MUSBO: Model-based Uncertainty Regularized and Sample Efficient Batch Optimization for Deployment Constrained Reinforcement Learning

DiJia Su[1], Jason D. Lee[1], John M. Mulvey[2], and H. Vincent Poor[1]

[1]Department of Electrical and Computer Engineering, Princeton University
[2]Department of Operation Research and Financial Engineering, Princeton University

Abstract

In many contemporary applications such as healthcare, finance, robotics, and recommendation systems, continuous deployment of new policies for data collection and online learning is either cost ineffective or impractical. We consider a setting that lies between pure offline reinforcement learning (RL) and pure online RL called deployment constrained RL in which the number of policy deployments for data sampling is limited. To solve this challenging task, we propose a new algorithmic learning framework called Model-based Uncertainty regularized and Sample Efficient Batch Optimization (MUSBO). Our framework discovers novel and high quality samples for each deployment to enable efficient data collection. During each offline training session, we bootstrap the policy update by quantifying the amount of uncertainty within our collected data. In the high support region (low uncertainty), we encourage our policy by taking an aggressive update. In the low support region (high uncertainty) when the policy bootstraps into the out-of-distribution region, we downweight it by our estimated uncertainty quantification. Experimental results show that MUSBO achieves state-of-the-art performance in the deployment constrained RL setting.

1 Introduction

Recent advances in deep learning have enabled reinforcement learning (RL) to achieve remarkable success in various applications (Silver et al., 2017; OpenAI et al., 2019b; Vinyals et al., 2019; OpenAI et al., 2019a). However, despite RL's success, it suffers many problems. In particular, traditional RL algorithms require the agent to interact with the real world to collect large amounts of online data with the latest learned policy. However, the online deployment of the agent to the real world might be impossible in many real world applications (Matsushima et al., 2020). In the field of robotics or self-driving cars, for example, the cost of deploying the agent to the field may be too risky for the agent itself as well as its surrounding environment. In quantitative finance, trading strategy is usually carefully back-tested and calibrated offline with historical data and simulation. Since the market data has a low signal-to-noise ratio (Chen et al., 2020), online training can easily lead the policy fits to the noise, which is dangerous and can potentially trigger unexpected large monetary loss. In the recommendation system setting (Peska and Vojtas, 2020), after the policy has been trained offline, it will be deployed across different servers for serving many users. In such setting, online training of policy is difficult because the resulting policy might become unstable and/or cause bad user experience. Thus, large chunk of data is collected for the deployed policy. Then, the data is used for offline training. During the next scheduled deployment, the production level (data collection) policy is then updated.

To address this challenge, training an RL agent in an offline fashion seems to offer solution. In offline RL (Levine et al., 2020), a static dataset is collected by a behavioral (or data collection) policy. Since

Preprint.

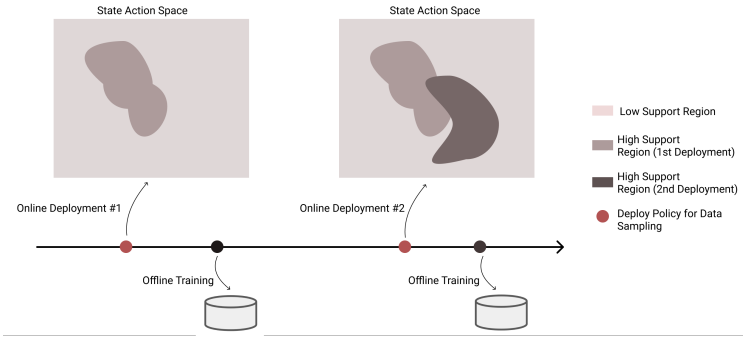

Figure 1: Illustration showing the deployment constrained RL setting. Large batch of data collection only occurs at the deployment point (showing two of them in red). The deployment and offline training occurs in an interleaved fashion. As the number of deployment increases, the state-action space will be explored more and more. Here, thicker-brown colored regions show the area of high support whereas the lighter color shows an under-explored (low support) region.

the RL agent has no access to the online environment, training the RL agent faces many challenges. First, because the learning policy and the behavioral policy have different state visitation frequencies, evaluation of the offline policy is difficult. Second, during the training process, this discrepancy of the distribution might increase over the training time, a phenomenon known as distributional shift. Third, there might be a large extrapolation error when the value function is bootstrapped into out-of-distribution actions (Kumar et al., 2019), a phenomenon that can lead to learning divergence and instability. There is a large body of research work on pure offline RL. Despite their empirical success, since the pure offline RL environment is fundamentally different than our setting, we suspect developing novel deployment constrained RL can possibly achieve a large headroom of improvement, since offline RL algorithms assume zero interaction with the environment.

In this work, we present Model-based Uncertainty regularized and Sample Efficient Batch Optimization (MUSBO), a general framework for doing batch policy optimization in the deployment constrained setting. To build an accurate model of the environment, our method encourages the data collection policy to *discover high quality and novel data batches during each online deployment*. During the offline training, MUSBO samples fictitious rollouts from the learned model and performs policy update weighted with *uncertainty regularized coefficient*, a term that quantifies the amount of uncertainty with respect to each state-action pair within the fictitious rollouts and the data. Our optimization process regularizes the update toward the high confidence regions. In areas of low confidence or low data support, MUSBO takes a pessimistic point of view and discounts the update by the uncertainty regularized coefficient when the policy bootstraps into out-of-distribution states and actions regions. We empirically compare our MUSBO to other strong baselines such as the state-of-the-art deployment constrained RL algorithms (BREMEN) (Matsushima et al., 2020), and we show that MUSBO is capable of achieving significant policy improvement while using smaller amounts of data and fewer deployment.

2 Related Works

The related research literature can be broadly categorized into three categories: deployment-constrained RL, offline RL, and model-based RL.

Deployment-constrained RL. To the best of our knowledge, Matsushima et al. (2020) is the first work that proposed the concept of deployment-constrained efficiency. In their paper, the authors proposed the algorithm Behavior-Regularized Model-ENsemble (BREMEN) that enforces KL-divergence between the learning policy and the behavioral policy for learning update. They compared their approach across numerous baseline such as Soft-Actor-Critic (SAC)(Haarnoja et al., 2018), Model-based-TRPO (METRPO)(Kurutach et al., 2018a), and show that their methodology provides the strongest empirical performance. Here in our paper, we compare our method MUSBO directly with a state-of-the-art method (BREMEN). Different from BREMEN, we first provide a theoretical analysis characterizing the improvement in terms of value function and optimization lower bounds. Second, our method is different because we emphasize at 1) propose to weight the contribution of each policy update by uncertainty quantification, 2) propose a method of measuring state action

uncertainty by utilizing a new set of dynamics models with next state estimation error, 3) propose to make use of uncertainty quantification to maximize the discovery of novel data transitions during each deployment.

On the other hand, there also some related research works such as Bai et al. (2020) and in semi-batch RL such as Ernst et al. (2005); Lange et al. (2012); Jaakkola et al. (1999); Chu and Kitani (2020).

Offline RL. Unlike deployment-constrained setting, offline RL assumes no interaction with the environment. Thus, the learning policy needs to reason about the behavioral policy and make policy update base on that. The two most related literatures are Yu et al. (2020) and Kidambi et al. (2020). In Yu et al. (2020), the authors proposed an uncertainty penalized MDPs in which the reward function was used to explicitly penalize the uncertainty. On the other hand, the authors from Kidambi et al. (2020) proposed a pessimistic MDP that divides the environment into two regions: known or unknown. When the agent is entering into the unknown region, the MDP undergoes a halt state (or absorbing state) in which a large negative reward will be assigned to penalize this action. In both cases, the proposed uncertainty penalization comes from the reward function, which is a totally different setup than what is being proposed in our paper. Here, our method enables a regularization approach. Our method regularizes our policy update toward high confidence region while down-weights or regularized it away when the policy bootstraps into the unfamiliar *state and actions* regions.

Besides these two, there is also a large body of offline RL research. In model-free offline RL, the common techniques are either 1) enforcing the learning policy to stay close with the behavioral policy as in Fujimoto et al. (2019); Kumar et al. (2019); Wu et al. (2019); Nachum et al. (2019b); Zhang et al. (2020); Nachum et al. (2019a) or 2) ensembles of Q values for stabilizing the learning and behaviors as in Ghasemipour et al. (2021); Wu et al. (2019); Nair et al. (2020).

Model-Based RL. In model-based approach, the world representation is learned first and then is used for generating imaginary rollouts. Related research works are Chua et al. (2018); Janner et al. (2019); Luo et al. (2019); Munos and Szepesvári (2008). However, direct application of MBRL methods into the offline setting can be challenging due to distribution shifts. Nevertheless, the closely related research is Kurutach et al. (2018a), in which an ensemble of estimated model dynamics is used for generating fictitious rollous for stabilizing effects. Similar approaches have also been investigated by Zhang et al. (2019); Kaiser et al. (2020); Veerapaneni et al. (2020); Feinberg et al. (2018).

2.1 Background

We consider a discounted, infinite horizon *Markov decision process (MDP)*, and denote $\Omega = (S, A, M, r, \mu_0, \gamma)$, where S is the state space, A is the action space, $M(s'|s,a)$ is the state transition kernel of transiting to a next state s' from state s while taking action a, $r(s,a)$ is the reward function, $\gamma \in (0,1)$ is the discounting factor, and μ_0 is the initial state distribution.

In RL, the goal is to optimize a policy $\pi(a|s)$ such that the expected discounted return $J_M(\pi)$ is maximized: $J_M(\pi) = \mathop{\mathbf{E}}_{\pi,M,\mu_0} \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)$. The value function is defined as $V_M^\pi(s) = \mathop{\mathbf{E}}_{\pi, s_{t+1}\sim M(s_t,a_t)} \{\sum_{t=0}^{\infty} \gamma^t r(s_t,a_t) | s_0 = s\}$. We denote the optimal policy $\pi^* = \text{argmax}_\pi J_M(\pi)$. We further define ρ_M^π to be the discounted state visited by π on M such that $\rho_M^\pi = (1-\gamma)\sum_{t=0} \gamma^t p_{S_t^\pi}$, where $p_{S_t^\pi}$ is the distribution of state at time t.

In the **deployment constrained** setting, there is a limitation on the number of times that the policy can be deployed online for data collection. In real world applications such as recommender system or robotic control, due to the associated cost with each deployment, it is desirable to minimize the number of deployments to as small as possible. Let I be the number of times of deployment, and let $|B|$ be the size of a large batch of training data collected when a policy has been taken online (or deployed), then, the total size of data collected throughout the entire process is $I \times |B|$. Further, let $B^{(i)}$ be the batch of data collected during the i_{th} deployment. In each batch B, we store the data transition $\{(s,a,r,s')\}$. Note the difference between the deployment constrained setting and the pure offline RL setting. In the pure offline RL setting, we consider it as a single batch of data with $I = 1$, whereas in the deployment constrained setting, the agent will have the opportunity to interact with the environment by deploying its learning or other behavioral policy for data collection purposes.

In **Model-Based RL** (MBRL), the world dynamics are learned from the sampled data. We estimate the ground truth environment transition dynamics M^* by learning a forward next state transition dynamics model $\hat{M} : S \times A \to S$. Let $\hat{M}^{(i)}$ be the estimated dynamics bootstrapped from data batch

$B^{(1)} \bigcup B^{(2)} \bigcup ... \bigcup B^{(i)}$, in which we assume the model is updated with the aggregated data batches up to the i_{th} deployment. We denote by $\hat{M}_{\theta}^{(i)}$ the estimated dynamics learned by neural networks parameterized by θ.

3 Algorithmic Framework

For the purpose of exposition, we start by presenting an idealized version of MUSBO algorithmic framework. Then, we describe a practical version that we have implemented and perform quite well. The detailed algorithm is summarized in Algorithm 1.

The heart of the algorithm relies on uncertainty estimation. We postulate that by having more diversified samples (data coverage), we can learn a more accurate representation of the environment, which is the key ingredient of MBRL. To get the most out of each deployment, our algorithm tries to explore the under-represented region (the low support area of our collected data) so as to minimize the amount of uncertainty or maximize the information gain. After building an accurate model, we can use it to generate fictitious rollouts for policy training. The idea is that we want to weight the contribution of each policy update by taking the uncertainty quantification on the fictitious rollouts (each state action pair) by **uncertainty regularized coefficient** ($U_{\hat{M}^{(i)},M^*}(s,a)$) which characterize the level of uncertainty results from model estimation error (eq. (2)). For fictitious rollouts that we have low data support, we discount the policy update by $U_{\hat{M}^{(i)},M^*}(s,a)$ and discourage the policy from bootstrapping into unknown regions. In doing this, we are *regularizing* the policy update into high support/confidence regions.

Theoretical Results. All the proofs in this section are deferred to the appendix. In the deployment constrained setting, we are interested in iterative improvement after each deployment. Let π_{ref} be a reference policy that we will deploy to collect the data, $d(\cdot,\cdot)$ be the closeness of the two policies, $D(\cdot,\cdot)$ be the discrepancy between the M^* and $\hat{M}$, we make the followings four assumptions:

$$V_{M^*}^{\pi} \geq V_{\hat{M}}^{\pi} - D_{\pi_{\text{ref}},\delta}(\hat{M},\pi),\ \text{s.t.} d(\pi,\pi_{\text{ref}}) \leq \delta \quad \text{(A1)} \qquad \hat{M} = M^* \implies D_{\pi_{\text{ref}}}(\hat{M},\pi) = 0 \quad \text{(A2)}$$

$$\text{L-Lipschitz in } V_{\hat{M}}^{\pi} \text{ w.r.t to some norm such that} |V_{\hat{M}}^{\pi}(s) - V_{\hat{M}}^{\pi}(s')| \leq L\|s - s'\|, \forall s, s' \quad \text{(A3)}$$

$$D_{\pi_{\text{ref}}}(\hat{M},\pi) \text{ is given by the form of } E_{\tau\sim\pi_{\text{ref}},M^*}\{f(\hat{M},\pi,\tau)\} \quad \text{(A4)}$$

Let $\hat{M}^{(i)}$ be the model estimated from the data collected up to the i_{th} deployment. Adapted from Schulman et al. (2017); Luo et al. (2019):

Lemma 1. *The difference of the value functions in-between each deployment is bounded by:*

$$|V_{\hat{M}^{(i+1)}}^{\pi} - V_{\hat{M}^{(i)}}^{\pi}| \leq \kappa L \mathop{\mathbf{E}}_{(s,a)\sim\rho_{\hat{M}^{(i+1)}}^{\pi}} (\|\hat{M}^{(i+1)}(s,a) - \hat{M}^{(i)}(s,a)\|) \quad (1)$$

We define the following function $U_{\hat{M}^{(i)},M^*}(s,a)$: $S \times A \to \mathbb{R}$ as an **uncertainty regularized coefficient**:

$$\mathop{\mathbf{E}}_{(s,a)\sim\rho_{\hat{M}^{(i)}}^{\pi}} \{U_{\hat{M}^{(i)},M^*}(s,a)\} \triangleq g\Big(\kappa \mathop{\mathbf{E}}_{(s,a)\sim\rho_{\hat{M}^{(i)}}^{\pi}} \|M^*(s,a) - \hat{M}^{(i)}(s,a)\|\Big) \quad (2)$$

where $g(x) \triangleq (V_{\hat{M}^{(i)}}^{\pi} - x)(V_{\hat{M}^{(i)}}^{\pi})^{-1}$, and $\kappa \triangleq (1-\gamma)^{-1}\gamma$.

Proposition 1. *Let $U_{\hat{M}^{(i)},M^*}(s,a)$ be the uncertainty regularized coefficient defined by eq.* (2)*, then the performance of π on the ground-truth M^* is lower bounded by that of the estimated $\hat{M}^{(i)}$, weighted by $U_{\hat{M}^{(i)},M^*}(s,a)$:*

$$V_{M^*}^{\pi} \geq V_{\hat{M}^{(i)}}^{\pi} \mathop{\mathbf{E}}_{(s,a)\sim\rho_{M^*}^{\pi}} \{U_{\hat{M}^{(i)},M^*}(s,a)\}. \quad (3)$$

Proposition.1 says that in optimizing the lower bound (the RHS of eq. (3)) with $V_{\hat{M}^{(i)}}^{\pi}$, we can maximize the overall performance of π on the real dynamics M^*.

Interpretation of $U_{\hat{M}^{(i)},M^*}(s,a)$ (uncertainty regularized coefficient). Here, the term can be interpreted as an uncertainty quantification measure as a result of *model estimation error* between $\hat{M}^{(i)}$ and M^*, with $\hat{M}^{(i)}$ being learned from the collected (and limited amount of) data up to the i_{th} deployment. On the regions where we have high data support, $\hat{M}^{(i)}(s,a)$ is close to $M^*(s,a)$, thus their discrepancy decreases and $U_{\hat{M}^{(i)},M^*}(s,a)$ approaches to unity. On the regions that are less certain, the discrepancy increases and thus $U_{\hat{M}^{(i)},M^*}(s,a)$ decreases.

Remark 1. *If our estimated model $\hat{M}^{(i)}$ is reasonably good, so that the model estimation error $\kappa\|M^*(s,a) - \hat{M}^{(i)}(s,a)\|$ is smaller than $V^{\pi}_{\hat{M}^{(i)}}$, and that $V^{\pi}_{\hat{M}^{(i)}}$ is positive, then $U_{\hat{M}^{(i)},M^*}(s,a)$ is a scalar between 0 and 1, and is approximately proportional to the accuracy of the model estimation.*

Thus to optimize $V^{\pi}_{M^*}$, we instead optimize its lower bound, which is the value function at the estimated model $V^{\pi}_{\hat{M}^{(i)}}$ weighted by the uncertainty regularized coefficient. We assign a higher weight at the regions of high confidence, and a lower weight at the regions of low confidence. Next, we utilize our result from eq. (3) for establishing our MUSBO lower bound optimization algorithm.

Algorithm 1 MUSBO: Model-based Uncertainty Regularized and Sample Efficient Batch Optimization

Given: Size of data batches B, Number of deployments I, $D_{all} \leftarrow \{\}$
for *i = 1, 2, 3, ..., I of deployments* **do**
  deploy π online for data collection
1 $D_{all} \leftarrow D_{all} \bigcup$ Collect-Data-Low-Support-Region(π)
2 $\hat{M}_\theta \leftarrow$ Learn approximate transition dynamics model
3 Train Uncertainty-Labeler(D_{all})
4 $\pi \leftarrow$ Train with MBRL(Uncertainty-Labeler, $\hat{M}_\theta, D_{all}$) as in section 3.2
return π

3.1 Practical Implementation

In this section, we explain the practical implementation of algorithm 1 in detail. For readers convenience, we have labeled the line number in the algorithm, and we will refer to the line number as we explain below.

Learning the transition dynamics (line 2): For estimating the transition dynamics $\hat{M}_\theta$, we use an ensemble of N deterministic dynamics models with multi-layer fully-connected perceptrons as in Mishra et al. (2017); Kurutach et al. (2018b) to reduce model bias. They are trained to predict the next state with L_2 loss:

$$\min_\theta \frac{1}{|D_{all}|} \sum_{(s_t,a_t,s_{t+1}) \in D_{all}} \|s_{t+1} - \hat{M}_\theta(s_t,a_t)\|_2^2. \tag{4}$$

Since we use the data collected up to i_{th} deployment, we drop the (i) superscript on $\hat{M}^{(i)}_\theta$ hereafter.

Uncertainty-Labeler (line 3). This module is responsible for characterizing the levels of uncertainty in the collected data and approximated $U_{\hat{M}^{(i)},M^*}(s,a)$. Specifically, we want to identify the regions in our data that have high support or low support with respect to uncertainty. Since the oracle is unavailable to us, here we can only approximate the uncertainty by state-actions visitation frequency. Shall the state-actions visited frequently, more pairs of them will show up in the data. If we were to train models on the batches of data to predict the next state transition dynamics, then, our prediction will be more accurate on the states that we have seen (in the collected data), and less accurate on the unfamiliar state. Thus, we have established that the uncertainty quantification measure as the next state prediction error.

In the actual implementation, we have used K ensembles of Probabilistic Neural Networks (PNN) (Chua et al., 2018) to *capture the uncertainty* within the data. Let $\hat{P}_\phi : S \times A \to S$ be the PNN with parameter ϕ. In PNN, the network has its output neurons parameterized by a Gaussian distribution in the effort of capturing the uncertainty. As explained above, we use $\hat{P}_\phi$ to quantify the uncertainty by prediction error. The lower the prediction error in relation to the ground truth, the higher the support. Our uncertainty-labeler approximates the uncertainty regularized coefficient $U_{\hat{M}^{(i)},M^*}(s,a)$ by $\hat{U}(a,s)$ (define below shortly), and we have dropped the dependency on subscript since $\hat{U}$ is trained with the data collected up to the i_{th} deployment.

The actual implementation of $\hat{U}(a,s)$ is motivated by remark 1. Let $\hat{\tau}$ be the fictitious trajectory generated by $\hat{M}_\theta$ (the learned dynamics), we use $\hat{P}_\phi$ to quantify the uncertainty. Since the ground truth state is unavailable to us in the fictitious rollouts, to calculate the prediction error, we further approximate it by the intra-discrepancy error within the ensembles. We randomly sample two models (a, b) from K Uncertainty-Labeler ensemble, for each state-action pairs in $\hat{\tau}$, we label them :

$$\widehat{U}(s_t, a_t) = \exp(-\alpha \|\hat{s}_{t+1}^{(a)} - \hat{s}_{t+1}^{(b)}\|_1), \text{for } (s_t, a_t) \in \hat{\tau} \tag{5}$$

where $\hat{s}_{t+1}^{(a)} = \hat{P}_\phi^{(a)}(s_t, a_t)$ is the next state predicted by the sampled a_{th} model from the Uncertainty-Labeler (and similarly for b index), and $\alpha > 0$ is a temperature parameter. Here, we have assumed that our estimated models $\hat{P}_\phi$ are operating on the "reasonably good" regime (as by remark 1).

Note that here, we have used different sets and different types of networks for estimating the model dynamics $\hat{M}_\theta$ and the uncertainty $\hat{P}_\phi$. We use $\hat{M}_\theta$ (deterministic networks) to generate fictitious rollouts, and we use $\hat{P}_\phi$ (PNN, a specialized uncertainty probabilistic network) as the "judge" to quantify the uncertainty for the generated fictitious rollouts. We separate out these two networks intentionally to avoid possible error propagation between the two modules. $\hat{P}_\phi$ is trained by (eq. (17)). For detail discussion, see Appendix F.

Collect Data from Low Support Region (line 1). To maximize the benefit out of each deployment and to build an accurate representation $\hat{M}_\theta$ of the environment, we emphasize that we want to achieve the maximal data coverage as well as exploring the under-explored region (the low support area).

Exploration in the deployment constrained setting is non-trivial because it is hard to evaluate which action-state pairs will fill the low support region and lead to high data coverage. During online data sampling, for each state that our agent encounters, we identify the amount of uncertainty by taking the actions which lead to maximal *prediction error* between our predicted next state ($\hat{s}'$) and the ground truth next state (as by eqn.16 in the Appendix E). Following a trajectory of maximal prediction error leads to novel experience discovery and reduce the number of unknown regions within the data. In the ablation study (Fig.4 a. and b.), we show that this strategy leads to more accurate learned dynamics $\hat{M}_\theta$. (For detailed implementation, please refer to Appendix E).

3.2 Offline Training with MBRL Method

Next, we define our offline model-based training method (line 4). In the function below, we provide a practical instantiation of MUSBO offline model-based method for learning a policy.

Our method utilizes trust region policy optimization (TRPO) (Schulman et al., 2017) with fictitious trajectory generated with learned ensembles of transition dynamics $\hat{M}_\theta$ weighted by Uncertainty-Labeler (or uncertainty regularized coefficient).

Function `MUSBO MBRL Training`($\hat{M}_\theta$, D_{all}, *Uncertainty-Labeler*)**:**

5 Initialize π_{init} with the data collection policy.

6 **for** *training iterations* **do**

7 Randomly sample a dynamics model from N ensembles of $\hat{M}_\theta$

for *optimization steps* **do**

8 $\hat{\tau} \leftarrow$ sample fictitious trajectory from $\hat{M}_\theta$

9 $\hat{\tau}_{labeled} \leftarrow$ label fictitious trajectory with Uncertainty-Labeler ($\hat{\tau}$) as in eq.(5)

10 train with uncertainty regularized TRPO with $\hat{\tau}_{labeled}$, π_{init} as in eq.(6)

Fictitious trajectory generated from learned dynamics (line 7 to line 8). After each deployment, the environment dynamics transitions are estimated by an ensemble of $\{\hat{M}_\theta\}_1^N$ trained using D_{all}. To generate a fictitious trajectory, we first randomly sample a model $j \in (1, 2, .., N)$, and then we roll-out the trajectory $\hat{\tau}$ by running the learning policy π with the next state as $\hat{s}_{t+1} = \hat{M}_{\theta_j}(\hat{s}_t, a_t = \pi(\hat{s}_t))$, for $t \in 1, ..., T$.

TRPO training with Uncertainty (line 9 to line 10). Next, we train the policy with uncertainty regularized TRPO. In the offline setting, TRPO is trained using imaginary rollouts generated from the learned dynamics $\hat{M}_\theta$. Utilizing Prop 1, we use TRPO to optimize the $V_{M^*}^\pi$ by improving its lower-bound (the RHS). Here, we replace $V_{\hat{M}^{(i)}}^\pi(s)$ by $A^{\pi_{\vartheta_k}}(s, a)$, and we approximated $U_{\hat{M}^{(i)}, M^*}(s, a)$ by $\widehat{U}(s, a)$ (eqn.5), with TRPO:

$$\begin{aligned} &\underset{\vartheta}{\text{argmax}}\ E_{\pi_\vartheta(s), s, \hat{P}_\phi^{(a,b)}, \hat{M}_\theta}\{\frac{\pi_\vartheta(a|s)}{\pi_{\vartheta_k}(a|s)} A^{\pi_{\vartheta_k}}(s, a)\widehat{U}(a, s)\} \\ &\text{s.t. } E_{a\sim\pi_\vartheta(s), s, \hat{P}_\phi^{(a,b)}, \hat{M}_\theta}\{D_{KL}(\pi_\vartheta(\cdot|s)\|\pi_{\vartheta_k}(\cdot|s))\} \leq \delta \end{aligned} \tag{6}$$

where we set $\pi_{\vartheta_0} = \pi_{\text{init}}$ as the initial policy of the TRPO at the first iteration (as a way to impose the constraint on π_{ref}). Here, $A^{\pi_{\vartheta_k}}(s, a)$ is the advantage function of policy π_{ϑ_k} following a fictitious trajectory generated by $\hat{M}_\theta$. Here, when the policy bootstraps into out-of-distribution states and actions regions (low support area), we discount the update. In doing this,our optimization process regularizes the update toward the high confidence regions.

4 Empirical Results

We empirically evaluate our proposed MUSBO algorithm with five continuous control benchmarks using the MuJOCO[1] physics simulator: Walker2d, Hopper, Half-Cheetah, Ant, and Cheetah-Run. **Baseline**. We compare our MUSBO with BREMEN, a state-of-the-art algorithm[2] that is designed

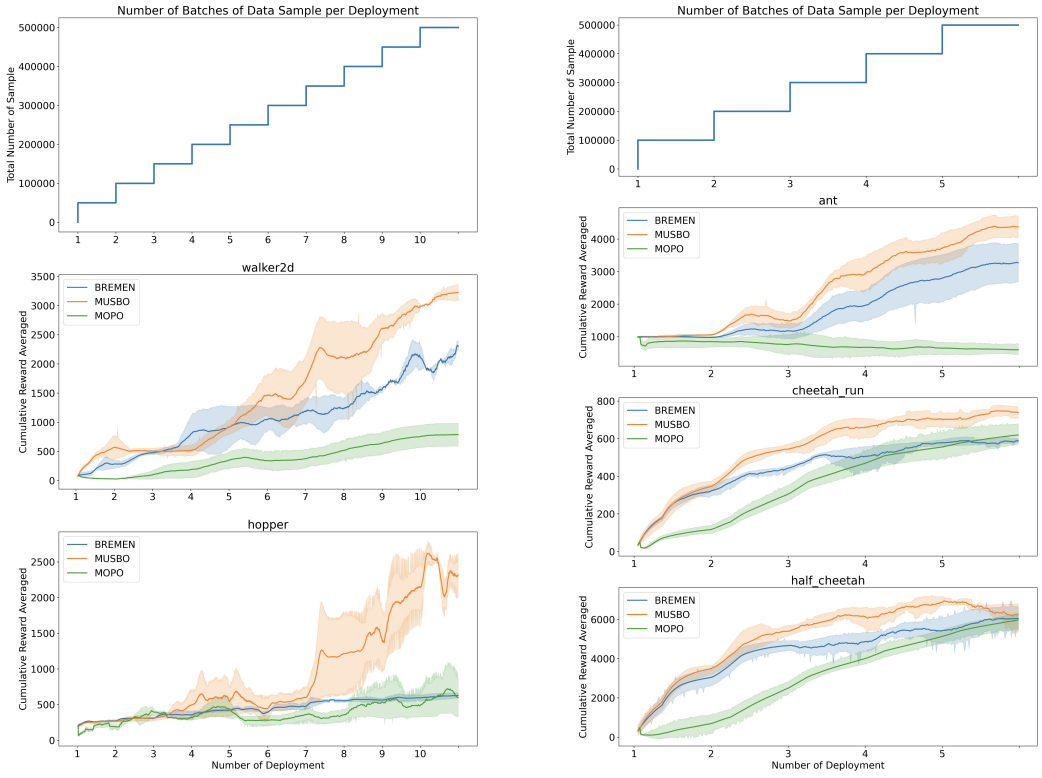

Fig a) Fig b)

Figure 2: Empirical Evaluations of a) Walker2D and Hopper for $I = 10$ deployments, $|B| = 50k$, and b) Ant, Half_Cheetah and Cheetah_Run for $I = 5$ deployments, $|B| = 100k$. Total data consumption in both settings is $I \times |B| = 500k$. The x-axis is aligned showing the number of deployment.

specifically for deployment constrained setting. For BREMEN, we used the exact same hyper-parameters as what was originally proposed in Matsushima et al. (2020). As for comparison, we also adapted MOPO (Yu et al., 2020), a pure-offline algorithm, to deployment-constrained setting. We used the latest learned policy for data collection.

Evaluation Setup. Following Matsushima et al. (2020), our evaluation set up consisted of 5 deployments for Half-Cheetah, Ant, Cheetah-Run and 10 deployments for Walker2D and Hopper. To see the trade-off between the number of deployment versus data sample size, we perform our empirical tests on two separate cumulative data sizes of *500k* and *250k*.

For the settings of *500k / (250k)* sample size experiments, the environments Half-Cheetah, Ant and Cheetah-Run have a per deployment data collection batch size $|B| = 100k/(50k)$ (with 5

[1] http://www.mujoco.org/

[2] BREMEN has been compared against to SAC, Model-Ensembles-TRPO, BCQ, and BRAC (all adapted to deployment-constrained setting) and shows state-of-the-art performance.

deployments in total). For the environments of Walker2d and Hopper, they have a per deployment data collection batch size of $|B| = 50k/(25k)$ (with 10 deployments in total).

Deployment Setup. In the deployment constrained setting, the data collection only happens during the deployment. No training happens during this period. Only until a data batch size of $|B|$ has been collected, the agent will be taken offline for training and policy update. At the first (initial) deployment, a random policy is used for data collection. After that, the data collection will be replaced by the updated policy and will be launched to deploy again. We plot our 500k data size results in Fig.2, and we plot our 250k data size results on Fig.3. For all experimental results, we averaged over 5 random seeds.

Deployment	2nd	3rd	4th	5th
MUSBO	**0.995**	**0.959**	**0.945**	**0.941**
Baseline(BREMEN)	0.972	0.903	0.891	0.858

Table 1: Table showing the amount of novelty of each data batches collected between each deployment for cheetah_run. Novelty is measured as cosine distance between each observation (state) in $B^{(i)}$ versus the previous aggregated batches ($B^{(1)}..\bigcup B^{(i-1)}$), averaged over the number of transitions.

Empirical Details: 500k data size. The top(first) figure of Fig.2 shows the total sample size of each deployment on the y-axis, and on the x-axis, we show the number of deployments. We align along the x-axis for all figures. Our method works the best on the Walker2D and Hopper, in which our MUSBO approach achieves significant cumulative rewards especially in the longer deployments (after 6_{th}). In the Hopper environment, we see that the baselines (BREMEN and MOPO) show incapable of learning a meaningful policy while our MUSBO significantly outperforms. On the other hand, in the Ant, Cheetah-Run and Half-Cheetah environment (as shown in part b. of Fig.2), our MUSBO is capable of achieving a higher performance using a smaller amount of data. For instance, in the Cheetah-Run environment, MUSBO already achieved 550 points in the second deployment but the baselines take four to five deployments to achieve the same score.

Empirical Details: 250k data size. Different than the previous set of experiments, here, we reduce the total data size by half to 250k. In general, when we reduce the data size, the performances of all algorithms will be reduced. Despite this, our MUSBO algorithm performs quite well even in this setting. Comparing to the 500k data size experiment, we also observe similar trends. In Fig.3 a), we plot the results for Walker2d and Hopper. For the former, we see a significant winning margin in Walker2d whereas in Hopper, the winning only happens in the $8_{th} \sim 9_{th}$ deployments. In Fig.3 b), we plot the results for Ant, Cheetah-Run and Half-Cheetah. We observe that our MUSBO algorithm achieves faster learning and stronger performance.

5 Ablation Study

In this section, we examine the MUSBO algorithm in terms of the following three aspects: 1) discovery of high quality and novel data batches during each deployment; 2) whether this will lead to more accurate learning of model dynamics; 3) isolation effect of having uncertainty coefficient only or data-collection strategy only . To assess the novelty of data-batches during each deployment, we calculate the average cosine distance between the current batch ($B^{(i)}$) versus the aggregation of all of the previous batches ($B^1 \bigcup B^2 \bigcup ...B^{(i-1)}$) and then show the result in Table.1. Note that deployment 1 is not shown because the initial data collection policy is a random policy. Our result shows that our MUSBO algorithm leads to much higher novel transitions discovery.

In Fig.4 a) and b) subplots, we compare the performance of the fictitious rollouts from the learned model dynamics versus the rollouts from the real environment. We first plot the energy distance in a) and we show the mean square error (MSE) of trajectory-wise rollouts in b). As the number of deployment increases, our method is capable of discovering high quality transitions which result in a better estimated model.

Lastly, in Fig.4, we isolate each component and show the effects of including either 1) only the uncertainty coefficient or 2) only our data collection strategy as in eqn.16. In terms of cumulative rewards, the former (coeff only) has a stronger effect than latter (data collection strategy only). In terms of the learned model dynamics, latter has a stronger effect. Combining both gives the optimal effect on the cumulative rewards (subplots c and d) and leads learning an accurate model (subplots a and b).

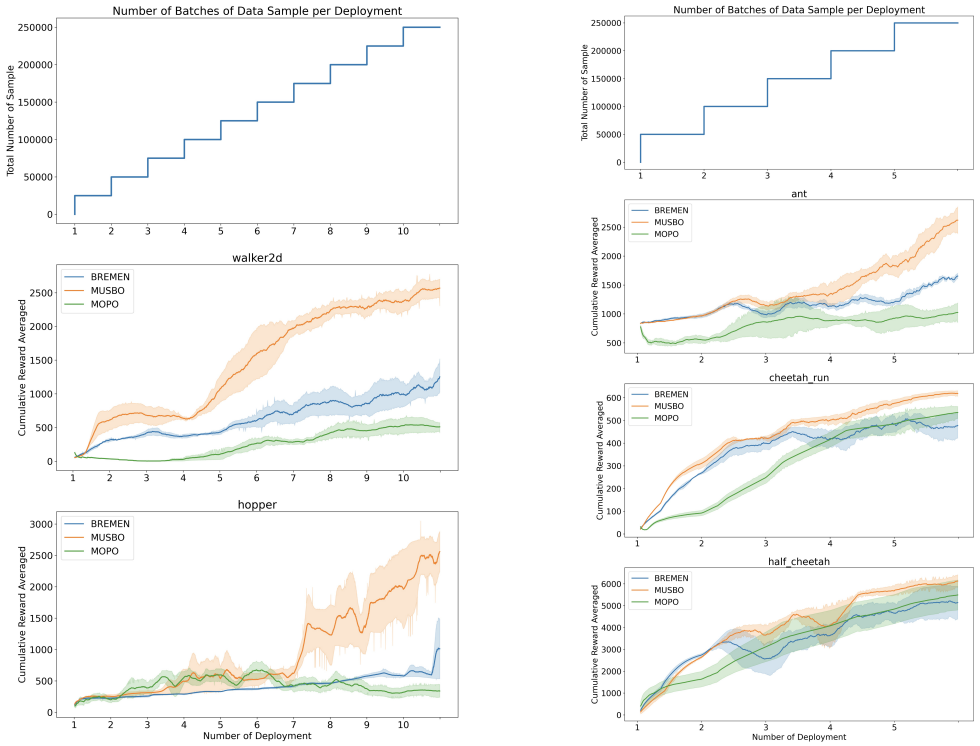

Figure 3: Empirical Evaluations of a) Walker2D and Hopper for $I = 10$ deployments, $|B| = 25k$ and b) Ant, Half_Cheetah and Cheetah_Run for $I = 5$ deployments, $|B| = 50k$. Total data consumption in both settings is $I \times |B| = 250k$. The x-axis is aligned showing the number of deployment.

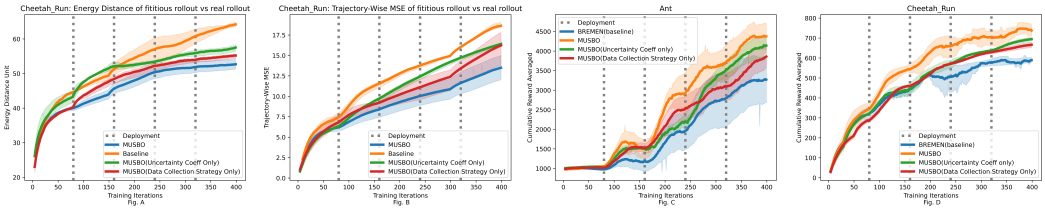

Figure 4: Ablation study on $I \times |B| = 500k$ setting: isolating the effects of having either uncertainty coeff only or data collection strategy only (as in eqn.16) on MUSBO. On subplots a) and b) we show quality of the learned model dynamics. We compare the fictitious rollouts against real rollouts in terms of energy distance (the **lower** the better) and trajectory-wise MSE (the **lower** the better) for the cheetah_run. We see that the data collection strategy gives stronger effect. We plot the cumulative rewards on fig. c) and d) for ant and cheetah_run. We see that the effect of uncertainty coeff is stronger than the effect of data collection strategy. The combined effects of two components (which becomes MUSBO) gives the best performance not only on the learned dynamics (subplots a and b), also on the cumulative rewards (subplots c and d).

6 Conclusion

In this paper, we have proposed the algorithmic framework MUSBO for optimizing the policy learning under the deployment-constrained setting. One limitation is that we have assume L-Lipschitz in the value function for our theoretical analysis which might not be the case in the practical setting. We have strong empirical result which justify it could be a viable solution. One potential negative social impact is the carbon footprint from training MBRL methods. Training policy offline consumes energy for extensive amount of time might be harmful to the environment. One way to mitigate is to train a smaller network using fewer iteration but at the cost of performance.

References

Yu Bai, Tengyang Xie, Nan Jiang, and Yu-Xiang Wang. 2020. Provably efficient q-learning with low switching cost.

Luyang Chen, Markus Pelger, and Jason Zhu. 2020. Deep learning in asset pricing.

Wen-Hsuan Chu and Kris M. Kitani. 2020. Neural batch sampling with reinforcement learning for semi-supervised anomaly detection. In *Computer Vision – ECCV 2020*, pages 751–766, Cham. Springer International Publishing.

Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. 2018. Deep reinforcement learning in a handful of trials using probabilistic dynamics models.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. 2005. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556.

Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I. Jordan, Joseph E. Gonzalez, and Sergey Levine. 2018. Model-based value estimation for efficient model-free reinforcement learning.

Scott Fujimoto, David Meger, and Doina Precup. 2019. Off-policy deep reinforcement learning without exploration.

Seyed Kamyar Seyed Ghasemipour, Dale Schuurmans, and Shixiang Shane Gu. 2021. Emaq: Expected-max q-learning operator for simple yet effective offline and online rl.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.

Tommi Jaakkola, Satinder Singh, and Michael Jordan. 1999. Reinforcement learning algorithm for partially observable markov decision problems. *Advances in Neural Information Processing Systems*, 7.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. 2019. When to trust your model: Model-based policy optimization.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. 2020. Model-based reinforcement learning for atari.

Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. 2020. Morel : Model-based offline reinforcement learning.

Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. 2019. Stabilizing off-policy q-learning via bootstrapping error reduction.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. 2018a. Model-ensemble trust-region policy optimization.

Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. 2018b. Model-ensemble trust-region policy optimization.

Sascha Lange, Thomas Gabel, and Martin Riedmiller. 2012. Batch reinforcement learning. *Reinforcement Learning: State of the Art*.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. 2020. Offline reinforcement learning: Tutorial, review, and perspectives on open problems.

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

Tatsuya Matsushima, Hiroki Furuta, Yutaka Matsuo, Ofir Nachum, and Shixiang Gu. 2020. Deployment-efficient reinforcement learning via model-based offline optimization.

Nikhil Mishra, Pieter Abbeel, and Igor Mordatch. 2017. Prediction and control with temporal segment models.

Rémi Munos and Csaba Szepesvári. 2008. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9(27):815–857.

Ofir Nachum, Yinlam Chow, Bo Dai, and Lihong Li. 2019a. Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections.

Ofir Nachum, Bo Dai, Ilya Kostrikov, Yinlam Chow, Lihong Li, and Dale Schuurmans. 2019b. Algaedice: Policy gradient from arbitrary experience.

Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. 2020. Accelerating online reinforcement learning with offline datasets.

OpenAI, :, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Jozefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Ponde de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. 2019a. Dota 2 with large scale deep reinforcement learning.

OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. 2019b. Solving rubik's cube with a robot hand.

Ladislav Peska and Peter Vojtas. 2020. Off-line vs. on-line evaluation of recommender systems in small e-commerce. In *Proceedings of the 31st ACM Conference on Hypertext and Social Media*, HT '20, page 291–300, New York, NY, USA. Association for Computing Machinery.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. 2017. Trust region policy optimization.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm.

Rishi Veerapaneni, John D. Co-Reyes, Michael Chang, Michael Janner, Chelsea Finn, Jiajun Wu, Joshua B. Tenenbaum, and Sergey Levine. 2020. Entity abstraction in visual model-based reinforcement learning.

Oriol Vinyals, I. Babuschkin, W. Czarnecki, Michaël Mathieu, Andrew Dudzik, J. Chung, D. Choi, R. Powell, Timo Ewalds, P. Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, L. Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, A. S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, D. Budden, Yury Sulsky, James Molloy, T. L. Paine, Caglar Gulcehre, Ziyu Wang, T. Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, Demis Hassabis, Chris Apps, and D. Silver. 2019. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, pages 1–5.

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

Yifan Wu, George Tucker, and Ofir Nachum. 2019. Behavior regularized offline reinforcement learning.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. 2020. Mopo: Model-based offline policy optimization.

Marvin Zhang, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew J. Johnson, and Sergey Levine. 2019. Solar: Deep structured representations for model-based reinforcement learning.

Ruiyi Zhang, Bo Dai, Lihong Li, and Dale Schuurmans. 2020. Gendice: Generalized offline estimation of stationary values.

A Proof of Lemma 1

Adapted from Schulman et al. (2017); Luo et al. (2019), we let W_j be the expected return when executing $\hat{M}^{(i+1)}$ for the first j steps, and then switch to $\hat{M}^{(i)}$ for the remaining steps.

Proof.

$$W_j = \mathop{\mathbf{E}}_{\substack{a_t\sim\pi(s_t) \\ \forall j>t\geq 0, s_{t+1}\sim\hat{M}^{(i+1)}(\cdot|s_t,a_t)\forall t\geq j, s_{t+1}\sim\hat{M}^{(i)}(\cdot|s_t,a_t)}} \{\sum_{t=0}\gamma^t r(s_t,a_t)|s_0=s\} \tag{7}$$

Thus, we have $W_0 = V^{\pi}_{\hat{M}^{(i)}}$, and $W_\infty = V^{\pi}_{\hat{M}^{(i+1)}}$. Next, we write:

$$V^{\pi}_{\hat{M}^{(i+1)}} - V^{\pi}_{\hat{M}^{(i)}} = \sum_{j=0}^{\infty}(W_{j+1} - W_j) \tag{8}$$

We expand W_j and W_{j+1} so that we can cancel the shared terms:

$$W_j = R_j + \mathop{\mathbf{E}}_{s_j,a_j\sim\pi,\hat{M}^{(i+1)}}\{\mathop{\mathbf{E}}_{s_{j+1}\sim\hat{M}^{(i)}(\cdot|s_t,a_t)}\{\gamma^{j+1}V^{\pi}_{\hat{M}^{(i)}}(s_{j+1})\}\}$$

$$W_{j+1} = R_j + \mathop{\mathbf{E}}_{s_j,a_j\sim\pi,\hat{M}^{(i+1)}}\{\mathop{\mathbf{E}}_{s_{j+1}\sim\hat{M}^{(i+1)}(\cdot|s_t,a_t)}\{\gamma^{j+1}V^{\pi}_{\hat{M}^{(i)}}(s_{j+1})\}\}$$

where R_j is the expected return of the first j time step. Next, we cancel the share terms so that:

$$W_{j+1} - W_j = \gamma^{j+1}\mathop{\mathbf{E}}_{s_j,a_j\sim\pi,\hat{M}^{(i+1)}}\Big\{\mathop{\mathbf{E}}_{s'\sim\hat{M}^{(i+1)}(\cdot|s_j,a_j)}\{V^{\pi}_{\hat{M}^{(i)}}(s')\} - \mathop{\mathbf{E}}_{s'\sim\hat{M}^{(i)}(\cdot|s_j,a_j)}\{V^{\pi}_{\hat{M}^{(i)}}(s')\}\Big\} \tag{9}$$

Thus, based on eq. (9), we have:

$$V^{\pi}_{\hat{M}^{(i+1)}} - V^{\pi}_{\hat{M}^{(i)}} = \kappa\mathop{\mathbf{E}}_{s_j,a_j\sim\pi,\hat{M}^{(i+1)}}\Big\{\mathop{\mathbf{E}}_{s'\sim\hat{M}^{(i+1)}(\cdot|s_j,a_j)}\{V^{\pi}_{\hat{M}^{(i)}}(s')\} - \mathop{\mathbf{E}}_{s'\sim\hat{M}^{(i)}(\cdot|s_j,a_j)}\{V^{\pi}_{\hat{M}^{(i)}}(s')\}\Big\} \tag{10}$$

Let $\nu(s,a) = \mathop{\mathbf{E}}_{s'\sim\hat{M}^{(i+1)}(\cdot|s_j,a_j)}\{V^{\pi}_{\hat{M}^{(i)}}(s')\} - \mathop{\mathbf{E}}_{s'\sim\hat{M}^{(i)}(\cdot|s_j,a_j)}\{V^{\pi}_{\hat{M}^{(i)}}(s')\}$, since we have assumed the Lipschitzeness of $V^{\pi}_{\hat{M}}$, we can bound $|\nu(s,a)| \leq L|\hat{M}^{(i+1)}(s,a) - \hat{M}^{(i)}(s,a)|$, then combine with triangle inequality, we have:

$$|V^{\pi}_{\hat{M}^{(i+1)}} - V^{\pi}_{\hat{M}^{(i)}}| \leq \kappa L\mathop{\mathbf{E}}_{(s,a)\sim\rho^{\pi}_{\hat{M}^{(i+1)}}}(\|\hat{M}^{(i+1)}(s,a) - \hat{M}^{(i)}(s,a)\|) \tag{11}$$

□

B Proof of Proposition 1

Proof. Starting from eqn.10, we substitute with $V^{\pi}_{M^*}$ and $V^{\pi}_{\hat{M}^{(i)}}$, and multiple both side with -1. Due to the Lipschitzeness assumption, we can then bound $|\nu(s,a)|$ (with the corresponding substitution) on eqn.10 and by the definition of $U_{\hat{M}^{(i)},M^*}(s,a)$, thus we have:

$$V^{\pi}_{M^*} \geq V^{\pi}_{\hat{M}^{(i)}}\mathop{\mathbf{E}}_{(s,a)\sim\rho^{\pi}_{M^*}}\{U_{\hat{M}^{(i)},M^*}(s,a)\}. \tag{12}$$

□

E Exploration to Collect Data from Low Support Region

During each deployment, we want to collect the data from the low support (or un-visited) regions. we make use of the uncertainty labeler to guide exploration to the un-visited regions for novel data discovery. This is achieved by injecting the $\zeta(a, s)$ as an exploration noise with the zero-mean normal distribution: $\mathcal{N}(0, \sigma = \zeta(a, s))$, where ζ is:

$$\zeta(a, s) = \max_{i \in \{\hat{P}_\phi\}_i^K} (\|s_{t+1} - \hat{s}_{t+1}\|_1) \tag{13}$$

where $\hat{s}_{t+1}$ is the predicted next state and we take the maximum prediction error of the model within $\{\hat{P}_\phi\}_i^K$ ensemble. In the Ablation Study (Section.5), we show that this exploration strategy leads to novel data discovery, and also contribute to better learned model dynamics.

Specifically, the action will be parameterized by a stochastic Gaussian policy (with parameter μ_ϑ) as:

$$a_t = \tanh(\mu_\vartheta(s_t)) + \epsilon_{\text{const}} + \epsilon_\zeta \tag{16}$$

where ϵ_{const} and ϵ_ζ are:

$$\epsilon_{\text{const}} \sim \mathcal{N}(\mu = 0, \sigma = 0.01)$$
$$\epsilon_\zeta \sim \mathcal{N}\Big(\mu = 0, \sigma = \zeta\big(a = \tanh(\mu_\vartheta(s_t)) + \epsilon_{\text{const}}, s = s_t\big)\Big)$$

Here, ϵ_{const} is an additive noise with a constant variance of 0.01, and on top of this, we also added another Gaussian noise with variance equal to $\zeta(a, s)$ from the uncertainty labeler to guide exploration.

F Detailed Implementation

We used ADAM as the optimizer with a learning rate of 1e-3 for the model dynamics $\hat{T}$ with ensembles size of $N = 5$. For the uncertainty-labeler, we use ensembles of PNN with $K = 3$ and a learning rate of 1e-3. For the behavioral cloning, we used a learning rate of 5e-4. For all the collected data, we divided them into 85% for training, and 15% for validation (for model validation). The $\hat{T}, \hat{P}$ are trained with early stopping when their performance on the validation set no longer improves after consecutive 3 episodes. We used this same setting for the behavioral cloning as well.

Model Architecture For our policy network, we parameterized it by two layers of fully-connected neural network with hidden units of 200. For the $\hat{T}$ model dynamics, we parameterized it by two layers of fully-connected neural network with hidden units of 1024. We used this same configuration with $\hat{P}$ uncertainty-labeler and implemented with PNN.

Training Time The overall training time differs for each environment. We train all models on Nvidia T4 GPU. For the $500k$ data size experiment, the entire training duration (1 run) for Walker2D and Hopper environments are 18 hours and 26 hours respectively. For the Half-Cheetah, Ant, and Cheetah-Run environment, it is a a lot more faster. It takes 7 hours, 12 hours, and 8 hours per run, respectively. For the $250k$ data size experiments, the training time is about 25 minutes faster than the $500k$ experiments.

Since we are applying Dyna-style update with neural network based dynamics models, following Wang et al. (2019) Matsushima et al. (2020), we used the following reward functions for our dynamics models (for model-based training only) as:

- Walker2d, Hopper: $\dot{x_t} - 0.001\|a_t\|_2^2 + 1$
- CheetahRun: $\max(0, \min(\dot{x_t}/10, 1))$
- Ant: $\dot{x_t} - 0.1\|a_t\|_2^2 - 3.0(z_t - 0.57)^2 + 1$
- HalfCheetah: $\dot{x_t} - 0.1\|a_t\|_2^2$

We enabled termination in rollouts for the Hopper and Walker2D environments, and disabled that of the Ant, HalfCheetah, and CheetahRun environments (with a maximum step of 1000 for each episode). For the CheetahRun task, we adopt it from the DM control suit[3].

[3]https://github.com/deepmind/dm_control

	L	Rollouts Length	TRPO's δ
Ant	2,000	250	0.05
HalfCheetah	2,000	250	0.1
Hopper	6,000	1,000	0.05
Walker2d	2,000	1,000	0.05
CheetahRun	2,000	250	0.05

Table 2: Hyper-parameters for MUSBO Algorithm

Other Hyper-parameters We searched α over the set of {0.28, 0.028, 0.0028} and we used the same $\alpha = 0.028$ (the temperature parameter for eq.(5)) for all environments. Similarly, for the action parameterization eq. (16), we used the same constant $\sigma = 0.01$ (variance term of ϵ_{const}) for all environments. The σ of ϵ_{const} is searched over the set of {0.01, 0.05, 0.1}.

We searched the rollout length on $\{250, 1000\}$, and the δ on $\{0.01, 0.05, 0.1\}$. We summarized these three parameters (L, Rollouts Length, δ) as above in table 2.

For discount factor γ and GAE λ, we used the same set of hyperparmaters as in Wang et al. (2019). Specifically, we used the same $\gamma = 0.99$ for all environment. Also, we used GAE $\lambda = 0.95$ for all environment except for Ant which has a GAE $\lambda = 0.97$.

Hyper-parameters for Baseline. For the BREMEN, we used the exact parameters as Matsushima et al. (2020). For MOPO, we adapted it to the deployment setting. We used the latest learned policy for deployment, and then launched it to collect data batch of size $|B|$. For the CheetahRun task, we used the same set of parameters of HalfCheetah. On all environments, we tried hyper-parameters search on the ranges as originally proposed by the paper Yu et al. (2020), we didn't find any improvement over the same set of parameters as originally proposed. Thus, we used the same set of parameters as originally proposed.

Training of the Probabilistic Neural Networks (PNN). In PNN (Chua et al., 2018), the network has its output neurons parameterized by a Gaussian distribution in the effort of capturing the uncertainty. This module is trained to minimize the following loss:

$$\text{loss}_{PNN}(\phi) = \sum_{n=1}^{K} \{\mu_\phi(s_n, a_n) - s_{n+1}\}^\top \Sigma_\phi^{-1}(s_n, a_n)(\mu_\phi(s_n, a_n) - s_{n+1}) + \log \det \Sigma_\phi(s_n, a_n) \tag{17}$$

where ϕ is the neural network learning parameters, μ_ϕ and Σ_ϕ are the mean and variance of the Gaussian distribution.