

Deep Model-Based Reinforcement Learning via Estimated Uncertainty and Conservative Policy Optimization

Qi Zhou, Houqiang Li, Jie Wang*

University of Science and Technology of China

zhouqida@mail.ustc.edu.cn

{lihq, jiewangx}@ustc.edu.cn

Abstract

Model-based reinforcement learning algorithms tend to achieve higher sample efficiency than model-free methods. However, due to the inevitable errors of learned models, model-based methods struggle to achieve the same asymptotic performance as model-free methods. In this paper, We propose a **Policy Optimization** method with **Model-Based Uncertainty** (POMBU)—a novel model-based approach—that can effectively improve the asymptotic performance using the uncertainty in Q-values. We derive an upper bound of the uncertainty, based on which we can approximate the uncertainty accurately and efficiently for model-based methods. We further propose an uncertainty-aware policy optimization algorithm that optimizes the policy conservatively to encourage performance improvement with high probability. This can significantly alleviate the overfitting of policy to inaccurate models. Experiments show POMBU can outperform existing state-of-the-art policy optimization algorithms in terms of sample efficiency and asymptotic performance. Moreover, the experiments demonstrate the excellent robustness of POMBU compared to previous model-based approaches.

1 Introduction

Model-free reinforcement learning has achieved remarkable success in sequential decision tasks, such as playing Atari games [21, 11] and controlling robots in simulation environments [19, 10]. However, model-free approaches require large amounts of samples, especially when using powerful function approximators, like neural networks. Therefore, the high sample complexity hinders the application of model-free methods in real-world tasks, not to mention data gathering is often costly. In contrast, model-based reinforcement learning is more sample efficient, as it can learn from the interactions with models and then find a near-optimal policy via models [14, 8, 17, 22]. However, these methods suffer from errors of learned models, which hurt the asymptotic performance [31, 1]. Thus, compared to model-free methods, model-based algorithms can learn more quickly but tend to learn suboptimal policies after plenty of trials.

Early model-based methods achieve impressive results using simple models, like linear models [2, 18] and Gaussian processes [16, 8]. However, these methods have difficulties in high-dimensional and non-linear environments due to the limited expressiveness of models. Recent methods use neural network models for better performance, especially for complicate tasks [29, 22]. Some methods further characterize the uncertainty in models via neural network ensembles [30, 15], or Bayesian neural networks [9]. Although the uncertainty in models improves the performance of model-based methods, recent research shows that these methods still struggle to achieve the comparable asymptotic performance to state-of-the-art model-free methods robustly [35].

Inspired by previous work that improves model-free algorithms via uncertainty-aware exploration [23], we propose a theoretically-motivated algorithm to estimate the uncertainty in Q-values and apply it to the exploration of model-based reinforcement learning. Moreover, we propose to optimize the policy conservatively by encouraging a large probability of performance improvement, which is also informed by the estimated uncertainty. Thus, we use the uncertainty in Q-values to enhance both exploration and policy optimization in our model-based algorithm.

Our contributions consist of three parts.

First, we derive an upper bound of the uncertainty in Q-values and present an algorithm to estimate it. Our bound is tighter than previous work [23], and our algorithm is feasible for deep model-based reinforcement learning, while many previous methods only focus on model-free cases [25, 26], or assume simple models [7].

*Corresponding author.

Second, we propose to optimize the policy conservatively based on an estimated probability of performance improvement, which is estimated via the uncertainty in Q-values. We found the conservative policy optimization is useful to prevent the overfitting to the biased models.

Third, we propose a **Policy Optimization** method with **Model-Based Uncertainty** (POMBU), which combines our uncertainty estimation algorithm with the conservative policy optimization algorithm. Experiments show that POMBU achieves excellent robustness and can outperform state-of-the-art policy optimization algorithms.

2 Background

A finite-horizon Markov decision process (MDP) \mathcal{M} is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, \rho, H)$. Here, \mathcal{S} is a finite set of states, \mathcal{A} is a finite set of actions, P is a third-order tensor that denotes the transition probabilities, R is a matrix that denotes the rewards, ρ denotes the distribution of initial states and H is the horizon length. More specifically, at the state s and selecting the action a , $P_{sas'} \in [0, 1]$ is the probability of transitioning to the state s' , and $R_{sa} \in [-R_{\max}, R_{\max}]$ is the obtained reward. We represent a posterior of MDPs as $(\Omega, \mathcal{F}, Pr)$, where Ω is the sample space containing all possible MDPs, \mathcal{F} is a σ -field consisting of subsets of Ω , and $Pr : \mathcal{F} \rightarrow [0, 1]$ measures the posterior probability of MDPs. We assume that each MDP in Ω is different from others only in terms of P and R . In this case, P is a random tensor and R is a random matrix. For any random variable, matrix or tensor X , $\mathbb{E}_{\mathcal{M}}[X]$ and $\mathbb{D}_{\mathcal{M}}[X]$ denotes its expectation and variance respectively. When without ambiguity, we write $\mathbb{E}_{\mathcal{M}}[X]$ as X for short. For example, \bar{P} denotes $\mathbb{E}_{\mathcal{M}}[P]$ and $\bar{P}_{sas'}$ denotes $\mathbb{E}_{\mathcal{M}}[P_{sas'}]$.

Let π denotes a policy. π_{sa} denotes the probability of taking the action a at the state s . Considering the posterior of MDPs, the expected return J_{π} is a random variable, which is defined by

$$J_{\pi} = \mathbb{E}_{\tau \sim (\mathcal{M}, \pi)} \left[\sum_{h=1}^H R_{s^h a^h} \right].$$

Here $\tau = (s^1, a^1, \dots, s^H, a^H)$ is a trajectory. $\tau \sim (\mathcal{M}, \pi)$ means that the trajectory is sampled from the MDP \mathcal{M} under policy π . That is, s^1 is sampled from the initial state distribution ρ of \mathcal{M} , a^h is sampled with the probability $\pi_{s^h a^h}$ and s^{h+1} is sampled with the probability $P_{s^h a^h s^{h+1}}$ in \mathcal{M} . Our goal is to find a policy maximizing J^{π} in real environment.

Given an MDP \mathcal{M} , we define the corresponding state-action value function Q , the state value function V and the advantage function A as follow:

$$\begin{aligned} V_{\pi}^h(s) &= \mathbb{E}_{\tau \sim (\mathcal{M}, \pi)} \left[\sum_{l=h}^H R_{s^l a^l} \mid s^h = s \right], \\ Q_{\pi}^h(s, a) &= \mathbb{E}_{\tau \sim (\mathcal{M}, \pi)} \left[\sum_{l=h}^H R_{s^l a^l} \mid s^h = s, a^h = a \right], \\ A_{\pi}^h(s, a) &= Q_{\pi}^h(s, a) - V_{\pi}^h(s). \end{aligned}$$

When the policy π is fixed, we write $V_{\pi}^h(s)$, $Q_{\pi}^h(s, a)$ and $A_{\pi}^h(s, a)$ as V_s^h , Q_{sa}^h and A_{sa}^h respectively for short. In this case, for any time-step h , V_s^h , Q_{sa}^h and A_{sa}^h are random variables mapping Ω to \mathbb{R} . Hence, V^h is a random vector. Q^h and A^h are random matrices.

3 Uncertainty Estimation

In this section, we consider a fixed policy π . Similarly to the uncertainty Bellman equation (UBE) [23], we regard the standard deviations of Q-values as the uncertainty. In this section, we derive an upper bound of $\mathbb{D}_{\mathcal{M}}[Q_{sa}^h]$ for each s, a, h , and prove that our upper bound is tighter than that of UBE. Moreover, we propose an uncertainty estimation algorithm for deep model-based reinforcement learning and discuss its advantages. We provide related proofs in Appendix A.1-A.4.

Upper Bound of Uncertainty in Q-values

To analyze the uncertainty, we first make two assumptions.

Assumption 1 *Each MDP in Ω is a directed acyclic graph.*

This assumption is common [27, 23]. It means that the agent cannot visit a state more than twice within the same episode. This assumption is weak because each finite horizon MDP violating the assumption can be converted into a similar MDP that satisfying the assumption [23].

Assumption 2 The random vector R_{s_1} and the random matrix P_{s_1} are independent of R_{s_2} and P_{s_2} if $s_1 \neq s_2$.

This assumption is used in the derivation of UBE [23]. It is consistent with the trajectory sampling strategies used in recent model-based algorithms [5, 15], which sample a model from the ensemble of models independently per time step to predict the next state and reward.

First, we derive an inequation from these assumptions.

Lemma 1 Under Assumption 1 and 2, for any $s \in \mathcal{S}$ and $a \in \mathcal{A}$, we have

$$\begin{aligned} \mathbb{D}_{\mathcal{M}}[Q_{sa}^h] &\leq u_{sa}^h + \sum_{s',a'} \pi_{s'a'} \bar{P}_{sas'} \mathbb{D}_{\mathcal{M}}[Q_{s'a'}^{h+1}], \\ \text{where } u_{sa}^h &= \mathbb{D}_{\mathcal{M}} \left[R_{sa} + \sum_{s'} P_{sas'} \bar{V}_{s'}^{h+1} \right]. \end{aligned}$$

We consider u_{sa}^h as a local uncertainty, because we can compute it locally with \bar{V} .

Then, we can derive our main theorem from this lemma.

Theorem 1 Under Assumption 1 and 2, for any policy π , there exists a unique solution U satisfying the following equation:

$$U_{sa}^h = u_{sa}^h + \sum_{s',a'} \pi_{s'a'} \bar{P}_{sas'} U_{s'a'}^{h+1} \quad (1)$$

for any (s, a) and $h = 1, 2, \dots, H$, where $U^{H+1} = \mathbf{0}$, and furthermore $U \geq \mathbb{D}_{\mathcal{M}}[Q]$ pointwise.

Theorem 1 means that we can compute an upper bound of $\mathbb{D}_{\mathcal{M}}[Q_{sa}^h]$ by solving the Bellman-style equation (1).

Moreover, we provide the following theorem to show the convergence when computing U iteratively.

Theorem 2 For arbitrary $(U)_1$, if

$$(U_{sa}^h)_{i+1} = (u_{sa}^h)_i + \sum_{s',a'} \pi_{s'a'} \bar{P}_{sas'} (U_{s'a'}^{h+1})_i,$$

for any (s, a) , $h = 1, 2, \dots, H$ and $i \geq 1$, where $(U^{H+1})_i = \mathbf{0}$ and $(u)_i$ converges to u pointwise, we have $(U)_i$ converges to U pointwise.

Theorem 2 shows that we can solve the equation (1) iteratively if the estimated local uncertainty is inaccurate per update but converges to the correct value, which is significant when we use an estimated \bar{V} to compute the uncertainty.

As U_{sa}^h is an upper bound of $\mathbb{D}_{\mathcal{M}}[Q_{sa}^h]$, $\sqrt{U_{sa}^h}$ is an upper bound of the uncertainty in Q_{sa}^h . We use the upper bound to approximate the uncertainty in our algorithm similarly to UBE. We need to analyze the accuracy of our estimates.

Here, we compare our upper bound U with that of UBE under the same assumptions, and hence we need to make an extra assumption used in UBE.

Assumption 3 R_s is independent of P_s for any $s \in \mathcal{S}$.

This assumption is not used to derive our upper bound of the uncertainty but is used in UBE. Under the assumption 2 and 3, we have R is independent of P .

The upper bound B derived in UBE satisfies

$$B_{sa}^h = \nu_{sa}^h + \sum_{s',a'} \pi_{s'a'} \bar{P}_{sas'} B_{s'a'}^{h+1},$$

where $\nu_{sa}^h = (Q_{\max})^2 \sum_{s'} \mathbb{D}_{\mathcal{M}}[P_{sas'}] / \bar{P}_{sas'} + \mathbb{D}_{\mathcal{M}}[R_{sa}]$. Here, Q_{\max} is an upper bound of all $|Q_{sa}^h|$ for any s, a, h and MDP. For example, we can regard $H R_{\max}$ as Q_{\max} .

Theorem 3 Under the assumption 1, 2 and 3, U_{sa}^h is a tighter upper bound of $\mathbb{D}_{\mathcal{M}}[Q_{sa}^h]$ than B_{sa}^h .

This theorem means that our upper bound U is a more accurate estimate of the uncertainty in Q-values than the upper bound B derived in UBE.

Uncertainty Estimation Algorithm

First, we characterize the posterior of MDPs approximatively using a deterministic model ensemble (please refer to the Section 5 for the details of training models). A deterministic ensemble is denoted by $(f_{w_1}, \dots, f_{w_K})$. Here, for any $i = 1, \dots, K$, $f_{w_i} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S} \times \mathbb{R}$ is a single model that predicts the next state and the reward, and w_i is its parameters. We define a posterior probability of MDPs by

$$Pr\{P_{sas'} = 1, R_{sa} = x\} = \frac{1}{K} \sum_{i=1}^K \mathbf{eq}((s', x), f_{w_i}(s, a)),$$

where \mathbf{eq} is defined by

$$\mathbf{eq}((s_1, x_1), (s_2, x_2)) = \begin{cases} 1, & \text{if } s_1 = s_2 \text{ and } x_1 = x_2, \\ 0, & \text{otherwise.} \end{cases}$$

Then, we can construct an MDP $\hat{\mathcal{M}}$ defined according to the posterior of MDPs, such that its transition tensor \hat{P} is equal to \bar{P} and its reward matrix \hat{R} is equal to \bar{R} . Hence, the state value matrix \hat{V} of the MDP $\hat{\mathcal{M}}$ is equal to \bar{V} .

Moreover, we use a neural network $\tilde{V}_\phi : \mathcal{S} \rightarrow \mathbb{R}$ to predict \hat{V}_s^h for any state s and time step h , which is equivalent to predicting \bar{V} . We train \tilde{V}_ϕ by minimizing ℓ_2 loss function

$$L_v(\phi) = \mathbb{E}_{\tau \sim (\hat{\mathcal{M}}, \pi)} \left[\frac{1}{H} \sum_{h=1}^H \left\| \tilde{V}_\phi(s^h) - \sum_{l=h}^H \hat{R}_{s^l a^l} \right\|_2^2 \right]. \quad (2)$$

Finally, given an imagined trajectory τ sampled from $\hat{\mathcal{M}}$ under π , we can estimate the uncertainty in Q-values via the algorithm 1. Note that for long-horizon tasks, we can introduce a discount factor γ similarly to previous work [23]. The modified uncertainty estimation method can be found in Appendix B.

Algorithm 1: Uncertainty Estimation for Q-values

Input : A approximate value function \tilde{V}_ϕ ; An ensemble model $\{f_{w_1}, f_{w_2}, \dots, f_{w_K}\}$; A trajectory $\tau = (s_1, a_1, \dots, s_H, a_H)$;

Output: Estimates of $(\sqrt{U_{s_1 a_1}^1}, \dots, \sqrt{U_{s_H a_H}^H})$;

$D^{H+1} = 0$;

for $i = H, H-1, \dots, 1$ **do**

for $j = 1, 2, \dots, K$ **do**

$(s_j, r_j) = f_{w_j}(s^i, a^i)$;

$q_j = r_j + \tilde{V}_\phi(s_j)$;

end

$q = \frac{1}{K} \sum_{j=1}^K q_j$;

$d^i = \frac{1}{K} \sum_{j=1}^K (q_j - q)^2$;

$D^i = d^i + D^{i+1}$;

end

return $(\sqrt{D^1}, \sqrt{D^2}, \dots, \sqrt{D^H})$;

Discussion

In this part, we discuss some advantages of our algorithm to estimate the uncertainty in Q-values.

Accuracy

Based on the Theorem 3, our upper bound of the uncertainty is tighter than that of UBE, which means a more accurate estimation. Intuitively, our local uncertainty depends on \bar{V}^h while that of UBE depends on Q_{\max} . Therefore, our local uncertainty has a weaker dependence on H and can provide a relatively accurate estimation for long-horizon tasks (see an example in Appendix C). Moreover, considering an infinite set of states, our method ensures the boundedness of the local uncertainty because \bar{V}^h and R are bounded. Therefore, our method has the potential to apply to tasks with continuous action spaces.

Applicability for Model-Based Methods

Our method to estimate the uncertainty in Q-values is effective for model-based reinforcement learning. In model-based cases, estimated Q-values are highly dependent on the models. Our method considers the model when computing the local uncertainty, while most of the existing methods estimate the uncertainty directly via the real-world samples regardless of the models. Ignoring models may lead to bad estimates of uncertainty in model-based cases. For example, the uncertainty estimated by a count-based method [3, 28] tends to decrease with the increase of the number of samples, while the true uncertainty keeps high even with a large amount of samples when modeling a complicate MDP using a simple model.

Computational Cost

Our method is much more computationally cheap compared with estimating the uncertainty via the empirical standard deviation of Q_{sa}^h . When MDP is given, estimating Q_{sa}^h requires plenty of virtual samples. Estimating the empirical standard deviation requires estimating Q_{sa}^h for several MDPs. Previous work reduces the computational cost by learning an ensemble of Q functions [4]. However, training an ensemble of Q functions requires higher computational overhead than training a single neural network \tilde{V}_ϕ .

Compatibility with Neural Networks

Previous methods that estimate uncertainty for model-based methods always assume simple models, like Gaussian processes [8, 7]. Estimating uncertainty using Theorem 1 only requires that the models can represent a posterior. This makes our method compatible with neural network ensembles and Bayesian neural networks. For instance, we propose Algorithm 1 with an ensemble of neural networks.

Propagation of Uncertainty

As discussed in previous work [24], Bellman equation implies the high dependency between Q-values. Ignoring this dependence will limit the accuracy of the estimates of uncertainty. Our method considers the dependency and propagates the uncertainty via a Bellman-style equation.

4 Conservative Policy Optimization

In this section, we first introduce surrogate objective and then modify it via uncertainty. The modified objective leads to conservative policy optimization because it penalizes the update in the high-uncertainty regions. $\pi_\theta : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ denotes a parameterized policy, and θ is its parameters. $\pi_\theta(a|s)$ is the probability of taking action a at state s .

Surrogate Objective

Recent reinforcement learning algorithms, like Trust Region Policy Optimization (TRPO) [32], Proximal Policy Optimization (PPO) [33], optimize the policy based on surrogate objective. We rewrite the surrogate objective in TRPO and PPO as follow:

$$L_{sr}(\theta) = \mathbb{E}_{\tau \sim (\mathcal{M}, \pi_{\theta_{old}})} \left[\sum_{h=1}^H r_\theta(s^h, a^h) A_{\text{old}}^h(s^h, a^h) \right],$$

where θ_{old} are the old policy parameters before the update, A_{old} is the advantage function of $\pi_{\theta_{\text{old}}}$ and

$$r_\theta(s, a) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}.$$

Previous work has proven the surrogate objective is the first order approximation to $J(\pi_\theta) - J(\pi_{\theta_{\text{old}}})$ when θ is around θ_{old} [32, 12]. That is, for any θ_{old} , we have the following theorem:

Theorem 4

$$\begin{aligned} L_{sr}(\theta)|_{\theta=\theta_{\text{old}}} &= J(\pi_\theta) - J(\pi_{\theta_{\text{old}}})|_{\theta=\theta_{\text{old}}}, \\ \nabla_\theta L_{sr}(\theta)|_{\theta=\theta_{\text{old}}} &= \nabla_\theta (J(\pi_\theta) - J(\pi_{\theta_{\text{old}}}))|_{\theta=\theta_{\text{old}}} \end{aligned}$$

(see proof in Appendix A.5). Therefore, maximizing $L_{sr}(\theta)$ can maximize $J(\pi_\theta)$ approximately when θ is around θ_{old} .

Uncertainty-Aware Surrogate Objective

To prevent the overfitting of the policy to inaccurate models, we introduce the estimated uncertainty in Q-values into the surrogate objective.

First, we need to estimate $\Pr\{J(\pi_\theta) > J(\pi_{\theta_{\text{old}}})\}$, which means the probability that the new policy outperforms the old one. Because of Theorem 4, $\Pr\{L_{sr}(\theta) > 0\}$ can approximate $\Pr\{J(\pi_\theta) > J(\pi_{\theta_{\text{old}}})\}$. We assume that a Gaussian can approximate the distribution of $L_{sr}(\theta) > 0$. Thus, $F\left(\mathbb{E}_{\mathcal{M}}[L_{sr}(\theta)] / \sqrt{\mathbb{D}_{\mathcal{M}}[L_{sr}(\theta)]}\right)$ is approximately equal to $\Pr\{L_{sr}(\theta) > 0\}$, where F is the probability distribution function of standard normal distribution.

Then, we need to construct an objective function for optimization. Here, we aims to find a new θ with a large $F\left(\mathbb{E}_{\mathcal{M}}[L_{sr}(\theta)] / \sqrt{\mathbb{D}_{\mathcal{M}}[L_{sr}(\theta)]}\right)$. As F is monotonically increasing, we can maximize $\mathbb{E}_{\mathcal{M}}[L_{sr}(\theta)]$ while minimize $\sqrt{\mathbb{D}_{\mathcal{M}}[L_{sr}(\theta)]}$. Therefore, we can maximize

$$\mathbb{E}_{\mathcal{M}}[L_{sr}(\theta)] - \alpha \sqrt{\mathbb{D}_{\mathcal{M}}[L_{sr}(\theta)]}, \quad (3)$$

where $\alpha \geq 0$ is a hyperparameter.

Moreover, we need to estimate the expectation and the variance of the surrogate objective. Because $L_{sr}(\theta)$ is equal to

$$\mathbb{E}_{\tau \sim (\mathcal{M}, \pi_{\theta_{\text{old}}})} \left[\sum_{h=1}^H (r_\theta(s^h, a^h) - 1) Q_{\text{old}}^h(s^h, a^h) \right],$$

we can approximate $\mathbb{E}_{\mathcal{M}}[L_{sr}(\theta)]$ and $\sqrt{\mathbb{D}_{\mathcal{M}}[L_{sr}(\theta)]}$ as $L_{\text{Exp}}(\theta)$ and $L_{\text{std}}(\theta)$ respectively, where

$$L_{\text{Exp}}(\theta) = \mathbb{E}_{\tau \sim (\hat{\mathcal{M}}, \pi_{\theta_{\text{old}}})} \left[\sum_{h=1}^H r_\theta(s^h, a^h) \bar{A}_{\text{old}}^h(s^h, a^h) \right], \quad (4)$$

$$L_{\text{std}}(\theta) = \mathbb{E}_{\tau \sim (\hat{\mathcal{M}}, \pi_{\theta_{\text{old}}})} \left[\sum_{h=1}^H |r_\theta(s^h, a^h) - 1| \sqrt{D^h} \right]. \quad (5)$$

Here $\hat{\mathcal{M}}$ is defined in Section 3 using a learned ensemble, $\bar{A}_{\text{old}}^h(s^h, a^h)$ can be approximated by $\sum_{l=h}^H \hat{R}_{s^l a^l} - \tilde{V}_\phi(s^h)$, and D^h is computed by Algorithm 1.

However, policy optimization without trust region may lead to unacceptable bad performance [32]. Thus, we clip $L_{\text{Exp}}(\theta)$ similarly to PPO. That is,

$$L_{\text{clip}}(\theta) = \mathbb{E}_{\tau \sim (\hat{\mathcal{M}}, \pi_{\theta_{\text{old}}})} \left[\sum_{h=1}^H \hat{r}_\theta(s^h, a^h) \bar{A}_{\text{old}}^h(s^h, a^h) \right]. \quad (6)$$

Here, we define $\hat{r}_\theta(s^h, a^h)$ as

$$\begin{cases} \max(1 - \epsilon, r_\theta(s^h, a^h)), & \text{if } \bar{A}_{\text{old}}^h(s^h, a^h) \leq 0, \\ \min(1 + \epsilon, r_\theta(s^h, a^h)), & \text{if } \bar{A}_{\text{old}}^h(s^h, a^h) > 0, \end{cases}$$

in which $\epsilon > 0$ is a hyperparameter.

Finally, we obtain the modified surrogate objective

$$L_\pi(\theta) = L_{\text{clip}}(\theta) - \alpha L_{\text{std}}(\theta).$$

Note that, the main difference of our objective from PPO is the uncertainty penalty $L_{\text{std}}(\theta)$. This penalty limits the ratio changes $|r_\theta(s^h, a^h) - 1|$ in high-uncertainty regions. Therefore, this objective is uncertainty-aware and leads to a conservative update.

5 Algorithm

In this section, we propose a **Policy Optimization method with Model-Based Uncertainty (POMBU)** in Algorithm 2. We detail each stage of our algorithm as following.

Exploration Policy

We train a set of exploration policies by maximizing the $L_{\text{clip}}(\theta)$. Different policies are trained with different virtual trajectories. To explore the unknown, we replace $\bar{A}_{\text{old}}^h(s^h, a^h)$ with $\bar{A}_{\text{old}}^h(s^h, a^h) + \beta \sqrt{D^h}$ in the equation (6). Here, $\beta \geq 0$ controlling the exploration to high-uncertainty regions.

Algorithm 2: POMBU

```
Initialize an ensemble  $\{\tilde{f}_{w_1} \dots \tilde{f}_{w_K}\}$  and a policy  $\pi_\theta$ ;  
Initialize a value function  $\tilde{V}_\phi$ ;  
Initialize the dataset  $S$  as a empty set;  
Sample  $N$  trajectories using  $\pi_\theta$ ;  
Add the sampled transitions to  $S$ ;  
repeat  
    Train the ensemble  $\{\tilde{f}_{w_1} \dots \tilde{f}_{w_K}\}$  using  $S$ ;  
    for  $i = 1, 2, \dots, M$  do  
        Sample virtual trajectories from  $\hat{\mathcal{M}}$  using  $\pi_\theta$ ;  
        Train  $\tilde{V}_\phi$  by minimizeing  $L_v(\phi)$ ;  
        Train  $\pi_\theta$  by maximizing  $L_\pi(\theta)$ ;  
    end  
    for  $i = 1, 2, \dots, N$  do  
        Sample virtual trajectories from  $\hat{\mathcal{M}}$  using  $\pi_\theta$ ;  
        Train an exploration policy  $\pi_{\text{exp}}$  based on  $\pi_\theta$ ;  
        Collect real-world trajectories using  $\pi_{\text{exp}}$ ;  
        Add the sampled transitions to  $S$ ;  
    end  
until  $\pi_\theta$  performs well in the real environment;
```

Model Ensemble

To predict the next state, a single neural network in the ensemble outputs the change in state and then adds the change to the current state [15, 22]. To predict the reward, we assume the reward in real environment is computed by a function μ such that $R_{s^h a^h} = \mu(s^h, a^h, s^{h+1})$, which is commonly true in many simulation control tasks. Then, we can predict the reward via the predicted next state. We train the model by minimizing ℓ_2 loss similarly to previous work [15, 22] and optimize the parameter using Adam [13]. Different models are trained with different train-validation split.

Policy Optimization

We use a Gaussian policy whose mean is computed by a forward neural network and standard deviation is represented by a vector of parameters. We optimizing all parameters by maximizing $L_\pi(\theta)$ via Adam.

6 Experiments

In this section, we fist evaluate our uncertainty estimation method. Second, we compare POMBU to state-of-the-arts. Then, we show how does the estimated uncertainty work by ablation study. Finally, we analyze the robustness of our method empirically. In the following experiments, we report the performance averaged over at least three random seeds. Please refer to Appendix D for the details of experiments. The source code and appendix of this work is available at <https://github.com/MIRALab-USC/RL-POMBU>.

Effectiveness of Uncertainty Estimation

We evaluate the effectiveness of our uncertainty estimation method in two environments: 2D-point and 3D-point. These environments have continuous state spaces and continuous action spaces. First, we train an ensemble model of the environment and sample state-action pairs from the model using a deterministic policy. Then, we estimate the Q-values of these pairs via the means of virtual returns (computed using the models), and estimate the uncertainty using the algorithm 1. Finally, we compute the real Q-values using the return in real world, compute the ratios of errors to the estimated uncertainties, and count the frequencies of these ratios to draw Figure 1. This figure shows the distribution of ratios is similar to a standard normal distribution after sufficient training of \tilde{V}_ϕ , which demonstrates the accuracy of the estimated uncertainty.

Comparison to State-of-the-Arts

We compare POMBU with state-of-the-art policy optimization algorithms in four continuous control tasks of Mujoco [34]: Swimmer, HalfCheetah, Ant, and Walker2d. Our method and our baselines optimize a stochastic policy to complete the tasks. Our baselines include: soft actor critic (SAC) [10]; proximal policy optimization (PPO); stochastic lower bounds

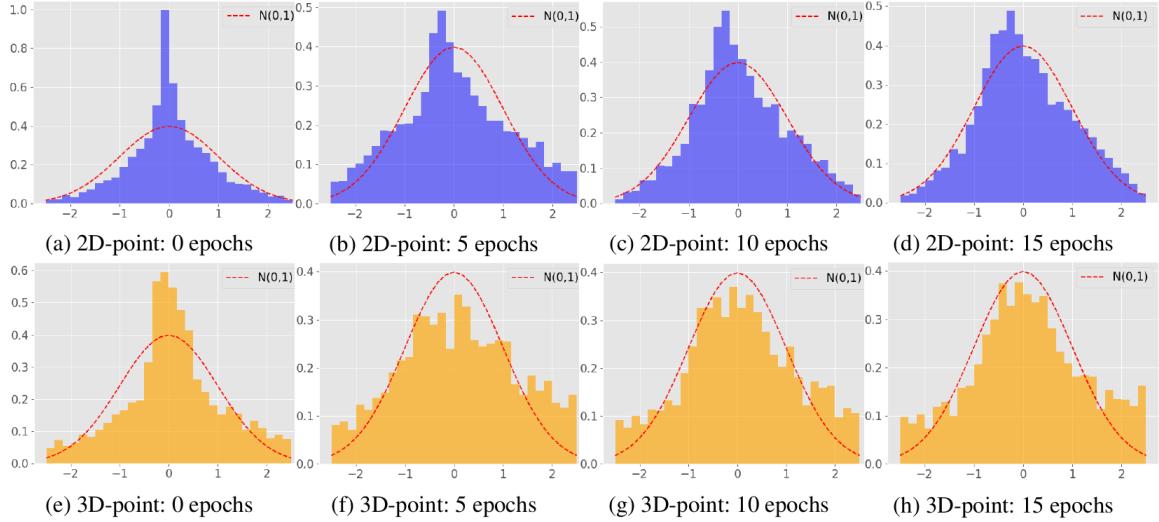


Figure 1: Frequency histograms of the ratios of errors to uncertainties after different numbers of epochs (training \tilde{V}_ϕ). The red dotted line means the probability density function of the standard normal distribution.

optimization (SLBO) [20]; model-ensemble trust region policy optimization (METRPO) [15]. To show the benefits of using uncertainty in model-based reinforcement learning, we also compare POMBU to model-ensemble proximal policy optimization (MEPPPO), which is equivalent to POMBU when $\alpha = 0$ and $\beta = 0$. We evaluate POMBU with $\alpha = 0.5$ and $\beta = 10$ for all tasks.

The result is shown in Figure 2. The solid curves correspond to the mean and the shaded region corresponds to the empirical standard deviation. It shows that POMBU achieves higher sample efficiency and better final performance than baselines, which highlights the great benefits of using uncertainty. Moreover, POMBU achieves comparable asymptotic performances with PPO and SAC in all tasks.

We also provide Table 1 that summarizes the performance, estimated wall-clock time and the number of used imagined samples and real-world samples in the HalfCheetah task ($H=200$). Compared to MEPPPO, the extra time used in POMBU is small (time: $10.17 \rightarrow 12.05$), while the improvement is significant (mean: $449 \rightarrow 852$; standard deviation: $226 \rightarrow 21$). Compared to SAC, POMBU achieve higher performance with about 5 times less real-world samples. Moreover, in our experiments, the total time to compute the uncertainty (not include the time to train \tilde{V}_ϕ) is about 1.4 minutes, which is ignorable compared with the overall time.

We further compare POMBU with state-of-the-art model-based algorithms in long-horizon tasks. The compared algorithms include model-based meta policy optimization (MBMPO) [6], probabilistic ensemble with trajectory sampling (PETS) [5] and stochastic ensemble value expansion (STEVE) [4] in addition. We directly use some of the results given by Tingwu Wang [35], and summarize all results in Table 2. The table shows that POMBU achieves comparable performance with STEVE and PETS, and outperforms other model-based algorithms. It demonstrates that POMBU is also effective in long-horizon tasks.

	POMBU	MEPPPO	METRPO	SLBO	SAC	PPO	SAC_max	PPO_max
Time (h)	12.05	10.17	6.35	3.91	0.87	0.04	4.18	0.19
Imagined	1.2e8	8e7	5e7	1e7	0	0	0	0
Real-world	2e5	2e5	2e5	2e5	2e5	2e5	9.89e5	9.78e5
Return	852 ± 21	449 ± 226	483 ± 136	704 ± 70	615 ± 64	-38 ± 16	741 ± 88	218 ± 63

Table 1: The performance, estimated wall-clock time and the number of used imagined samples and real-world samples in the HalfCheetah task ($H=200$). We conduct all experiments with one GPU Nvidia GTX 2080Ti.

Environment	POMBU	STEVE	MBMPO	SLBO	METRPO	PETS
Ant	2010 ± 91	552 ± 190	706 ± 147	728 ± 123	282 ± 18	1852 ± 141
HalfCheetah	3672 ± 8	7965 ± 1719	3639 ± 1186	1098 ± 166	2284 ± 900	2795 ± 880
Swimmer	144.4 ± 22.6	149 ± 81	85.0 ± 98.9	41.6 ± 18.4	30.1 ± 9.7	22.1 ± 25.2
Walker2d	-565 ± 129	-26 ± 328	-1546 ± 217	-1278 ± 428	-1609 ± 658	260 ± 537

Table 2: The performance of 200k time-step training. The horizons of all environments are 1000.

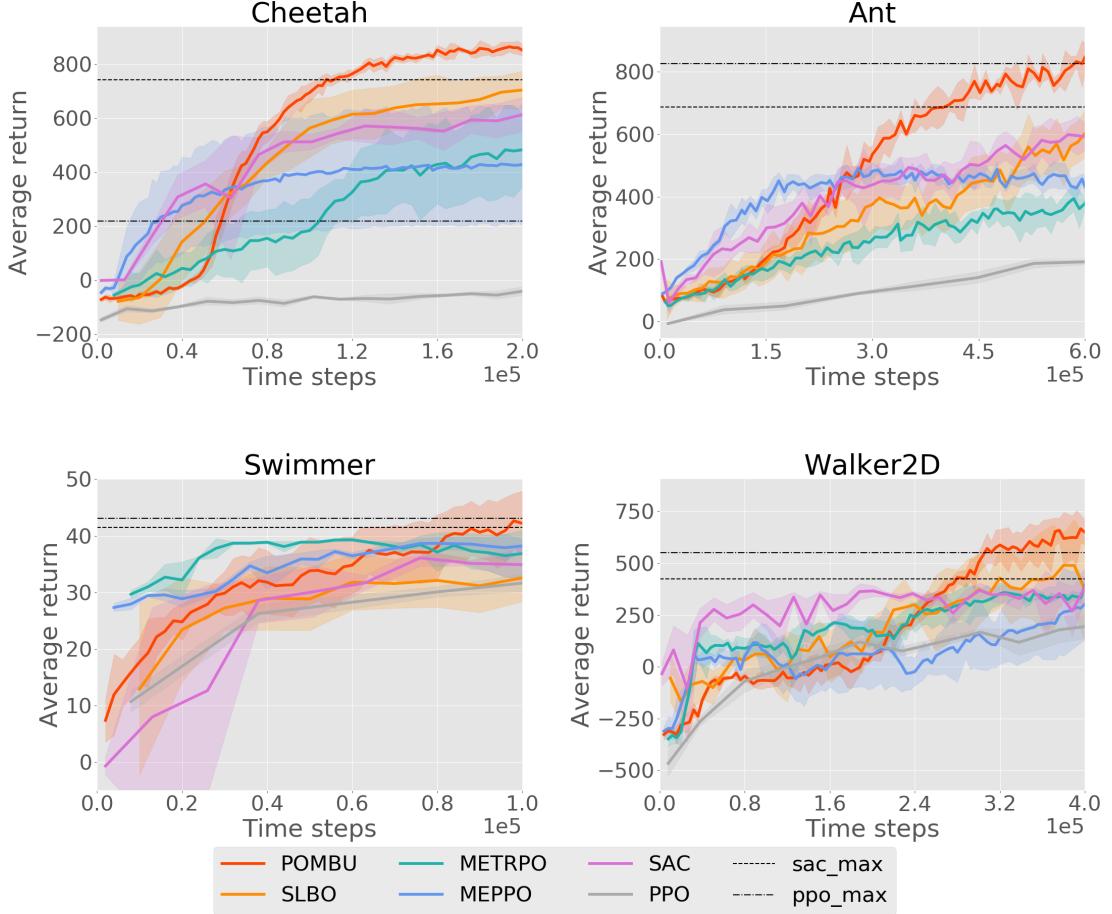


Figure 2: The training curve of our method and baselines. The horizons of all tasks are 200. The number of total steps is selected to ensure most model-based algorithms converge. We train the policy via PPO and SAC with at least 1 million samples and report the best averaged performance as "max".

Ablation Study

We provide an ablation study to show how the uncertainty benefits the performance. In our algorithm, we employ the uncertainty in policy optimization (controlled by α) and exploration (controlled by β). Therefore, we compare the performance with different α and β .

The results are shown in Figure 3 and 4. Setting α as 0.5 or 0.75 achieves the best final performance and the best robustness with 200K samples. Note that a large α may result in poorer performance in the early stage, because the uncertainty is high in the early stage and a large α tends to choose a small step size when uncertainty is high. Using $\beta = 10$ can improve the performance (larger mean and smaller standard deviation), which demonstrate the effectiveness of uncertainty-aware exploration.

Robustness Analyses

We demonstrate the excellent robustness of POMBU in two ways. First, we evaluate algorithms in noisy environments. In these environments, we add Gaussian noise to the observation with the standard deviation σ . This noise will affect the accuracy of the learned models. Second, we evaluate algorithms in long-horizon tasks. In these tasks, models need to generate long trajectories, and the error is further exacerbated due to the difficulty of longterm predictions.

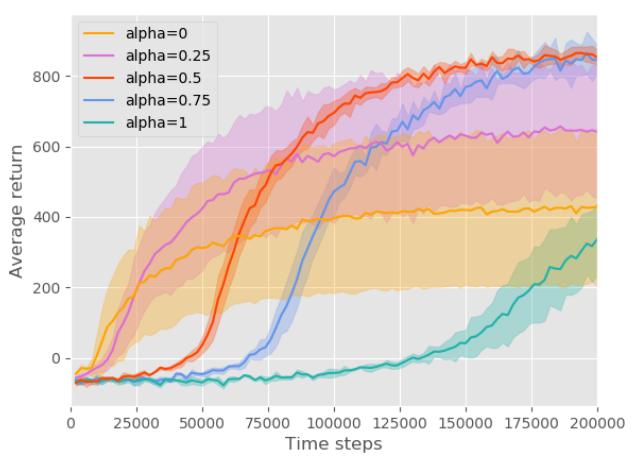


Figure 3: The development of the average return during training with different α in the Cheetah task ($H=200$).

We report the results in Figure 5. Experiments show that our algorithm achieves similar performance with different random seeds, while the performance of METRPO varies greatly with the random seeds. Moreover, in Figure 5, the worst performance of POMBU beats the best of METRPO. This implies that our method has promising robustness, even in noisy environments and long-horizon environments.

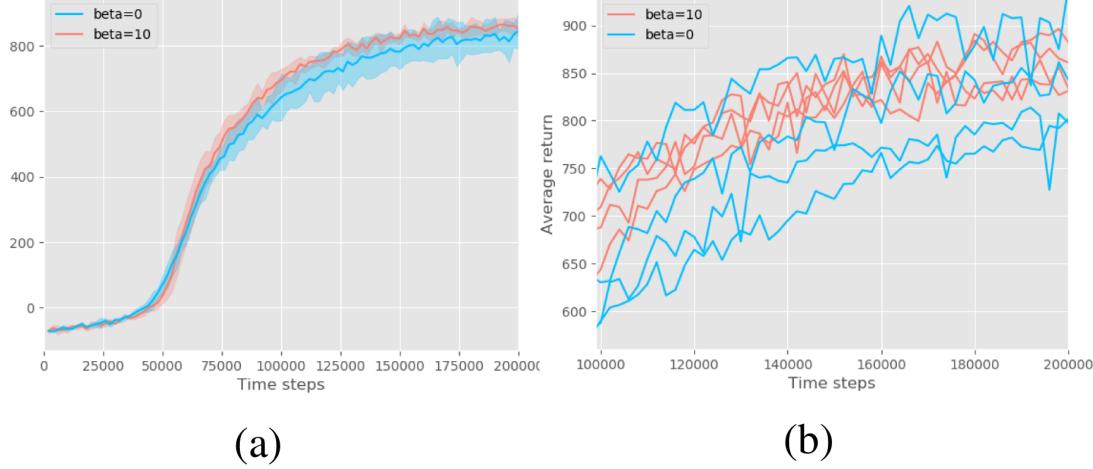


Figure 4: (a): The development of average return with $\beta = 10$ and $\beta = 0$. (b): The performance after $1e5$ time-step training with different random seeds.

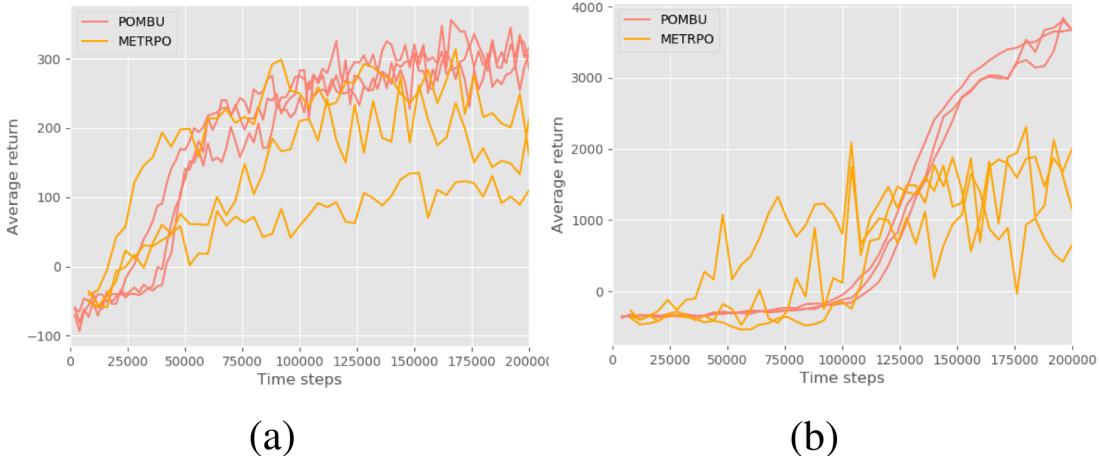


Figure 5: The training curves of POMBU and METRPO with different random seeds. (a) Comparison in a noisy Cheetah task ($\sigma = 0.1$). (b) Comparison in a long-horizon Cheetah task ($H = 1000$).

7 Conclusion

In this work, we propose a Policy Optimization method with Model-Based Uncertainty (POMBU), which is a novel uncertainty-aware model-based algorithm. This method estimates uncertainty using a model ensemble and then optimizes policy Conservatively considering the uncertainty. Experiments demonstrate that POMBU can achieve comparable asymptotic performance with SAC and PPO while using much fewer samples. Compared with other model-based methods, POMBU is robust and can achieve better performance. We believe that our approach will bring new insights into model-based reinforcement learning. An enticing direction for further work is the combination of our uncertainty estimation method with other kinds of models like Bayesian neural networks. Another exciting direction is to modify other advanced model-based algorithms like STEVE and PETS using our uncertainty estimation method.

References

- [1] P. Abbeel, M. Quigley, and A. Y. Ng. Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 1–8. ACM, 2006.

- [2] J. A. Bagnell and J. G. Schneider. Autonomous helicopter control using reinforcement learning policy search methods. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, volume 2, pages 1615–1620. IEEE, 2001.
- [3] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.
- [4] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. In *Advances in Neural Information Processing Systems*, pages 8224–8234, 2018.
- [5] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765, 2018.
- [6] I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, and P. Abbeel. Model-based reinforcement learning via meta-policy optimization. *arXiv preprint arXiv:1809.05214*, 2018.
- [7] R. Dearden, N. Friedman, and D. Andre. Model based bayesian exploration. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 150–159. Morgan Kaufmann Publishers Inc., 1999.
- [8] M. Deisenroth and C. 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.
- [9] S. Depeweg, J. M. Hernández-Lobato, F. Doshi-Velez, and S. Udluft. Learning and policy search in stochastic dynamical systems with bayesian neural networks. *arXiv preprint arXiv:1605.07127*, 2016.
- [10] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.
- [11] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [12] S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *ICML*, volume 2, pages 267–274, 2002.
- [13] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [14] J. Kocijan, R. Murray-Smith, C. E. Rasmussen, and A. Girard. Gaussian process model based predictive control. In *Proceedings of the 2004 American Control Conference*, volume 3, pages 2214–2219. IEEE, 2004.
- [15] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*, 2018.
- [16] M. Kuss and C. E. Rasmussen. Gaussian processes in reinforcement learning. In *Advances in neural information processing systems*, pages 751–758, 2004.
- [17] S. Levine and P. Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *Advances in Neural Information Processing Systems*, pages 1071–1079, 2014.
- [18] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.
- [19] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.
- [20] Y. Luo, H. Xu, Y. Li, Y. Tian, T. Darrell, and T. Ma. Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. *arXiv preprint arXiv:1807.03858*, 2018.
- [21] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.
- [22] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.
- [23] B. O’Donoghue, I. Osband, R. Munos, and V. Mnih. The uncertainty bellman equation and exploration. *arXiv preprint arXiv:1709.05380*, 2017.
- [24] I. Osband, J. Aslanides, and A. Cassirer. Randomized prior functions for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 8617–8629, 2018.
- [25] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pages 4026–4034, 2016.
- [26] I. Osband, B. Van Roy, D. Russo, and Z. Wen. Deep exploration via randomized value functions. *arXiv preprint arXiv:1703.07608*, 2017.
- [27] I. Osband, B. Van Roy, and Z. Wen. Generalization and exploration via randomized value functions. *arXiv preprint arXiv:1402.0635*.
- [28] G. Ostrovski, M. G. Bellemare, A. van den Oord, and R. Munos. Count-based exploration with neural density models. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2721–2730. JMLR. org, 2017.

- [29] A. Punjani and P. Abbeel. Deep learning helicopter dynamics models. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3223–3230. IEEE, 2015.
- [30] A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine. Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*, 2016.
- [31] J. G. Schneider. Exploiting model uncertainty estimates for safe dynamic control learning. In *NIPS*, pages 1047–1053. 1997.
- [32] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.
- [33] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [34] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.
- [35] T. Wang, X. Bao, I. Clavera, J. Hoang, Y. Wen, E. Langlois, S. Zhang, G. Zhang, P. Abbeel, and J. Ba. Benchmarking model-based reinforcement learning, 2019.

Deep Model-Based Reinforcement Learning via Estimated Uncertainty and Conservative Policy Optimization

Appendix

In this appendix, we first present the detailed proof of all the lemmas and theorems. Then, we provide an algorithm to estimate the uncertainty in Q-values in long-horizon tasks. Moreover, we provide a empirical comparison between our method and UBE. Finally, we provide the details of our experiments settings and additional experimental results.

A Proof

In this section, We provide all the proof mentioned in the body of our paper.

Proof of Lemma 1

Lemma 1 *Under Assumption 1 and 2, for any $s \in \mathcal{S}$ and $a \in \mathcal{A}$, we have*

$$\mathbb{D}_{\mathcal{M}}[Q_{sa}^h] \leq u_{sa}^h + \sum_{s', a'} \pi_{s' a'} \bar{P}_{sas'} \mathbb{D}_{\mathcal{M}}[Q_{s'a'}^{h+1}],$$

where $u_{sa}^h = \mathbb{D}_{\mathcal{M}}[R_{sa} + \sum_{s'} P_{sas'} \bar{V}_{s'}^{h+1}]$.

Proof. Let $\sum_i a_i = 1$ and each X_i is a random variable. By using Jensen's inequality , we have

$$\begin{aligned} \mathbb{D}\left[\sum_i a_i X_i\right] &= \mathbb{E}\left[\left(\sum_i a_i X_i - \mathbb{E}\left[\sum_i a_i X_i\right]\right)^2\right] \\ &= \mathbb{E}\left[\left(\sum_i a_i (X_i - \mathbb{E}[X_i])\right)^2\right] \\ &\leq \mathbb{E}\left[\sum_i a_i (X_i - \mathbb{E}[X_i])^2\right] \\ &= \sum_i a_i \mathbb{D}[X_i]. \end{aligned} \tag{7}$$

By applying the inequation (7) to the Bellman equation, we have

$$\mathbb{D}_{\mathcal{M}}[V_s^h] = \mathbb{D}_{\mathcal{M}}\left[\sum_a \pi_{sa} Q_{sa}^h\right] \leq \sum_a \pi_{sa} \mathbb{D}_{\mathcal{M}}[Q_{sa}^h]. \tag{8}$$

By using the law of total variance, we have

$$\begin{aligned} \mathbb{D}_{\mathcal{M}}[Q_{sa}^h] &= \mathbb{D}_{\mathcal{M}}\left[R_{sa} + \sum_{s'} P_{sas'} V_{s'}^{h+1}\right] \\ &= \mathbb{D}_{P_{sa}, R_{sa}}\left[\mathbb{E}\left[R_{sa} + \sum_{s'} P_{sas'} V_{s'}^{h+1} \mid P_{sa}, R_{sa}\right]\right] + \mathbb{E}_{P_{sa}, R_{sa}}\left[\mathbb{D}\left[R_{sa} + \sum_{s'} P_{sas'} V_{s'}^{h+1} \mid P_{sa}, R_{sa}\right]\right] \end{aligned} \tag{9}$$

Because Assumption 1 and 2 implies that $\mathbb{E}\left[V_{s'}^{h+1} \mid P_{sa}, R_{sa}\right] = \mathbb{E}_{\mathcal{M}}[V_{s'}^{h+1}]$ when $s' \neq s$, we have

$$\begin{aligned} &\mathbb{D}_{P_{sa}, R_{sa}}\left[\mathbb{E}\left[R_{sa} + \sum_{s'} P_{sas'} V_{s'}^{h+1} \mid P_{sa}, R_{sa}\right]\right] \\ &= \mathbb{D}_{P_{sa}, R_{sa}}\left[R_{sa} + \sum_{s'} P_{sas'} \mathbb{E}\left[V_{s'}^{h+1} \mid P_{sa}, R_{sa}\right]\right] \\ &= \mathbb{D}_{P_{sa}, R_{sa}}\left[R_{sa} + \sum_{s'} P_{sas'} \bar{V}_{s'}^{h+1}\right] = u_{sa}^l \end{aligned} \tag{10}$$

By using the inequation (7), we have

$$\begin{aligned}
& \mathbb{E}_{P_{sa}, R_{sa}} \left[\mathbb{D} \left[R_{sa} + \sum_{s'} P_{sas'} V_{s'}^{h+1} \middle| P_{sa}, R_{sa} \right] \right] \\
& \leq \mathbb{E}_{P_{sa}, R_{sa}} \left[\sum_{s'} P_{sas'} \mathbb{D} \left[V_{s'}^{h+1} \middle| P_{sa}, R_{sa} \right] \right] \\
& = \sum_{s'} \bar{P}_{sas'} \mathbb{D} \left[V_{s'}^{h+1} \middle| P_{sa}, R_{sa} \right] \\
& = \sum_{s'} \bar{P}_{sas'} \mathbb{D}_{\mathcal{M}} \left[V_{s'}^{h+1} \right], \tag{11}
\end{aligned}$$

where the last step holds because $V_{s'}^{h+1}$ is independent of P_{sa}, R_{sa} when $s' \neq s$ according to Assumption 1 and 2. Combining 8, 9, 10 and 11, we obtain the Lemma 1. \blacksquare

Proof of Theorem 1

Theorem 1 Under Assumption 1 and 2, for any policy π , there exists a unique solution U satisfying the following equation:

$$U_{sa}^h = u_{sa}^h + \sum_{s', a'} \pi_{s'a'} \bar{P}_{sas'} U_{s'a'}^{h+1} \tag{12}$$

for any (s, a) and $h = 1, 2, \dots, H$, where $U^{H+1} = \mathbf{0}$, and furthermore $U \geq \mathbb{D}_{\mathcal{M}}[Q]$ pointwise.

Proof. First, the solution of U^H exists and is unique because $U^{H+1} = \mathbf{0}$ and U^H is a linear combinations of U^{H+1} . Moreover, we know that $\mathbb{D}_{\mathcal{M}}[Q_{sa}^H] \leq u_{sa}^H = U_{sa}^H$.

Then, there exists a unique solution of $U^{H-(i+1)}$ if there exists a unique solution of U^{H-i} because $U^{H-(i+1)}$ is a linear combinations of U^{H-i} and $u^{H-(i+1)}$. Additionally, by using Lemma 1, we have

$$\mathbb{D}_{\mathcal{M}}[Q_{sa}^{H-(i+1)}] \leq u_{sa}^{H-(i+1)} + \sum_{s', a'} \pi_{s'a'} \bar{P}_{sas'} \mathbb{D}_{\mathcal{M}}[Q_{s'a'}^{H-i}] \leq u_{sa}^{H-(i+1)} + \sum_{s', a'} \pi_{s'a'} \bar{P}_{sas'} U_{s'a'}^{H-i} = U_{sa}^{H-(i+1)}$$

if $U^{H-i} \geq \mathbb{D}_{\mathcal{M}}[Q^{H-i}]$ pointwise.

Finally, we obtain Theorem 1 by induction. \blacksquare

Proof of Theorem 2

Theorem 2 For arbitrary $(U)_1$, if

$$(U_{sa}^h)_{i+1} = (u_{sa}^h)_i + \sum_{s', a'} \pi_{s'a'} \bar{P}_{sas'} (U_{s'a'}^{h+1})_i,$$

for any (s, a) , $h = 1, 2, \dots, H$ and $i \geq 1$, where $(U^{H+1})_i = \mathbf{0}$ and $(u)_i$ converges to u pointwise, we have $(U)_i$ converges to U pointwise.

Proof. $(U^H)_i$ is converges to U^H because $(u^H)_i$ converges to u^H and $(U^H)_i = (u^H)_i$.

For any $j > 0$, if $(U^{H-j+1})_i$ converges to U^{H-j+1} , $(U^{H-j})_i$ converges to U^{H-j} with the assumption $(u^{H-j})_i$ converges to u^{H-j} because $(U^{H-j})_i$ is a linear combinations of $(U^{H-j+1})_i$ and $(u^{H-j})_i$.

Then, we obtain the conclusion by induction. \blacksquare

Proof of Theorem 3

Theorem 3 Under the assumption 1, 2 and 3, U_{sa}^h is a tighter upper bound of $\mathbb{D}_{\mathcal{M}}[Q_{sa}^h]$ than B_{sa}^h .

Proof. Here, we only show that $U \leq B$ pointwise (see UBE [23] for the proof that B is an upper bound of uncertainty).

Because $\sum_{s'} \mathbb{E}_{\mathcal{M}} \bar{P}_{sas'} = 1$, by using inequation (8), we have

$$\begin{aligned}
\mathbb{D}_{\mathcal{M}} \left[\sum_{s'} P_{sas'} \bar{V}_{s'}^{h+1} \right] &= \mathbb{D}_{\mathcal{M}} \left[\sum_{s'} \frac{\bar{P}_{sas'}}{\bar{P}_{sas}} P_{sas'} \bar{V}_{s'}^{h+1} \right] \\
&\leq \sum_{s'} \bar{P}_{sas'} \mathbb{D}_{\mathcal{M}} \left[\frac{P_{sas'}}{\bar{P}_{sas}} \bar{V}_{s'}^{h+1} \right] \\
&= \sum_{s'} \mathbb{D}_{\mathcal{M}} \left[\bar{P}_{sas'} \bar{V}_{s'}^{h+1} \right] / \bar{P}_{sas'} \\
&\leq \sum_{s'} \mathbb{D}_{\mathcal{M}} \left[\bar{P}_{sas'} Q_{\max} \right] / \bar{P}_{sas'} \\
&= (Q_{\max})^2 \sum_{s'} \mathbb{D}_{\mathcal{M}} [P_{sas'}] / \bar{P}_{sas'}. \tag{13}
\end{aligned}$$

Under the Assumption 1, 2 and 3, we have

$$\begin{aligned}
u_{sa}^h &= \mathbb{D}_{\mathcal{M}} \left[R_{sa} + \sum_{s'} P_{sas'} \bar{V}_{s'}^{h+1} \right] \\
&= \mathbb{D}_{\mathcal{M}} [R_{sa}] + \mathbb{D}_{\mathcal{M}} \left[\sum_{s'} P_{sas'} \bar{V}_{s'}^{h+1} \right] \\
&\leq \mathbb{D}_{\mathcal{M}} [R_{sa}] + (Q_{\max})^2 \sum_{s'} \mathbb{D}_{\mathcal{M}} [P_{sas'}] / \bar{P}_{sas'} \\
&= \nu_{sa}^h.
\end{aligned}$$

Then $U^H = u^h \leq \nu^h \leq B^H$ pointwise. If $U^{H-i} \leq B^{H-i}$ pointwise, we have

$$U_{sa}^{H-(i+1)} = u_{sa}^{H-(i+1)} + \sum_{s', a'} \pi_{s' a'} \bar{P}_{sas'} U_{s'a'}^{H-i} \leq \nu_{sa}^{H-(i+1)} + \sum_{s', a'} \pi_{s' a'} \bar{P}_{sas'} B_{s'a'}^{H-i} = B_{sa}^{H-(i+1)}.$$

Moreover we have $U^{H+1} = B^{H+1} = \mathbf{0}$.

Then, we obtain the conclusion by induction. ■

Proof of Theorem 4

Theorem 4

$$\begin{aligned}
L_{sr}(\theta)|_{\theta=\theta_{old}} &= J(\pi_\theta) - J(\pi_{\theta_{old}})|_{\theta=\theta_{old}}, \\
\nabla_\theta L_{sr}(\theta)|_{\theta=\theta_{old}} &= \nabla_\theta (J(\pi_\theta) - J(\pi_{\theta_{old}}))|_{\theta=\theta_{old}}.
\end{aligned}$$

Proof. Note that

$$A_{\text{old}}^h(s, a) = \mathbb{E}_{\tau \sim (\mathcal{M}, \pi_\theta)} \left[R_{s^h a^h} + V_{\text{old}}^{h+1}(s^{h+1}) - V_{\text{old}}^h(s^h) \mid s^h = s, a^h = a \right].$$

Therefore, we have

$$\begin{aligned}
\mathbb{E}_{\tau \sim (\mathcal{M}, \pi_\theta)} \left[\sum_{h=1}^H A_{\text{old}}^h(s^h, a^h) \right] &= \mathbb{E}_{\tau \sim (\mathcal{M}, \pi_\theta)} \left[\sum_{h=1}^H \left(R_{s^h a^h} + V_{\text{old}}^{h+1}(s^{h+1}) - V_{\text{old}}^h(s^h) \right) \right] \\
&= \mathbb{E}_{\tau \sim (\mathcal{M}, \pi_\theta)} \left[-V_{\text{old}}^1(s^1) + \sum_{h=1}^H R_{s^h a^h} \right] \\
&= -\mathbb{E}_{\tau \sim (\mathcal{M}, \pi_\theta)} \left[V_{\text{old}}^1(s^1) \right] + \mathbb{E}_{\tau \sim (\mathcal{M}, \pi_\theta)} \left[\sum_{h=1}^H R_{s^h a^h} \right] \\
&= -J(\pi_{\text{old}}) + J(\pi_\theta). \tag{14}
\end{aligned}$$

When $\pi_{\text{old}} = \pi_\theta$, we have

$$L_{sr}(\theta) = \mathbb{E}_{\tau \sim (\mathcal{M}, \pi_{\theta_{\text{old}}})} \left[\sum_{h=1}^H r_\theta(s^h, a^h) A_{\text{old}}^h(s^h, a^h) \right] = \mathbb{E}_{\tau \sim (\mathcal{M}, \pi_\theta)} \left[\sum_{h=1}^H A_{\text{old}}^h(s^h, a^h) \right] = J(\pi_\theta) - J(\pi_{\text{old}}).$$

Additionally, we have

$$\begin{aligned}
\nabla_{\theta} \mathbb{E}_{\tau \sim (\mathcal{M}, \pi_{\theta})} \left[\sum_{h=1}^H A_{\text{old}}^h(s^h, a^h) \right] &= \nabla_{\theta} \sum_{h=1}^H \mathbb{E}_{\tau \sim (\mathcal{M}, \pi_{\theta})} \left[A_{\text{old}}^h(s^h, a^h) \right] \\
&= \nabla_{\theta} \sum_{h=1}^H \mathbb{E}_{\tau \sim (\mathcal{M}, \pi_{\theta_{\text{old}}})} \left[\left(\prod_{l=1}^h r_{\theta}(s^l, a^l) \right) \times A_{\text{old}}^h(s^h, a^h) \right] \\
&= \sum_{h=1}^H \mathbb{E}_{\tau \sim (\mathcal{M}, \pi_{\theta_{\text{old}}})} \left[\left(\nabla_{\theta} \prod_{l=1}^{h-1} r_{\theta}(s^l, a^l) \right) \times r_{\theta}(s^h, a^h) \times A_{\text{old}}^h(s^h, a^h) \right] + \\
&\quad \sum_{h=1}^H \mathbb{E}_{\tau \sim (\mathcal{M}, \pi_{\theta_{\text{old}}})} \left[\left(\prod_{l=1}^{h-1} r_{\theta}(s^l, a^l) \right) \times \nabla_{\theta} r_{\theta}(s^h, a^h) \times A_{\text{old}}^h(s^h, a^h) \right]. \tag{15}
\end{aligned}$$

When $\pi_{\text{old}} = \pi_{\theta}$, we have

$$\begin{aligned}
&\mathbb{E}_{\tau \sim (\mathcal{M}, \pi_{\theta_{\text{old}}})} \left[\left(\nabla_{\theta} \prod_{l=1}^{h-1} r_{\theta}(s^l, a^l) \right) \times r_{\theta}(s^h, a^h) \times A_{\text{old}}^h(s^h, a^h) \right] \\
&= \mathbb{E}_{\tau \sim (\mathcal{M}, \pi_{\theta_{\text{old}}})} \left[\left(\nabla_{\theta} \prod_{l=1}^{h-1} r_{\theta}(s^l, a^l) \right) \times \left(Q_{\text{old}}^h(s^h, a^h) - V_{\text{old}}^h(s^h) \right) \right] \\
&= \mathbb{E}_{\tau \sim (\mathcal{M}, \pi_{\theta_{\text{old}}})} \left[\left(\nabla_{\theta} \prod_{l=1}^{h-1} r_{\theta}(s^l, a^l) \right) \times \left(V_{\text{old}}^h(s^h) - V_{\text{old}}^h(s^h) \right) \right] \\
&= 0 \tag{16}
\end{aligned}$$

and

$$\begin{aligned}
&\sum_{h=1}^H \mathbb{E}_{\tau \sim (\mathcal{M}, \pi_{\theta_{\text{old}}})} \left[\left(\prod_{l=1}^{h-1} r_{\theta}(s^l, a^l) \right) \times \nabla_{\theta} r_{\theta}(s^h, a^h) \times A_{\text{old}}^h(s^h, a^h) \right] \\
&= \sum_{h=1}^H \mathbb{E}_{\tau \sim (\mathcal{M}, \pi_{\theta_{\text{old}}})} \left[\nabla_{\theta} r_{\theta}(s^h, a^h) \times A_{\text{old}}^h(s^h, a^h) \right] \\
&= \nabla_{\theta} \mathbb{E}_{\tau \sim (\mathcal{M}, \pi_{\theta_{\text{old}}})} \left[\sum_{h=1}^H r_{\theta}(s^h, a^h) A_{\text{old}}^h(s^h, a^h) \right] \\
&= \nabla_{\theta} L_{\text{surr}}(\theta). \tag{17}
\end{aligned}$$

Combine equations (14), (15), (16) and (17), we have $\nabla_{\theta} L_{\text{sr}}(\theta)|_{\theta=\theta_{\text{old}}} = \nabla_{\theta} (J(\pi_{\theta}) - J(\pi_{\theta_{\text{old}}}))|_{\theta=\theta_{\text{old}}}$. ■

B Uncertainty Estimation with a Discount Factor

When considering a discount factor $\gamma \in (0, 1]$, we train \tilde{V}_ϕ by minimizing ℓ_2 loss function

$$L_v(\phi) = \mathbb{E}_{\tau \sim (\hat{\mathcal{M}}, \pi)} \left[\frac{1}{H} \sum_{h=1}^H \left\| \tilde{V}_\phi(s^h) - \sum_{l=h}^H \gamma^{l-h} \hat{R}_{s^l a^l} \right\|_2^2 \right]. \quad (18)$$

and estimate uncertainty using the algorithm 3.

Algorithm 3: Uncertainty Estimation for Q-values with a discount factor γ

Input : A approximate value function \tilde{V}_ϕ ; An ensemble model $\{f_{w_1}, f_{w_2}, \dots, f_{w_K}\}$; A discount factor γ ; A trajectory $\tau = (s_1, a_1, \dots, s_H, a_H)$;

Output: Estimates of $(\sqrt{U_{s_1 a_1}^1}, \dots, \sqrt{U_{s_H a_H}^H})$;

```

 $D^{H+1} = 0;$ 
for  $i = H, H-1, \dots, 1$  do
    for  $j = 1, 2, \dots, K$  do
         $(s_j, r_j) = f_{w_j}(s^i, a^i);$ 
         $q_j = r_j + \gamma \tilde{V}_\phi(s_j);$ 
    end
     $q = \frac{1}{K} \sum_{j=1}^K q_j;$ 
     $d^i = \frac{1}{K} \sum_{j=1}^K (q_j - q)^2;$ 
     $D^i = d^i + \gamma^2 D^{i+1};$ 
end
return  $(\sqrt{D^1}, \sqrt{D^2}, \dots, \sqrt{D^H});$ 

```

C Empirical Comparison with Different Horizons

Here, we compare our uncertainty estimation method with UBE empirically. We consider a simple posterior of models denoted by $(\Omega, \mathcal{F}, Pr)$. Each MDP $\mathcal{M} \in \Omega$ has a same finite action space $\{a_0, a_1\}$ and a same finite state space $\{s_t, s_0, s_1, \dots, s_H\}$, where s_t is the terminal state. We use a deterministic ensemble $\{f_1, f_2\}$ to represent the probability measure function like discussed in Section 3. The ensemble are defined by $f_1(s, a) = (s_t, 0)$ and

$$f_2(s, a) = \begin{cases} (s_{i-1}, 1), & \text{if } a = a_1 \text{ and } s = s_i \text{ for any } i > 0, \\ (s_t, 0), & \text{otherwise.} \end{cases}$$

More intuitively, we represent those possible MDPs as the picture 6.

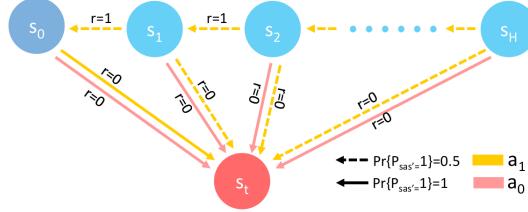


Figure 6: An intuitive definition of the probability space.

We compute the value and the uncertainty under a policy π . This policy satisfies that $\pi_{sa} = 0.5$ for any state s , action a . We compare U (estimated by our method) and B (estimated by UBE) with the groundtruth $\mathbb{D}_{\mathcal{M}}[Q]$. The results are shown in Figure 7. It shows that both our method and UBE esimate upper bounds of the true uncertainty. The upper bound estimated by our method is much tighter, especially when using a long horizon.

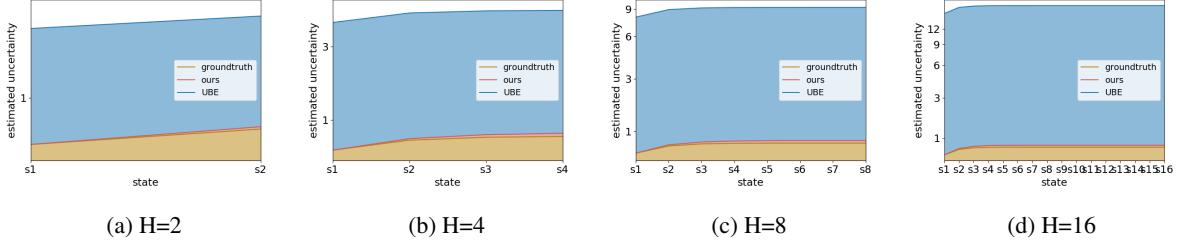


Figure 7: The comparison between estimated uncertainties with different horizon.

D Experiments

We describe the details of our experiments and provide additional experimental results in this part.

Effectiveness in 2D-point 3D-point Environments

We first introduce the 2D-point environment. Our 2D-point environment is similar to that defined by Ignasi Clavera [6].

- **State space:** \mathbb{R}^2 .
- **Action space:** $[-0.1, 0.1]^2$.
- **Initial state distribution:** Uniform distribution over $[-2, 2]^2$.
- **Transition function:** The distribution of the next state is $\delta(s + a)$, where δ means a dirac distribution.
- **Reward function:** $R_{sa} = -\|s + a\|_2^2$.
- **Horizon:** $H = 30$.

The 3D-point environment is defined similarly to 2D-point environment while the dimensionality of its state space and action space is 3.

Then we introduce how we estimate the error and value. We sample 5000 states from the initial state distribution and select actions using a deterministic policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ (parameterized by a random initialized neural network). We use an ensemble containing 5 models, each of which has the hidden layer sizes (64,64). The approximate value function \tilde{V}_w has the layer sizes (32,32). We collect 3000 real-world trajectories to train the models using mini-batch size 500 and learning rate 0.0002. We collect 100 virtual trajectories to train the value function per epoch using mini-batch size 200 and learning rate 0.00005.

We also show the relation between error and uncertainty using scatters (sample 500 pairs of data) in Figure 8 (2-D point).

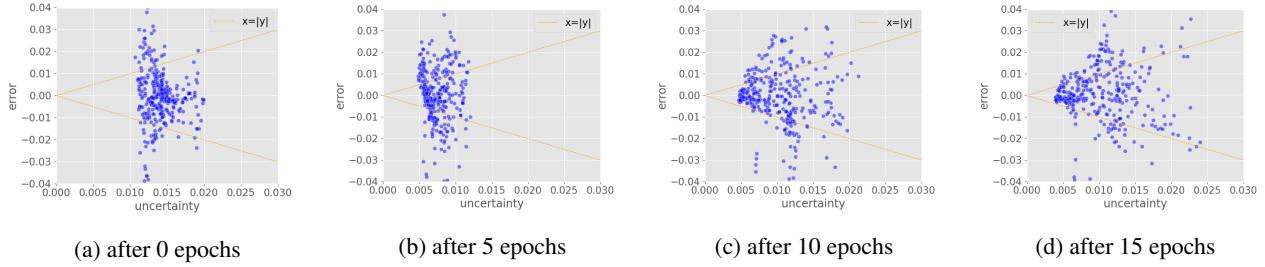


Figure 8: Relation between error and uncertainty. We show the results after different number of epochs (training \tilde{V}_w).

Mujoco Environments

We use the environments adopted from Tingwu Wang's implements [35]. The details are as follow.

Environment Name	Observation Space Dimension	Action Space Dimension	Horizon
Ant	28	8	200
Walker2D	17	6	200
HalfCheetah	17	6	200
Swimmer	8	2	200

Table 3: Dimensions of observation and action space, and horizon length.

Environment Name	Reward Function
Ant	$\dot{x} - 0.1\ a\ _2^2 - 3.0 \times (z - 0.57)^2$
Walker2D	$\dot{x} - 0.1\ a\ _2^2 - 3.0 \times (z - 1.3)^2$
HalfCheetah	$\dot{x} - 0.1\ a\ _2^2$
Swimmer	$\dot{x} - 0.0001\ a\ _2^2$

Table 4: Reward functions. In this table, \dot{x} denotes the x direction velocity, a is the action and z denotes the height.

Implement and Hyperparameter Setting

First we provide the hyperparameter setting of our method for the experiments in Section 6.2.

Dynamics Model: In all models, we used weight normalization and ReLu nonlinearities. We composed an ensemble using 5 models. We used the Adam optimizer with a batch-size of 1000 and learning rate 0.0002 to minimize the $L_m(\phi)$ of each model. For each model, we randomly split the dataset using a 4-to-1 ratio for training and validation datasets. Except for the first iteration, all models were trained with warm starts using the parameters of the previous iteration. We trained each model individually and stopped the training individually until the validation loss has not decreased by 25 epochs (we validated the training every 5 epochs). We also normalized the input and output to train the models.

Policy Optimization: We represented the policy π_θ using a 2-hidden-layer neural network with hidden sizes (64,64) and tanh nonlinearities for all the environments. We used the $\epsilon = 0.15$ to clip the objective function and used $\alpha = 0.5$ for all short horizon tasks. For training exploration policy, we used $\beta = 10$. All policies were optimized by Adam optimizer with a linear decreasing learning rate. For each step of update, we trained the policy in 10 epochs.

Value and Uncertainty Estimation: We represented the value function \tilde{V}_w using a 2-hidden-layer neural network with hidden sizes (64,64) and tanh nonlinearities for all the environments. We normalized the rewards using the standard deviation of returns. We trained the approximate value function using Adam optimizer with a linear decreasing learning rate. For each step, we trained the approximate value function in 10 epochs.

In the long horizon tasks, we show the different hyperparameters in the Tabel 6. For long-horizon HalfCheetah task, we set $\alpha = 0.75$, and for long-horizon Ant task, we set $\alpha = 0.25$. The other hyperparameters are same as the original.

Hyperparameter	Ant	Walker2D	HalfCheetah	Swimmer
hidden layers (model)	(512,512,512)	(512,512)	(512,512)	(512,512)
number of real-world trajectories per iteration	20	20	10	10
number of updates per iteration	30	30	20	20
number of iterations	150	100	100	50
number of virtual trajectories per update	500	500	200	200
mini-batch size (π_θ and \tilde{V}_w)	1250	1250	500	500
learning rate in i th iter. (π_θ and \tilde{V}_w)	$\frac{160-i}{1.6 \times 10^6}$	$\frac{110-i}{1.1 \times 10^6}$	$\frac{110-i}{1.1 \times 10^6}$	$\frac{60-i}{0.6 \times 10^6}$

Table 5: Hyperparameter setting in short horizon tasks ($H=200$).

Hyperparameter	long-horizon tasks ($H = 1000$)
number of real-world trajectories per iteration	4
number of updates per iteration	30
number of iterations	50
number of virtual trajectories per update	50
mini-batch size (π_θ and \tilde{V}_w)	500
learning rate in i th iter. (π_θ and \tilde{V}_w)	$\frac{60-i}{0.6 \times 10^6}$

Table 6: Different hyperparameter setting in long horizon tasks ($H = 1000$).

Then we introduce our baselines. We chose the state-of-the-art policy optimization algorithms (including both model-base and model-free methods that optimize parameterized policies) as our baselines. We did not compare our method to model-based meta policy optimization (MBMPO) [6] in short horizon tasks because of its high computation cost. It is worth noting that the probabilistic ensembles with trajectory sampling (PETS) [5] select actions via model predictive control (MPC) instead of a parameterized policy. Therefore, PETS is another kind of algorithm different from our method. We also did not compare our method to PETS in short horizon tasks. There are the implements that we used to draw figure 2.

- **Soft actor critic (SAC):** [10] A state-of-the-art model-free algorithm for continuous control in terms of sample efficiency. We use the implement from rlkit².

²<https://github.com/vitchyr/rlkit>

- **Proximal policy optimization (PPO):** [33] A simple and efficient model-free algorithm. We use Tingwu Wang’s implement [35].
- **Stochastic Lower Bounds Optimization (SLBO):** [20] A deep model-based algorithm with a theoretical guarantee of monotone improvement to a local maximum of the expected reward. We use the office implement from the author.
- **Model-Ensemble Trust Region Policy Optimization (METRPO):** [15] A deep model-based algorithm that optimizes policy via trust region policy optimization algorithm using imaginary data. We use Tingwu Wang’s implement [35].
- **Model-Ensemble Proximal Policy Optimization (MEPPPO):** A standard deep model-based algorithm that optimizes policy via proximal policy optimization algorithm using imaginary data. we implement it based on the code of our method by setting $\alpha = 0$ and $\beta = 0$.

For all those model-based baselines, we used the default of other hyperparameters. For our method and MEPPPO, we evaluate the performance using the average return of 20 trajectories.

How does POMBu Work?

To further show how our algorithm work, we provide the details during training in Figure 9. In this picture, we show the development of different kinds of values with the increase of iterations.

entropy: The entropy of policy. Here, we report the value at the end of each iteration.

kl: The averaged Kullback-Leibler (KL) divergence between the post-update policy and the pre-update policy. During a iteration, we update the policy multiple times and estimate the averaged KL divergence for each update. A large value of KL means a large the step size of update. We report the average for each iteration.

penalty: $\mathbb{E}_{\tau \sim (\mathcal{M}, \pi_{\theta_{\text{old}}})} \left[\sum_{h=1}^H \left| r_{\theta}(s^h, a^h) - 1 \right| \sqrt{U_{sa}^h} \right]$. The estimated uncertainty over the surrogate objective $L_{sr}(\theta)$. For each iteration, we report the average during this iteration.

p_improve: $L_{clip}(\theta)$. An estimated probability of performance improvement. For each iteration, we report the average during this iteration.

return_model: The average returns of virtual trajectories generated by the models. We report the value at the end of each iteration.

return_real: The average returns of real-world trajectories. We report the value at the end of each iteration.

With the increase of α , the step size decreases. In the early stage, the value of kl is approximately 0 due to the high inaccuracy of models. This show the conservatism of our method. The small step size leads to a slow rate of descent in entropy. This ensures better exploration after multiple iterations. Small α results in higher uncertainty over the surrogate objective and lower probability of performance improvement. When using $\alpha = 0$, the return_model is unstable and much higher than return_real in the early stage, which means overfitting. Using $\alpha > 0$ can significantly alleviate this overfitting, even using $\alpha = 0.25$. It is worth noting that, updating policy with small stepsize—for example, $kl \leq 0.001$ when $\alpha = 0.75$, which is **10 times** smaller than most implements of TRPO choose—can achieve promising sample efficiency.

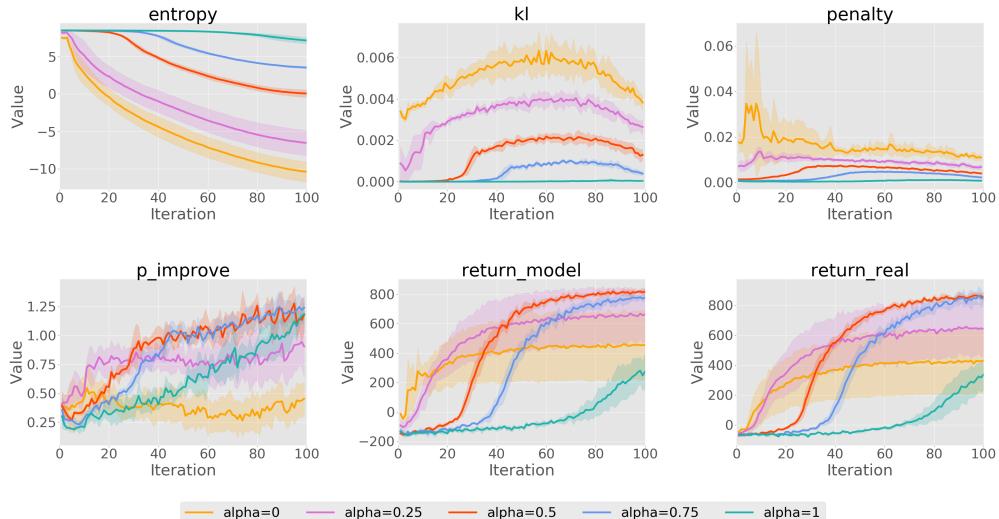
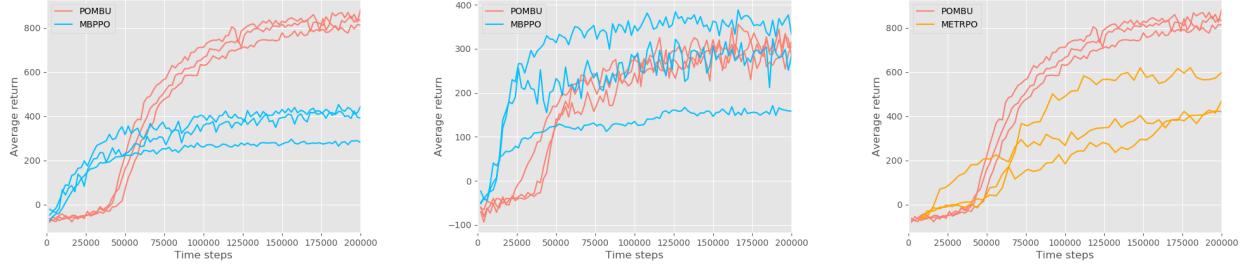


Figure 9: The details during training using different α in HalfCheetah task ($H = 200$).

E Robust Analysis

Here, we provide more results for robust analysis.



(a) We compare POMBU to MEPPPO in a noisy environment ($\sigma = 0.01$). (b) We compare POMBU to MEPPPO in a noisy environment ($\sigma = 0.1$). (c) We compare POMBU to METRPO in a noisy environment ($\sigma = 0.1$).

Figure 10: Robustness comparisons in noisy HalfCheetah tasks .

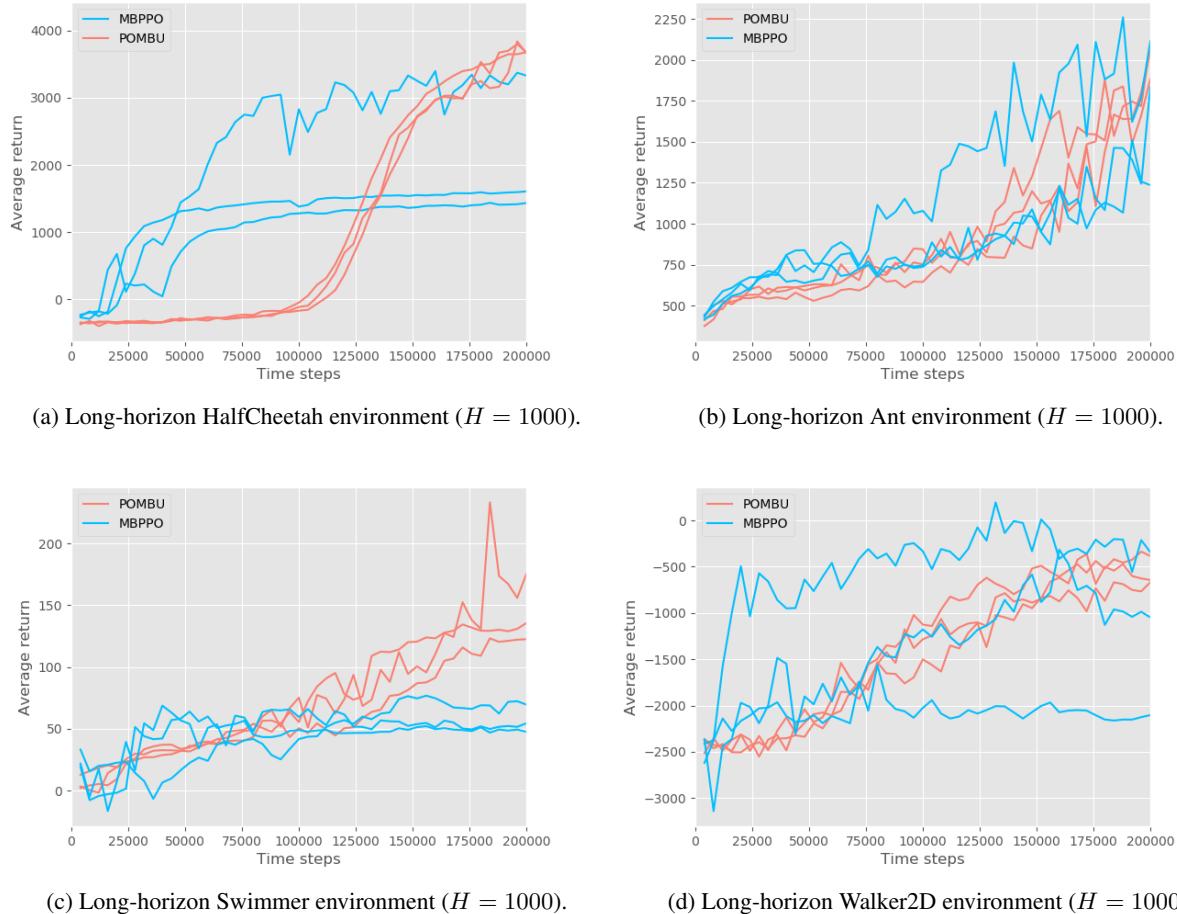


Figure 11: Robustness comparisons between our method and MEPPPO in long-horizon tasks.