.

[40] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4906–4913. IEEE, 2012.

[41] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.

[42] Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 300–306. IEEE, 2005.

[43] Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew J Johnson, and Sergey Levine. Solar: Deep structured latent representations for model-based reinforcement learning. *arXiv preprint arXiv:1808.09105*, 2018.

A Appendix

A.1 Algorithm Diagrams

To better illustrate the algorithm variants of our proposed methods, we summarize them in Algorithm 1, 2, 3.

Algorithm 1 POPLIN-A-Init

Initialize policy network parameters θ, dynamics network parameters ϕ, data-set $\mathcal{D}$
while Training iterations not Finished **do**
  for i^{th} time-step of the agent **do** ▷ Sampling Data
    Initialize reference action sequence $\{\hat{a}_i, \hat{a}_{i+1}, ..., \hat{a}_{i+\tau}\}$. ▷ Using Equation 3
    Initialize action-sequence noise distribution. $\mu = \mu_0$, $\Sigma = \sigma_0^2 \boldsymbol{I}$
    for j^{th} CEM Update **do** ▷ CEM Planning
      Sample action noise sequences $\{\boldsymbol{\delta}_i\}$ from $\mathcal{N}(\mu, \Sigma)$.
      for Every candidate $\boldsymbol{\delta}_i$ **do** ▷ Trajectory Predicting
        for $t = i$ to $i + \tau$, $s_{t+1} = f_\phi(s_{t+1}|s_t, a_t = \hat{a}_t + \delta_t)$
        Evaluate expected reward of this candidate.
      end for
      Fit distribution of the elite candidates as μ', Σ'.
      Update noise distribution $\mu = (1-\alpha)\mu + \alpha\mu'$, $\Sigma = (1-\alpha)\Sigma + \alpha\Sigma'$
    end for
    Execute the first action from the optimal candidate action sequence.
  end for
  Update ϕ using data-set $\mathcal{D}$ ▷ Dynamics Update
  Update θ using data-set $\mathcal{D}$ ▷ Policy Distillation
end while

Algorithm 2 POPLIN-A-Replan

Initialize policy network parameters θ, dynamics network parameters ϕ, data-set $\mathcal{D}$
while Training iterations not Finished **do**
  for i^{th} time-step of the agent **do** ▷ Sampling Data
    Initialize action-sequence noise distribution. $\mu = \mu_0$, $\Sigma = \sigma_0^2 \boldsymbol{I}$
    for j^{th} CEM Update **do** ▷ CEM Planning
      Sample action noise sequences $\{\boldsymbol{\delta}_i\}$ from $\mathcal{N}(\mu, \Sigma)$.
      for Every candidate $\boldsymbol{\delta}_i$ **do** ▷ Trajectory Predicting
        for $t = i$ to $i + \tau$, $s_{t+1} = f_\phi(s_{t+1}|s_t, a_t = \pi_\theta(s_t) + \delta_t)$
        Evaluate expected reward of this candidate.
      end for
      Fit distribution of the elite candidates as μ', Σ'.
      Update noise distribution $\mu = (1-\alpha)\mu + \alpha\mu'$, $\Sigma = (1-\alpha)\Sigma + \alpha\Sigma'$
    end for
    Execute the first action from the optimal candidate action sequence.
  end for
  Update ϕ using data-set $\mathcal{D}$ ▷ Dynamics Update
  Update θ using data-set $\mathcal{D}$ ▷ Policy Distillation
end while

A.2 Bench-marking Environments

In the original PETS paper [5], the authors only experiment with 4 environments, which are namely Reacher3D, Pusher, Cartpole and Cheetah. In this paper, we experiment with the 9 more environments based on the standard bench-marking environments from OpenAI Gym [2]. More specifically, we experiment with InvertedPendulum, Acrobot, Pendulum, Ant, Hopper, Swimmer, Walker2d. We also note that the Cheetah environment in PETS [5] is different from the standard HalfCheetah-v1 in OpenAI Gym. Therefore we experiment with both versions in our paper, where the Cheetah from

Algorithm 3 POPLIN-P

Initialize policy network parameters θ, dynamics network parameters ϕ, data-set $\mathcal{D}$
while Training iterations not Finished **do**
  for i^{th} time-step of the agent **do** ▷ Sampling Data
    Initialize parameter-sequence noise distribution. $\mu = \mu_0$, $\Sigma = \sigma_0^2 \boldsymbol{I}$
    for j^{th} CEM Update **do** ▷ CEM Planning
      Sample parameter noise sequences $\{\boldsymbol{\omega}_i\}$ from $\mathcal{N}(\mu, \Sigma)$.
      for Every candidate $\boldsymbol{\omega}_i$ **do** ▷ Trajectory Predicting
        for $t = i$ to $i + \tau$, $s_{t+1} = f_\phi(s_{t+1}|s_t, a_t = \pi_{\theta+\omega_t}(s_t))$
        Evaluate expected reward of this candidate.
      end for
      Fit distribution of the elite candidates as μ', Σ'.
      Update noise distribution $\mu = (1-\alpha)\mu + \alpha\mu'$, $\Sigma = (1-\alpha)\Sigma + \alpha\Sigma'$
    end for
    Execute the first action from the optimal candidate action sequence.
  end for
  Update ϕ using data-set $\mathcal{D}$ ▷ Dynamics Update
  Update θ using data-set $\mathcal{D}$ ▷ Policy Distillation
end while

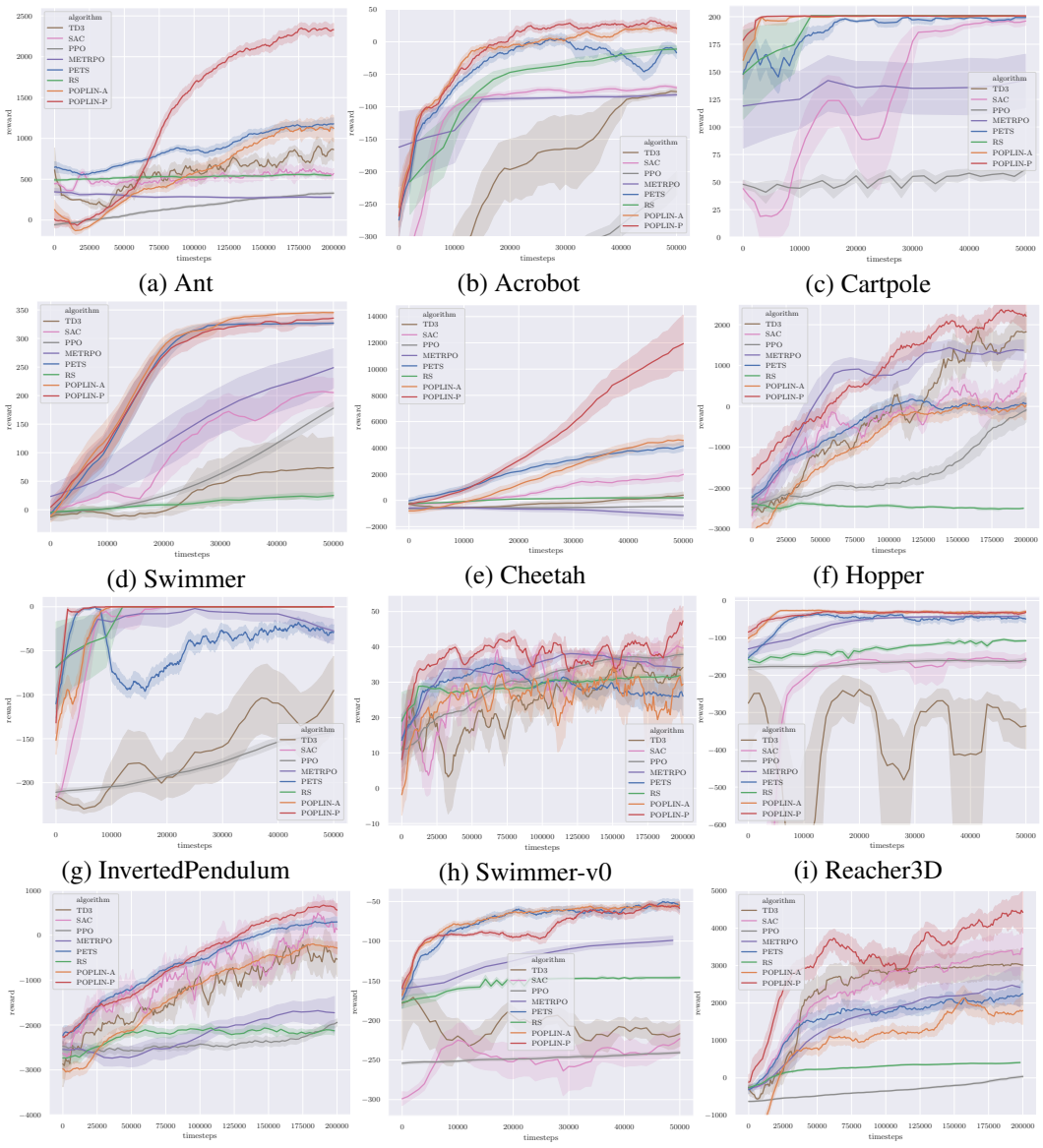
	Cheetah	Ant	Hopper	Swimmer	Cheetah-v0	Walker2d	Swimmer-v0
POPLIN-P	12227.9 ± 5652.8	2330.1 ± 320.9	2055.2 ± 613.8	334.4 ± 34.2	4235.0 ± 1133.0	597.0 ± 478.8	37.1 ± 4.6
POPLIN-A	4651.1 ± 1088.5	1148.4 ± 438.3	202.5 ± 962.5	344.9 ± 7.1	1562.8 ± 1136.7	-105.0 ± 249.8	26.7 ± 13.2
PETS	4204.5 ± 789.0	1165.5 ± 226.9	114.9 ± 621.0	326.2 ± 12.6	2288.4 ± 1019.0	282.5 ± 501.6	29.7 ± 13.5
RS	191.1 ± 21.2	535.5 ± 37.0	-2491.5 ± 35.1	22.4 ± 9.7	421.0 ± 55.2	-2060.3 ± 228.0	26.8 ± 2.3
MBMF	-459.5 ± 62.5	134.2 ± 50.4	-1047.4 ± 1098.7	110.7 ± 45.6	126.9 ± 72.7	-2218.1 ± 437.7	30.6 ± 4.9
TRPO	-412.4 ± 33.3	323.3 ± 24.9	-2100.1 ± 640.6	47.8 ± 11.1	-12.0 ± 85.5	-2286.3 ± 373.3	26.3 ± 2.6
PPO	-483.0 ± 46.1	321.0 ± 51.2	-103.8 ± 1028.0	155.5 ± 14.9	17.2 ± 84.4	-1893.6 ± 234.1	24.7 ± 4.0
GPS	129.4 ± 140.4	445.5 ± 212.9	-768.5 ± 200.9	-30.9 ± 6.3	52.3 ± 41.7	-1730.8 ± 441.7	8.2 ± 10.2
METRPO	-744.8 ± 707.1	282.2 ± 18.0	1272.5 ± 500.9	225.5 ± 104.6	2283.7 ± 900.4	-1609.3 ± 657.5	35.4 ± 2.2
TD3	218.9 ± 593.3	870.1 ± 283.8	1816.6 ± 994.8	72.1 ± 130.9	3015.7 ± 969.8	-516.4 ± 812.2	17.0 ± 12.9
SAC	1745.9 ± 839.2	548.1 ± 146.6	788.3 ± 738.2	204.6 ± 69.3	3459.8 ± 1326.6	164.5 ± 1318.6	23.0 ± 17.3
Random	-284.2 ± 83.3	478.0 ± 47.8	-2768.0 ± 571.6	-12.4 ± 12.8	-312.4 ± 44.2	-2450.1 ± 406.5	2.4 ± 12.0
Time-step	50000	200000	200000	50000	200000	200000	200000

Table 2: Performance of each algorithm on environments based on OpenAI Gym [2] MuJoCo[41] environments. In the table, we record the performance at 200,000 time-step.

	Reacher3D	Pusher	Pendulum	InvertedPendulum	Acrobot	Cartpole
POPLIN-P	-29.0 ± 25.2	-55.8 ± 23.1	167.9 ± 45.9	-0.0 ± 0.0	23.2 ± 27.2	200.8 ± 0.3
POPLIN-A	-27.7 ± 25.2	-56.0 ± 24.3	178.3 ± 19.3	-0.0 ± 0.0	20.5 ± 20.1	200.6 ± 1.3
PETS	-47.7 ± 43.6	-52.7 ± 23.5	155.7 ± 79.3	-29.5 ± 37.8	-18.4 ± 46.3	199.6 ± 4.6
RS	-107.6 ± 5.2	-146.4 ± 3.2	161.2 ± 11.5	-0.0 ± 0.0	-12.5 ± 14.3	201.0 ± 0.0
MBMF	-168.6 ± 23.2	-285.8 ± 15.2	163.7 ± 15.2	-202.3 ± 17.0	-146.8 ± 29.9	22.5 ± 67.7
TRPO	-176.5 ± 24.3	-235.5 ± 6.2	158.7 ± 9.1	-134.6 ± 6.9	-291.2 ± 6.7	46.3 ± 6.0
PPO	-162.2 ± 15.7	-243.2 ± 6.9	160.9 ± 12.5	-137.3 ± 12.4	-205.4 ± 51.5	68.8 ± 4.9
GPS	-552.8 ± 577.7	-151.2 ± 1.3	164.3 ± 4.1	-14.7 ± 20.7	-214.3 ± 15.3	-18.7 ± 101.1
METRPO	-43.5 ± 3.7	-98.5 ± 12.6	174.8 ± 6.2	-29.3 ± 29.5	-78.7 ± 5.0	138.5 ± 63.2
TD3	-331.6 ± 134.6	-216.4 ± 39.6	168.6 ± 12.7	-102.9 ± 101.0	-76.5 ± 10.2	-409.2 ± 928.8
SAC	-161.6 ± 43.7	-227.6 ± 42.2	159.5 ± 12.1	-0.2 ± 0.1	-69.4 ± 7.0	195.5 ± 8.7
Random	-183.1 ± 41.5	-199.0 ± 10.0	-249.5 ± 228.4	-205.9 ± 12.1	-374.1 ± 15.6	31.3 ± 36.3
Time-step	50000	50000	50000	50000	50000	50000

Table 3: Performance of each algorithm on environments based on OpenAI Gym [2] classic control environments. In the table, we record the performance at 50000 time-step.

PETS is named as "Cheetah", and the HalfCHeetah from OpenAI Gym is named as "Cheetah-v0". Empirically, Cheetah is much easier to solve than Cheetah-v0, as show in Table 2 and Table 3. We also include two swimmer, which we name as Swimmer and Swimmer-v0, which we explain in section A.2.1.



(a) Ant (b) Acrobot (c) Cartpole

(d) Swimmer (e) Cheetah (f) Hopper

(g) InvertedPendulum (h) Swimmer-v0 (i) Reacher3D

(j) Walker2d (k) Pusher (l) Cheetah-v0

Figure 8: Full Performance of POPLIN-P, POPLIN-A and other state-of-the-art algorithms on 12 different bench-marking environments. In the figure, we include baselines such as TD3, SAC, PPO, METRPO, PETS, RS and our proposed algorithm.

A.2.1 Fixing the Swimmer Environments

We also notice that after an update in the Gym environments, the swimmer became unsolvable for almost all algorithms. The reward threshold for solving is around 340 for the original swimmer, but almost all algorithms, including the results shown in many published papers [34], will be stuck at the 130 reward local-minima. We note that this is due the fact that the velocity sensor is on the neck of the swimmer, making swimmer extremely prone to this performance local-minimum. We provide a fixed swimmer, which we name as Swimmer, by moving the sensor from the neck to the head. We believe this modification is necessary to test the effectiveness of the algorithms.

A.3 Full Results of Bench-marking Performance

In this section, we show the figures of all the environments in Figure 8. We also include the final performance in the Table 2 and 3. As we can see, POPLIN has consistently the best performance among almost all the environments. We also include the time-steps we use on each environment for all the algorithms in Table 2 and 3.

A.3.1 Hyper-parameters

In this section, we introduce the hyper-parameters we search during the experiments. One thing to notice is that, for all of the experiments on PETS, POPLIN, we use the model type PE (probabilistic ensembles) and propagation method of E (expectation). While other combinations of model type and propagation methods might result in better performance, they are usually prohibitively computationally expensive. For example, the combination of PE-DS requires a training time of about 68 hours for one random seed, for PETS to train with 200 iteration, which is 200,000 time-step. As a matter of fact, PE-E is actually one of the best combination in many environments. Since POPLIN is based on PETS, we believe this is a fair comparison for all the algorithms.

We show the hyper-parameter search we perform for PETS in the paper in Table 4. For the hyper-parameters specific to POPLIN, we summarize them in 5 and 6.

Hyper-parameter	Value Tried
Population Size	100, 200, ..., 2000
Planning Horizon	30, 50, 100
Initial Distribution Sigma	0.01, 0.03, 0.1, 0.25, 0.3, 0.5
CEM Iterations	5, 8, 10, 20
ELite Size ξ	50, 100, 200

Table 4: Hyper-parameter grid search options for PETS.

Hyper-parameter	Value Tried
Training Data	real data, hallucination data
Variant	Replan, Init
Initial Distribution Sigma	0.001, 0.003, 0.01, 0.03, 0.1

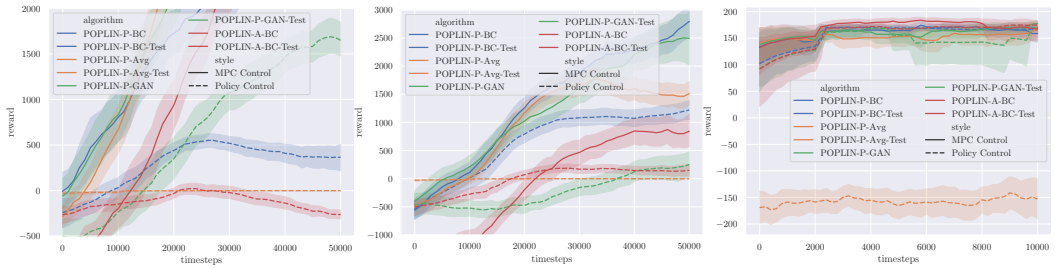
Table 5: Hyper-parameter grid search options for POPLIN-A.

Hyper-parameter	Value Tried
Training Data	real data, hallucination data
Training Variant	BC, GAN, Avg
Noise Variant	Uni, Sep
Initial Distribution Sigma	0.001, 0.003, 0.01, 0.03, 0.1

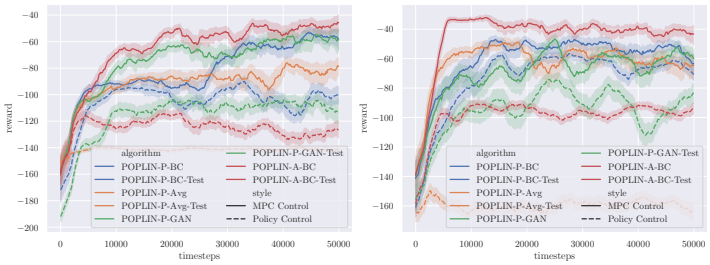
Table 6: Hyper-parameter grid search options for POPLIN-P. We also experiment with using WGAN in [32] to train the policy network, which does not results in good performance and is not put into the article.

A.4 Full Results of Policy Control

Due to the space limit, we are not able to put all of the results of policy control in the main article. More specifically, we add the figure for the original Cheetah-v0 compared to the figures shown in the main article, as can be seen in 9 (b). Again, we note that POPLIN-P-BC and POPLIN-P-GAN are comparable to each other, as mentioned in the main article. POPLIN-P-BC and POPLIN-P-GAN are the better algorithms respectively in Cheetah and Cheetah-v0, which are essentially the same environment with different observation functions.

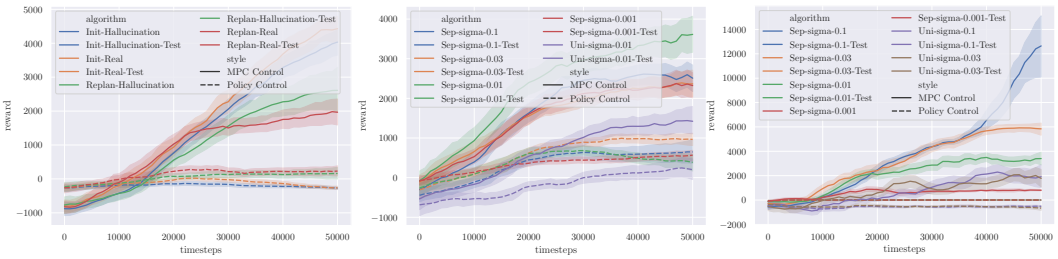


(a) Cheetah (b) Cheetah-v0 (c) Pendulum

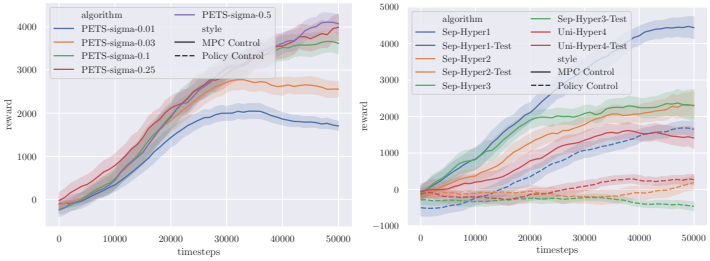


(e) Pusher (f) Reacher3D

Figure 9: The planning performance and the testing performance of the proposed POPLIN-A, and POPLIN-P with its three training schemes, which are namely behavior cloning (BC), generative adversarial network training (GAN) and setting parameter average (Avg).



(a) POPLIN-A (b) POPLIN-P-BC (c) POPLIN-P-Avg

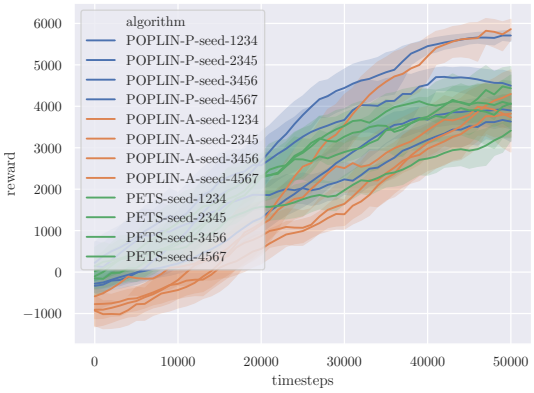


(d) PETS (d) POPLIN-P-GAN

Figure 10: The performance of POPLIN-A, POPLIN-P-BC, POPLIN-P-Avg, POPLIN-P-GAN using different hyper-parameters.

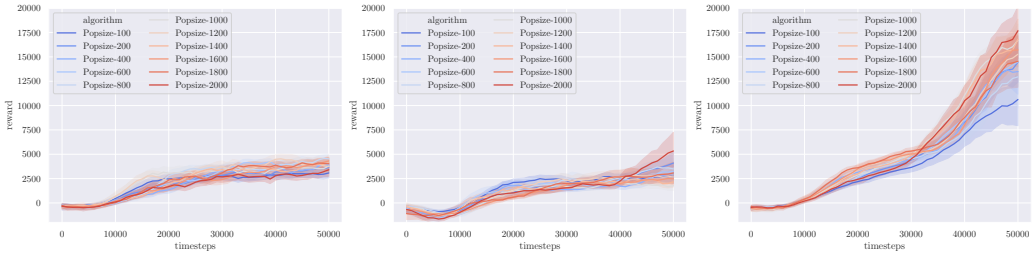
A.5 Ablation Study for Different Variant of POPLIN

In this section, we show the results of different variant of our algorithm. In Figure 11, the performances of different random seeds are visualized, where we show that POPLIN has similar randomness in performance to PETS. Additionally, we visualize POPLIN-P-BC in Figure 10 (b), whose best distribution variance for policy planning is 0.01, while the best setting for testing is 0.03.

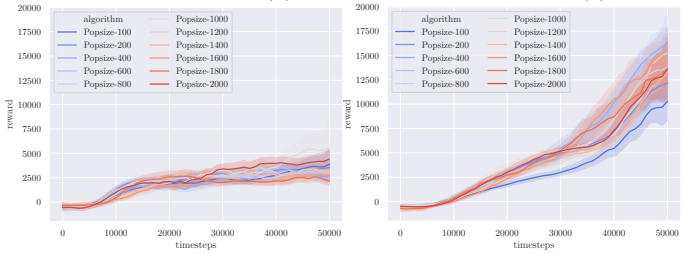


(e) Random seeds

Figure 11: The performance of POPLIN-A, POPLIN-P, and PETS of different random seeds.



(a) CEM (b) POPLIN-A (c) POPLIN-P-AVG



(d) POPLIN-P-BC (e) POPLIN-P-ZeroWeight

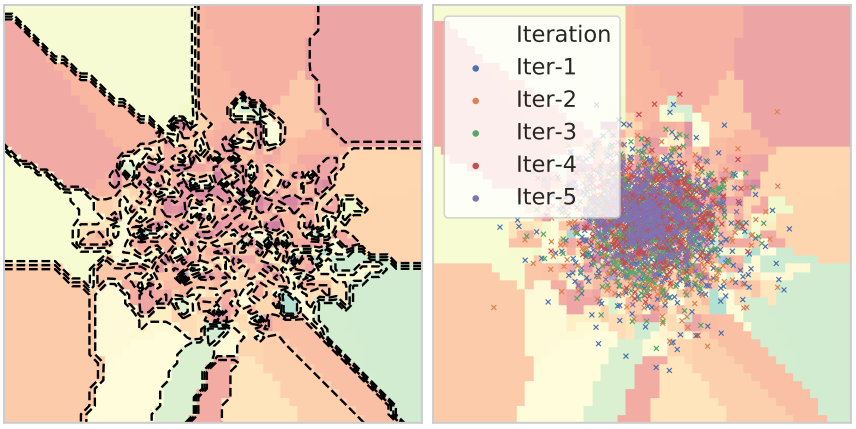
Figure 12: The performance of PETS, POPLIN-A, POPLIN-P-Avg, POPLIN-P-BC and POPLIN-P whose network has fixed parameters of zeros. The variance of the candidates trajectory σ in POPLIN-P is set to 0.1.

A.6 Population Size

In Figure 12, we include more detailed figures of the performance of different algorithms with different population size. One interesting finding is that even with fixed parameters of zeros, POPLIN-P can still performance very efficient search. This is indicating that the efficiency in optimization of POPLIN-P, especially of POPLIN-P-AVG, is the key reasons for successful planning. However, this scheme naturally sacrifices the policy distillation and thus cannot be applied without planning.

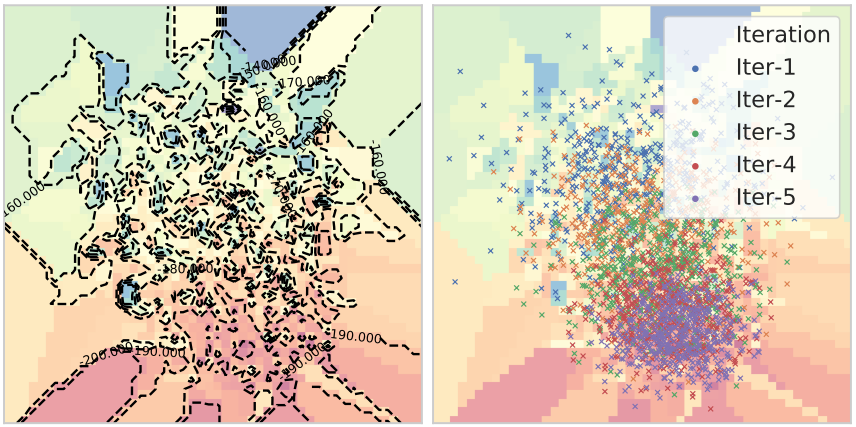
A.7 The Reward Surface of Different Algorithm

In this section, we provide a more detailed description of the reward surface with respect the the solution space (action space for PETS and POPLIN-A, and parameter space for POPLIN-P) in Figure 13, 14, 15, 16, 17. As we can see, variants of POPLIN-A are better at searching, but the reward surface is still not smooth. POPLIN-A-Replan is more efficient in searching than POPLIN-A-Init, but the errors in dynamics limit its performance. We also include the results for POPLIN-P using a 1-layer neural network in solution space in Figure 16 (g), (h). The results indicate that the deeper the network, the better the search efficiency.



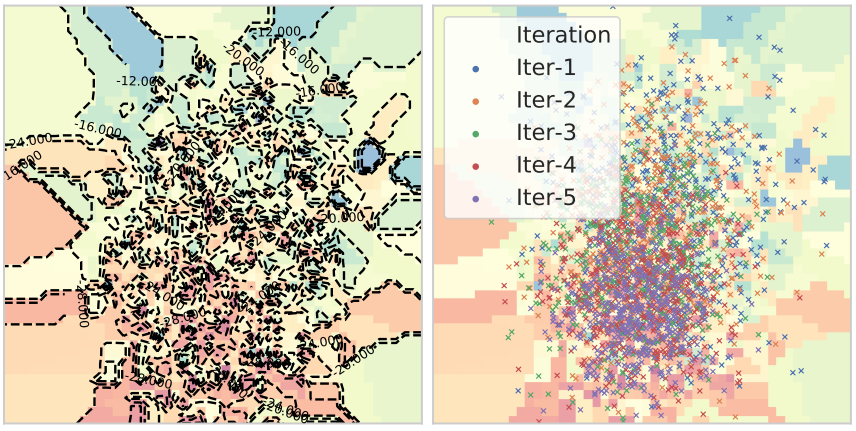
(a) PETS reward surface (b) PETS candidates

Figure 13: Reward surface in solution space (action space) for PETS algorithm.



(c) POPLIN-A-Replan reward surface (d) POPLIN-A-Replan candidates

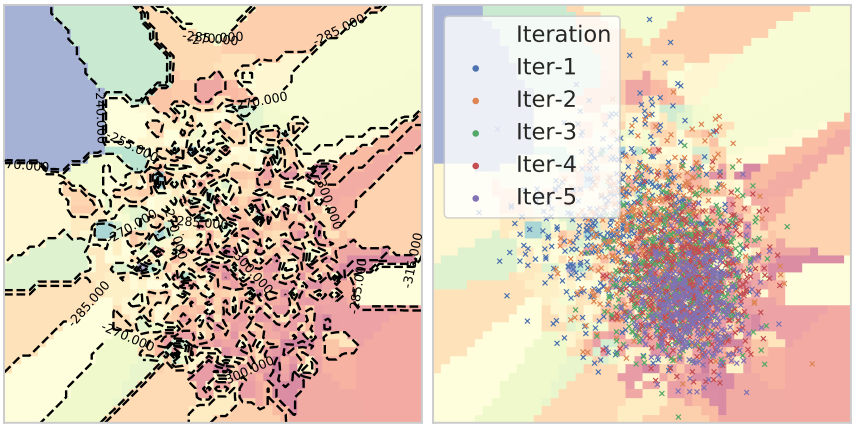
Figure 14: Reward surface in solution space (action space) for POPLIN-A-Replan.



(e) POPLIN-A-Init reward surface (f) POPLIN-A-Init candidates

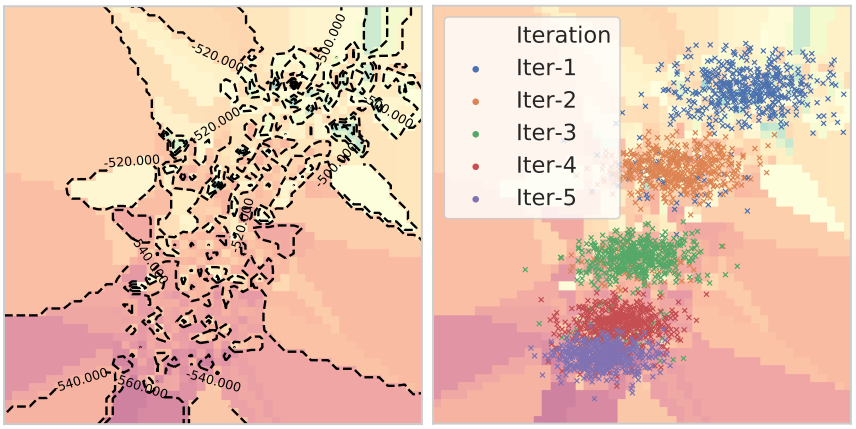
Figure 15: Reward surface in solution space (action space) for POPLIN-A-Init.

We also provide more detailed version of Figure 1 in Figure 18. We respectively show the surface for PETS, POPLIN-P-P using 1 and 0 hidden layers. Their planned trajectories across different CEM updates are visualized in Figure 19, 20, 21. Originally in Figure 1, we use the trajectories in iteration



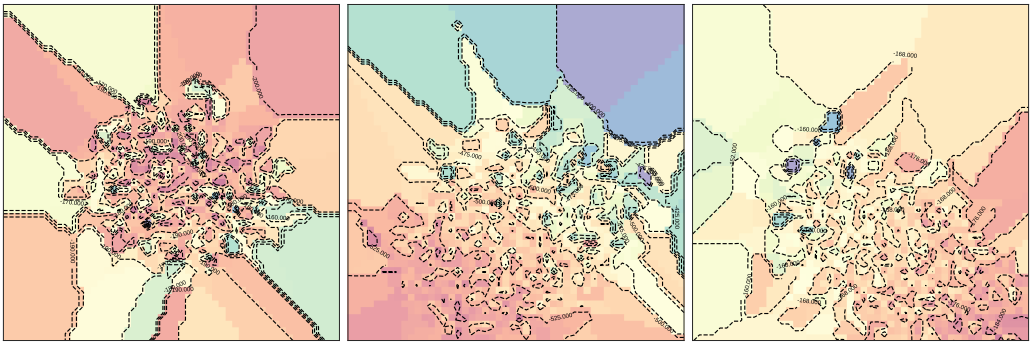
(g) POPLIN-P-Layer0 reward surface (h) POPLIN-P-Layer0 candidates

Figure 16: Reward surface in solution space (parameter space) for POPLIN-P with 0 hidden layer.



(i) POPLIN-P reward surface (j) POPLIN-P candidates

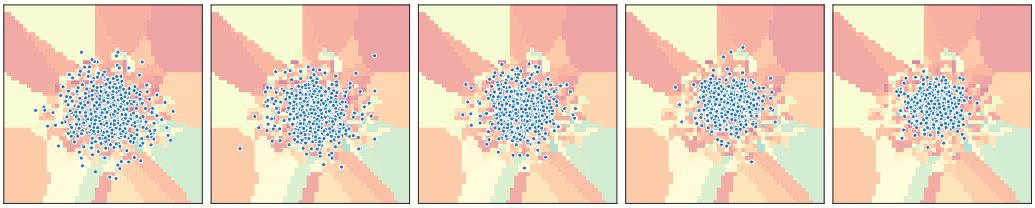
Figure 17: Reward surface in solution space (parameter space) for POPLIN-P using 1 hidden layer.



(a) PETS Surface (b) POLINA Surface (c) POLINA-L0 Surface

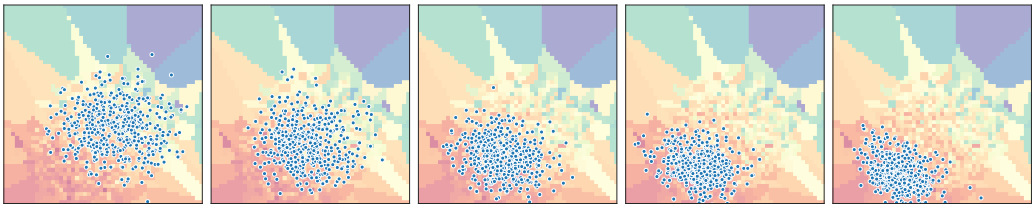
Figure 18: The color indicates the expected cost (negative of expected reward). We emphasis that all these figures are visualized in the action space. And all of them are very unsmooth. For the figures visualized in solution space, we refer to Figure 13.

1, 3, 5 for better illustration. In the appendix, we also provide all the iteration data. Again, the color indicates the expected cost (negative of expected reward). From left to right, we show the updated the trajectories in each iteration with blue scatters.



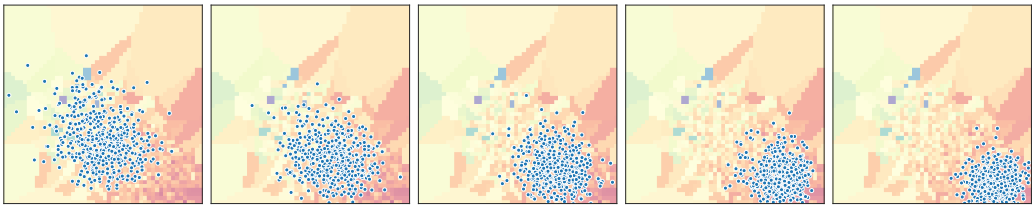
(a1) PETS Iteration 1 | (a2) PETS Iteration 2 | (a3) PETS Iteration 3 | (a4) PETS Iteration 4 | (a5) PETS Iteration 5

Figure 19: The figures are the planned trajectories of PETS.



(b1) POPLIN Iteration 1 | (b2) POPLIN Iteration 2 | (b3) POPLIN Iteration 3 | (b4) POPLIN Iteration 4 | (b5) POPLIN Iteration 5

Figure 20: The figures are the planned trajectories of POPLIN-P using 1 hidden layer MLP.



(b2) POPLIN-L0 Iteration 1 | (b2) POPLIN-L0 Iteration 2 | (b3) POPLIN-L0 Iteration 3 | (b4) POPLIN-L0 Iteration 4 | (b4) POPLIN-L0 Iteration 5

Figure 21: The figures are the planned trajectories of POPLIN-P using 0 hidden layer MLP.