Meta-Model-Based Meta-Policy Optimization

Takuya Hiraoka TAKUYA-H1@NEC.COM
Central Research Laboratories, NEC Corporation, Kanagawa, Japan

Takahisa Imagawa IMAGAWA.T@AIST.GO.JP
Artificial Intelligence Research Center, National Institute of Advanced Industrial Science and Technology, Tokyo, Japan

Voot Tangkaratt VOOT.TANGKARATT@RIKEN.JP
Center for Advanced Intelligence Project, RIKEN, Tokyo, Japan

Takayuki Osa OSA@BRAIN.KYUTECH.AC.JP
Department of Human Intelligence Systems, Kyushu Institute of Technology, Fukuoka, Japan
Center for Advanced Intelligence Project, RIKEN, Tokyo, Japan

Takashi Onishi TAKASHI.ONISHI@NEC.COM
Central Research Laboratories, NEC Corporation, Kanagawa, Japan

Yoshimasa Tsuruoka TSURUOKA@LOGOS.T.U-TOKYO.AC.JP
Department of Information and Communication Engineering, The University of Tokyo, Tokyo, Japan

Editors: Vineeth N Balasubramanian and Ivor Tsang

Abstract

Model-based meta-reinforcement learning (RL) methods have recently been shown to be a promising approach to improving the sample efficiency of RL in multi-task settings. However, the theoretical understanding of those methods is yet to be established, and there is currently no theoretical guarantee of their performance in a real-world environment. In this paper, we analyze the performance guarantee of model-based meta-RL methods by extending the theorems proposed by Janner et al. (2019). On the basis of our theoretical results, we propose Meta-Model-Based Meta-Policy Optimization (M3PO), a model-based meta-RL method with a performance guarantee. We demonstrate that M3PO outperforms existing meta-RL methods in continuous-control benchmarks.

Keywords: Reinforcement Learning, Meta Learning

1. Introduction

Reinforcement learning (RL) in multi-task problems requires a large number of training samples, and this is a serious obstacle to its practical application (Rakelly et al., 2019). In the real world, we often want a policy that can solve multiple tasks. For example, in robotic arm manipulation (Yu et al., 2019), we may want the policies for controlling the arm to "grasp the wood box," "grasp the metal box," "open the door," and so on. In such multi-task problems, standard RL methods independently learn a policy for individual tasks with millions of training samples (Andrychowicz et al., 2020). This independent policy learning with a large number of samples is often too costly in practical RL applications.

Meta-RL methods have recently gained attention as promising methods to reduce the number of samples required in multi-task problems (Finn et al., 2017). In meta-RL methods, the structure shared in the tasks is learned by using samples collected across the tasks. Once learned, it is leveraged for adapting quickly to new tasks with a small number of samples. Various meta-RL methods have previously been proposed in both model-free and model-based settings.

Many **model-free meta-RL** methods have been proposed, but they usually require a large number of samples to learn a useful shared structure. In model-free meta-RL methods, the shared structure is learned as being embedded into the parameters of a context-aware policy (Duan et al., 2017; Mishra et al., 2018; Rakelly et al., 2019; Wang et al., 2016), or into the prior of policy parameters (Al-Shedivat et al., 2018; Finn and Levine, 2018; Finn et al., 2017; Gupta et al., 2018; Rothfuss et al., 2019; Stadie et al., 2018). The policy adapts to a new task by updating context information or parameters with recent trajectories. Model-free meta-RL methods have better sample efficiency than independently learning a policy for each task. However, these methods still require millions of training samples in total to learn the shared structure sufficiently usefully for the adaptation (Mendonca et al., 2019).

On the other hand, **model-based meta-RL** methods have been demonstrated to be more sample efficient than model-free counterparts. In model-based meta-RL methods, the shared structure is learned as being embedded into the parameters of a context-aware transition model (Sæmundsson et al., 2018; Perez et al., 2020), or into the prior of the model parameter (Nagabandi et al., 2019a,b). The model adapts to a new task by updating context information or its parameters with recent trajectories. The adapted models are used for action optimization in model predictive control (MPC). In the aforementioned literature, model-based meta-RL methods have been empirically shown to be more sample efficient than model-free meta-RL methods.

Despite these empirical findings, these model-based meta-RL methods lack performance guarantees. Here, the performance guarantee means the theoretical property that specifies the relationship between the planning performance in the model and the actual planning performance in a real environment. Note that the planning performance in the model is generally different from the actual performance due to the model bias. That is, even if planning performs well in the model, it does not necessarily perform well in a real environment. Such a performance guarantee for model-based meta-RL methods has not been analyzed in the literature.

In this paper, we propose a model-based meta-RL method called Meta-Model-based Meta-Policy Optimization (M3PO) which is equipped with a performance guarantee (Figure 1). Specifically, we develop M3PO by firstly formulating a meta-RL setting on the basis of partially observable Markov decision processes (POMDPs) in Section 4. Then, in Section 5, we conduct a theoretical analysis to justify the use of branched rollouts, which is a promising model-based rollout method proposed by Janner et al. (2019), in the meta-RL setting. We compare the theorems for performance guarantees of the branched rollouts and full-model based rollouts, which is a naive baseline model-based rollout method, in the meta-RL setting. Our comparison result shows that the performance of the branched rollout is more tightly guaranteed than that of the full model-based rollouts. Based on this, in Section 6 we derive M3PO, a practical method that uses the branch rollouts. Lastly, in

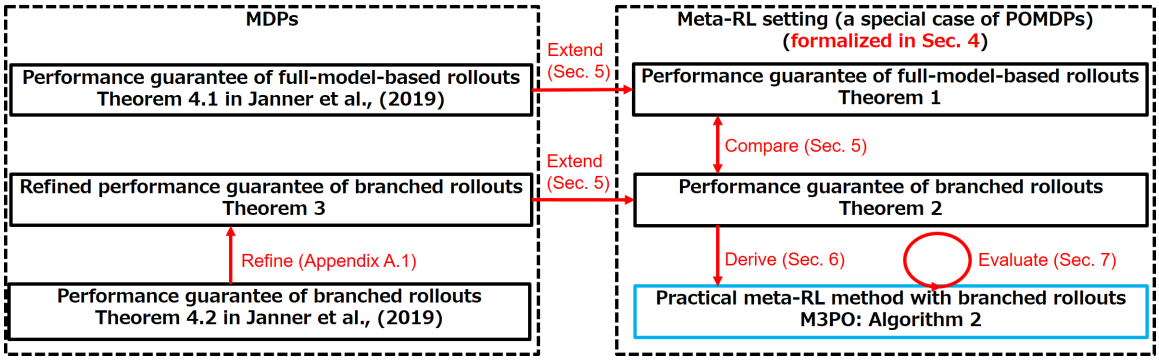


Figure 1: The executive summary of our work. Our primary contribution is proposing a model-based meta-RL method (M3PO: Algorithm 2) with a performance guarantee. In Section 4, we formulate our model-based meta-RL setting as solving (a special case of) POMDPs. In Section 5, we derive the theorems of performance guarantees by extending the theorems proposed in Janner et al. (2019) into the meta-RL setting. We then compare the derived theorems to justify using branched rollouts for model-based meta-RL. In Section 6, on the basis of our theoretical results, we derive a practical model-based meta-RL method (M3PO: Algorithm 2). In Section 7, we experimentally evaluate M3PO.

Section 7 we experimentally demonstrate the performance of M3PO in continuous-control benchmarks.

We make four primary contributions in both theoretical and practical frontiers.

Theoretical contributions: (1) We provide the performance guarantee for model-based meta-RL (Section 5). To the best of our knowledge, our work is the first attempt to provide a performance guarantee for model-based meta-RL. (2) We conduct theoretical analysis and provide the insight that *short-steps branched rollouts are better than full-model based rollouts* (Section 5 and Appendix A). We refine analyses in Janner et al. (2019) by considering multiple-model-based rollout factors (Appendix A.1), and extend our analyses results into a meta-RL setting (Section 5). Therefore, our work complements and strengthens their analyses and insight.

Practical contributions: (3) We propose a practical model-based RL method (M3PO) for meta-RL (Section 6) and demonstrate that it outperforms existing meta-RL methods (Section 7). Notably, we demonstrate that M3PO can work successfully even on complex tasks (e.g., Humanoid-direc). (4) We provide a practically interesting finding that, in M3PO (Dyna-style RL method), dynamically adjusting the mixture ratio of real/fictitious training data substantially improves long-term learning performance (Section 7.1 and Appendix A.8). In existing Dyna-style RL methods (e.g. Janner et al. (2019); Shen et al. (2020); Yu et al. (2020)), this mixture ratio is fixed during the training phase. Our finding promotes assessing the effect of dynamically adjusting the mixture ratio for the other Dyna-style RL methods to improve these methods.

2. Related work

Meta-RL is a popular approach for solving multi-tasks RL problems. Recently, researchers have formulated meta-RL as solving special cases of POMDPs (Duan et al., 2017; Humplik et al., 2019; Zintgraf et al., 2020). However, the performance guarantee of meta-RL under the POMDPs framework has not been established. In this paper, we presents bounds of the performance of meta-RL under the POMDPs framework.

We derive the above-mentioned performance bound in the context of model-based meta-RL. Existing works have focused on the performance bound of model-based RL (Feinberg et al., 2018; Henaff, 2019; Janner et al., 2019; Luo et al., 2018; Rajeswaran et al., 2020), while ignoring model-based meta-RL. Specifically, in existing works, Markov decision processes (MDPs) are assumed, and a meta-RL setting is not discussed.

3. Preliminaries

3.1. Meta-reinforcement learning

Meta-RL aims to learn the structure shared in tasks (Finn and Levine, 2018; Nagabandi et al., 2019a). Here, a task is defined as a MDP $\langle \mathcal{S}, \mathcal{A}, p_{\mathrm{st}}, r, \gamma \rangle$ with a state space $\mathcal{S}$, an action space $\mathcal{A}$, a transition probability $p_{\mathrm{st}} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$, a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and a discount factor $\gamma \in [0, 1)$. We assume that there are infinitely many tasks with the same state and action spaces but different transition probabilities and reward functions. Information about task identity cannot be observed by the agent. Hereinafter, the task identity and its set are denoted by τ and $\mathcal{T}$, respectively.

A meta-RL process is composed of meta-training and meta-testing. In meta-training, the structure shared in the tasks is learned as being embedded in either or both of a policy and a transition model. In meta-testing, on the basis of the meta-training result, they adapt to a new task. For the adaptation, the trajectory observed from the beginning of the new task to the current time step is leveraged.

3.2. Partially observable Markov decision processes

The POMDP framework extends the MDP framework by assuming that the state itself is hidden, and the agent receives an observation instead of a state. Formally, a POMDP is defined as a tuple $\langle \mathcal{O}, \mathcal{S}, \mathcal{A}, p_{\mathrm{ob}}, r, \gamma, p_{\mathrm{st}} \rangle$. Here, $\mathcal{O}$ is an observation space, and $p_{\mathrm{ob}} : \mathcal{O} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ is the observation probability. At time step t, the functions contained in POMDP are used as $p(s_t|s_{t-1}, a_{t-1})$, $p(o_t|s_t,\ a_{t-1})$, and $r_t = r(s_t,\ a_t)$. The agent selects an action on the basis of a policy $\pi(a_t|h_t)$. Here, h_t is a history (of the past trajectory) defined as $h_t = \{o_0, a_0, ..., o_t\}$. We denote the set of the histories by $\mathcal{H}$. Given the definition of the history, the history transition probability $p(h_{t+1}|a_t, h_t)$ can be evaluated by using $p(o_{t+1}|a_t, h_t)$. Here, $p(o_{t+1}|a_t, h_t) = \sum_{s_{t+1}} \sum_{s_t} p(s_t|h_t) p(s_{t+1}|s_t, a_t) p(o_{t+1}|s_{t+1}, a_t)$. The goal of RL in the POMDP is to find the optimal policy π^* that maximizes the expected return: $\pi^* = \arg\max_{\pi} \mathbb{E}_{a \sim \pi, h \sim p}[R]$, where $R = \sum_{t=0}^{\infty} \gamma^t r_t$.

3.3. Branched rollouts (Janner et al., 2019)

Branched rollouts are Dyna-style rollouts (Sutton, 1991), in which model-based rollouts are run as being branched from real trajectories. It is seemingly similar to the MPC-style rollout used in the previous model-based meta-RL methods (Sæmundsson et al., 2018; Perez et al., 2020; Nagabandi et al., 2019a,b). However, it can be used for off-policy optimization with respect to an infinite planning horizon, which is a strong advantage over the MPC-style rollout. A theorem for the performance guarantee of the branched rollout in MDPs are provided as Theorem 4.2 in Janner et al. (2019). We refer to the formal statement of the theorem in Appendix A.1.

4. Formulating model-based meta-RL

In this section, we formulate model-based meta-RL as a special case of POMDPs. In our formulation, the hidden state is assumed to be composed by the task and observation: $\mathcal{S} = \mathcal{T} \times \mathcal{O}$. With this assumption, the transition probability, observation probability, and reward function can be written as follows: $p(s_{t+1}|s_t, a_t) = p(\tau_{t+1}, o_{t+1}|\tau_t, o_t, a_t)$, $p(o_{t+1}|s_{t+1}, a_t) = p(o_{t+1}|\tau_{t+1}, o_{t+1}, a_t) = 1$ and $r(s_t, a_t) = r(\tau_t, o_t, a_t)$. As in Nagabandi et al. (2019a,b), we assume that the task can change during an episode (i.e., the value of τ_{t+1} is not necessarily equal to that of τ_t). In addition, following previous studies (Finn and Levine, 2018; Finn et al., 2017; Rakelly et al., 2019), we assume that the task set $\mathcal{T}$ and the initial task distribution $p(\tau_0)$ do not change in meta-training and meta-testing. Owing to this assumption, in our analysis and method, meta-training and meta-testing can be seen as identical.

We define a parameterized policy and model as $\pi_\phi(a_t|h_{L:t})$ and $p_\theta(r_t, o_{t+1}|a_t, h_{L:t})$, respectively. Here, ϕ and θ are learnable parameters, and $h_{L:t}$ is the truncated history $h_{L:t} = \left\{o_{\max(t-L,0)}, a_{\max(t-L,0)}, ..., o_t\right\}$, where L is a hyper-parameter. As with Nagabandi et al. (2019a), we use the history for this period for policy and model adaptation. In addition, we assume that r_t and o_{t+1} are conditionally independent given a_t and $h_{L:t}$, i.e., $p_\theta(r_t, o_{t+1}|a_t, h_{L:t}) = p_\theta(r_t|a_t, h_{L:t}) \cdot p_\theta(o_{t+1}|a_t, h_{L:t})$. For analysis in the next section, we use the model for the history $p_\theta(h_{t+1}|a_t, h_t)$ that can be evaluated by using $p_\theta(o_{t+1}|a_t, h_{L:t})$.

We use the parameterized model and policy as shown in Algorithm 1. This algorithm is composed of 1) training the model, 2) collecting trajectories from the real environment, and 3) training policy to maximize the model return $\mathbb{E}_{(a,h)\sim m(\pi_\phi, p_\theta, \mathcal{D}_{\text{env}})}[R]$. Here, $m(\pi_\phi, p_\theta, \mathcal{D}_{\text{env}})$ is the history-action visitation probability based upon an abstract model-based rollout scheme, which can be calculated on the basis of π_ϕ, p_θ, and $\mathcal{D}_{\text{env}}$. In the next section, we will analyze the algorithm equipped with two specific model-based rollout schemes: the full-model-based rollout and branched rollout.

Our model-based meta-RL formulation via POMDPs introduced in this section is summarized in Figure 2. In the following sections, we present theoretical analysis and empirical experiments on the basis of this formulation.

5. Performance guarantees

In the previous section, we formulate our meta-RL setting. In this section, we justify the use of the branched rollout in the meta-RL setting. To do so, we show the performance

Algorithm 1 Model-based meta-RL

1: Initialize policy π_ϕ, model p_θ, environment dataset $\mathcal{D}_{\text{env}}$
2: **for** N epochs **do**
3: Train p_θ with $\mathcal{D}_{\text{env}}$
4: Collect trajectories from environment in accordance with π_ϕ: $\mathcal{D}_{\text{env}} = \mathcal{D}_{\text{env}} \cup \{(h_t, a_t, o_{t+1}, r_t)\}$
5: Train π_ϕ to maximize $\mathbb{E}_{(a,h)\sim m(\pi_\phi, p_\theta, \mathcal{D}_{\text{env}})}[R]$
6: **end for**

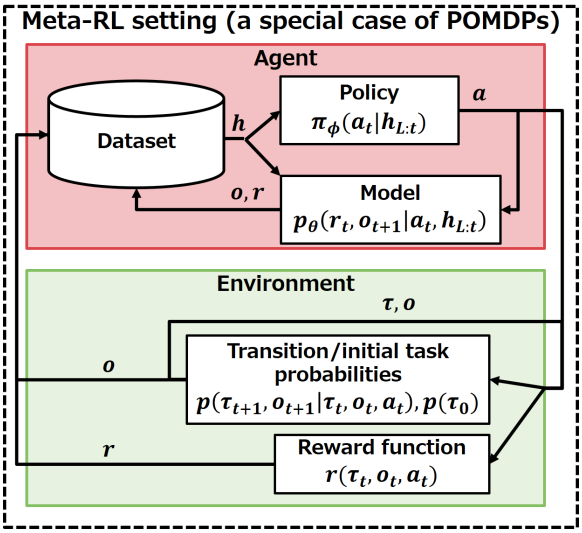


Figure 2: Model-based meta-RL formulation via POMDPs. Here, τ is a task, o is an observation, h is a (truncated) history, a is an action, and r is a reward.

guarantee of the branched rollout can be tighter than that of full-model based rollout in the meta-RL setting.

The performance guarantees is defined as "planning performance in a real environment $\geq$ planning performance in the model – discrepancy." Formally, it is represented as:

$$\mathbb{E}_{a\sim\pi_\phi, h\sim p}[R] \geq \mathbb{E}_{(a,h)\sim m(\pi_\phi, p_\theta, \mathcal{D}_{\text{env}})}[R] - C(\epsilon_m, \epsilon_\pi). \tag{1}$$

Here, $\mathbb{E}_{a\sim\pi_\phi, h\sim p}[R]$ is a true return (i.e., the planning performance in the real environment). $\mathbb{E}_{(a,h)\sim m(\pi_\phi, p_\theta, \mathcal{D}_{\text{env}})}[R]$ is a model return (i.e., the planning performance in the model). $C(\epsilon_m, \epsilon_\pi)$ is the discrepancy between the returns, which can be expressed as the function of two error quantities ϵ_m and ϵ_π.

Definition 1 (Model error) *A model error of model p_θ is defined as $\epsilon_m = \max_t \mathbb{E}_{a_t\sim\pi_\mathcal{D}, h_t\sim p}\left[D_{TV}\left(p||p_\theta\right)\right]$. Here, $D_{TV}(p||p_\theta)$ is the total variation distance between $p(h_{t+1}|a_t, h_t)$ and $p_\theta(h_{t+1}|a_t, h_t)$. $\pi_\mathcal{D}$ is the data-collection policy.*

Definition 2 (Maximal policy divergence) *The maximal policy divergence between π_D and π_ϕ is defined as $\epsilon_\pi = \max_{h_t} D_{TV}\left(\pi_\mathcal{D}||\pi_\phi\right)$. Here, $D_{TV}\left(\pi_\mathcal{D}||\pi_\phi\right)$ is the total variation distance of $\pi_\mathcal{D}(a_t|h_t)$ and $\pi_\phi(a_t|h_{L:t})$.*

Table 1: Return discrepancies in our theorems in Section 5. The discrepancies measure how much different the model return is from the true return. A low discrepancy means that the difference between them is small and the performance is tightly guaranteed. C_{Th1} and C_{Th2} are discrepancies in Theorems 1 and 2, respectively.

$C_{\mathrm{Th1}} = r_{\max}\left\{\frac{2\gamma\left(\epsilon_m + 2\epsilon_\pi\right)}{(1-\gamma)^2} + \frac{4\epsilon_\pi}{(1-\gamma)}\right\}$
$C_{\mathrm{Th2}} = r_{\max}\left\{\frac{1+\gamma^2}{(1-\gamma)^2}2\epsilon_\pi + \frac{\gamma - k\gamma^k + (k-1)\gamma^{k+1}}{(1-\gamma)^2}\left(\epsilon_\pi + \epsilon_m\right) + \frac{\gamma^k - \gamma}{\gamma - 1}(\epsilon_\pi + \epsilon_m) + \frac{\gamma^k}{1-\gamma}(k+1)(\epsilon_\pi + \epsilon_m)\right\}$

In the following, we firstly provide the performance guarantees in the meta-RL setting by deriving the discrepancies $C(\epsilon_m, \epsilon_\pi)$ of the full model-based and branched rollouts. Then, we analyze these discrepancies and show that the discrepancy of the branched rollout can yield a tighter performance guarantee than that of the full model-based rollout.

5.1. Performance guarantee of the full-model-based rollout

In the full-model-based rollout scheme, model-based rollouts are run from the initial state to the infinite horizon without any interaction with the real environment. The performance guarantee of model-based meta-RL with the full-model-based rollout scheme is as follow:

Theorem 1 *Under the full-model-based rollouts, the following inequality holds,*

$$\mathbb{E}_{a\sim\pi_\phi, h\sim p}\left[R\right] \geq \mathbb{E}_{(a,h)\sim m_f(\pi_\phi, p_\theta, \mathcal{D}_{env})}[R] - C_{\mathrm{Th1}}(\epsilon_m, \epsilon_\pi).$$

Here $\mathbb{E}_{(a,h)\sim m_f(\pi_\phi, p_\theta, \mathcal{D}_{\mathrm{env}})}\left[R\right]$ is a model return in the full model-based rollout scheme. $C_{\mathrm{Th1}}(\epsilon_m, \epsilon_\pi)$ is the discrepancy between the returns, and its expanded representation is shown in Table 1. r_{max} is the constant that bounds the expected reward: $r_{max} > \max_{a_t, h_t}\left|\sum_{\tau_t} p(\tau_t|h_t) r(\tau_t, o_t, a_t)\right|$. The proof of Theorem 1 is given in Appendix A.2, where we extend the result of Janner et al. (2019) to the meta-RL setting.

5.2. Performance guarantee of the branched rollout

Next, we provide a theorem for the performance guarantee of the branched rollout in the meta-RL setting, where k-step model-based rollouts are run as being branched from real trajectories (see Figure 3).

To provide our theorem, we extend the notion and theorem of the branched rollout proposed in Janner et al. (2019) into the meta-RL setting. Specifically, we extend the branched rollout defined originally in the state-action space (Janner et al., 2019) to a branched rollout defined in the history-action space (Figure 3). Then, we translate our meta-RL setting into

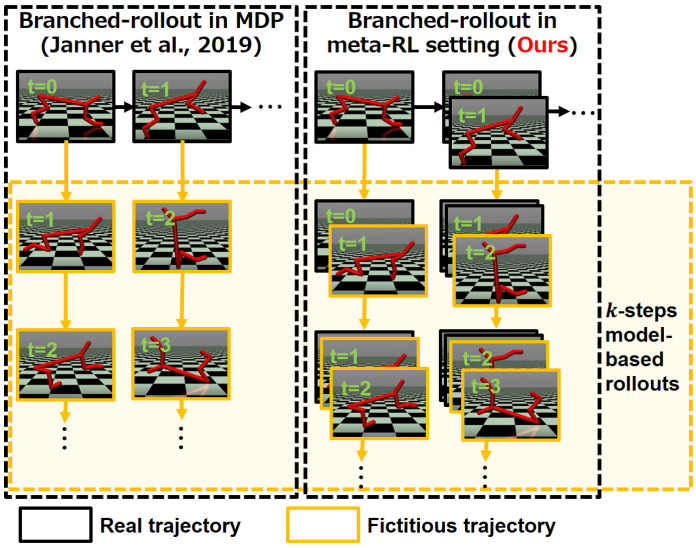


Figure 3: The branched rollouts in a MDP and the meta-RL setting. The branched rollout (Janner et al., 2019) is a kind of Dyna-style rollouts (Sutton, 1991), in which k-step model-based rollouts are run as being branched from real trajectories. The fictitious trajectories generated by the model-based rollouts are used for policy optimization. In Janner et al. (2019), branched rollouts are defined on a MDP (i.e., the state-action space). We extend it to a meta-RL setting (the history-action space).

equivalent MDPs with histories as states, and then utilize the theorem to obtain a result in the meta-RL setting (more details are given in Appendix A.2).

Our resulting theorem bounds the performance of model-based meta-RL under k-steps branched rollouts as follows.

Theorem 2 *Under the $k \in \mathbb{N}_{>0}$ steps branched rollouts, the following inequality holds,*

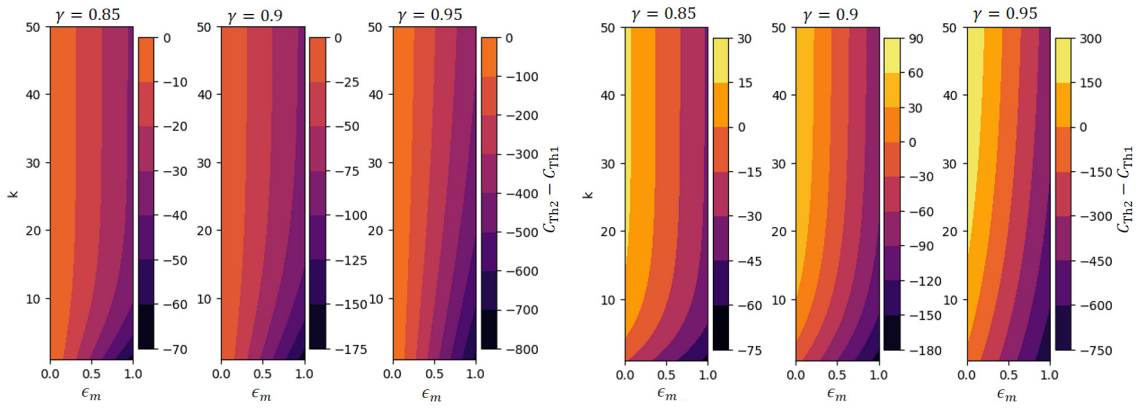
$$\mathbb{E}_{a\sim\pi_\phi,h\sim p}\left[R\right] \geq \mathbb{E}_{(a,h)\sim m_b(\pi_\phi,p_\theta,\mathcal{D}_{env})}\left[R\right] - C_{\mathrm{Th2}}(\epsilon_m,\epsilon_\pi).$$

Here $\mathbb{E}_{(a,h)\sim m_b(\pi_\phi,p_\theta,\mathcal{D}_{\mathrm{env}})}\left[R\right]$ is a model return in the branched rollout scheme. $C_{\mathrm{Th2}}(\epsilon_m,\epsilon_\pi)$ is the discrepancy between the returns, and its expanded representation is shown in Table 1. The value of $C_{\mathrm{Th2}}(\epsilon_m,\epsilon_\pi)$ tends to monotonically increase as the value of k increases, regardless of the values of ϵ_m and ϵ_π. This implies that the optimal value of k is 1.

5.3. Comparison of the performance guarantees

We conduct theoretical and empirical comparisons, focusing on the magnitude correlation of discrepancies in Theorem 1 and that of Theorem 2 (i.e., C_{Th1} and C_{Th2}). In theoretical analysis, we compare the magnitude correlation between the terms relying on ϵ_m in C_{Th1} and ones in C_{Th2}. In empirical analysis, we compare the empirical value of C_{Th1} and C_{Th2} by varying the values of various factors.

A theoretical comparison result is provided as follows:



(a) $\epsilon_\pi = 0$ (b) $\epsilon_\pi = 1 - \epsilon_m$

Figure 4: Comparison results of discrepancies in our theorems in Section 5. We compare the discrepancy (C_{Th1}) in Theorem 1 with that (C_{Th2}) in Theorem 2 by subtracting the former from the later. The comparison results in cases of $\gamma = 0.85$, $\gamma = 0.9$, and $\gamma = 0.95$ are shown in the figures. In each figure, the vertical axis represents the model-based rollout length k (from 1 to 50), and the horizontal axis represents the model errors ϵ_m (from 0.0 to 1.0). In Figure (a), to focus on the model error ϵ_m effect, we set ϵ_π value as $\epsilon_\pi = 0$. In Figure (b), to illustrate the effect of both ϵ_m and ϵ_π, we set ϵ_π value as $\epsilon_\pi = 1 - \epsilon_m$. In both figures, we set $r_{\max}$ value as $r_{\max} = 1$. A key insight with the figures is that, by setting k to 1, the branched rollouts reduce the discrepancy more significantly than the full model-based rollouts (i.e., $C_{\text{Th2}} - C_{\text{Th1}}$ is always smaller than zero).

Corollary 1 *The terms relying on ϵ_m at $k = 1$ in C_{Th2} are equal to or smaller than those relying on ϵ_m in C_{Th1}.*

The proof is given in Appendix A.3. This result implies that the model error is less harmful in the branched rollout with $k = 1$. In addition, in empirical comparison results shown in Figure 4, we can see that C_{Th2} with $k = 1$ is always smaller than C_{Th1}. These results motivate us to use *branched rollouts with $k = 1$ for the meta-RL setting.*

6. A practical model-based meta-RL method with the branched rollout

In the previous section, we demonstrated the usefulness of the branched rollout. In this section, we propose Meta-Model-Based Meta-Policy Optimization[1] (M3PO), which is a model-based meta-RL method with the branched rollout.

M3PO is described in Algorithm 2. In line 3, the model is learned via maximum likelihood estimation. In line 5, trajectories are collected from the environment with policy π_ϕ and stored into the environment dataset $\mathcal{D}_{\text{env}}$. In lines 6–9, k-step branched rollouts are run to generate fictitious trajectories, and the generated ones are stored into the model dataset $\mathcal{D}_{\text{model}}$. In lines 10–12, the policy π_ϕ is learned to improve the model return.

1. The "meta-model" and "meta-policy" come from the use of $p_\theta(r_t, o_{t+1}|a_t, h_{L:t})$ and $\pi_\phi(a_t|h_{L:t})$ in this method. Following previous work (Clavera et al., 2018), we refer to this type of history-conditioned policy as a **meta-policy**. Similarly, we refer to this type of history-conditioned model as a **meta-model**.

Algorithm 2 Meta-Model-Based Meta-Policy Optimization (M3PO)

Initialize policy π_ϕ, model p_θ, environment dataset $\mathcal{D}_{\text{env}}$, model dataset $\mathcal{D}_{\text{model}}$
for N epochs **do**
  Train p_θ with $\mathcal{D}_{\text{env}}$: $\theta \leftarrow \arg\max_\theta \mathbb{E}_{r_t, o_{t+1}, a_t, h_{L:t} \sim \mathcal{D}_{\text{env}}} [p_\theta(r_t, o_{t+1} | a_t, h_{L:t})]$
  for E steps **do**
    Collect trajectories from environment in accordance with π_ϕ: $\mathcal{D}_{\text{env}} = \mathcal{D}_{\text{env}} \cup \{(h_{L:t}, a_t, o_{t+1}, r_t)\}$
    for M model-based rollouts **do**
      Sample $h_{L:t}$ uniformly from $\mathcal{D}_{\text{env}}$
      Perform k-step model-based rollouts starting from $h_{L:t}$ using π_ϕ, and then add fictitious trajectories to $\mathcal{D}_{\text{model}}$
    end for
    for G gradient updates **do**
      Update policy parameters with $\mathcal{D}_{\text{model}}$: $\phi \leftarrow \phi - \nabla_\phi J_{\mathcal{D}_{\text{model}}}(\phi)$
    end for
  end for
end for

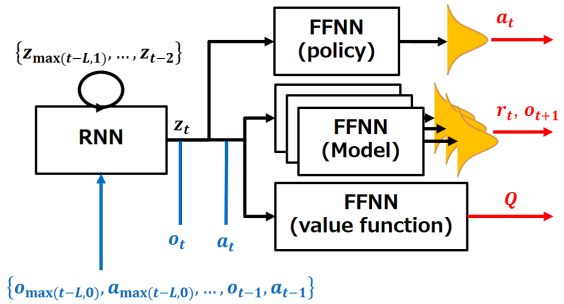


Figure 5: A summary of our implementation of the model, policy, and value network for M3PO. We use a recurrent neural network (RNN) to encode the history. RNN's hidden-unit output z is fed into the model, policy and value network. We use a feed-forward neural network (FFNN) for the model, policy and value network. Especially, we use an ensemble of the networks with B network heads for the model.

The model is implemented as a bootstrap ensemble of B diagonal Gaussian distributions: $p_\theta(r_t, o_{t+1} | a_t, h_{L:t}) = \frac{1}{B} \sum_{i=1}^{B} \mathcal{N}\left(r_{t+1}, o_{t+1} | \mu_\theta^i(a_t, h_{L:t}), \sigma_\theta^i(a_t, h_{L:t})\right)$. Here, μ_θ^i and σ_θ^i are the mean and standard deviation, respectively. The model ensemble technique is used to consider both epistemic and aleatoric uncertainty of the model into prediction (Chua et al., 2018). μ_θ^i and σ_θ^i are implemented with a recurrent neural network (RNN) and a feed-forward neural network (FFNN). At each prediction, $\{o_{\max(t-L,0)}, a_{\max(t-L,0)}, \ldots, o_{t-1}, a_{t-1}\}$ in $h_{L:t}$ is fed to the RNN. Then its hidden-unit outputs z_t are fed to the FFNN together with o_t and a_t. The FFNN outputs the mean and standard deviation of the Gaussian distribution, and the next observation and reward are sampled from its ensemble.

The policy is implemented as $\pi_\phi(a_t | h_{L:t}) = \mathcal{N}(a_t | \mu_\phi(h_{L:t}), \sigma_\phi(h_{L:t}))$. Here, μ_ϕ and σ_ϕ are the mean and standard deviation, respectively. The network architecture for them is the

same as that for the model, except that the former does not contain a_t as input. To train ϕ, we use a soft-value policy improvement objective (Haarnoja et al., 2018): $J_{\mathcal{D}_{\text{model}}}(\phi) = \mathbb{E}_{h_{L:t} \sim \mathcal{D}_{\text{model}}} [D_{KL} (\pi_\phi || \exp (Q - V))]$. Here, D_{KL} is the Kullback-Leibler divergence, and Q and V are soft-value functions: $Q(a_t, h_{L:t}) = \mathbb{E}_{(r_t, h_{L:t+1}) \sim \mathcal{D}_{\text{model}}} [r_t + \gamma V(h_{L:t+1}) | a_t, h_{L:t}]$ and $V(h_{L:t}) = \mathbb{E}_{a \sim \pi_\phi} [Q(a, h_{L:t}) - \log \pi_\phi(a|h_{L:t}) | h_{L:t}]$. The network architecture for Q is identical to that for the model. We do not prepare the network for V, and it is directly calculated by using Q and π_ϕ.

Figure 5 gives a summary of our implementation of the model, policy and value network for M3PO.

7. Experiments

In this section, we report our experiments [2].

7.1. Comparison against meta-RL baselines

In our first experiments, we compare our method (M3PO) with two baseline methods: **probabilistic embeddings for actor-critic reinforcemnet learning (PEARL)** (Rakelly et al., 2019) and **learning to adapt (L2A)** (Nagabandi et al., 2019a). More detailed information on the baselines is described in Appendix A.4. As with Nagabandi et al. (2019a), our primary interest lies in improving the performance of meta-RL methods in short-term training. Thus, we compare the aforementioned methods on the basis of their performance in short-term training (within 200k training samples). Nevertheless, for complementary analysis, we compare the performances of PEARL and M3PO in long-term training (with 2.5m to 5m training samples, or until earlier learning convergence). Their long-term performances are denoted by **M3PO-long** and **PEARL-long**.

We compare the methods in simulated robot environments based on the MuJoCo physics engine (Todorov et al., 2012). For this, we consider environments proposed in the literature of meta-RL (Finn and Levine, 2018; Nagabandi et al., 2019a; Rakelly et al., 2019; Rothfuss et al., 2019): **Halfcheetah-fwd-bwd**, **Halfcheetah-pier**, **Ant-fwd-bwd**, **Ant-crippled-leg**, **Walker2D-randomparams** and **Humanoid-direc** (Figure 6). In these environments, the agent is required to adapt to a fluidly changing task that the agent cannot directly observe. Detailed information about each environment is described in Appendix A.5. In addition, the hyperparameter settings of M3PO for each environment are shown in Appendix A.7.

Our experimental results demonstrate that M3PO outperforms existing meta-RL methods. Figure 7 shows learning curves of M3PO and existing meta-RL methods (L2A and PEARL). These learning curves indicate that M3PO has better sample efficiency than L2A and PEARL. The performance (average return) of L2A remains poor and does not improve even when the number of training samples increases. On the other hand, the performance of PEARL improves as the number of training samples increases, but the degree of improvement is smaller. Note that, in an early stage of the training phase, unseen tasks appear in

2. Source code to replicate the experiments is available at https://github.com/TakuyaHiraoka/Meta-Model-Based-Meta-Policy-Optimization

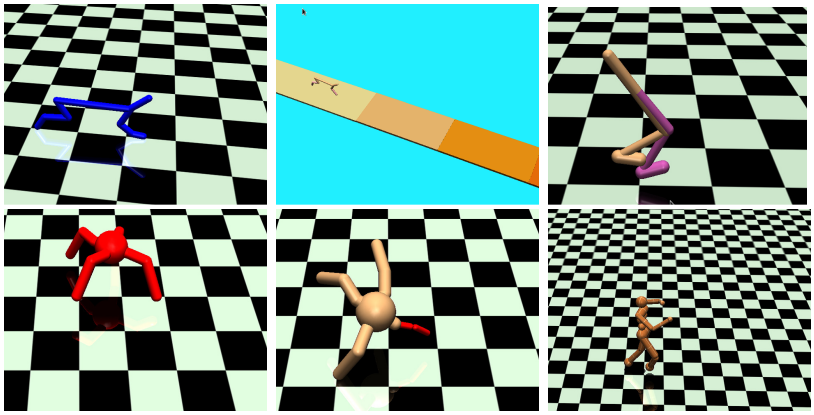


Figure 6: Environments for our experiment. Halfcheetah-fwd-bwd (upper left), Halfcheetah-pier (upper center), Walker2D-randomparams (upper right), Ant-fwd-bwd (lower left), Ant-crippled-leg (lower center) and Humanoid-direc (lower right).

many test episodes. Therefore, the improvement of M3PO over L2A and PEARL at the early stage of training indicates M3PO's high adaptation capability for unseen tasks.

Interestingly, in Halfcheetah-pier, Walker2D-randomparams, and Humanoid-direc, M3PO (M3PO-long) has worse long-term performance than PEARL (PEARL-long). That is, the improvement of M3PO over PEARL, which is a model-free approach and does not depend on the model, becomes smaller as the learning epoch elapses. This is perhaps due to the model bias. This result indicates that gradually making M3PO less dependent on the model needs to be considered to improve overall performance. Motivated by this observation, we equip M3PO with a gradual transition mechanism. Specifically, we replace $\mathcal{D}_{\text{model}}$ in line 11 in Algorithm 2 with the mixture of $\mathcal{D}_{\text{model}}$ and $\mathcal{D}_{\text{env}}$. During training, the mixture ratio of $\mathcal{D}_{\text{model}}$ is gradually reduced. As this ratio reduces, the M3PO becomes less dependent on the model. More details of our modification for M3PO are described in Appendix A.8. The long-term performance of the modified M3PO is shown as **M3PO-h-long** in Figure 7. We can see that it is the same as or even better than that of PEARL-long.

Figure 8 shows an example of policies learned by M3PO with 200k training samples in Humanoid-direc and indicates that the learned policy successfully adapts to tasks. Additional examples of the policies learned by PEARL and M3PO are shown in the video at the following link: https://drive.google.com/file/d/1DRA-pmIWnHGNv5G_gFrml8YzKCtMcGnu/view?usp=sharing

7.2. Ablation study

In the next experiment, we perform ablation study to evaluate M3PO with different rollout length k and M3PO with full model-based rollout. The evaluation results (Figure 9) show that 1) the performance of M3PO tends to degrade when its model-based rollout length is long, and 2) the performance of M3PO with $k = 1$ is mostly the best in all environments. These results are consistent with our theoretical result in Section 5.2.

Next, we evaluate a variant of M3PO where we use the full-model-based rollout instead of the branched rollout. The learning curves of this instance are plotted as **Full** in Figure 9.

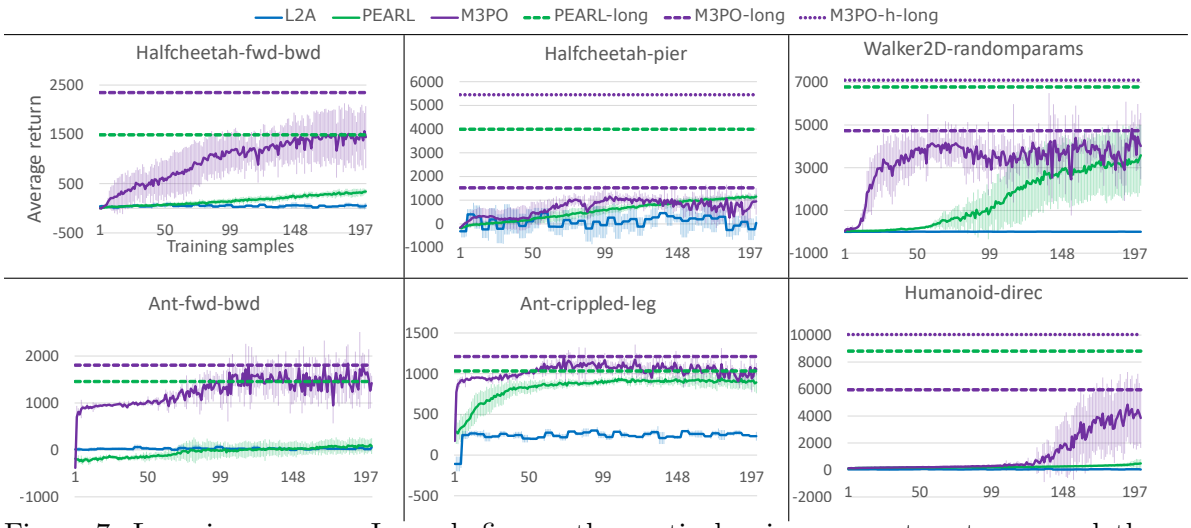


Figure 7: Learning curves. In each figure, the vertical axis represents returns, and the horizontal axis represents numbers of training samples (one on the scale is equal to 1000 samples). The policy and model are fixed and evaluated in terms of their average return on 50 episodes at every 5000 training samples for L2A and 1000 training samples for the other methods. Each method is evaluated in six trials, and average returns on the 50 episodes are further averaged over the trials. The averaged returns and their standard deviations are plotted.

Figure 8: Example of a policy learned by M3PO with 200k training samples, which is equal to around 11 hours of real-world experience, in Humanoid-direc. The humanoid is highlighted in accordance with a task (red: move to left, and blue: move to right). The figures are time-lapse snapshots, and the first figure on the left is the snapshot at the beginning time. The green arrows show the humanoid's movement direction. Figures show the policy successfully adapting to the task change.

We can see that the branched rollouts (M3PO) with $k = 1$ is much better than the full-model-based rollout (Full) in all environments.

8. Conclusion

In this paper, we analyzed the performance guarantee of the model-based meta-reinforcement learning (RL) method. We first formulated model-based meta-RL as solving a special case of partially observable Markov decision processes. We then theoretically analyzed the performance guarantee of the branched rollout in the meta-RL setting. We showed that the branched rollout has a more tightly guaranteed performance than the full model-based rollouts. Motivated by the theoretical result, we proposed Meta-Model-Based Meta-Policy

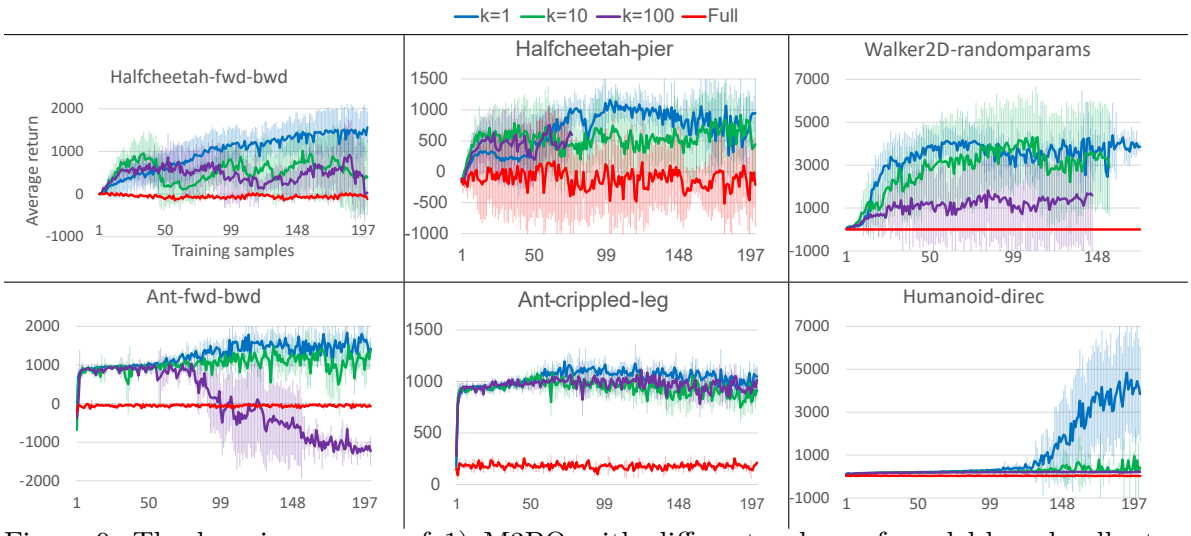


Figure 9: The learning curves of 1) M3PO with different values of model-based rollout length k, and 2) a variant of M3PO where we use the full-model-based rollout instead of the branched rollout.

Optimization (M3PO), a practical model-based meta-RL method based on branched rollouts. Our experimental results show that M3PO outperforms PEARL and L2A.

References

Maruan Al-Shedivat, Trapit Bansal, Yuri Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. Continuous adaptation via meta-learning in nonstationary and competitive environments. In *Proc. ICLR*, 2018.

Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020. doi: 10.1177/0278364919887447.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proc. EMNLP*, 2014.

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

Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization. In *Proc. CoRL*, pages 617–629, 2018.

Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. RL^2: Fast reinforcement learning via slow reinforcement learning. In *Proc. ICLR*, 2017.

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

Chelsea Finn and Sergey Levine. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. In *Proc. ICLR*, 2018.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proc. ICML*, pages 1126–1135, 2017.

Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine. Meta-reinforcement learning of structured exploration strategies. In *Proc. NeurIPS*, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *Proc. ICML*, pages 1856–1865, 2018.

Mikael Henaff. Explicit explore-exploit algorithms in continuous state spaces. In *Proc. NeurIPS*, 2019.

Jan Humplik, Alexandre Galashov, Leonard Hasenclever, Pedro A Ortega, Yee Whye Teh, and Nicolas Heess. Meta reinforcement learning as task inference. *arXiv preprint arXiv:1905.06424*, 2019.

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

Yuping Luo, Huazhe Xu, Yuanzhi Li, Yuandong Tian, Trevor Darrell, and Tengyu Ma. Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. In *Proc. ICLR*, 2018.

Russell Mendonca, Abhishek Gupta, Rosen Kralev, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Guided meta-policy search. In *Proc. NeurIPS*, pages 9653–9664, 2019.

Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *Proc. ICLR*, 2018.

Anusha Nagabandi, Ignasi Clavera, Simin Liu, Ronald S Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Learning to adapt in dynamic, real-world environments via meta-reinforcement learning. In *Proc. ICLR*, 2019a.

Anusha Nagabandi, Chelsea Finn, and Sergey Levine. Deep online learning via meta-learning: Continual adaptation for model-based RL. In *Proc. ICLR*, 2019b.

Christian F Perez, Felipe Petroski Such, and Theofanis Karaletsos. Generalized hidden parameter MDPs transferable model-based RL in a handful of trials. In *Proc. AAAI*, 2020.

Aravind Rajeswaran, Igor Mordatch, and Vikash Kumar. A game theoretic framework for model based reinforcement learning, 2020.

Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *Proc. ICML*, pages 5331–5340, 2019.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Jonas Rothfuss, Dennis Lee, Ignasi Clavera, Tamim Asfour, and Pieter Abbeel. ProMP: Proximal meta-policy search. In *Proc. ICLR*, 2019.

Steindór Sæmundsson, Katja Hofmann, and Marc Peter Deisenroth. Meta reinforcement learning with latent variable Gaussian processes. *arXiv preprint arXiv:1803.07551*, 2018.

Jian Shen, Han Zhao, Weinan Zhang, and Yong Yu. Model-based policy optimization with unsupervised model adaptation. In *Proc. NeurIPS*, 2020.

David Silver and Joel Veness. Monte-Carlo planning in large POMDPs. In *Proc. NIPS*, pages 2164–2172, 2010.

Bradly C Stadie, Ge Yang, Rein Houthooft, Xi Chen, Yan Duan, Yuhuai Wu, Pieter Abbeel, and Ilya Sutskever. Some considerations on learning to explore via meta-reinforcement learning. *arXiv preprint arXiv:1803.01118*, 2018.

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

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *Proc. IROS*, pages 5026–5033. IEEE, 2012.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

Grady Williams, Andrew Aldrich, and Evangelos Theodorou. Model predictive path integral control using covariance variable importance sampling. *arXiv preprint arXiv:1509.01149*, 2015.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Proc. CoRL*, 2019.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. MOPO: model-based offline policy optimization. In *Proc. NeurIPS*, 2020.

Luisa Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. VariBAD: A very good method for Bayes-adaptive deep RL via meta-learning. In *Proc. ICLR*, 2020.