Replication of "When to Trust Your Model: Model-Based Policy Optimization"

Xiaoyu Jiang[*], **Qiuxuan Chen**[*], **Shiyi Han**[*], **Mingxuan Li**[*], **Jingyan Dong**[*], **Ruochen Zhang**[*]
Department of Computer Science
Brown University
Providence, RI 02906
{xjiang5,qchen32,shan43,mli115,jdong20,rzhang73}@cs.brown.edu

Abstract

In "When to Trust You Model: Model-Based Policy Optimization," Janner et al. [5] establish a theoretical framework encouraging model usage in reinforcement learning and propose an algorithm that surpasses the performance of state-of-the-art model-based and model-free algorithms. Specifically, the model-based policy optimization (MBPO) algorithm uses short model rollouts branched from real data to reduce the compounding errors of inaccurate models and decouple the model horizon from task horizon, allowing it to achieve higher performance, faster convergence, and better generalizability in complex environments. Our report studies the replicability of the algorithm, examining if the algorithm is able to achieve the performance as the paper claims and if any unstated assumptions about implementation benefits generalization. We reproduce the graphs comparing the learning curve of MBPO with various state-of-the-art model-based and model-free algorithms in the MuJoCo environment, and the result confirms that of the original paper. We also find additional specifications and limitations of the algorithm, both theoretically and empirically.

1 Introduction

Reinforcement learning algorithms mainly consist of two general categories, model-based and model-free methods, with their respective advantages. Model-free algorithms depend on state descriptions to generate a direct mapping from states to actions, making the techniques generalizable to various environments at the cost of high sample-complexity. Conversely, model-based approaches have demonstrated their efficiency by formulating a model imitating the dynamics of the environment along the learning process, which, as previous literature shows, require orders-of-magnitude fewer samples than model-free approaches to achieve good performance [5]. Despite the seemingly promising performance, model-based algorithms mostly fail to extend their successes into noisy, complex environments due to the compounding error between the model and the real dynamics. Thus, many recent studies focus on generalizing model-based algorithms into complex environments while retaining their sample efficiency.

Based on previous works on model-based learning, Janner et al. [5] find that model rollouts of full-task lengths dampen model performance, resulting in failures when facing complex, long-horizon tasks. However, purely model-free learning also cannot catch the rich dynamics in complex tasks with limited samples. Thus, the original paper seeks for a way to combine model-free and model-based methods by constraining the lengths of model rollouts. Specifically, the authors offer a theoretical proof of a bound on model errors that motivates the limited use of truncated, but nonzero-length, model rollouts for learning. In light of the theoretical analysis, the paper introduces Model-based

[*]The authors contributed equally to this paper.

Preprint. Under review.

policy optimization (MBPO), an algorithm that learns through short rollouts and makes limited use of the predictive model. Through empirical analysis, the paper demonstrates that MBPO maintains speedy learning in complex environments, shedding light on the possibility of designing model-based approaches that are both efficient and generalizable.

In this work, we reproduce the original paper's experiments that compare the performance of MBPO with other state-of-the-art model-based and model-free learning algorithms. By presenting our implementation and theoretical analysis of MBPO, we get a better understanding of the algorithm as well as its strengths and weaknesses that Janner et al. [5] do not thoroughly discuss.

2 Background

The original paper uses five baselines to conduct comparative analysis. Among the baseline algorithms, two of them are model-free (SAC and PPO), and three of them are model-based (PETS, STEVE, and SLBO).

SAC Soft actor-critic (SAC) is an off-policy model-free RL algorithm based on the maximum entropy reinforcement learning framework. It combines an actor-critic architecture with separate policy and value function networks, an off-policy formulation that improves data efficiency, and entropy maximization to encourage exploration and increase randomness in the policy. By approximating a maximum entropy variant of the policy iteration formulation, SAC evaluates the Q-function and updates the current policy through an off-policy gradient update [3]. SAC can also extend its application to complex and high-dimensional tasks because it makes use of two Q-functions to mitigate positive bias in the policy update step, which effectively speeds up training and improves performance. MBPO uses SAC in its policy optimization procedure but outperforms SAC in sample efficiency by incorporating a model to generate fictitious training data. The usage of model-generated data in MBPO effectively speeds up learning because it reduces the risk of overfitting and thus increases the ratio between the number of gradient updates and environment samples.

PPO We choose not to include PPO in our replication due to the following reasons. Firstly, the paper introducing SAC uses PPO as its baseline, so we believe the inclusion of SAC alone would be sufficient to fulfill the purpose of the experiment. Secondly, while the design of MBPO refers to the strengths of other baseline algorithms, it has little connection with PPO, thus making its inclusion less informative.

PETS Probabilistic ensembles with trajectory sampling (PETS) is a model-based algorithm that relies on dynamics estimation and planning. Specifically, it combines a high-capacity neural network model, which incorporates uncertainty via an ensemble of deep learning models, with sampling-based uncertainty propagation. It models the aleatoric uncertainty using a probabilistic NN, whose outputs parameterize a Gaussian distribution. It also captures the epistemic uncertainty by replacing the single learned model with an ensemble of learned models and uses a separate sampling of the dataset of true environment interaction to train each model [2]. Although MBPO uses an ensemble of probabilistic neural networks similar to PETS, it uses the learned model to train a policy instead of to plan directly.

STEVE Stochastic ensemble value expansion (STEVE) is a model-based technique that integrates with model-free learning to achieve high performance with low sample complexity while preventing errors in the imperfect dynamics model from degrading the performance. Specifically, the technique uses a dynamic model to compute rollouts that are used to improve the targets for temporal difference learning. By interpolating between different horizon lengths and favoring those whose estimates have lower uncertainty (short-horizon model-based rollouts), the technique incorporates data from these rollouts into value estimation and reduces the error of the model [1]. Compared with MBPO, STEVE does not use model rollouts to learn the policy but uses them to improve the quality of target values of samples collected from the real environment, which lacks directness.

SLBO Stochastic lower bounds optimization (SLBO) is a variant of model-based reinforcement learning algorithms. Its strength lies in its theoretical performance guarantee of monotonic improvement to a local maximum of the expected reward, which inspires the MBPO paper to lay out its framework of proving the proposed algorithm's guarantee of monotonic improvement [6]. Similar to

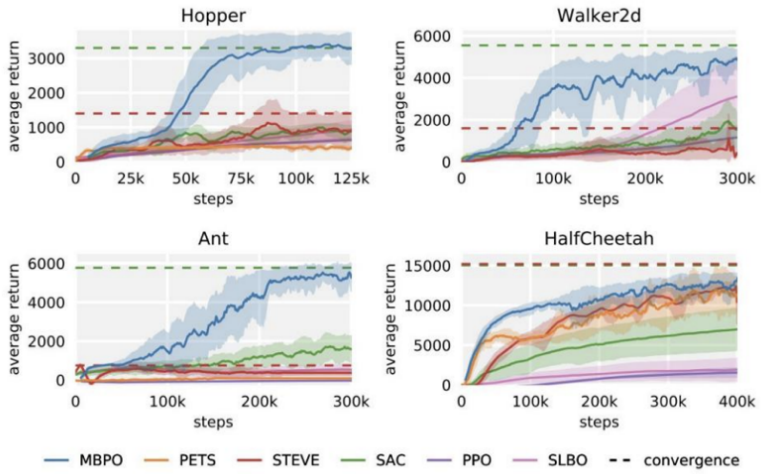

Figure 1: Graph from the original paper indicating the training curves of MBPO and five baselines on continuous control benchmarks. Solid curves depict the mean of five trials, and shaded regions correspond to standard deviation among trials. We evaluate all algorithms on the standard 1000-step versions of the benchmarks.

MBPO, the algorithm also performs model rollouts to evaluate possible next actions in stochastic environments. However, SLBO begins the model rollouts from the initial state distribution, while MBPO uses the branched rollout method to begin each rollout from a state from the previous policy's state distribution. Although the former approach seems to reflect the real model more faithfully as it optimizes a policy purely under the state distribution of the model, it compounds the model rollout length with the task horizon, making it evaluate only on truncated versions of benchmarks due to the limited length of each rollout [5]. By beginning the rollouts from the state distribution of a different policy under the true environment dynamics, MBPO disentangles the model horizon and task horizon by replacing long rollouts from the initial state distribution with many short rollouts starting from the replay buffer states, which relieves the limitation that SLBO faces.

3 Graph

In the original paper, one of the major concerns is how MBPO performs asymptotically on benchmarks compared with other state-of-the-art algorithms. The authors evaluate the performance of MBPO on a set of MuJoCo continuous tasks along with five other baselines.

As Figure 1 shows, MBPO outperforms the state-of-the-art model-free algorithms in that it learns in substantially fewer steps while retaining an asymptotic performance comparable with the model-free baselines. It also obtains significantly better performance than the other three model-based algorithms in high complexity tasks such as Ant.

4 Methods

4.1 Overview

We reimplement this algorithm according to the pseudo-code in the paper. In our evaluation, we aim to confirm the original conclusion that MBPO outperforms state-of-the-art model-based and model-free algorithms in that it has higher sample efficiency and faster convergence.

4.2 Model Description

The general concept of MBPO is to optimize a policy under a learned model – it generates fictitious training data using that policy and trains a new model leveraging the generated data. While con-

ceptually simple, the actual implementation of MBPO can be hard to understand. The predicted returns of the policy under the learned model and the real dynamics can be quite different due to the compounding model errors.

In our replication, we instantiate the algorithm by replacing the single learned model with a bootstrap ensemble of the dynamic model. Each member of the ensemble is a probabilistic NN, whose outputs parametrize a Gaussian distribution to model the aleatoric uncertainty resulting from the inherent variance of the sampling data. The entire bootstrapping procedure handles the epistemic uncertainty, the subjective uncertainty led by a lack of knowledge about model distribution [2]. By handling these two uncertainties properly, we can bridge the gap between the true returns and the model returns. Yet, in cases where the model error is high and improvement is not guaranteed, handling the two uncertainties is not sufficient to improve the model return. To cope with these situations, MBPO takes an approach that interpolates between model-based and model-free updates. In detail, it executes a nonstationary policy which starts from random states seen previously during interaction with the environment and propagates short trajectories under the current policy [5]. In this way, we trade a few long trajectories with large compounding errors with more short trajectories with smaller errors.

For policy optimization, MBPO uses SAC to alternate between a policy evaluation, which computes the value function for a policy using the Bellman backup operator, and a policy improvement steps, which uses the value function obtained previously to get a new policy minimizing the expected KL-divergenc [3]. Unlike similar model-based algorithms that start the model rollouts from the initial state of the distribution, MBPO adopts a branching strategy starting the model rollouts from the state distribution of a different policy under the true environment dynamics [5]. This strategy disentangles model rollout length with the task horizon, thus eliminating the need for full-length rollouts.

4.3 Theoretical Reanalysis

To better replicate the algorithm, we carefully review the theoretical analysis and try to find out the inherent logic chain of the paper's idea. The paper first formulates the learning target as the minimum improvement to gain on the learned model to improve in real dynamics. It turns out that two terms dominate this improvement (Theorem 4.1), which are the generalization error $_m$ and the distribution shift . Theorem 1 shows that if we use model-based learning agents with full-length rollouts, performance improvements in real dynamics are guaranteed by a conservative bound. To tighten this bound, Janner et al. [5] introduce constrained model rollouts into the analysis to deduce Theorem 4.2. Note that the authors first illustrate a more complicated bound in the main body of the paper (Theorem 4.2), but we think it is more reasonable to deduce Theorem 4.2 from Theorem 4.3, as shown in the Appendix.

The main difference between Theorem 4.2 and Theorem 4.3 is that they acquire different assumptions of model error bounds. Theorem 4.2 requires model error to be bounded simply on the distribution of data collecting policy $_D$ while the latter requires the error to be bounded on the distribution of updated policy $_t$ (learned and updated by the model in every time step t). In a supervised learning setting, we cannot get access to model error based on the updated policy because our reply buffer consists of data based on data collecting policy $_D$ instead of the updated $_t$. A simpler model bound can give rise to a good algorithm design, but it is unhelpful if the assumption is unrealistic. To address this problem, in chapter 4.3 the author claims that empirically we can estimate updated model error $_{m'}$ with $_m$ and $\frac{d'_m}{d}$. Thus, we are safe to use Theorem 4.3 to guide our model design.

4.3.1 Reprove of Lemma B.4, Theorem 4.2 and 4.3

In Lemma B.4, following the definition of "before the branch" and "after the branch", we assume that the time step before the branch point k is "before the branch" and the time step after the point k is "after the branch". Thus, we should instead swap the notations of "pre" and "post" in the proof. That is:

For t k:

$$D_{TV}(d_1^t(s;a)||d_2^t(s;a)) \quad t(\,{}^{pre}_{m} + \,{}^{pre}) + \,{}^{pre}$$

and for t k:

$$D_{TV}(d_1^t(s;a)||d_2^t(s;a)) \quad (t \quad k)(\,{}^{post}_{m} + \,{}^{post}) + k(\,{}^{pre}_{m} + \,{}^{pre}) + \,{}^{pre} + \,{}^{post}$$

The new bound induced by these inequalities should be,

$$j \quad {}_1 \quad {}_2 j \quad 2r_{max}\left[\frac{k+1}{(1 \quad)^2}\left(\begin{smallmatrix} post \\ m \end{smallmatrix} + \quad {}^{post}\right) + \frac{k}{1}\left(\begin{smallmatrix} pre \\ m \end{smallmatrix} + \quad {}^{pre}\right) + \frac{k+1}{1} \quad {}^{post} + \frac{1}{1} \quad {}^{pre}\right]$$

Applying the new Lemma B.4 in Theorem 4.2 and Theorem 4.3, following the same proof, we have new results for Theorem 4.2 and Theorem 4.3. That is,

(New Theorem 4.2)

$$[\ ] \quad {}^{branch}[\ ] \quad 2r_{max}\left[\frac{k+1}{(1 \quad)^2}\left(\begin{smallmatrix} \\ m \end{smallmatrix} + 2 \quad\right) + \frac{2}{1}^{k+1} \quad + \frac{k+1}{1} \quad\right]$$

(New Theorem 4.3)

$$[\ ] \quad {}^{branch}[\ ] \quad 2r_{max}\left[\frac{k+1}{1} \quad + \frac{k+1}{(1 \quad)^2} \begin{smallmatrix} \prime \\ m \end{smallmatrix}\right]$$

4.3.2 Insight on proper length of model rollouts

For both the original experiments and our replication, we observe that the model performance benefits from using a short model horizon like 1 in the first place and gradually increasing the model horizon in later epochs. From the reproved version of Theorem 4.3 above, we can see that it is the distribution shift error instead of the generalization error that contributes to the linear term. Since the policy distribution changes rapidly at the beginning and slowly converges as total learning steps grow, using a small k initially will give a tighter bound, resulting in easier improvements on real dynamics. As time evolves, the agent will collect enough samples in the true dynamics and be less likely to encounter unseen states during model training, which means is sufficiently small. Thus, we can then increase k to suppress generalization error $\begin{smallmatrix} \prime \\ m \end{smallmatrix}$ to get a better error bound. It is worthy of mentioning that if we follow the old Theorem 4.2 and Theorem 4.3, a good hyperparameter tuning scheme would be setting a big k at the beginning and gradually decreasing it instead.

4.4 Replication Effort

In our replication, we test on the four MuJoCo tasks that the original graph includes. We use Hopper-v2 as a baseline environment to verify the correctness of our reimplementation. After we get approximately similar performance, we apply the baselines to the other three environments.

Our implementation contains two models - the environment model and the SAC model. In the environment model, we have seven bootstrapped NNs, each with four fully-connected layers. The model receives the concatenation of the sampled state and the action generated by the policy net as its input and outputs the estimated reward and the difference between the next state and the current state. We set the reward scale to 1 to improve the training efficiency. To emulate the uncertainty of the real environment when generating the output, we use a Gaussian distribution similar to the original paper, in which the model generates the mean value and the standard deviation. In the SAC model, we follow the standard settings in the original paper and fine-tune the hyperparameters to achieve good performance. We refer to the paper by Henderson et al. [4] in the process of reproducing the model.

4.4.1 Environment Setup

To replicate the experiments, we use four continuous control tasks (Hopper-v2, Halfcheetah, Ant, Walker2d) in the Mujoco physics simulator, as the original paper does to evaluate MBPO and other baselines. For each task, we run MBPO along with one model-free baseline (SAC) and three model-based baselines (PETS, STEVE, and SLBO) and plot their average returns within the same range of steps. Same as the original experiments, we use the standard full-length version of the tasks and make privileged information, such as fully observable states and the reward function for offline evaluation, unknown to MBPO to evaluate its capability of learning in complex environments.

4.4.2 Parameter Setup

We followed the parameter settings in Appendix C of the original paper. We first implement the learning rule as the original paper, performing M rollouts at each environment step. However, after several attempts, because of the long time that MBPO requires with the original setting, we change the algorithm to conduct model rollouts less frequently, which improves the speed.

4.4.3 Barriers

We replicate the results of the comparative experiment, which demonstrates MBPO's strengths over other baselines. We also enhance the time efficiency of MBPO, which enables us to evaluate its reproducibility on a broader range of problems. To achieve these goals, we make several modifications and assumptions about the implementation unstated by the original paper:

Replay Buffer Size To allow the algorithm to take full advantage of environment exploration, we assume that the policy optimization algorithm of MBPO (SAC) uses a new set of data for training during each epoch. To achieve this, we set the size of the replay buffer in each epoch to be the maximum size of the model dataset (D_{model}), which is E (environment steps per epoch) M (model rollouts per environment step) k (the rollout length). As the algorithm increases k over epochs, for each epoch, we adjust the size of the buffer concerning this rule. In this way, we guarantee a refreshed buffer at every epoch during training using SAC.

Frequency of Rollouts To achieve higher real-time efficiency, we make MBPO perform model rollouts with a lower frequency than the original pseudo-code indicates, as mentioned in Section 4.3.2. Specifically, we define a new parameter f, which controls the interval (the number of environment steps that the algorithm takes) between two calls of the rollout action. For every step, we let MBPO act in the environment and perform gradient updates as the pseudo-code specifies. However, instead of conducting model rollouts in each environment step, we batch the rollout actions together and perform f M model rollouts only in the first step of every f steps. In this way, we empirically speed up the learning process in real-time while performing the same number of rollouts for each epoch.

Configuration of Environments and Baseline Parameters During replication, we use the existing implementations [1][2][3][6] of the baselines along with our implementation of MBPO to plot the graph. However, because Janner et al. [5] do not specify the versions and specific configurations of each task and the hyperparameters of each baseline, we have a hard time making the baselines achieve similar performance as the original graphs indicate. While we manage to replicate the benchmark results for SAC, STEVE, and SLBO, PETS does not show any sign of learning during training, so we choose not to include it in our replication graph. Many possible factors might cause the situation. For example, PETS might require a specific set of hyperparameters to achieve high performance, which we fail to encounter during our experiment. The intricacies of each task's environment configuration might also cause the failure, as the setup of tasks in our replication may not be the same as the settings in the original paper, which may create unknown barriers for PETS to learn.

4.4.4 Adapted Pseudo-code of Our Replication

Algorithm 1 Model-Based Policy Optimization with Deep Reinforcement Learning (Adapted)

Initialize policy , predictive model p , environment dataset D_{env}, model dataset D_{model}
for N epochs **do**
  Train model p on D_{env} via maximum likelihood
  for E steps **do**
    Take action in environment according to ; add to D_{env}
    if E % f = 0 **then**
      for f M model rollouts **do**
        Sample s_t uniformly from D_{env}
        Perform k-step model rollout starting from s_t using policy ; add to D_{model}
    for G gradient updates **do**
      Update policy parameters on model data: $\hat{r}$ J (; D_{model})
      Update parameters of the Q-functions

4.4.5 Graph of the Replication Result

We evaluate all four algorithms on the standard 1000-step versions of the benchmarks. Our replicated graph (Figure 2) demonstrates similar results that the graph in the original paper (Figure 1) shows. Generally speaking, MBPO has a better asymptotic performance than the other three baselines, and

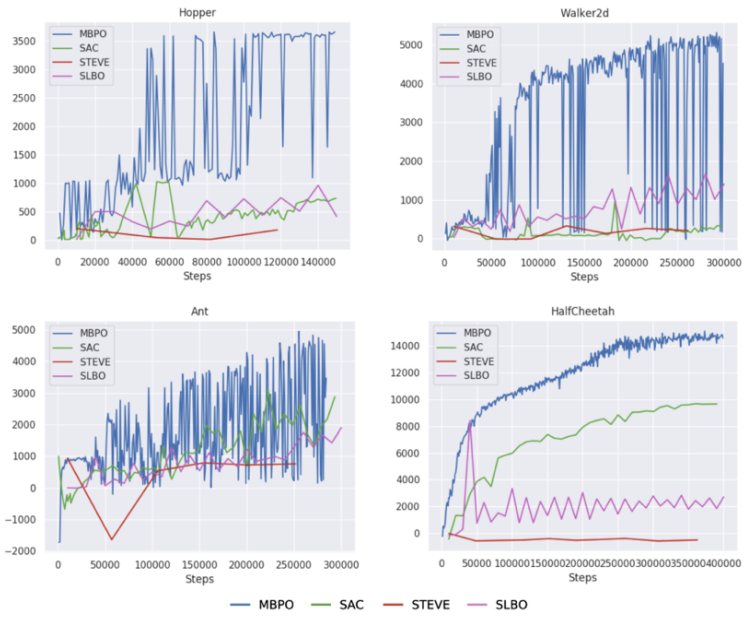

Figure 2: Our replication graph of training curves of MBPO and three baselines on four continuous control benchmarks.

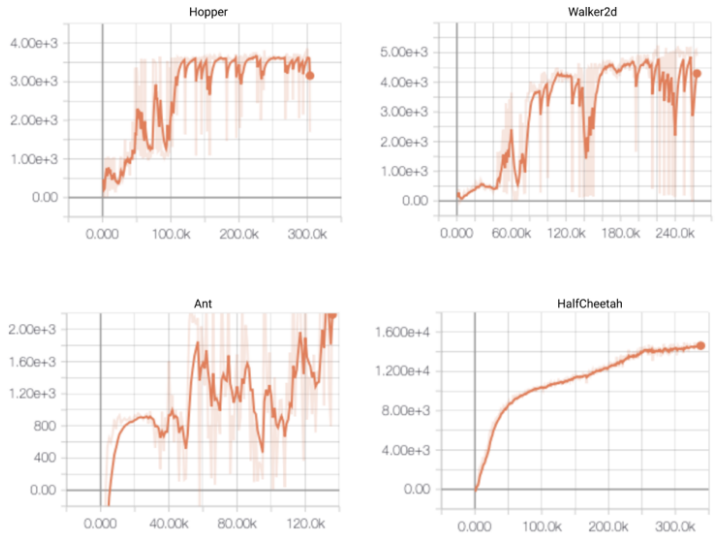

Figure 3: Learning curves of MBPO on four continuous control benchmarks with smoothing applied.

it takes significantly fewer steps to converge compared to the baselines. For example, MBPO's performance on the Ant task at 50 thousand steps matches that of SAC at 3 million steps. To get a clearer view of MBPO's performance, we apply smoothing when plotting Figure 3.

4.4.6 Non-reproduced experiment

We do not replicate the ablation tests due to time and computational resources limitation, despite further insights that they may offer. Based on the original discussion of the paper, these experiments

justify the use of a dynamic model in learning, show that short model-based rollouts can alleviate compounding modeling errors to reach a good performance, and experiment on using rollouts for value expansion. They also show that short rollouts prevent a policy from exploiting the simulated, imperfect model. While these findings are valuable to help us further understand the strengths of MBPO, our primary goal is to test if the core idea of the algorithm has the advantages that the paper claims (high performance, generalizability, faster learning, etc.). Thus, our replication focuses on theoretical justification and the comparative analysis between MBPO and other model-based and model-free algorithms. Our analysis provides insights into algorithm's real performance on a broader range of tasks and evaluates its contribution in pushing forward the boundary of state of the art, which we believe to be more valuable.

5 Results

Theoretically, we review the proof of the paper and illustrate the underlying logic chain that forms the inherent idea of the method. By correcting some possible notation misuses and proofs, we justify the relation between model's rollout length schedule (k) and the two error terms of the bound ($_m$;).

Empirically, we evaluate MBPO's rate of convergence, asymptotic performance, and generalizability to complex environments on four benchmark tasks. We demonstrate that MBPO converges substantially faster than prior state-of-the-art model-free algorithms such as SAC on all four benchmarks, manifesting the benefits of incorporating a model in the training process. MBPO also mitigates the limitations on model accuracy of many model-based algorithms on model accuracy as it reduces compounding errors by replacing long erroneous rollouts with many short rollouts with much smaller errors. For example, MBPO performs significantly better on all tasks than SLBO, which uses longer rollout lengths during training and thus suffers from the inaccuracy of the simulated model. MBPO also maintains desirable performance on high complexity tasks with long task horizons, as its branching technique disentangles the model rollout length with the task horizon. For instance, in the Ant task, where the complexity of the model is high as many parameters vary independently in the environment, MBPO performs well while other model-based algorithms degrade significantly due to the inaccuracy of the simulated models.

Although MBPO requires fewer steps to converge, it takes longer in real-time to perform the same number of steps, which is due to the large number of rollout steps in each epoch, despite our attempts to speed up the algorithm with a lower rollout frequency. Thus, in terms of real-time efficiency, MBPO may perform less desirably than the state-of-the-art baselines.

6 Reflection

In our replication, we demonstrate the strengths of MBPO by reproducing the experiment in the original paper, which compares MBPO's performance with various state-of-the-art model-based and model-free algorithms in the MuJoCo environment. The replication results confirm that MBPO converges faster, performs asymptotically better, and generalizes better to complex environments than the other three baselines (SAC, STEVE, SLBO).

Yet, we notice that sample efficiency does not indicate real-time efficiency. Although MBPO takes the fewest steps to converge in comparison with the other three baselines as the original paper shows, it takes the longest in real-time to perform the same number of steps. Thus, the algorithm is most suitable for tasks with sampling difficulty and high complexity. However, for tasks that require speedy learning, other more time-efficient algorithms may be more preferable.

We also find one unstated assumption of MBPO's implementation that would improve the sampling quality during training. Specifically, by setting the maximum size of the replay buffer in each epoch to be the largest possible size of the model dataset that the rollout process would produce, the mechanism ensures that the algorithm will avoid using the same information repeatedly and always use newly sampled data for training. This modification is valuable especially in data-limited regimes, as it extracts as much information as possible from a limited number of samples.

We also provide justifications and corrections to some proofs in the original Appendix, leading to a more reasonable justification for the hyper-parameter tuning scheme.

References

[1] Buckman, Jacob, et al. "Sample-efficient reinforcement learning with stochastic ensemble value expansion." *Advances in Neural Information Processing Systems*. 2018.

[2] Chua, Kurtland, et al. "Deep reinforcement learning in a handful of trials using probabilistic dynamics models." *Advances in Neural Information Processing Systems*. 2018.

[3] Haarnoja, Tuomas, et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." *arXiv preprint arXiv:1801.01290* (2018).

[4] Henderson, Peter, et al. "Deep reinforcement learning that matters." *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018.

[5] Janner, Michael, et al. "When to Trust Your Model: Model-Based Policy Optimization." *arXiv preprint arXiv:1906.08253* (2019).

[6] Luo, Yuping, et al. "Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees." *arXiv preprint arXiv:1807.03858* (2018).

Appendix

Possible Notation Misuse in Lemma

In Lemma B.2, the author actually uses a bounded TV-distance in the proof. Thus, the condition should instead be "the expected TV-distance between two transition distributions is bounded as $max_t E_{s\sim p_1^t(s)} D_{TV}(p_1(s'|s,a)||p_2(s'|s,a)) \quad \delta$"

In Lemma B.2, page 16 line 2, it should be $\quad \frac{1}{2}\sum_s(\ )$. Line 5 should be $\epsilon_0 + \sum_{i=1}^t \delta_i$, and the same for Line 6.

In Lemma B.3, the condition should be "the expected TV-distance between two dynamics distributions is bounded as $max_t E_{s\sim p_1^t(s)} D_{TV}(p_1(s'|s,a)||p_2(s'|s,a)) \quad \epsilon_m$"

In Lemma B.4, the condition should be "... the dynamics distributions are bounded as

$$max_t E_{s\sim p_1^t(s)} D_{TV}(p_1^{pre}(s'|s,a)||p_2^{pre}(s'|s,a)) \quad \epsilon_m^{pre}$$

and after the branch as $max_t E_{s\sim p_1^t(s)} D_{TV}(p_1^{post}(s'|s,a)||p_2^{post}(s'|s,a)) \quad \epsilon_m^{post} \ldots$"

In Lemma B.4, the last part of the proof, the first inequality should be,

$$D_{TV}(d_1(s,a)||d_2(s,a)) \quad (1 \quad \gamma)\sum_{t=0}^{\infty}\gamma^t D_{TV}(d_1^t(s,a)||d_2^t(s,a))$$

Justifications

In this section, we provide some justification for the part of proof that we find confusing at the first place.

In Lemma B.3, the authors claim that "...We now apply Lemma B.2, using $\delta = \epsilon_m + \epsilon_\pi \ldots$ (via Lemma B.1)". To use this bound, we have to make sure that $max_t E_{s\sim p_1^t(s)} D_{TV}(p_1(s'|s)||p_2(s'|s)) \quad \delta$. We can show that,

$$\begin{aligned}
&max_t E_{s\sim p_1^t(s)} D_{TV}(p_1(s'|s)||p_2(s'|s)) \\
&\quad max_t E_{s\sim p_1^t(s)} D_{TV}(p_1(s'|s,a)||p_2(s'|s,a)) \\
&\quad + max_t E_{s\sim p_1^t(s)} max_s D_{TV}(\pi_1(a|s)||\pi_2(a|s)) \\
=&max_t E_{s\sim p_1^t(s)} D_{TV}(p_1(s'|s,a)||p_2(s'|s,a)) \\
&\quad + max_s D_{TV}(\pi_1(a|s)||\pi_2(a|s)) \\
&\quad \epsilon_m + \epsilon_\pi
\end{aligned}$$

In Lemma B.4, the authors claim that

For $t \quad k$:

$$D_{TV}(d_1^t(s,a)||d_2^t(s,a)) \quad t(\epsilon_m^{post} + \epsilon_\pi^{post}) + \epsilon_\pi^{post}$$

This can be verified using Lemma B.1 and the proof step in Lemma B.3 we justified above,

$$\begin{aligned}
D_{TV}(d_1^t(s,a)||d_2^t(s,a)) \quad & D_{TV}(p_1^t(s)||p_2^t(s)) + max_s D_{TV}(\pi_1(a|s)||\pi_2(a|s)) \\
& t(\epsilon_m^{post} + \epsilon_\pi^{post}) + \epsilon_\pi^{post}
\end{aligned}$$

It is the same for the case of $t \quad k$.