Overcoming Model-Bias in Reinforcement Learning

Ignasi Clavera Gilaberte

December 19, 2020

OVERCOMING MODEL-BIAS IN REINFORCEMENT LEARNING

IGNASI CLAVERA GILABERTE

Fall 2020

A dissertation submitted in partial satisfaction of the requirements for the degree of Doctor of Philosophy in Computer Science in the Graduate Division of the University of California, Berkeley

Overcoming Model-Bias in Reinforcement Learning

by

Ignasi Clavera Gilaberte

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Fall 2020

Overcoming Model-Bias in Reinforcement Learning

Abstract

Overcoming Model-Bias in Reinforcement Learning

by

Ignasi Clavera Gilaberte

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Pieter Abbeel, Chair

Autonomous skill acquisition has the potential to dramatically expand the tasks robots can perform in settings ranging from manufacturing to household robotics. Reinforcement learning offers a general framework that enables skill acquisition solely from environment interaction with little human supervision. As a result, reinforcement learning presents itself as a scalable approach for widespread adoption of robotic agents. While reinforcement learning has achieved tremendous success, it has been limited to simulated domains; such as video games, computer graphics, and board games. Its most promising methods typically require large amount of interaction with the environment to learn optimal policies. In real robotic systems, significant interaction can cause wear and tear, create unsafe scenarios during the learning process, or become prohibitively time consuming to enable potential applications.

One promising venue to minimize the interaction between the agent and environment are the methods under the umbrella of model-based reinforcement learning. Model-based methods are characterized by learning a predictive model of the environment that is used for learning a policy or planning. By exploiting the structure of the reinforcement learning problem and making a better use of the collected data, model-based methods can achieve better sample complexity. Previous to this work, model-based methods were limited to simple environments and tended to achieve lower performance than model-free methods. In here, we illustrate the model-bias problem: the set of difficulties that prevent typical model-based methods to achieve optimal policies; and propose solutions that tackle model-bias. The methods proposed are able to achieve the same asymptotic performance as model-free methods while being two orders of magnitude more sample efficient. We unify these methods into an asynchronous model-based framework that allow fast and efficient learning. We successfully learn manipulation policies, such as block stacking and shape matching, on the real PR2 robot within 10 min of wall-clock time. Finally, we take a further step towards real-world robotics and propose a method that can efficiently adapt to changes in the environment. We showcase it on a real 6-legged robot navigating on different terrains, like grass and rock.

ACKNOWLEDGMENTS

Four years ago amidst the nervousness of leaving home for the first time and the excitement that comes with it, I crossed the globe to come to Berkeley. At that time, as an undergrad about to do my bachelor's thesis. Little did I know the profound effect of that voyage. These last three years of Ph.D I have grown to an extent that I could not have thought possible, both professionally and personally. For that, I am extremely grateful to my advisor Pieter Abbeel for having had faith in an undergrad with almost no experience in machine learning. His work ethic, generosity, and constant support has shaped the AI researcher I have become. I would like to thank my collaborators, without whom this thesis would not have been possible – Yiming Ding, Chelsea Finn, Violet Fu, Thanard Kurutach, Dennis Lee, Sergey Levine, Simin Liu, Anusha Nagabandi, Jonas Rothfuss, Boren Tsai, Yunzhi Zhang.

However, this journey began many years ago. Back in high school, thanks to emphasis on knowledge, instead of grades, and the beauty of mathematics of my teacher Alex Torio I turned from an average to a top student. Of course, the constant competition with Alberto Cejudo pushed me to further excel. During those times, the learning process would not have been nearly as fun without Raimon Gaspar, Pol Hernandez, and Ander Estruch. In college, I was fortunate enough to be in the CFIS, where I found a brilliant and fun group of friends, like Xavier Cabanes and Alex Nowak. Especially, I would like to thank Arnau Sanroma and Carla Domenech with whom I spent long days going back and forth between the Math and Engineering faculty. They became my family during those college years. I am grateful to Joan Sola and Juan Andrade, my first research advisors, for their patience and guidance. This journey would not have been possible without Carme Torras, who advised me in research opportunities and put me in contact with my current advisor Pieter Abbeel. I would like to thank everyone in the BAIR community, with whom I have had endless research discussions. During these years of Ph.D, I could not have had a better roommate: Abhishek Gupta, who, in some sense, has been my secondary advisor providing guidance and support. In Berkeley, I found a family away from home: Carlos, Maria, Kate, Agnes, Kate, Cashmere, Mostafa, Kim, Dan, Edna, Adam, Vicenç. The constant adventures and dinners with them have made these years extremely exciting and fun!

I want to thank my parents Albert and Lydia, and my brother Albert, without whom I would not be the person that I am today. Finally, I would like to thank Anna for sticking with these last 11 years, not an easy endeavour. For her love, support, and understanding – specially, during our four years apart.

CONTENTS

1 INTRODUCTION

Autonomous skill acquisition has the potential to dramatically expand the tasks robots can perform in settings ranging from manufacturing to household robotics. The current use of robotics is limited to perfectly controlled environments, such as warehouses or factories, where one has a mapping of the environment and the robot encounters staged situations with limited variability. However, for a widespread adoption of robotic agents, they need to be able to handle the variability of the world. For instance, in a household, they should be able to operate whether the kitchen is organized or dirty, be able to carry out the task while people move around them, and function regardless the house they are in. How can we accomplish this degree of generality?

Robotic applications have been driven by three main methods in the past: trajectory optimization (Khatib, 1986; Schulman et al., 2014; Warren, 1989), hand-crafted controllers (Maitin-Shepard et al., 2010; Moudgal et al., 1994; Hopcroft et al., 2007; Fikes et al., 1972), and imitation learning (T. Zhang et al., 2018; Ross et al., 2010; Schaal, 1999; Calinon, 2009; Coates et al., 2008; Abbeel et al., 2010). However, I would argue that none of these approaches are suitable to get algorithms that enable such degree of generality. First, trajectory optimization, while it can accomplish complex and precise motions it needs a model of the environment. The need of the model limits its scalability. First, it is not feasible to know in advance every single element of the world. And second, there are elements in the world that are hard to model, such as clothes or liquids. On the other hand, hand-crafted controllers are not bottlenecked by needing a description of the world. However, defining the optimal controller in some scenarios might be extremely hard, or we might not even know which is the optimal. For instance, in the game of chess while we have good heuristics of how to play it, we do not know the optimal action. Finally, imitation learning provides a framework to easily teach robots and it also does not need to model the environment. In imitation learning, the human needs to provide supervision of which is the right action to take in each task. The strong human supervision needed for this framework hinders the scalability of it. For instance, T. Zhang et al. (2018) needed 200 demonstration to learn a simple reaching task with 90% success rate.

Reinforcement learning (RL) is a general framework to learn the optimal behavior of an agent through interaction with the environment. Reinforcement learning methods overcome the previously described limitations and present themselves as a scalable approach for autonomous skill acquisition. In the RL paradigm, there is no need to know a dynamics model of the world, minimal supervision is provided through the reward function, and an optimal policy with respect to such reward is obtained. In recent years, reinforcement learning methods have accomplished tremendous success in the areas of video games, computer graphics, and board games (Mnih et al., 2015; Schulman et al., 2015b; Silver et al., 2016; Silver et al., 2017). This success, however, has been limited to areas where gathering data can be done at scale; namely, in simulated environments. In order to achieve high performance, these methods typically require a large amount of data. For instance, the number of Go games played in Silver et al. (2017) amount to 8 years, if played sequentially. In real robotic systems, significant interaction can cause wear and tear, create unsafe scenarios during the learning process, or become prohibitively time consuming to enable applications.

One promising venue to minimize the interaction between the agent and environment are the methods under the umbrella of model-based reinforcement learning. Model-based methods, contrary to model-free ones, are characterized by learning a predictive model of the environment that is used for learning a policy or planning. By exploiting the structure of the reinforcement learning problem and making a better use of the collected data, model-based methods can achieve better sample complexity. Previous to this work, model-based methods were limited to simple environments and tended to achieve lower performance as model-free methods. In this thesis, we illustrate the model-bias problem (Deisenroth and Rasmussen, 2011; Schneider, 1997; Atkeson and Santamaria, 1997): the set of challenges that prevent typical model-based methods to achieve optimal policies.

In this work, we propose a set of methods that tackle the model-bias problem. We present sample efficient algorithms that achieve high asymptotic performance. By substantially decreasing the interaction with the environment, these methods decrease the risk of wear and tear, and the exposure to hazardous situations. We further enable the applicability of model-based methods by developing an algorithm that is able to learn in long horizons tasks efficiently by making use of a value function and the model derivatives. In order to decrease the training time, we propose an asynchronous framework for model-based learning that reduces by a factor of 10 the wall clock time of these methods. The result of our work is a set of fast and data-efficient policy learning algorithms. Finally, we move one step closer to general real-world robotics, by introducing an algorithm that allows the robotic agent to adapt to unseen environments. We showcase it on a real 6-legged robot navigating on different terrains, like grass and rock.

The main contributions of this thesis are the following:

- In Chapter 2 we introduce the reinforcement learning (RL) framework. In this chapter, we also introduce model-based RL and give an overview of previous model-based methods. Finally, we summarize meta-reinforcement learning, which will be helpful background for our proposed methods.
- Chapter 3, explains the model-bias problem. We decompose the problem into three specific issues: estimation errors, compounding errors, and over-optimism.
- In Chapter 4 we propose an algorithm that tackles the over-optimism issue. Our method model-based trust-region policy optimization tackles overoptimism by learning an ensemble of dynamics models and training a robust policy on top of them.
- In Chapter 5, model-based RL via meta-policy optimization is proposed. The method, instead of tackling the approximation and compounding errors, harness them by learning an adaptive policy to different dynamics. This approach uses meta-learning to learn a policy that is able to quickly adapt to the real environment.
- In Chapter 6, we presented a theoretically justified meta-learning algorithm that allows for better adaptation. Our method is able to optimize for the exploration strategy that would result in the best adaptation.
- In Chapter 7, we make use of model-free methods; namely, actor-critic, to extend the horizon at which model-based methods can perform. By using the models to predict short horizons and the value function for longer horizon we are able to learn in tasks with a longer horizons than the methods in Chapters 4 and 5. The work presented is characterized by efficiently making use of the derivatives of the learned model.
- In Chapter 8, we introduce our asynchronous model-based training framework. Asynchronous training reduces the wall-clock time of model-based training by a factor of 10. Furthermore, we notice that the sample complexity is also reduced when using asynchronous learning. These results are showcased with a block stacking and shape matching task with a real PR2 robot.
- In Chapter 9, a method that is able to adapt to different dynamics is proposed. Our method uses meta-learning on temporal segments to identify the current environment given the recent experience of the agent. The proposed algorithm is able to generalize to out-of-distribution tasks, as we demonstrate with a real 6-legged

robot.

- Finally, in Chapter 10.2, we conclude with future directions for model-based reinforcement learning.

The work presented in Chapter 4 was originally published at the International Conference on Learning Representations (ICLR), 2018 (Kurutach et al., 2018), lead by Thanard Kurutach. The work presented in Chapter 5 was originally published at the Conference on Robot Learning (CORL), 2018 (Clavera et al., 2018) and it was co-first authored by Ignasi Clavera* and Jonas Rothfuss*. The works in Chapters 6 and 9 were presented at ICLR 2019 (Rothfuss et al., 2018; Nagabandi et al., 2018a). (Rothfuss et al., 2018) was co-first authored by Jonas Rothfuss*, Dennis Lee*, and Ignasi Clavera*. (Nagabandi et al., 2018a) was co-first authored by Anusha Nagabandi* and Ignasi Clavera*. Chapter 8 was originally published at CORL 2019 (Y. Zhang et al., 2019) and co-first authored by Yunzhi Zhang* and Ignasi Clavera*. Finally, the work in Chapter 7 was published at ICLR 2020 (Clavera et al., 2020).

2

PROBLEM STATEMENT

2.1 REINFORCEMENT LEARNING

The term reinforcement learning (RL) is simultaneously used to refer to a problem, a class methods of that aim to solve the problem, an a field that studies these problems and methods. In a nut shell, reinforcement learning aims to learn an optimal behaviour by trial-and-error, i.e., by repeatedly interacting with the environment, of a given metric. In here, we will formalize the problem of reinforcement learning.

The problem of reinforcement learning is a sequential decision problem where an agent must maximize a function, named expected return, by interacting with an environment. Mathematically, the problem is specified by defining its Markov decision process (MDP). In this work, we assume that the decisions are made at discrete intervals and the horizon is finite; hence, a discrete-time finite Markov decision process. An MDP, $\mathcal{M}$ is defined by the tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma, p_0, H)$. Here, $\mathcal{S}$ is the state space, i.e., all the possible states that the agent can visit; and $\mathcal{A}$ define the action space, the actions allowed to the agent. The transition distribution, also known as dynamics, is defined by $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ and determines the probability of going to a state $\mathbf{s}'$ given that the current state is $\mathbf{s}$ and the agent took action $\mathbf{a}$. The reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ specifies the value of a particular state, while the discount factor $\gamma \in [0, 1]$ accounts for the depreciation of future states. Finally, p_0 is the density function of the initial state and H is the horizon of the task. The agent then, from an initial state sampled from p_0 takes in each step and action $\mathbf{a}_t \in \mathcal{A}$ which result in the agent transition to a state $\mathbf{s}_{t+1}$, according to the dynamics p. As a result of this transition, the agent experiences a reward $r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$. The agent proceeds taking actions for H (horizon) time-steps. A policy policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ is density function that determines the probability of an action $\mathbf{a}$ for a given state $\mathbf{s}$. The reinforcement problem is the optimization problem that aims to find the policy that maximize the discounted sum of rewards, that is:

$$\max_{\pi \in \Pi} J(\pi) = \max_{\pi \in \Pi} \mathbb{E}_{\substack{s_0 \sim p_0 \\ a_t \sim \pi(s_t) \\ s_{t+1} \sim p(s_{t+1}|s_t, a_t)}} \left[\sum_{t=0}^{H} \gamma^t r(s_t, a_t, s_{t+1}) \right]$$

In here, Π is the family of functions that our policy π belongs to. In this work, we mainly consider parametrized policies, we refer them by π_θ. For a more thorough review of reinforcement learning and its history, we refer the reader to (Sutton and Barto, 2018).

Typically, reinforcement learning methods can be classified in two categories: model-free and model-based. Model-free approaches are simple and learn purely from trial-and-error, that is the optimal mapping from state to actions is learned by directly interacting with the environment. Model-free approaches, when combined with deep learning, have resulted in tremendous success. These relatively simple methods tend to scale well, hence successes such as in Silver et al., 2016; Silver et al., 2017; Mnih et al., 2015; Schulman et al., 2015b have been accomplished using these class of methods. On the other hand, model-based methods are characterized by learning a predictive model of the environment, i.e., they aim to learn the transition distribution p. The learned model $\hat{p}$ is then used to perform planning or policy improvement. Model-based methods make a better use of the collected data and fully exploit the structure of the reinforcement learning problem. As a result, they tend to be more efficient in terms of interactions with the environment. However, learning a policy or doing planning on a learn model leads to more complex algorithms and can lead to unstable learning. In this thesis, we aim to tackle the instabilities of model-based RL, namely the model-bias (Chapter 3).

2.2 MODEL-BASED REINFORCEMENT LEARNING

Model-based reinforcement learning is the set of RL methods that learn the transition distribution from the experience gathered by the agent. By constructing this learned model, model-based methods exploit the sequential nature of the MDP, which results in more sample efficient algorithms. There are two main ways of making use of the model, the model can be used as a simulator on top of which a global policy is learned, if the learned model is differientable one can use its gradient information (Sutton, 1990; Deisenroth and Rasmussen, 2011; Depeweg et al., 2017b; Sutton, 1991a). The second approach is to use the model for planning. In this case, there is no policy learning and and the model is used to locally plan ahead the best sequence of actions. In this thesis, we will mainly focus on the former, where the model is used to learn a global policy.

However, Chapter 9 makes use of the latter, and more details about it can be found in the chapter.

2.2.1 *Related Work*

There has been a large body of work on model-based reinforcement learning. They differ by the choice of model parameterization, which is associated with different ways of utilizing the model for policy learning. Interestingly, the most impressive robotic learning applications so far were achieved using the simplest possible model parameterization, namely linear models (Bagnell and Schneider, 2001; Abbeel et al., 2006; Levine and Abbeel, 2014; Watter et al., 2015; Levine et al., 2016a; Kumar et al., 2016), where the model either operates directly over the raw state, or over a feature representation of the state. Such models are very data efficient, and allows for very efficient policy optimization through techniques from optimal control. However, they only have limited expressiveness, and do not scale well to complicated nonlinear dynamics or high-dimensional state spaces, unless a separate feature learning phase is used (Watter et al., 2015).

An alternative is to use nonparametric models such as Gaussian Processes (GPs) (Rasmussen, Kuss, et al., 2003; Ko et al., 2007; Deisenroth and Rasmussen, 2011). Such models can effectively maintain uncertainty over the predictions, and have infinite representation power as long as enough data is available. However, they suffer from the curse of dimensionality, and so far their applications have been limited to relatively low-dimensional settings. The computational expense of incorporating the uncertainty estimates from GPs into the policy update also imposes an additional challenge.

Deep neural networks have shown great success in scaling up model-free reinforcement learning algorithms to challenging scenarios (Mnih et al., 2015; Silver et al., 2016; Schulman et al., 2015b; Schulman et al., 2016b). However, there has been only limited success in applying them to model-based RL. Although many previous studies have shown promising results on relatively simple domains (Nguyen and Widrow, 1990; Schmidhuber and Huber, 1991; Jordan and Rumelhart, 1992; Gal et al., 2016), so far their applications on more challenging domains have either required a combination with model-free techniques (Oh et al., 2015; Heess et al., 2015; Nagabandi et al., 2017), or domain-specific policy learning or planning algorithms (Lenz et al., 2015; Agrawal et al., 2016; Levine et al., 2016b; Finn and Levine, 2017a; Ashvin Nair et al., 2017).

2.2.2 *Vanilla Model-Based Method*

In the following, we describe the vanilla model-based reinforcement learning algorithm (see Algorithm 1). The vanilla algorithm iterates through the steps until convergence: data collection, model learning, and policy improvement.

2.2.2.1 *Data Collection*

The first step of any reinforcement learning method is to collect data. The data collection can be in principle done by an exploratory policy; however, in practice, the latest trained policy of the algorithm is used. Using the latest policy allows to gather data in regions of the environment where the model is being over-optimistic, correcting its predictions in the subsequent model learning step. The data collected by the policy π_θ is stored in the data buffer $\mathcal{D}$.

2.2.2.2 *Model learning*

Once we have stored the transitions experienced by the agent in the buffer $\mathcal{D}$, we use those to learn a predictive model of the environment. In this vanilla approach, as well as in the rest of this manuscript, the learned model $\hat{p}_\phi$ corresponds to a feed-forward neural network parametrized by the weights $\boldsymbol{\phi}$. As is standard, the neural network parametrizes the change in state, i.e., $\mathbf{s}_{t+1} - \mathbf{s}_t$, instead of the next state given the current state and action. This relieves the neural network from learning small deltas of the identity func-

Algorithm 1 Vanilla Model-Based Deep Reinforcement Learning

```
1:  Initialize a policy $\pi_\theta$ and a model $\hat{p}_\phi$.
2:  Initialize an empty dataset $\mathcal{D}$.
3:  repeat
4:      Collect samples from the real environment $p$ using $\pi_\theta$ and add them to $\mathcal{D}$.
5:      Train the model $\hat{p}_\phi$ using $\mathcal{D}$.
6:      repeat
7:          Collect fictitious samples from $\hat{p}_\phi$ using $\pi_\theta$.
8:          Update the policy using BPTT on the fictitious samples.
9:          Estimate the performance $J(\theta)$.
10:     until the performance stop improving.
11: until the policy performs well in real environment $f$.
```

tion, especially when the change is small (Deisenroth and Rasmussen, 2011; Fu et al., 2016; Nagabandi et al., 2017). For notational convenience however, we will denote $\hat{p}_\phi$ the density function of the next state, given the current state and action.

In the model learning step, we train the model by maximizing the likelihood of the collected data:

$$\max_{\phi} \mathbb{E}_{(s',a,s)\sim\mathcal{D}} \left[\log \hat{p}_\phi(s'|s,a)\right] \tag{1}$$

Hence, the model is learned with standard supervised learning. While some works use multi-step losses, standard methods (and hence the vanilla method) learn one-step prediction models[1]. The model used in most of the cases follows a Gaussian distribution with an fixed diagonal covariance, hence resulting in the minimization of the standard mean squared error. Standard techniques to prevent overfitting and facilitate learning are used: early stopping with a validation dataset, and normalizing inputs and outputs.

2.2.2.3 *Policy Improvement*

The last step of the vanilla algorithm is to use the learned model to train a policy. The beauty of model-based methods is that now we can consider the MDP $\hat{\mathcal{M}}$ formed by $(\mathcal{S}, \mathcal{A}, \hat{p}_\phi, r, \gamma, p_0, H)$ and use all the machinery of model-free RL, control, or planning. Furthermore, given that the learned model is differiantable one can assume gradients of the dynamics. Hence, in the policy improvement step we maximze the objective specified in Equation 2.1 with the MDP $\hat{\mathcal{M}}$:

$$\max_{\theta} J(\theta) = \max_{\theta} \mathbb{E}_{\substack{s_0\sim p_0 \\ s_{t+1}\sim\hat{p}_\phi \\ a_t\sim\pi_\theta}} \left[\sum_{t=0}^{H} \gamma^t r(s_t, a_t, s_{t+1})\right]$$

The gradient of Equation 2.2.2.3 can be estimated by Monte-Carlo methods

$$\nabla\theta J(\theta) = \mathbb{E}_{\substack{s_0\sim p_0 \\ s_{t+1}\sim\hat{p}_\phi \\ a_t\sim\pi_\theta}} \left[\nabla_\theta \sum_{t=0}^{H} \gamma^t r(s_t, a_t, s_{t+1})\right]$$

The gradient of each sample is computed by propagating the gradient of the last step to the initial step, named backpropation through time (BPTT).

1 We found that multi-step prediction loss did not significantly improve the policy learning results.

2.3 META-LEARNING

The algorithms proposed in Chapters 5, 6, and 9 are build on top of meta-learning algorithms. Here, we provide an overview of the meta-learning problem. For a more thorough overview and methods, we refer to the reader to Schmidhuber, 1987; Finn, 2018; Duan, 2017.

Meta-learning is concerned with automatically learning, learning algorithms that are more efficient and effective than learning from scratch. These algorithms leverage data from *previous* tasks to acquire a learning procedure that can quickly adapt to new tasks. These methods operate under the assumption that the previous meta-training tasks and the new meta-test tasks are drawn from the same task distribution $\rho(\mathcal{M})$ and share a common structure that can be exploited for fast learning. In the supervised learning setting, we aim to learn a function $f_{\boldsymbol{\phi}}$ with parameters $\boldsymbol{\phi}'$ that minimizes a loss $L_{\mathcal{M}}$. Then, the goal of meta-learning is to find a learning procedure, denoted as $\boldsymbol{\phi}' = u_{\boldsymbol{\psi}}(\mathcal{D}_{\mathcal{M}}^{\text{train}}, \boldsymbol{\phi})$, that can learn a range of tasks $\mathcal{M}$ from small datasets $\mathcal{D}_{\mathcal{M}}^{\text{train}}$.

We can formalize this meta-learning problem setting as optimizing for the parameters of the function and learning procedure, $\boldsymbol{\phi}' and \boldsymbol{\psi}$ respectively:

$$\min_{\boldsymbol{\phi},\boldsymbol{\psi}} \mathbb{E}_{\mathcal{M}\sim\rho(\mathcal{M})}\left[L(\mathcal{D}_{\mathcal{M}}^{\text{test}}, \boldsymbol{\phi}')\right] \quad \text{s.t.} \quad \boldsymbol{\phi}' = u_{\boldsymbol{\psi}}(\mathcal{D}_{\mathcal{M}}^{\text{train}}, \boldsymbol{\phi}) \tag{2}$$

where $\mathcal{D}_{\mathcal{M}}^{\text{train}}, \mathcal{D}_{\mathcal{M}}^{\text{test}}$ are sampled without replacement from the meta-training dataset $\mathcal{D}_{\mathcal{M}}$.

Once meta-training optimizes for the parameters $\boldsymbol{\phi}'_*, \boldsymbol{\psi}_*$, the learning procedure $u_{\boldsymbol{\psi}_*}(\cdot, \boldsymbol{\phi}_*)$ can then be used to learn new held-out tasks from small amounts of data. We will also refer to the learning procedure u as the update function.

Gradient-based meta-learning. Gradient-based methods (Finn et al., 2017) aims to learn the initial parameters of a neural network such that taking one or several gradient descent steps from this initialization leads to effective generalization (or few-shot generalization) to new tasks. Then, when presented with new tasks, the model with the meta-learned initialization can be quickly fine-tuned using a few data points from the new tasks. Using the notation from before:

$$u_{\boldsymbol{\psi}}(\mathcal{D}_{\mathcal{M}}^{\text{train}}, \boldsymbol{\phi}) = \boldsymbol{\phi} - \alpha\nabla_{\boldsymbol{\phi}} L(\mathcal{D}_{\mathcal{M}}^{\text{train}}, \boldsymbol{\phi}') \tag{3}$$

The learning rate α may be a learnable parameter (in which case $\boldsymbol{\psi} = \alpha$) or fixed as a hyperparameter, leading to $\boldsymbol{\psi} = \varnothing$. Despite the update rule being fixed, a learned initialization of an overparameterized deep network followed by gradient descent is as expressive as update rules represented by deep recurrent networks (Finn and Levine, 2017b).

Recurrence-based meta-learning. Another approach to meta-learning is to use recurrent models. In this case, the update function is always learned, and ψ corresponds to the weights of the recurrent model that update the hidden state. The parameters ϕ of the prediction model correspond to the remainder of the weights of the recurrent model and the hidden state. Both gradient-based and recurrence-based meta-learning methods have been used for meta model-free RL (Finn et al., 2017; Duan et al., 2016b).

2.3.1 *Meta-Learning for Reinforcement Learning*

One can extend the concept of meta-learning to RL (J. Wang et al., 2016; Z. Xu et al., 2018; Finn et al., 2017; Duan et al., 2016b), as we do in Chapters 5 and 6. In that case, the different tasks $\mathcal{M}$ correspond to different MDPs and one optimizes the objective defined in Equation 2.1 for each of the tasks. Then, meta-RL aims to learn a learning procedure which is able to quickly learn optimal policies in a set of MDPs $\{\mathcal{M}_k\}$ drawn from a distribution $\rho(\mathcal{M})$. The MDPs $\mathcal{M}_k$ may differ in their reward function $r_k(\mathbf{s}, \mathbf{a})$ and/or transition distribution $p_k(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, but share action space $\mathcal{A}$ and state space $\mathcal{S}$.

In the context of meta-reinforcement learning, we focus on gradient-based meta-learning algorithms (Finn et al., 2017). They train a parametric policy π_θ to quickly improve its performance on a new task with one or a few vanilla policy gradient steps Peters and Schaal, 2006. The meta-training objective for gradient-based meta-learning can be written as:

$$\max_{\theta} \; \mathbb{E}_{\substack{\mathcal{M}_k \sim \rho(\mathcal{M}) \\ \mathbf{s}_{t+1} \sim p_k \\ \mathbf{a}_t \sim \pi_{\theta'}}} \left[\sum_{t=0}^{H-1} r_k(\mathbf{s}_t, \mathbf{a}_t) \right] \quad \text{s.t.:} \; \theta' = \theta + \alpha \, \nabla_\theta \mathbb{E}_{\substack{\mathbf{s}_{t+1} \sim p_k \\ \mathbf{a}_t \sim \pi_\theta}} \left[\sum_{t=0}^{H-1} r_k(\mathbf{s}_t, \mathbf{a}_t) \right] \tag{4}$$

3

MODEL-BIAS

Model-based reinforcement learning offers the possibility to efficiently learn new tasks in an autonomous way. When compared with model-free methods, this efficiency is obtained by making a better use of the available data and exploiting the structure of the problem. Since the reinforcement learning problem is set within the Markov decision process framework, we know of the existence of a dynamics function that just depends on the current state an action. Therefore, model-based RL aims to learn that dynamics function using the transitions that the agent experience. The learn model is then substituted with the real dynamics, and a policy is learned on to of them. In contrast, model-free RL ignores the dynamics of it and shortcuts the learning process by directly learning the mapping from states to actions, a more costly learning process. However, because of the extra approximation that model-based RL does, the asymptotic performance of it is typically well bellow its model-free counter part, limiting its applicability to simpler domains. The reason for this gap in performance is due to the commonly known *model-bias* problem (Deisenroth and Rasmussen, 2011; Schneider, 1997; Atkeson and Santamaria, 1997; Schaal, 1997). Model-bias has been an umbrella term that encompasses all the reasons of why model-based reinforcement learning suffers from poor performance when compared with model-free methods. In the following, we decompose the model-bias problem into three main issues: estimation errors, compounding errors, and over-optimism.

3.1 ESTIMATION ERRORS

Estimation errors occur during the model learning step, and are present in any supervised learning method. Estimation errors can also be decomposed into structural errors, and approximation errors. Structural errors arise from the fact that our function class has limited capacity; hence, failing to capture the complexity of the real environment.

Essentially, they measure the estimation error our model would have if infinite data was available. They can be mitigated by using higher capacity models, such as neural networks (Nguyen and Widrow, 1990; Schmidhuber and Huber, 1991). Instead, approximation errors measure how close can our model class can get to the capture the real environment given the limited availability of data.

As we explained in the previous section, model-based RL first learn a model of the environment. Then it uses this model to improve the policy that will be executed in the real-environment.

$$\max_{\theta} J(\theta) = \max_{\theta} \mathbb{E}_{\substack{s_0 \sim p_0 \\ s_{t+1} \sim \hat{p}_\phi \\ a_t \sim \pi_\theta}} \left[\sum_{t=0}^{H} \gamma^t r(s_t, a_t, s_{t+1}) \right]$$

Notice that in the expectation, samples are drawn from the learned model. Model-based RL uses this proxy objective to optimize the RL objective (Equation 2.1):

$$\max_{\theta} J(\theta) = \max_{\theta} \mathbb{E}_{\substack{s_0 \sim p_0 \\ s_{t+1} \sim p \\ a_t \sim \pi_\theta}} \left[\sum_{t=0}^{H} \gamma^t r(s_t, a_t, s_{t+1}) \right]$$

As a result, large model errors, i.e., large $D_{KL}(p \| \hat{p}_\phi)$, will result in a policy learned on top of wrong dynamics that will not transfer to the real-environment.

3.2 COMPOUNDING ERRORS

Estimation errors are exacerbated by their usage in model-based RL. Learned dynamics are used to plan ahead into the future or to simulate trajectories (Sutton, 1991a; Nagabandi et al., 2017; Kurutach et al., 2018). The predictions of the model are bootstrapped to predict further into the future states with a horizon of ranging from hundreds to thousands of steps. During these multiple step predictions, errors are compounded resulting in state predictions on regions that the model has not been trained on. Then, the predictions of the model are completely arbitrary and they are claimed with full confidence, see the conceptual sketch in Figure 1.

While learning more accurate models might relieve the compounding error issue, it is still present when we use them for long horizons. Previous approaches have tackled this problem by learning probabilistic models (Deisenroth and Rasmussen, 2011) or multi-step prediction models (Asadi et al., 2019).

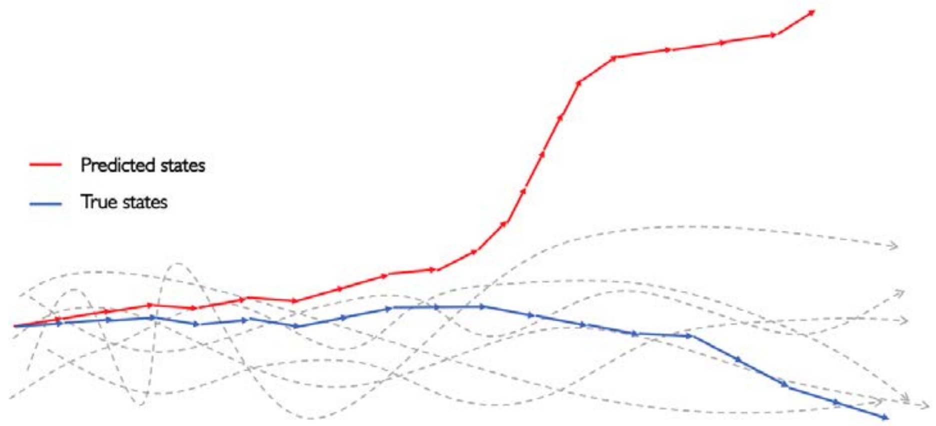


Figure 1: Conceptual example. The model trained on collected trajectories (grey lines) is used to predict a trajectory (red line). While the initial predictions present small error, the compounding effect of these errors leads the predictions to regions where the model has not been trained on, completely diverging from the real trajectory (blue line).

3.3 OVER-OPTIMISM

While the two previous issues are also present in supervised learning, the over-optimism problem is just particular to model-based RL. Over-optimism refers to the common problem when a policy optimized in a learned model finds “shortcuts" not realizable, or sub-optimal in the real environment, due to the lack of training data. For instance, consider the conceptual example in Figure 2, where the agent’s goal is to reach the star. In the first step of data collection, we gather trajectories that do not cover the entire state-space. As a result, the learned function is able to model the regions around the gathered trajectories, but it is not able to predict the environment correctly in the other regions. When we optimize for the optimal policy in the learned environment, it finds the shortest trajectory in the learned model. However, the shortest trajectory in the model is not feasible in the real maze.

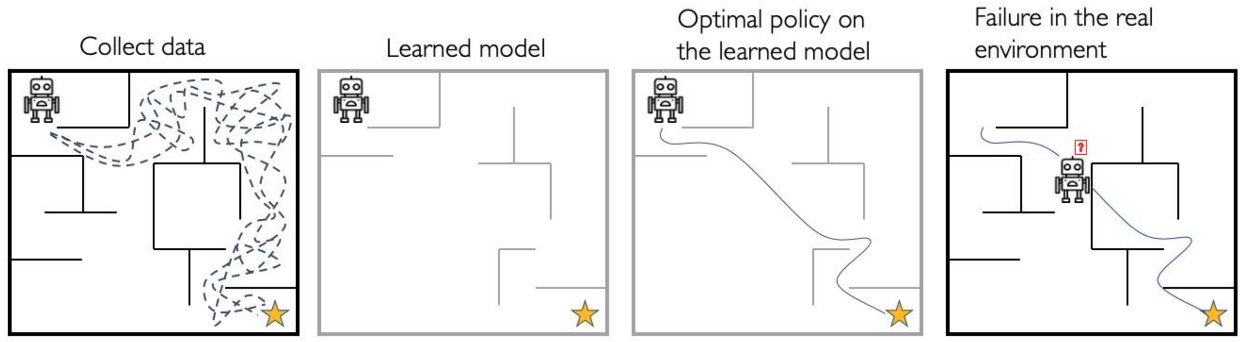

Collect data
Learned model
Optimal policy on the learned model
Failure in the real environment

Figure 2: Conceptual example. The learned model (grey maze) captures the regions where data has been collected (dashed lines), but cannot capture how the environment is in regions where no data has been collected. When the policy is trained on top of the learned model it will exploit inaccuracies in it to achieve the optimal policy (solid line) in the learned environment. This results in a policy that can fail in the real environment.

Optimizing reinforcement learning agents on top of learned models is a hard task because the policy chooses its own distribution. The agent then can very easily exploit the errors of the model to its favour, and traditional regularization techniques do not work (Kukacka et al., 2017). Because of these three problems, model-based reinforcement learning has been limited to simple domains and little success has been accomplished.

4 MODEL-ENSEMBLE TRUST-REGION POLICY OPTIMIZATION

In this first work, we aim to tackle the over-optimism problem in model-bias. As mentioned in the previous chapter, over-optimism is the result of the agent being able to choose their own distribution on the learned model, which allows it to exploit the inaccuracies of the model to achieve higher rewards. Because this shortcuts that the agent finds in the learned model do not exist in the real environment, the agent fails to achieve the task. In here, instead of training the policy in our best model, we learned the policy in an ensemble of them. We learn multiple models, each slightly different, and aim to learn a policy that is optimal in all of them. Since each model is different, it will present different predictions in regions where it has not been trained on; and since we are training a policy that is optimal in all of them, it cannot exploit the shortcuts of each model that do not exist in the real environment.

4.1 OVERVIEW

The standard approach for model-based reinforcement learning alternates between model learning and policy optimization. In the model learning stage, samples are collected from interaction with the environment, and supervised learning is used to fit a dynamics model to the observations. In the policy optimization stage, the learned model is used to search for an improved policy. The underlying assumption in this approach, henceforth termed *vanilla* model-based RL, is that with enough data, the learned model will be accurate enough, such that a policy optimized on it will also perform well in the real environment.

Although vanilla model-based RL can work well on low-dimensional tasks with relatively simple dynamics, we find that on more challenging continuous control tasks, performance was highly unstable. The reason is that the policy optimization tends to exploit regions where insufficient data is available to train the model, leading to catastrophic failures. Previous work has pointed out this issue as *model bias*, i.e. (Deisenroth and Rasmussen, 2011; Schneider, 1997; Atkeson and Santamaria, 1997). While this issue can be regarded as a form of overfitting, we emphasize that standard countermeasures from the supervised learning literature, such as regularization or cross validation, are not sufficient here – supervised learning can guarantee generalization to states from the same distribution as the data, but the policy optimization stage steers the optimization exactly towards areas where data is scarce and the model is inaccurate. This problem is severely aggravated when expressive models such as deep neural networks are employed.

To resolve this issue, we propose to use an ensemble of deep neural networks to maintain model uncertainty given the data collected from the environment. During model learning, we differentiate the neural networks by varying their weight initialization and training input sequences. Then, during policy learning, we regularize the policy updates by combining the gradients from the imagined stochastic roll-outs. Each imagined step is uniformly sampled from the ensemble predictions. Using this technique, the policy learns to become robust against various possible scenarios it may encounter in the real environment. To avoid overfitting to this regularized objective, we use the model ensemble for early stopping policy training.

Standard model-based techniques require differentiating through the model over many time steps, a procedure known as backpropagation through time (BPTT). It is well-known in the literature that BPTT can lead to exploding and vanishing gradients (Bengio et al., 1994). Even when gradient clipping is applied, BPTT can still get stuck in bad local optima. We propose to use likelihood ratio methods instead of BPTT to estimate the gradient, which only make use of the model as a simulator rather than for direct gradient computation. In particular, we use Trust Region Policy Optimization (TRPO) (Schulman et al., 2015b), which imposes a trust region constraint on the policy to further stabilize learning.

In this work, we propose model-ensemble trust-region policy optimization (ME-TRPO), a model-based algorithm that achieves the same level of performance as state-of-the-art model-free algorithms with 100$\times$ reduction in sample complexity. We show that the model ensemble technique is an effective approach to overcome the challenge of model bias in model-based reinforcement learning. We demonstrate that replacing BPTT by TRPO yields significantly more stable learning and much better final performance. Finally, we provide an empirical analysis of vanilla model-based RL using neural networks as function approximators, and identify its flaws when applied to challenging continuous control tasks.

4.2 METHOD

The vanilla model-based approach tends to suffer from model-bias (Chapter 3). This issue can be partly alleviated by early stopping on validation initial states. However, we found this insufficient, since the performance is still evaluated using the same learned model, which tends to make consistent mistakes. Furthermore, another complication that arises when learning a policy with the vanilla approach is the instability from the exploding and vanishing gradients. Since the gradients are backpropagated throughout the entire trajectory their compouning results in extremely large or small values. While gradient clipping can resolve exploding gradients, the backpropagation through time (BPTT) still suffers from vanishing gradients, which cause the policy to get stuck in bad local optima (Bengio et al., 1994; Pascanu et al., 2013). This problem is exacerbated in when learning over long horizons.

In this section we present model ensemble trust region policy optimization (ME-TRPO), pseudo-code is shown in Algorithm 2. ME-TRPO tackles the aforementioned difficulties by characterizing the epistemic uncertainty with an ensemble of models and by early stopping of the policy. Both techniques prevent the policy from presenting model-bias. Finally, we use the models as proxies of the real-world; hence, drawing samples from them instead of backpropagating through them, preventing the common issue of exploding and vanishing gradients. Specifically, we use the trust-region policy optimization (TRPO) algorithm to learn the policy over the ensemble of models. These modifications are described in detail below.

Model learning. we fit a set of dynamics models $\{\hat{p}_{\phi_1}, \ldots, \hat{p}_{\phi_K}\}$ (termed a *model ensemble*) using the same real world data. These models are trained via standard supervised learning as described in Section 2.2.2.2, and they only differ by the initial weights and the order in which mini-batches are sampled.

Policy optimization. To overcome the issues with BPTT, we use likelihood-ratio methods from the model-free RL literature. We evaluated using Vanilla Policy Gradient (VPG) (Peters and Schaal, 2006), Proximal Policy Optimization (PPO) (Schulman et al., 2017), and Trust Region Policy Optimization (TRPO) (Schulman et al., 2015b). The best results were achieved by TRPO. In order to estimate the gradient, we use the learned models to simulate trajectories as follows: in every step, we randomly choose a model to predict the next state given the current state and action. This avoids the policy from overfitting to any single model during an episode, leading to more stable learning.

Policy validation. We monitor the policy's performance using the K learned models.

Algorithm 2 Model Ensemble Trust Region Policy Optimization (ME-TRPO)

```
1: Initialize a policy $\pi_\theta$ and all models $\hat{p}_{\phi_1}, \hat{p}_{\phi_1}, ..., \hat{p}_{\phi_K}$.
2: Initialize an empty dataset $\mathcal{D}$.
3: repeat
4:    Collect samples from the real system $f$ using $\pi_\theta$ and add them to $D$.
5:    Train all models using $\mathcal{D}$.
6:    repeat
7:       Optimize $\pi_\theta$ using all models.
8:       Collect fictitious samples from $\{\hat{p}_{\phi_i}\}_{i=1}^K$ using $\pi_\theta$.
9:       Update the policy using TRPO on the fictitious samples.
10:      Estimate the performances $J_i(\theta)$ for $i = 1, ..., K$.
11:   until the performances stop improving.
12: until the policy performs well in real environment $f$.
```

Specifically, we compute the ratio of models in which the policy improves:

$$\frac{1}{K}\sum_{k=1}^{K} \mathbb{1}[J_k(\theta_{\text{new}}) > J_k(\theta_{\text{old}})]. \tag{5}$$

The current iteration continues as long as this ratio exceeds a certain threshold. In practice, we validate the policy after every 5 gradient updates and we use 70% as the threshold. If the ratio falls below the threshold, a small number of updates is tolerated in case the performance improves, and otherwise the current iteration is terminated. Then, we repeat the overall process of using the policy to collect more real-world data, optimizing the model ensemble, and using the model ensemble to improve the policy. This process continues until the desired performance is reached in the real environment.

The model ensemble serves as effective regularization for policy learning: by using the model ensemble for policy optimization and validation, the policy is forced to perform well over a vast number of possible alternative futures. Even though any of the individual models can still incur model bias, our experiments below suggests that combining these models yields stable and effective policy improvement.

4.3 EXPERIMENTS

The algorithm described in the previous section aims to tackle the model-bias problem of classical model-based methods. Here, we aim to evaluate if our method successfully

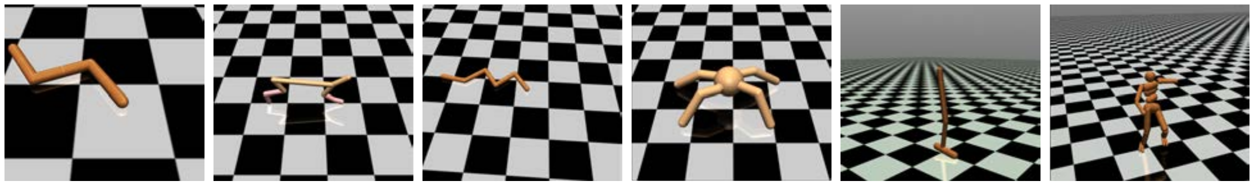


Figure 3: Mujoco environments used in our experiments. Form left to right: Swimmer, Half Cheetah, Snake, Ant, Hopper, and Humanoid.

overcomes the common pitfalls of model-based methods. Specifically, we design the experiments to know (1) what are the failure cases of the vanilla methods? (2) how does ME-TRPO overcome these failers?

Finally, we compare our approach to state-of-the-art model-free methods. The algorithms are evaluated on six standard continuous control benchmark tasks (Duan et al., 2016a; Dhariwal et al., 2017) in Mujoco (Todorov et al., 2012): Swimmer, Snake, Hopper, Ant, Half Cheetah, and Humanoid, shown in Figure 3. The details of the tasks can be found in Appendix A.2. In the Appendix A.4 an ablation study to characterize the effect of each component of our algorithm.

4.3.1 *From Vanilla to ME-TRPO*

In this section we explain and quantify the failure cases of vanilla model-based reinforcement learning, and how our approach overcomes such failures. We analyze the effect of each of our proposed modifications by studying the learning behavior of replacing BPTT with TRPO in vanilla model-based RL using just a single model, and then the effect of using an ensemble of models.

As discussed above, BPTT suffers from exploding and vanishing gradients, especially when optimizing over long horizons. Furthermore, one of the principal drawbacks of BPTT is the assumption that the model derivatives should match that of the real dynamics, even though the model has not been explicitly trained to provide accurate gradient information. In Figure 4 we demonstrate the effect of using policy gradient methods that make use of a score function estimator, such as VPG and TRPO, while using a single learned model. The results suggest that in comparison with BPTT, policy gradient methods are more stable and lead to much better final performance. By using such model-free algorithms, we require less information from the learned model, which only acts as a simulator. Gradient information through the dynamics model is not needed anymore to optimize the policy.

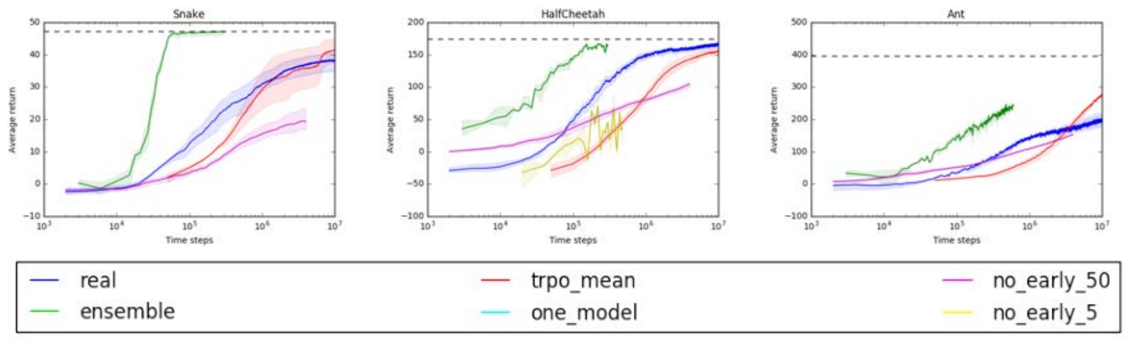


Figure 4: Comparison among different policy optimization techniques with one model. Using TRPO for model-based optimization leads to the best policy learning across the different domains (Best viewed in color).

However, while replacing BPTT by TRPO helps optimization, the learned policy can still suffer from model bias. The learning procedure tends to steer the policy towards regions where it has rarely visited, so that the model makes erroneous predictions to its advantage. The estimated performances of the policy often end up with high rewards according to the learned model, and low rewards according to the real one (see Appendix A.3 for further discussion). In Figure 5, we analyze the effect of using various numbers of ensemble models for sampling trajectories and validating the policy's performance. The results indicate that as more models are used in the model ensemble, the learning is better regularized and the performance continually improves. The improvement is even more noticeable in more challenging environments like HalfCheetah and Ant, which require more complex dynamics models to be learned, leaving more room for the policy to exploit when model ensemble is not used.

4.3.2 *Comparison to State-of-the-Art*

We compare our method with the following state-of-the-art reinforcement learning algorithms in terms of sample complexity and performance: trust region policy optimization (TRPO) (Schulman et al., 2015b), proximal policy optimization (PPO) (Schulman et al., 2017), deep deterministic policy gradient (DDPG) (Lillicrap et al., 2015), and stochastic value gradient (SVG) (Heess et al., 2015).

The results are shown in Figure 6. Prior model-based methods appears to achieve worse performance compared with model-free methods. In addition, we find that model-based methods tend to be difficult to train over long horizons. In particular, SVG(1), not

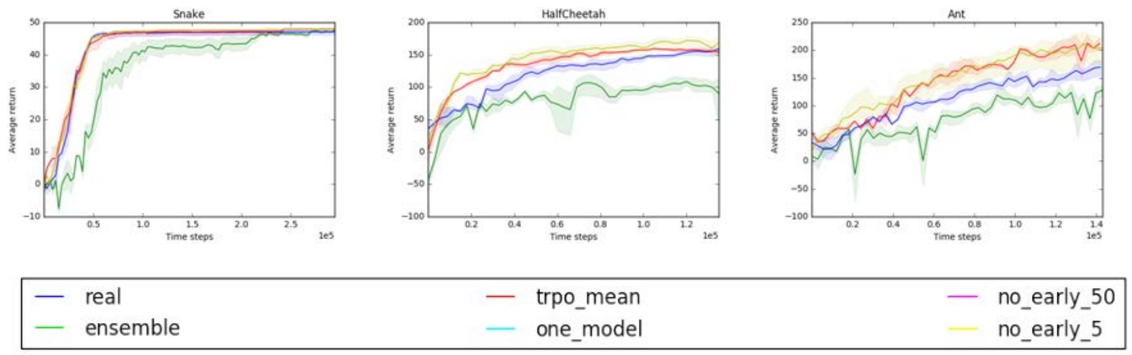


Figure 5: Comparison among different number of models that the policy is trained on. TRPO is used for the policy optimization. We illustrate the improvement when using 5, 10 and 20 models over a single model (Best viewed in color).

presented in the plots, is very unstable in our experiments. While SVG(∞) is more stable, it does not scale up to more complex tasks. In contrast, our proposed method reaches the same level of performance as model-free approaches with $\approx 100\times$ less data. To the best of our knowledge, it is the first purely model-based approach that can optimize policies over high-dimensional locomotion tasks such as Humanoid. For experiment details please refer to Appendix A.

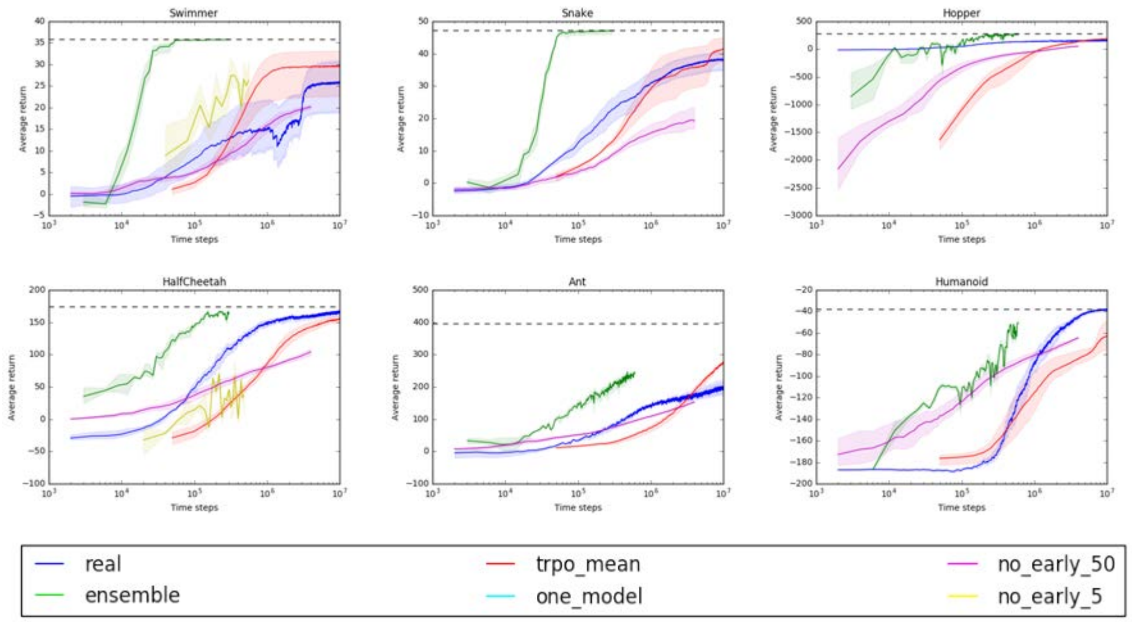


Figure 6: Learning curves of our method versus state-of-the-art methods. The horizontal axis, in log-scale, indicates the number of time steps of real world data. The vertical axis denotes the average return. These figures clearly demonstrate that our proposed method significantly outperforms other methods in comparison (best viewed in color).

4.4 DISCUSSION

In this work, we present a simple and robust model-based reinforcement learning algorithm that is able to learn neural network policies across different challenging domains. We show that our approach significantly reduces the sample complexity compared to state-of-the-art methods while reaching the same level of performance. In comparison, our analyses suggests that vanilla model-based RL tends to suffer from the over-optimism in model bias and numerical instability, and fails to learn a good policy. We further evaluate the effect of each key component of our algorithm, showing that both using TRPO and model ensemble are essential for successful applications of deep model-based RL.

5

MODEL-BASED REINFORCEMENT LEARNING VIA META-POLICY OPTIMIZATION

In the previous section we presented model-ensemble trust-region policy optimization (ME-TRPO). The aforementioned method is able to surpass previous model-based reinforcement methods by (1) characterizing the uncertainty by the means of an ensemble of models, and (2) sampling from the model instead of backpropagating through it. In the following work we challenge underlying assumption of (1). ME-TRPO uses the model to prevent the policy from overfitting to the learned dynamics, while it accomplishes this task it prevents the policy from achieving optimal performance. First, the learned model will not mimic perfectly the world dynamics; hence, some performance gap is bound to exists. Second, this same fact is exacerbated by the posed optimization problem. We are aiming to learn the optimal policy across different dynamics models. The result of it is a policy that presents a gap in perfomance when compared against model-free methods; gap which widens as environments become more complex.

In the following work, we acknowledge that learning accurate representations of the dynamics is a nearly impossible task. Nevertheless, we are still interested in learning optimal policies with minimal data requirements. We propose to add another layer of abstraction in the learning process and relying on the generalization of such layer. In model-based meta-policy optimization we consider the model-based reinforcement learning process from the point of view of meta-learning (Finn et al., 2017; Duan et al., 2016b), akin to dynamics randomization (Peng et al., 2017). In essence, we meta-learn a policy that is able to adapt to different dynamics. This policy is trained entirely on an ensemble of dynamics models, being each model a different dynamics randomization of the real-world dynamics.

5.1 OVERVIEW

Most of the recent success in reinforcement learning was achieved using model-free reinforcement learning algorithms (Schulman et al., 2015b; Lillicrap et al., 2015; Silver et al., 2016). Model-free (MF) algorithms tend to achieve optimal performance, are generally applicable, and are easy to implement. However, this is achieved at the cost of being data intensive, which is exacerbated when combined with high-capacity function approximators like neural networks. Their high sample complexity presents a major barrier to their application to robotic control tasks, on which data gathering is expensive.

In contrast, model-based (MB) reinforcement learning methods are able to learn with significantly fewer samples by using a learned model of the environment dynamics against which policy optimization is performed. Learning dynamics models can be done in a sample efficient way since they are trained with standard supervised learning techniques, allowing the use of off-policy data. However, accurate dynamics models can often be far more complex than good policies. For instance, pouring water into a cup can be achieved by a fairly simple policy while modeling the underlying dynamics of this task is highly complex. Hence, model-based methods have only been able to learn good policies on a much more limited set of problems, and even when good policies are learned, they typically saturate in performance at a level well below their model-free counterparts (Deisenroth et al., 2013; Pong et al., 2018).

Model-based approaches tend to rely on accurate (learned) dynamics models to solve a task. If the dynamics model is not sufficiently precise, the policy optimization is prone to overfit on the deficiencies of the model, leading to suboptimal behavior or even to catastrophic failures. This problem is known in the literature as model-bias (Deisenroth and Rasmussen, 2011). Previous work has tried to alleviate model-bias by characterizing the uncertainty of the models and learning a robust policy (Deisenroth and Rasmussen, 2011; Rajeswaran et al., 2016; Zhou et al., 1996; Lim et al., 2013; Kurutach et al., 2018), often using ensembles to represent the posterior. This paper also uses ensembles, but very differently.

We propose model-based meta-policy-optimization (MB-MPO), an orthogonal approach to previous model-based RL methods: while traditional model-based RL methods rely on the learned dynamics models to be sufficiently accurate to enable learning a policy that also succeeds in the real world, we forego reliance on such accuracy. We are able to do so by learning an ensemble of dynamics models and framing the policy optimization step as a meta-learning problem. Meta-learning, in the context of RL, aims to learn a policy that adapts fast to new tasks or environments (Finn et al., 2017; Duan et al., 2016b;

J. Wang et al., 2016; Mishra et al., 2018; Sung et al., 2017). Using the models as learned simulators, MB-MPO learns a policy that can be quickly adapted to any of the fitted dynamics models with one gradient step. This optimization objective steers the meta-policy towards internalizing the parts of the dynamics prediction that are consistent among the ensemble while shifting the burden of behaving optimally w.r.t discrepancies between models towards the adaptation step. This way, the learned policy exhibits less model-bias without the need to behave conservatively. While much is shared with previous MB methods in terms of how trajectory samples are collected and the dynamic models are trained, the use of (and reliance on) learned dynamics models for the policy optimization is fundamentally different.

In this paper we show that 1) model-based policy optimization can learn policies that match the asymptotic performance of model-free methods while being substantially more sample efficient, 2) MB-MPO consistently outperforms previous model-based methods on challenging control tasks, 3) learning is still possible when the models are strongly biased. The low sample complexity of our method makes it applicable to real-world robotics. For instance, we are able learn an optimal policy in high-dimensional and complex quadrupedal locomotion within seven hours of real-world data. Note that the amount of data required to learn such policy using model-free methods is $10\times$ - $100\times$ higher, and, to the best knowledge of the authors, no prior model-based method has been able to attain the model-free performance in such tasks.

5.2 METHOD

Enabling complex and high-dimensional real robotics tasks requires extending current model-based methods to the capabilities of mode-free while, at the same time, maintaining their data efficiency. Our approach, model-based meta-policy-optimization (MB-MPO), attains such goal by framing model-based RL as meta-learning a policy on a distribution of dynamic models, advocating to maximize the policy adaptation, instead of robustness, when models disagree. This not only removes the arduous task of optimizing for a single policy that performs well across differing dynamic models, but also results in better exploration properties and higher diversity of the collected samples, which leads to improved dynamic estimates.

We instantiate this general framework by employing an ensemble of learned dynamic models and meta-learning a policy that can be quickly adapted to any of the dynamic models with one policy gradient step. In the following, we first describe how the models are learned, then explain how the policy can be meta-trained on an ensemble of models, and, finally, we present our overall algorithm.

5.2.1 *Model Learning*

A key component of our method is learning a distribution of dynamics models, in the form of an ensemble, of the real environment dynamics. In order to decorrelate the models, each model differs in its random initialization and it is trained with a different randomly selected subset $\mathcal{D}_k$ of the collected real environment samples. In order to address the distributional shift that occurs as the policy changes throughout the meta-optimization, we frequently collect samples under the current policy, aggregate them with the previous data $\mathcal{D}$, and retrain the dynamic models with warm starts.

In our experiments, we consider the dynamics models to be a deterministic function of the current state $\mathbf{s}_t$ and action $\mathbf{a}_t$, employing a feed-forward neural network to approximate them. We follow the standard practice in model-based RL of training the neural network to predict the change in state $\Delta\mathbf{s} = \mathbf{s}_{t+1} - \mathbf{s}_t$ (rather than the next state $\mathbf{s}_{t+1}$) (Nagabandi et al., 2017; Deisenroth and Rasmussen, 2011). We denote by $\hat{p}_{\boldsymbol{\phi}}$ the function approximator for the next state, which is the sum of the input state and the output of the neural network. The objective for learning each model $\hat{p}_{\boldsymbol{\phi}_k}$ of the ensemble is to find the parameter vector $\boldsymbol{\phi}_k$ that minimizes the ℓ_2 one-step prediction loss:

$$\min_{\boldsymbol{\phi}_k} \frac{1}{|\mathcal{D}_k|} \sum_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \in \mathcal{D}_k} \|\mathbf{s}_{t+1} - \hat{p}_{\boldsymbol{\phi}_k}(\mathbf{s}_t, \mathbf{a}_t)\|_2^2 \tag{6}$$

where $\mathcal{D}_k$ is a sampled subset of the training data-set $\mathcal{D}$ that stores the transitions which the agent has experienced. Standard techniques to avoid overfitting and facilitate fast learning are followed; specifically, 1) early stopping the training based on the validation loss, 2) normalizing the inputs and outputs of the neural network, and 3) weight normalization (Salimans and Kingma, 2016).

5.2.2 *Meta-Reinforcement Learning on Learned Models*

Given an ensemble of learned dynamic models for a particular environment, our core idea is to learn a policy which can adapt quickly to any of these models. To learn this policy, we use gradient based meta-learning with MAML (described in Section 2.3.1). To properly formulate this problem in the context of meta-learning, we first need to define an appropriate task distribution. Considering the models $\{\hat{p}_{\boldsymbol{\phi}_1}, \hat{p}_{\boldsymbol{\phi}_2}, ..., \hat{p}_{\boldsymbol{\phi}_K}\}$, which approximate the dynamics of the true environment, we can construct a uniform task distribution by embedding them into different MDPs $\mathcal{M}_k = (\mathcal{S}, \mathcal{A}, \hat{p}_{\phi_k}, r, \gamma, p_0)$ using these learned dynamics models. We note that, unlike the experimental considerations of prior methods Duan et al., 2016b; Finn et al., 2017; Mishra et al., 2018, in our work the reward function remains the same across tasks while the dynamics vary. Therefore, each task constitutes a different belief about what the dynamics in the true environment could be. Finally, we pose our objective as the following meta-optimization problem:

$$\max_{\theta} \quad \frac{1}{K}\sum_{k=0}^{K} J_k(\theta'_k) \qquad \text{s.t.:} \quad \theta'_k = \theta + \alpha\, \nabla_\theta J_k(\theta) \tag{7}$$

with $J_k(\theta)$ being the expected return under the policy π_θ and the estimated dynamics model $\hat{p}_{\phi_k}$.

$$J_k(\theta) = \mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)}\left[\sum_{t=0}^{H-1} r(s_t, a_t) \middle| s_{t+1} = \hat{p}_{\phi_k}(s_t, a_t)\right] \tag{8}$$

For estimating the expectation in Eq. 8 and computing the corresponding gradients, we sample trajectories from the imagined MDPs. The rewards are computed by evaluating the reward function, which we assume as given, in the predicted states and actions $r(\hat{p}_{\phi_k}(s_{t-1}, a_{t-1}), a_t)$. In particular, when estimating the adaptation objectives $J_k(\theta)$, the meta-policy π_θ is used to sample a set of imaginary trajectories $\mathcal{T}_k$ for each model $\hat{p}_{\phi_k}$. For the meta-objective $\frac{1}{K}\sum_{k=0}^{K} J_k(\theta'_k)$, we generate trajectory roll-outs $\mathcal{T}'_k$ with the models $\hat{p}_{\phi_k}$ and the policies $\pi_{\theta'_k}$ obtained from adapting the parameters θ to the k-th model. Thus, no real-world data is used for the data intensive step of meta-policy optimization.

In practice, any policy gradient algorithm can be chosen to perform the meta-update of the policy parameters. In our implementation, we use Trust-Region Policy Optimization (TPRO) (Schulman et al., 2015b) for maximizing the meta-objective, and employ vanilla policy gradient (VPG) Peters and Schaal, 2006 for the adaptation step. To reduce the variance of the policy gradient estimates a linear reward baseline is used.

5.2.3 *Algorithm*

In the following, we describe the overall algorithm of our approach (see Algorithm 3). First, we initialize the models and the policy with different random weights. Then, we proceed to the data collection step. In the first iteration, a uniform random controller

Algorithm 3 MB-MPO

```
Require: Inner and outer step size $\alpha$, $\beta$
 1: Initialize the policy $\pi_\theta$, the models $\hat{p}_{\phi_1}, \hat{p}_{\phi_2}, ..., \hat{p}_{\phi_K}$ and $\mathcal{D} \leftarrow \emptyset$
 2: repeat
 3:     Sample trajectories from the real environment with the adapted policies
        $\pi_{\theta'_1}, ..., \pi_{\theta'_K}$. Add them to $\mathcal{D}$.
 4:     Train all models using $\mathcal{D}$.
 5:     for all models $\hat{p}_{\phi_k}$ do
 6:         Sample imaginary trajectories $\mathcal{T}_k$ from $\hat{p}_{\phi_k}$ using $\pi_\theta$
 7:         Compute adapted parameters $\theta'_k = \theta + \alpha \nabla_\theta J_k(\theta)$ using trajectories $\mathcal{T}_k$
 8:         Sample imaginary trajectories $\mathcal{T}'_k$ from $\hat{p}_{\phi_k}$ using the adapted policy $\pi_{\theta'_k}$
 9:     end for
10:     Update $\theta \to \theta - \beta \frac{1}{K} \sum_k \nabla_\theta J_k(\theta'_k)$ using the trajectories $\mathcal{T}'_k$
11: until the policy performs well in the real environment
12: return Optimal pre-update parameters $\theta^*$
```

is used to collect data from the real-world, which is stored in a buffer $\mathcal{D}$. At subsequent iterations, trajectories from the real-world are collected with the adapted policies $\{\pi_{\theta'_1}, ..., \pi_{\theta'_K}\}$, and then aggregated with the trajectories from previous iterations. The models are trained with the aggregated real-environment samples following the procedure explained in section 5.2.1. The algorithm proceeds by imagining trajectories from each the ensemble of models $\{\hat{p}_{\phi_1}, ..., \hat{p}_{\phi_k}\}$ using the policy π_θ. These trajectories are are used to perform the inner adaptation policy gradient step, yielding the adapted policies $\{\pi_{\theta'_1}, ..., \pi_{\theta'_K}\}$. Finally, we generate imaginary trajectories using the adapted policies $\pi_{\theta'_k}$ and models $\hat{p}_{\phi_k}$, and optimize the policy towards the meta-objective (as explained in section 5.2.2). We iterate through these steps until desired performance is reached. The algorithm returns the optimal pre-update parameters θ^*.

5.3 BENEFITS OF THE ALGORITHM

Meta-learning a policy over an ensemble of dynamic models using imaginary trajectory roll-outs provides several benefits over traditional model-based and model-based model-free approaches. In the following we discuss several such advantages, aiming to provide intuition for the algorithm.

Regularization effect during training. Optimizing the policy to adapt within one policy gradient step to any of the fitted models imposes a regularizing effect on the policy learning (as Nichol et al., 2018 observed in the supervised learning case). The meta-optimization problem steers the policy towards higher plasticity in regions with high dynamics model uncertainty, shifting the burden of adapting to model discrepancies towards the inner policy gradient update.

We consider plasticity as the policy's ability to change its (conditional) distribution with a small change (i.e. gradient update) in the parameter space. The policy plasticity is manifested in the statistical distance between the pre- and post-update policy. In section 5.5.4 we analyze the connection between model uncertainty and the policy plasticity, finding a strong positive correlation between the model ensembles predictive variance and the KL-divergence between π_θ and $\pi_{\theta'_k}$. This effect prevents the policy to learn sub-optimal behaviors that arise in robust policy optimization. More importantly, this regularization effect fades away once the dynamics models get more accurate, which leads to asymptotic optimal policies if enough data is provided to the learned models. In section 5.5.3, we show how this property allows us to learn from noisy and highly biased models.

Tailored data collection for fast model improvement. Since we sample real-environment trajectories using the different policies $\{\pi_{\theta'_1}, ..., \pi_{\theta'_K}\}$ obtained by adaptation to each model, the collected training data is more diverse which promotes robustness of the dynamic models. Specifically, the adapted policies tend to exploit the characteristic deficiencies of the respective dynamic models. As a result, we collect real-world data in regions where the dynamic models insufficiently approximate the true dynamics. This effect accelerates correcting the imprecision of the models leading to faster improvement. In Appendix B.1, we experimentally show the positive effect of tailored data collection on the performance.

Fast fine-tuning. Meta-learning optimizes a policy for fast adaptation Finn et al., 2017 to a set of tasks. In our case, each task corresponds to a different believe of what the real environment dynamics might be. When optimal performance is not achieved, the ensemble of models will present high discrepancy in their predictions increasing the likelihood of the real dynamics to lie in the believe distribution's support. As a result, the learned policy is likely to exhibit high adaptability towards the real environment, and fine-tuning the policy with VPG on the real environment leads to faster convergence than training the policy from scratch or from any other MB initialization.

Simplicity. Our approach, contrary to previous methods, is simple: it does not rely on parameter noise exploration, careful reinitialization of the model weights or policy's entropy, hard to train probabilistic models, and it does not need to address the model distribution mismatch (Chua et al., 2018; Kurutach et al., 2018; Feinberg et al., 2018a).

5.4 RELATED WORK

In this section, we discuss related work in addressing model inaccuracies in model-based RL as well as recent advances in the field of meta-learning.

Model-Based Reinforcement Learning: Addressing Model Inaccuracies. Impressive results with model-based RL have been obtained using simple linear models (Bagnell and Schneider, 2001; Abbeel et al., 2006; Levine and Abbeel, 2014; Levine et al., 2016a). However, like Bayesian models (Deisenroth and Rasmussen, 2011; Nguyen-Tuong et al., 2009; Kamthe and Deisenroth, 2017), their application is limited to low-dimensional domains. Our approach, which uses neural networks (NNs), is easily able to scale to complex high dimensional control problems. NNs for model learning offer the potential to scale to higher dimensional problems with impressive sample complexity (Nagabandi et al., 2017; Chua et al., 2018; Punjani and Abbeel, 2015; Wahlström et al., 2015). A major challenge when using high-capacity dynamics models is preventing policies from exploiting model inaccuracies. Several works approach this problem of model-bias by learning a distribution of models (Depeweg et al., 2017a; Rajeswaran et al., 2016; Kurutach et al., 2018; Chua et al., 2018), or by learning adaptive models (Nagabandi et al., 2018a; Fu et al., 2016; Gu et al., 2016b). We incorporate the idea of reducing model-bias by learning an ensemble of models. However, we show that these techniques do not suffice in challenging domains, and demonstrate the necessity of meta-learning for improving asymptotic performance.

Past work has also tried to overcome model inaccuracies through the policy optimization process. Model Predictive Control (MPC) compensates for model imperfections by re-planning at each step (Lenz et al., 2015), but it suffers from limited credit assignment and high computational cost. Robust policy optimization (Rajeswaran et al., 2016; Zhou et al., 1996; Lim et al., 2013) looks for a policy that performs well across models; as a result policies tend to be over-conservative. In contrast, we show that MB-MPO learns a robust policy in the regions where the models agree, and an adaptive one where the models yield substantially different predictions.

Meta-Learning. Our approach makes use of meta-learning to address model inaccuracies. Meta-learning algorithms aim to learn models that can adapt to new scenarios or tasks with few data points. Current meta-learning algorithms can be classified in three categories. One approach involves training a recurrent or memory-augmented network that ingests a training dataset and outputs the parameters of a learner model (Schmidhuber, 1987; Andrychowicz et al., 2016). Another set of methods feeds the dataset followed by the test data into a recurrent model that outputs the predictions for the test inputs (Duan et al., 2016b; Santoro et al., 2016). The last category embeds the structure of optimization problems into the meta-learning algorithm (Finn et al., 2017; Hüsken and Goerick, 2000; Ravi and Larochelle, 2018). These algorithms have been extended to the context of RL (Duan et al., 2016b; J. Wang et al., 2016; Sung et al., 2017; Finn et al., 2017). Our work builds upon MAML Finn et al., 2017. However, while in previous meta-learning methods each task is typically defined by a different reward function, each of our tasks is defined by the dynamics of different learned models.

5.5 EXPERIMENTS

The aim of our experimental evaluation is to examine the following questions: 1) How does MB-MPO compare against state-of-the-art model-free and model-based methods in terms of sample complexity and asymptotic performance? 2) How does the model uncertainty influence the policy's plasticity? 3) How robust is our method against imperfect models?

To answer the posed questions, we evaluate our approach on six continuous control benchmark tasks in the Mujoco simulator Todorov et al., 2012. A depiction of the environments as well a detailed description of the experimental setup can be found in Appendix B.3. In all of the following experiments, the pre-update policy is used to report the average returns obtained with our method. The performance reported are averages over at least three random seeds. The source code and the experiments data is available on our supplementary website [1].

5.5.1 *Comparison to State-of-the-Art: Model-Free*

We compare our method in sample complexity and performance to four state-of-the-art model free RL algorithms: Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015), Trust Region Policy Optimization (Schulman et al., 2015b), Proximal Policy Optimization (PPO) (Schulman et al., 2017), and Actor Critic using Kronecker-Factored Trust Region (ACKTR) (Wu et al., 2017). The results are shown in Figure 7.

In all the locomotion tasks we are able to achieve maximum performance using between 10 and 100 times less data than model-free methods. In the most challenging domains: ant, hopper, and walker2D; the data complexity of our method is two orders of magnitude less than the MF. In the easier tasks: the simulated PR2 and swimmer, our method achieves the same performance of MF using 20-50$\times$ less data. These results

1 https://sites.google.com/view/mb-mpo

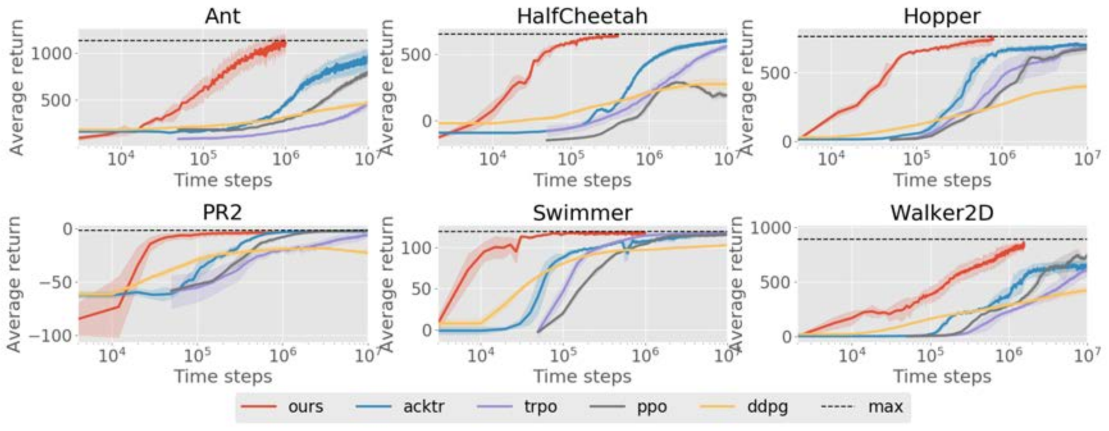


Figure 7: Learning curves of MB-MPO ("ours") and four state-of-the-art model-free methods in six different Mujoco environments with a horizon of 200. MB-MPO is able to match the asymptotic performance of model-free methods with two orders of magnitude less samples.

highlight the benefit of MB-MPO for real robotics tasks; the amount of real-world data needed for attaining maximum return corresponds to 2 hours in the case of easier domains and to 7 hours in the more complex ones.

5.5.2 *Comparison to State-of-the-Art: Model-Based*

We also compare our method against recent model-based work: Model-Ensemble Trust-Region Policy Optimization (ME-TRPO) Kurutach et al., 2018, and the model-based approach introduced in (Nagabandi et al., 2017), which uses MPC for planning (MB-MPC).

The results, shown in Figure 8, highlight the strength of MB-MPO in complex tasks. MB-MPC struggles to perform well on tasks that require robust planning, and completely fails in tasks where medium/long-term planning is necessary (as in the case of hopper). In contrast, ME-TRPO is able to learn better policies, but the convergence to such policies is slower when compared to MB-MPO. Furthermore, while ME-TRPO converges to suboptimal policies in complex domains, MB-MPO is able to achieve max-performance.

5.5.3 *Robustness to Imperfect Dynamic Models and Compounding Errors*

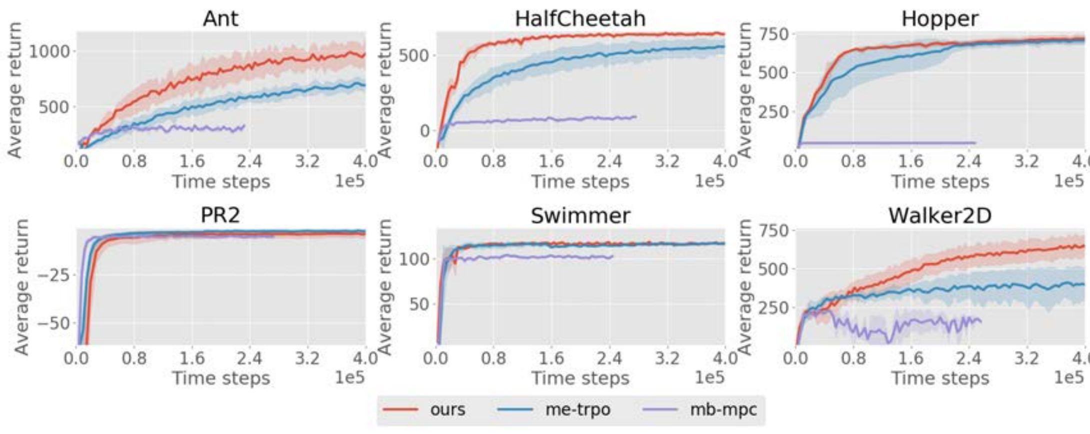


Figure 8: Learning curves of MB-MPO ("ours") and two MB methods in 6 different Mujoco environments with a horizon of 200. MB-MPO achieves better asymptotic performance and faster convergence rate than previous MB methods.

We pose the question of how robust our proposed algorithm is w.r.t. imperfect dynamics predictions. We examine it in two ways. First, with an illustrative example of a model with clearly wrong dynamics. Specifically, we add biased Gaussian noise $\mathcal{N}(b, 0.1^2)$ to the next state prediction, whereby the bias $b \sim \mathcal{U}(0, b_{\max})$ is re-sampled in every iteration for each model. Second, we present a realistic case on which long horizon predictions are needed. Bootstrapping the model predictions for long horizons leads to high compounding errors, making policy learning on such predictions challenging.

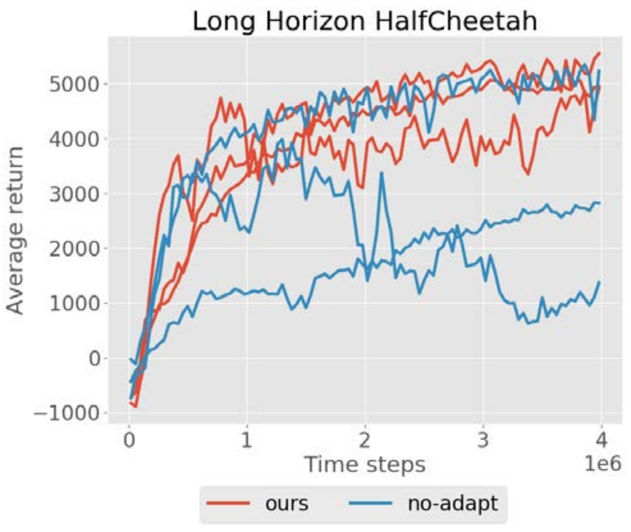


Figure 9: Comparison of our method with and without adaptation. Learning curves of three random seeds on the half-cheetah environment with a horizon of 1000 time steps.

Figure 10 depicts the performance comparison between our method and ME-TRPO on the half-cheetah environment for various values of $b_{\max}$. Results indicate that our method consistently outperforms

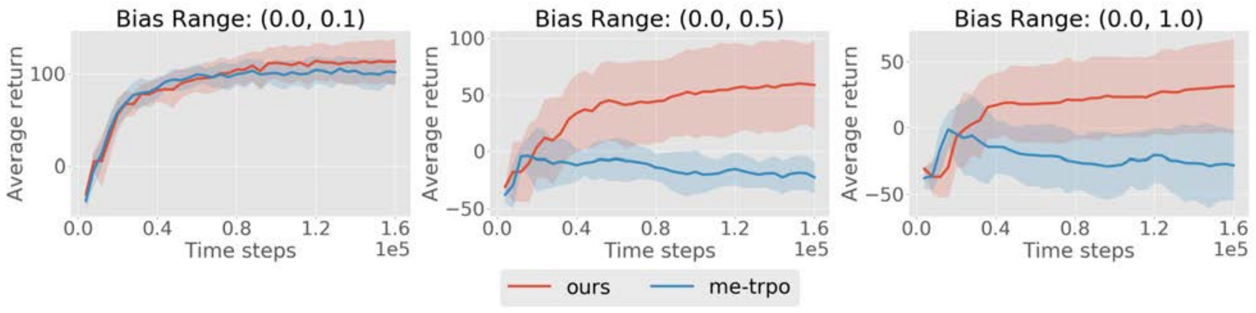


Figure 10: Comparison of MB-MPO and ME-TRPO using 5 biased and noisy dynamic models in the half-cheetah environment with a horizon of 100 time steps. A bias term b is sampled uniformly from a denoted interval in every iteration. During the iterations we add to the predicted observation a Gaussian noise $\mathcal{N}(b, 0.1)$.

ME-TRPO when exposed to biased and noisy dynamics models. ME-TPRO catastrophically fails to learn a policy in the presence of strong bias (i.e. $b_{max} = 0.5$ and $b_{max} = 1.0$), but our method, despite the strongly compromised dynamic predictions, is still able to learn a locomotion behavior with a positive forward velocity.

This property also manifests itself in long horizon tasks. Figure 9 compares the performance of our approach with inner learning rate $\alpha = 10^{-3}$ against the edge case $\alpha = 0$, where no adaption is taking place. For each random seed, MB-MPO steadily converges to maximum performance. However, when there is no adaptation, the learning becomes unstable and different seeds exhibit different behavior: proper learning, getting stuck in sub-optimal behavior, and even unlearning good behaviors.

5.5.4 *Model Uncertainty and Policy Plasticity*

In section 5.5.4 we hypothesize that the meta-optimization steers the policy towards higher plasticity in regions with high dynamics model uncertainty while embedding consistent model predictions into the pre-update policy. To empirically analyze this hypothesis, we conduct an experiment in a simple 2D-Point environment where the agent, starting uniformly from $[-2, 2]^2$, must go to the goal position $(0, 0)$. We use the average KL-divergence between π_θ and the different adapted policies $\pi_{\theta'}k$ to measure the plasticity conditioned on the state $\mathbf{s}$.

Figure 11 depicts the KL-divergence between the pre- and post-update policy, as well as the standard deviation of the predictions of the ensemble over the state space. Since the agent steers towards the center of the environment, more transition data is available in this region. As a result the models present higher accuracy in the center. The results indicate a strong positive correlation between model uncertainty and the KL-divergence between pre- and post-update policy. We find this connection between policy plasticity and predictive uncertainty consistently throughout the training and among different hyper-parameter configurations.

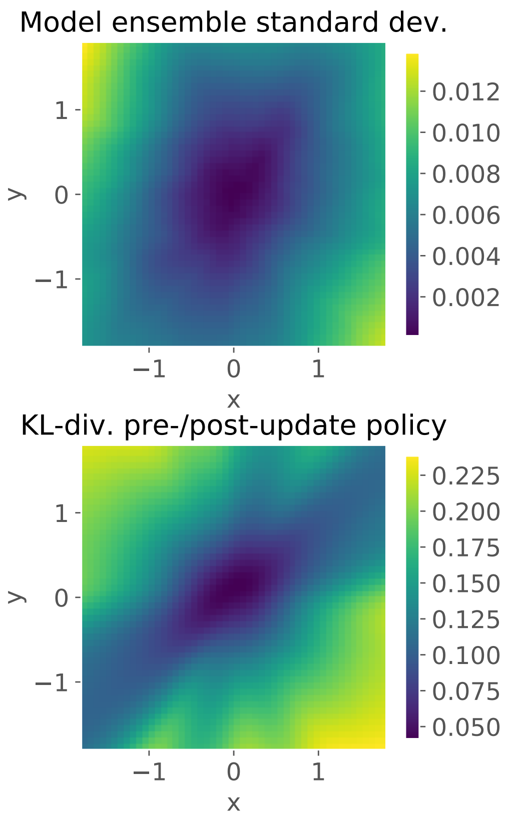


Figure 11: Upper: Standard deviation of model ensemble predictions. Lower: KL-divergence between pre- and post-update policy. The x and y axis denote the state-space dimensions of the 2-D Point environment

5.6 CONCLUSION

In this work, we present a simple and generally applicable algorithm, model-based meta-policy optimization (MB-MPO), that learns an ensemble of dynamics models and meta-optimizes a policy for adaptation in each of the learned models. Foregoing the reliance in accurate models, MB-MPO harness model errors to learn more adaptable policies that can successfully adapt to the real environment. Our experimental results demonstrate that meta-learning a policy over an ensemble of learned models provides the recipe for reaching the same level of performance as state-of-the-art model-free methods with substantially lower sample complexity. We also compare our method against previous model-based approaches, obtaining better performance and faster convergence. Our analysis demonstrate the ineffectiveness of prior approaches to combat model-bias, and showcases the robustness of our method against imperfect models. As a result, we are able to extend model-based to more complex domains and longer horizons.

6 PROMP: PROXIMAL META-POLICY SEARCH

In this chapter, we delve into the meta-learning procedure that model-based meta-policy optimization (MB-MPO) uses. Specifically, we analyze the update rule by which adaptation occurs. We point out that in the reinforcement learning setting, the commonly used gradient based meta-learning from (Finn et al., 2017) is a first order approximation. The approximation neglects the exploration strategy that the pre-udpate policy should follow so the adapted policy is optimal. Instead, (Finn et al., 2017) pre-update policy's just tries to maximize the reward. However, purely maximizing the reward might not result in any information about the environment at hand, thus failing to adapt to it.

In the following work, we analyze the different objectives used in gradient based meta-learning and present an objective that allows to learn the optimal exploration strategy for adaptation. Furthermore, we incorporate our objective into an efficient and stable meta-learning algorithm named Proximal Meta-Policy Search (ProMP).

6.1 OVERVIEW

A remarkable trait of human intelligence is the ability to adapt to new situations in the face of limited experience. In contrast, our most successful artificial agents struggle in such scenarios. While achieving impressive results, they suffer from high sample complexity in learning even a single task, fail to generalize to new situations, and require large amounts of additional data to successfully adapt to new environments. Meta-learning addresses these shortcomings by learning how to learn. Its objective is to learn an algorithm that allows the artificial agent to succeed in an unseen task when only limited experience is available, aiming to achieve the same fast adaptation that humans possess (Schmidhuber, 1987; Thrun and Pratt, 1998).

Despite recent progress, deep reinforcement learning (RL) still relies heavily on hand-crafted features and reward functions as well as engineered problem specific inductive bias. meta-RL aims to forego such reliance by acquiring inductive bias in a data-driven manner. Recent work proves this approach to be promising, demonstrating that meta-RL allows agents to obtain a diverse set of skills, attain better exploration strategies, and learn faster through meta-learned dynamics models or synthetic returns (Duan et al., 2016b; Z. Xu et al., 2018; Gupta et al., 2018b; Sæmundsson et al., 2018).

meta-RL is a multi-stage process in which the agent, after a few sampled environment interactions, adapts its behavior to the given task. Despite its wide utilization, little work has been done to promote theoretical understanding of this process, leaving meta-RL grounded on unstable foundations. Although the behavior prior to the adaptation step is instrumental for task identification, the interplay between pre-adaptation sampling and posterior performance of the policy remains poorly understood. In fact, prior work in gradient-based meta-RL has either entirely neglected credit assignment to the pre-update distribution (Finn et al., 2017) or implemented such credit assignment in a naive way (Al-Shedivat et al., 2018; Stadie et al., 2018).

To our knowledge, we provide the first formal in-depth analysis of credit assignment w.r.t. pre-adaptation sampling distribution in meta-RL. Based on our findings, we develop a novel meta-RL algorithm. First, we analyze two distinct methods for assigning credit to pre-adaptation behavior. We show that the recent formulation introduced by (Al-Shedivat et al., 2018) and (Stadie et al., 2018) leads to poor credit assignment, while the MAML formulation (Finn et al., 2017) potentially yields superior meta-policy updates. Second, based on insights from our formal analysis, we highlight both the importance and difficulty of proper meta-policy gradient estimates. In light of this, we propose the low variance curvature (LVC) surrogate objective which yields gradient estimates with a favorable bias-variance trade-off. Finally, building upon the LVC estimator we develop Proximal Meta-Policy Search (ProMP), an efficient and stable meta-learning algorithm for RL. In our experiments, we show that ProMP consistently outperforms previous meta-RL algorithms in sample-efficiency, wall-clock time, and asymptotic performance.

6.2 BACKGROUND

Meta-Reinforcement Learning aims to learn a learning algorithm which is able to quickly learn the optimal policy for a task $\mathcal{M}$ drawn from a distribution of tasks $\rho(\mathcal{M})$. Each task $\mathcal{M}$ corresponds to a different MDP. Typically, it is assumed that the distribution of tasks share the action and state space, but may differ in their reward function or their dynamics.

Gradient-based meta-learning aims to solve this problem by learning the parameters

θ of a policy π_θ such that performing a single or few steps of vanilla policy gradient (VPG) with the given task leads to the optimal policy for that task. This meta-learning formulation, also known under the name of MAML, was first introduced by (Finn et al., 2017). We refer to it as formulation I which can be expressed as maximizing the objective

$$J^{I}(\theta) = \mathbb{E}_{\mathcal{M}\sim\rho(\mathcal{M})}\left[\mathbb{E}_{\boldsymbol{\tau}'\sim p_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')}\left[R(\boldsymbol{\tau}')\right]\right] \quad \text{with} \quad \theta' := U(\theta, \mathcal{M}) = \theta + \alpha\nabla_\theta \mathbb{E}_{\boldsymbol{\tau}\sim p_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\left[R(\boldsymbol{\tau})\right]$$

In that U denotes the update function which depends on the task $\mathcal{M}$, and performs one VPG step towards maximizing the performance of the policy in $\mathcal{M}$. For national brevity and conciseness we assume a single policy gradient adaptation step. Nonetheless, all presented concepts can easily be extended to multiple adaptation steps.

Later work proposes a slightly different notion of gradient-based Meta-RL, also known as E-MAML, that attempts to circumvent issues with the meta-gradient estimation in MAML (Al-Shedivat et al., 2018; Stadie et al., 2018):

$$J^{II}(\theta) = \mathbb{E}_{\mathcal{M}\sim\rho(\mathcal{M})}\left[\mathbb{E}_{\substack{\boldsymbol{\tau}^{1:N}\sim p_{\mathcal{M}}(\boldsymbol{\tau}^{1:N}|\theta) \\ \boldsymbol{\tau}'\sim p_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')}}\left[R(\boldsymbol{\tau}')\right]\right] \quad \text{with} \quad \theta' := U(\theta, \boldsymbol{\tau}^{1:N}) = \theta + \alpha\nabla_\theta \sum_{n=1}^{N}\left[R(\boldsymbol{\tau}^{(n)})\right]$$

Formulation II views U as a deterministic function that depends on N sampled trajectories from a specific task. In contrast to formulation I, the expectation over pre-update trajectories $\boldsymbol{\tau}$ is applied outside of the update function. Throughout this paper we refer to π_θ as pre-update policy, and $\pi_{\theta'}$ as post-update policy.

6.3 SAMPLING DISTRIBUTION CREDIT ASSIGNMENT

This section analyzes the two gradient-based meta-RL formulations introduced in Section 2.3.1. Figure 12 illustrates the stochastic computation graphs (Schulman et al., 2015a) of both formulations. The red arrows depict how credit assignment w.r.t the pre-update sampling distribution $p_{\mathcal{M}}(\boldsymbol{\tau}|\theta)$ is propagated. Formulation I (left) propagates the credit assignment through the update step, thereby exploiting the full problem structure. In contrast, formulation II (right) neglects the inherent structure, directly assigning credit from post-update return R' to the pre-update policy π_θ which leads to noisier, less effective credit assignment.

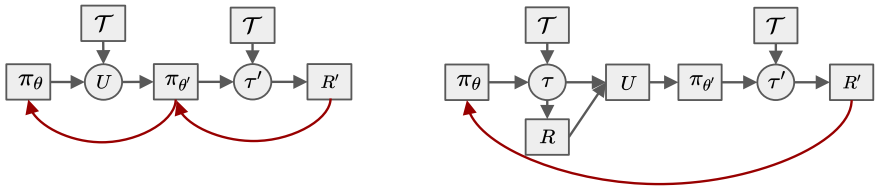


Figure 12: Stochastic computation graphs of meta-learning formulation I (left) and formulation II (right). The red arrows illustrate the credit assignment from the post-update returns R' to the pre-update policy π_θ through $\nabla_\theta J_{\text{pre}}$. (Deterministic nodes: Square; Stochastic nodes: Circle)

Both formulations optimize for the same objective, and are equivalent at the 0^{th} order. However, because of the difference in their formulation and stochastic computation graph, their gradients and the resulting optimization step differs. In the following, we shed light on how and where formulation II loses signal by analyzing the gradients of both formulations, which can be written as (see Appendix C.1 for more details and derivations)

$$\nabla_\theta J(\theta) = \mathbb{E}_{\mathcal{M}\sim\rho(\mathcal{M})}\left[\mathbb{E}_{\substack{\boldsymbol{\tau}\sim p_{\mathcal{M}}(\boldsymbol{\tau}|\theta)\\ \boldsymbol{\tau}'\sim p_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')}}\left[\nabla_\theta J_{\text{post}}(\boldsymbol{\tau},\boldsymbol{\tau}') + \nabla_\theta J_{\text{pre}}(\boldsymbol{\tau},\boldsymbol{\tau}')\right]\right] \tag{9}$$

The first term $\nabla_\theta J_{\text{post}}(\boldsymbol{\tau},\boldsymbol{\tau}')$ is equal in both formulations, but the second term, $\nabla_\theta J_{\text{pre}}(\boldsymbol{\tau},\boldsymbol{\tau}')$, differs between them. In particular, they correspond to

$$\nabla_\theta J_{\text{post}}(\boldsymbol{\tau},\boldsymbol{\tau}') = \underbrace{\nabla_{\theta'}\log\pi_\theta(\boldsymbol{\tau}')R(\boldsymbol{\tau}')}_{\nabla_{\theta'}J^{\text{outer}}}\underbrace{\left(I + \alpha R(\boldsymbol{\tau})\nabla_\theta^2\log\pi_{\theta'}(\boldsymbol{\tau})\right)}_{\text{transformation from } \theta' \text{ to } \theta} \tag{10}$$

$$\nabla_\theta J^{\text{II}}_{\text{pre}}(\boldsymbol{\tau},\boldsymbol{\tau}') = \alpha\nabla_\theta\log\pi_\theta(\boldsymbol{\tau})R(\boldsymbol{\tau}') \tag{11}$$

$$\nabla_\theta J^{\text{I}}_{\text{pre}}(\boldsymbol{\tau},\boldsymbol{\tau}') = \alpha\nabla_\theta\log\pi_\theta(\boldsymbol{\tau})\Big(\underbrace{(\nabla_\theta\log\pi_\theta(\boldsymbol{\tau})R(\boldsymbol{\tau}))^\top}_{\nabla_\theta J^{\text{inner}}}\underbrace{(\nabla_{\theta'}\log\pi_{\theta'}(\boldsymbol{\tau}')R(\boldsymbol{\tau}'))}_{\nabla_{\theta'}J^{\text{outer}}}\Big) \tag{12}$$

$\nabla_\theta J_{\text{post}}(\boldsymbol{\tau},\boldsymbol{\tau}')$ simply corresponds to a policy gradient step on the post-update policy $\pi_{\theta'}$ w.r.t θ', followed by a linear transformation from post- to pre-update parameters. It corresponds to increasing the likelihood of the trajectories $\boldsymbol{\tau}'$ that led to higher returns. However, this term does not optimize for the pre-update sampling distribution, i.e., which trajectories $\boldsymbol{\tau}$ led to better adaptation steps.

The credit assignment w.r.t. the pre-updated sampling distribution is carried out by the second term. In formulation II, $\nabla_\theta J^{\text{II}}_{\text{pre}}$ can be viewed as standard reinforcement learning on π_θ with $R(\boldsymbol{\tau}')$ as reward signal, treating the update function U as part of the unknown dynamics of the system. This shifts the pre-update sampling distribution to better adaptation steps.

Formulation I takes the causal dependence of $p_\mathcal{M}(\boldsymbol{\tau}'|\theta')$ on $p_\mathcal{M}(\boldsymbol{\tau}|\theta)$ into account. It does so by maximizing the inner product of pre-update and post-update policy gradients (see Eq. 12). This steers the pre-update policy towards 1) larger post-updates returns 2) larger adaptation steps $\alpha\nabla_\theta J^{\text{inner}}$, 3) better alignment of pre- and post-update policy gradients (Z. Li et al., 2017; Nichol et al., 2018). When combined, these effects directly optimize for adaptation. As a result, we expect the first meta-policy gradient formulation, J^I, to yield superior learning properties.

6.4 LOW VARIANCE CURVATURE ESTIMATOR

In the previous section we show that the formulation introduced by (Finn et al., 2017) results in superior meta-gradient updates, which should in principle lead to improved convergence properties. However, obtaining correct and low variance estimates of the respective meta-gradients proves challenging. As discussed by (Foerster et al., 2018), and shown in Appendix C.2.3, the score function surrogate objective approach is ill suited for calculating higher order derivatives via automatic differentiation toolboxes. This important fact was overlooked in the original RL-MAML implementation (Finn et al., 2017) leading to incorrect meta-gradient estimates[1]. As a result, $\nabla_\theta J_{\text{pre}}$ does not appear in the gradients of the meta-objective (i.e. $\nabla_\theta J = \nabla_\theta J_{\text{post}}$). Hence, MAML does not perform any credit assignment to pre-adaptation behavior.

But, even when properly implemented, we show that the meta-gradients exhibit high variance. Specifically, the estimation of the hessian of the RL-objective, which is inherent in the meta-gradients, requires special consideration. In this section, we motivate and introduce the low variance curvature estimator (LVC): an improved estimator for the hessian of the RL-objective which promotes better meta-policy gradient updates. As we show in Appendix C.1.1, we can write the gradient of the meta-learning objective as

$$\nabla_\theta J^I(\theta) = \mathbb{E}_{\mathcal{M}\sim\rho(\mathcal{M})}\Big[\mathbb{E}_{\boldsymbol{\tau}'\sim p_\mathcal{M}(\boldsymbol{\tau}'|\theta')}\big[\nabla_{\theta'}\log p_\mathcal{M}(\boldsymbol{\tau}'|\theta')R(\boldsymbol{\tau}')\nabla_\theta U(\theta,\mathcal{M})\big]\Big] \qquad (13)$$

Since the update function U resembles a policy gradient step, its gradient $\nabla_\theta U(\theta,\mathcal{M})$ involves computing the hessian of the reinforcement learning objective, i.e., $\nabla^2_\theta\, \mathbb{E}_{\boldsymbol{\tau}\sim p_\mathcal{M}(\boldsymbol{\tau}|\theta)}\left[R(\boldsymbol{\tau})\right]$.

1 Note that MAML is theoretically sound, but does not attend to correctly estimating the meta-policy gradients. As consequence, the gradients in the corresponding implementation do not comply with the theory.

Estimating this hessian has been discussed in (Bartlett and Baxter, 2011) and (Furmston et al., 2016). In the infinite horizon MDP case, (Bartlett and Baxter, 2011) derived a decomposition of the hessian. We extend their finding to the finite horizon case, showing that the hessian can be decomposed into three matrix terms (see Appendix C.2.2 for proof):

$$\nabla_\theta U(\theta, \mathcal{M}) = I + \alpha \nabla^2_\theta \, \mathbb{E}_{\boldsymbol{\tau} \sim p_{\mathcal{M}}(\boldsymbol{\tau}|\theta)} \left[R(\boldsymbol{\tau})\right] = I + \alpha\big(H_1 + H_2 + H_{12} + H_{12}^\top\big) \tag{14}$$

whereby

$$H_1 = \mathbb{E}_{\boldsymbol{\tau} \sim p_{\mathcal{M}}(\boldsymbol{\tau}|\theta)} \left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_t, \mathbf{s}_t) \nabla_\theta \log \pi_\theta(\mathbf{a}_t, \mathbf{s}_t)^\top \left(\sum_{t'=t}^{H-1} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) \right]$$

$$H_2 = \mathbb{E}_{\boldsymbol{\tau} \sim p_{\mathcal{M}}(\boldsymbol{\tau}|\theta)} \left[\sum_{t=0}^{H-1} \nabla^2_\theta \log \pi_\theta(\mathbf{a}_t, \mathbf{s}_t) \left(\sum_{t'=t}^{H-1} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) \right]$$

$$H_{12} = \mathbb{E}_{\boldsymbol{\tau} \sim p_{\mathcal{M}}(\boldsymbol{\tau}|\theta)} \left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_t, \mathbf{s}_t) \nabla_\theta Q_t^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t)^\top \right]$$

Here $Q_t^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{\boldsymbol{\tau}^{t+1:H-1} \sim p_{\mathcal{M}}(\cdot|\theta)} \left[\sum_{t'=t}^{H-1} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t \right]$ denotes the expected state-action value function under policy π_θ at time t.

Computing the expectation of the RL-objective is in general intractable. Typically, its gradients are computed with a Monte Carlo estimate based on the policy gradient theorem. In practical implementations, such an estimate is obtained by automatically differentiating a surrogate objective (Schulman et al., 2015a). However, this results in a highly biased hessian estimate which just computes H_2, entirely dropping the terms H_1 and $H_{12} + H_{12}^\top$. In the notation of the previous section, it leads to neglecting the $\nabla_\theta J_{\text{pre}}$ term, ignoring the influence of the pre-update sampling distribution.

The issue can be overcome using the DiCE formulation, which allows to compute unbiased higher-order Monte Carlos estimates of arbitrary stochastic computation graphs (Foerster et al., 2018). The DiCE-RL objective can be rewritten as follows

$$J^{\text{DiCE}}(\boldsymbol{\tau}) = \sum_{t=0}^{H-1} \left(\prod_{t'=0}^{t} \frac{\pi_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'})}{\perp(\pi_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'}))} \right) r(\mathbf{s}_t, \mathbf{a}_t) \quad \boldsymbol{\tau} \sim p_{\mathcal{M}}(\boldsymbol{\tau}) \tag{15}$$

$$\mathbb{E}_{\boldsymbol{\tau} \sim p_{\mathcal{M}}(\boldsymbol{\tau}|\theta)} \left[\nabla^2_\theta J^{\text{DiCE}}(\boldsymbol{\tau}) \right] = H_1 + H_2 + H_{12} + H_{12}^\top \tag{16}$$

In that, $\perp$ denotes the “stop_gradient" operator, i.e., $\perp(f_\theta(x)) \to f_\theta(x)$ but $\nabla_\theta \perp(f_\theta(x)) \to 0$. The sequential dependence of $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ within the trajectory, manifesting itself through the product of importance weights in (15), results in high variance estimates of the hessian $\nabla^2_\theta \, \mathbb{E}_{\boldsymbol{\tau} \sim p_{\mathcal{M}}(\boldsymbol{\tau}|\theta)} [R(\boldsymbol{\tau})]$. As noted by (Furmston et al., 2016), H_{12} is particularly difficult to estimate, since it involves three nested sums along the trajectory. In section 6.7.2 we empirically show that the high variance estimates of the DiCE objective lead to noisy meta-policy gradients and poor learning performance.

To facilitate a sample efficient meta-learning, we introduce the low variance curvature (LVC) estimator:

$$J^{\text{LVC}}(\boldsymbol{\tau}) = \sum_{t=0}^{H-1} \frac{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\perp(\pi_\theta(\mathbf{a}_t|\mathbf{s}_t))} \left(\sum_{t'=t}^{H-1} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) \right) \quad \boldsymbol{\tau} \sim p_{\mathcal{M}}(\boldsymbol{\tau}) \tag{17}$$

$$\mathbb{E}_{\boldsymbol{\tau} \sim p_{\mathcal{M}}(\boldsymbol{\tau}|\theta)} \left[\nabla^2_\theta J^{\text{LVC}}(\boldsymbol{\tau}) \right] = H_1 + H_2 \tag{18}$$

By removing the sequential dependence of $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ within trajectories, the hessian estimate neglects the term $H_{12} + H_{12}^\top$ which leads to a variance reduction, but makes the estimate biased. The choice of this objective function is motivated by findings in (Furmston et al., 2016): under certain conditions the term $H_{12} + H_{12}^\top$ vanishes around local optima θ^*, i.e., $\mathbb{E}_{\boldsymbol{\tau}}[\nabla^2_\theta J^{\text{LVC}}] \to \mathbb{E}_{\boldsymbol{\tau}}[\nabla^2_\theta J^{\text{DiCE}}]$ as $\theta \to \theta^*$. Hence, the bias of the LVC estimator becomes negligible close to local optima. The experiments in section 6.7.2 underpin the theoretical findings, showing that the low variance hessian estimates obtained through J^{LVC} improve the sample-efficiency of meta-learning by a significant margin when compared to J^{DiCE}. We refer the interested reader to Appendix C.2 for derivations and a more detailed discussion.

6.5 METHOD

Building on the previous sections, we develop a novel meta-policy search method based on the low variance curvature objective which aims to solve the following optimization problem:

$$\max_\theta \quad \mathbb{E}_{\mathcal{M} \sim \rho(\mathcal{M})} \left[\mathbb{E}_{\boldsymbol{\tau}' \sim p_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')} \left[R(\boldsymbol{\tau}') \right] \right] \quad \text{with} \quad \theta' := \theta + \alpha \, \nabla_\theta \mathbb{E}_{\boldsymbol{\tau} \sim p_{\mathcal{M}}(\boldsymbol{\tau}|\theta)} \left[J^{\text{LVC}}(\boldsymbol{\tau}) \right] \tag{19}$$

Prior work has optimized this objective using either vanilla policy gradient (VPG) or TRPO (Schulman et al., 2015b). TRPO holds the promise to be more data efficient and stable during the learning process when compared to VPG. However, it requires computing the Fisher information matrix (FIM). Estimating the FIM is particularly problematic in the meta-learning set up. The meta-policy gradients already involve second order derivatives; as a result, the time complexity of the FIM estimate is cubic in the number of policy parameters. Typically, the problem is circumvented using finite difference methods, which introduce further approximation errors.

The recently introduced PPO algorithm (Schulman et al., 2017) achieves comparable results to TRPO with the advantage of being a first order method. PPO uses a surrogate clipping objective which allows it to safely take multiple gradient steps without re-sampling trajectories.

$$J_{\mathcal{M}}^{\text{CLIP}}(\theta) = \mathbb{E}_{\boldsymbol{\tau} \sim p_{\mathcal{M}}(\boldsymbol{\tau}, \theta_o)} \left[\sum_{t=0}^{H-1} \min \left(\frac{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\theta_o}(\mathbf{a}_t|\mathbf{s}_t)} A^{\pi_{\theta_o}}(\mathbf{s}_t, \mathbf{a}_t) \ , \ \text{clip}_{1-\epsilon}^{1+\epsilon} \left(\frac{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\theta_o}(\mathbf{a}_t|\mathbf{s}_t)} \right) A^{\pi_{\theta_o}}(\mathbf{s}_t, \mathbf{a}_t) \right) \right]$$

In case of meta-RL, it does not suffice to just replace the post-update reward objective with $J_{\mathcal{M}}^{\text{CLIP}}$. In order to safely perform multiple meta-gradient steps based on the same sampled data from a recent policy π_{θ_o}, we also need to 1) account for changes in the pre-update action distribution $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$, and 2) bound changes in the pre-update state visitation distribution (Kakade and Langford, 2002).

We propose Proximal Meta-Policy Search (ProMP) which incorporates both the benefits of proximal policy optimization and the low variance curvature objective (see Alg. 4.) In order to comply with requirement 1), ProMP replaces the "stop gradient" importance weight $\frac{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\perp(\pi_\theta(\mathbf{a}_t|\mathbf{s}_t))}$ by the likelihood ratio $\frac{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\theta_o}(\mathbf{a}_t|\mathbf{s}_t))}$, which results in the following objective

$$J_{\mathcal{M}}^{LR}(\theta) = \mathbb{E}_{\boldsymbol{\tau} \sim p_{\mathcal{M}}(\boldsymbol{\tau}, \theta_o)} \left[\sum_{t=0}^{H-1} \frac{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\theta_o}(\mathbf{a}_t|\mathbf{s}_t)} A^{\pi_{\theta_o}}(\mathbf{s}_t, \mathbf{a}_t) \right] \tag{20}$$

An important feature of this objective is that its derivatives w.r.t θ evaluated at θ_o are identical to those of the LVC objective, and it additionally accounts for changes in the pre-update action distribution. To satisfy condition 2) we extend the clipped meta-objective with a KL-penalty term between π_θ and π_{θ_o}. This KL-penalty term enforces a soft local "trust region" around π_{θ_o}, preventing the shift in state visitation distribution to become large during optimization. This enables us to take multiple meta-policy gradient steps without re-sampling. Altogether, ProMP optimizes

$$J_{\mathcal{M}}^{\text{ProMP}}(\theta) = J_{\mathcal{M}}^{\text{CLIP}}(\theta') - \eta \bar{\mathcal{D}}_{KL}(\pi_{\theta_o}, \pi_\theta) \quad \text{s.t.} \quad \theta' = \theta + \alpha \, \nabla_\theta J_{\mathcal{M}}^{LR}(\theta) \ , \quad \mathcal{M} \sim \rho(\mathcal{M}) \tag{21}$$

ProMP consolidates the insights developed throughout the course of this paper, while at the same time making maximal use of recently developed policy gradients algorithms. First, its meta-learning formulation exploits the full structural knowledge of gradient-based meta-learning. Second, it incorporates a low variance estimate of the RL-objective

Algorithm 4 Proximal Meta-Policy Search (ProMP)

```
Require: Task distribution $\rho$, step sizes $\alpha$, $\beta$, KL-penalty coefficient $\eta$, clipping range $\epsilon$
 1: Randomly initialize $\theta$
 2: while $\theta$ not converged do
 3:    Sample batch of tasks $\mathcal{M}_i \sim \rho(\mathcal{M})$
 4:    for step $n = 0, ..., N-1$ do
 5:       if $n = 0$ then
 6:          Set $\theta_o \leftarrow \theta$
 7:          for all $\mathcal{M}_i \sim \rho(\mathcal{M})$ do
 8:             Sample pre-update trajectories $\mathcal{D}_i = \{\boldsymbol{\tau}_i\}$ from $\mathcal{M}_i$ using $\pi_\theta$
 9:             Compute adapted parameters $\theta'_{o,i} \leftarrow \theta + \alpha \nabla_\theta J^{LR}_{\mathcal{M}_i}(\theta)$ with $\mathcal{D}_i = \{\boldsymbol{\tau}_i\}$
10:             Sample post-update trajectories $\mathcal{D}'_i = \{\boldsymbol{\tau}'_i\}$ from $\mathcal{M}_i$ using $\pi_{\theta'_{o,i}}$
11:          end for
12:       end if
13:       Update $\theta \leftarrow \theta + \beta \sum_{\mathcal{M}_i} \nabla_\theta J^{ProMP}_{\mathcal{M}_i}(\theta)$ using each $\mathcal{D}'_i = \{\boldsymbol{\tau}'_i\}$
14:    end for
15: end while
```

hessian. Third, ProMP controls the statistical distance of both pre- and post-adaptation policies, promoting efficient and stable meta-learning. All in all, ProMP consistently outperforms previous gradient-based meta-RL algorithms in sample complexity, wall clock time, and asymptotic performance (see Section 6.7.1).

6.6 RELATED WORK

Meta-Learning concerns the question of “learning to learn", aiming to acquire inductive bias in a data driven manner, so that the learning process in face of unseen data or new problem settings is accelerated (Schmidhuber, 1987; Schmidhuber et al., 1997; Thrun and Pratt, 1998).

This can be achieved in various ways. One category of methods attempts to learn the “learning program" of an universal Turing machine in form of a recurrent / memory-augmented model that ingests datasets and either outputs the parameters of the trained model (Hochreiter et al., 2001; Andrychowicz et al., 2016; Chen et al., 2017; Ravi and Larochelle, 2018) or directly outputs predictions for given test inputs (Duan et al., 2016b; Santoro et al., 2016; Mishra et al., 2018). Though very flexible and capable of learning very efficient adaptations, such methods lack performance guarantees and are difficult to train on long sequences that arise in meta-RL.

Another set of methods embeds the structure of a classical learning algorithm in the meta-learning procedure, and optimizes the parameters of the embedded learner during the meta-training (Hüsken and Goerick, 2000; Finn et al., 2017; Nichol et al., 2018; Miconi et al., 2018). A particular instance of the latter that has proven to be particularly successful in the context of RL is gradient-based meta-learning (Finn et al., 2017; Al-Shedivat et al., 2018; Stadie et al., 2018). Its objective is to learn an initialization such that after one or few steps of policy gradients the agent attains full performance on a new task. A desirable property of this approach is that even if fast adaptation fails, the agent just falls back on vanilla policy-gradients. However, as we show, previous gradient-based meta-RL methods either neglect or perform poor credit assignment w.r.t. the pre-update sampling distribution.

A diverse set of methods building on meta-RL, has recently been introduced. This includes: learning exploration strategies (Gupta et al., 2018b), synthetic rewards (Sung et al., 2017; Z. Xu et al., 2018), unsupervised policy acquisition (Gupta et al., 2018a), model-based RL (Clavera et al., 2018; Saemundsson et al., 2018), learning in competitive environments (Al-Shedivat et al., 2018) and meta-learning modular policies (Frans et al., 2018; Alet et al., 2018). Many of the mentioned approaches build on previous gradient-based meta-learning methods that insufficiently account for the pre-update distribution. ProMP overcomes these deficiencies, providing the necessary framework for novel applications of meta-RL in unsolved problems.

6.7 EXPERIMENTS

In order to empirically validate the theoretical arguments outlined above, this section provides a detailed experimental analysis that aims to answer the following questions: (i) How does ProMP perform against previous Meta-RL algorithms? (ii) How do the lower variance but biased LVC gradient estimates compare to the high variance, unbiased DiCE estimates? (iii) Do the different formulations result in different pre-update exploration properties? (iv) How do formulation I and formulation II differ in their meta-gradient estimates and convergence properties?

To answer the posed questions, we evaluate our approach on six continuous control Meta-RL benchmark environments based on OpenAI Gym and the Mujoco simulator (Dhariwal et al., 2017; Todorov et al., 2012). A description of the experimental setup is found in Appendix C.4. In all experiments, the reported curves are averaged over at

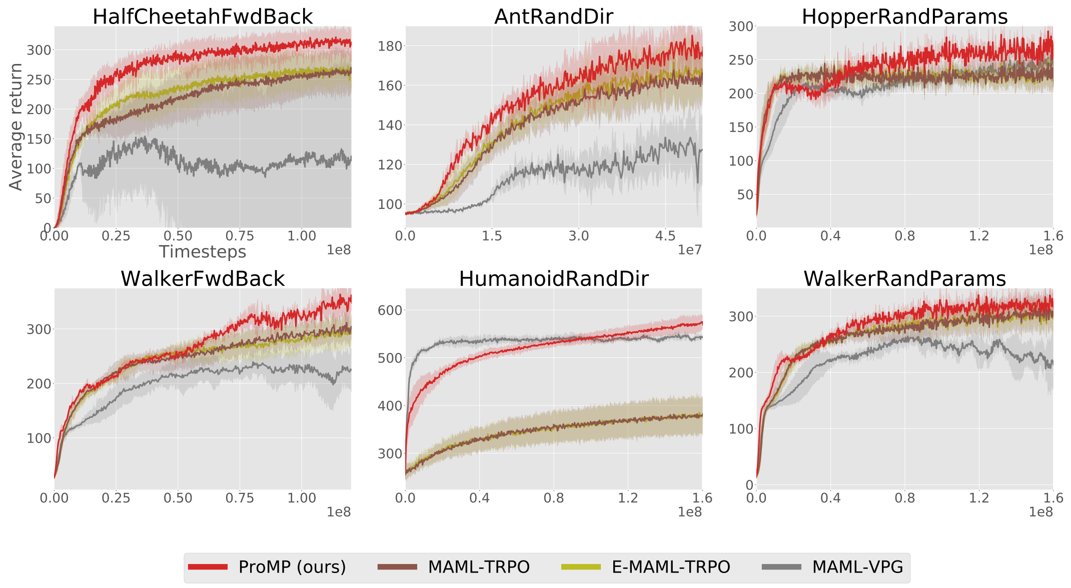


Figure 13: Meta-learning curves of ProMP and previous gradient-based meta-learning algorithms in six different MuJoCo environments. ProMP outperforms previous work in all the the environments.

least three random seeds. Returns are estimated based on sampled trajectories from the adapted post-update policies and averaged over sampled tasks. The source code and the experiment data are available on our supplementary website.[2]

6.7.1 *Meta-Gradient Based Comparison*

We compare our method, ProMP, in sample complexity and asymptotic performance to the gradient-based meta-learning approaches MAML-TRPO (Finn et al., 2017) and E-MAML-TRPO (see Fig. 13). Note that MAML corresponds to the original implementation of RL-MAML by (Finn et al., 2017) where no credit assignment to the pre-adaptation policy is happening (see Appendix C.2.3 for details). Moreover, we provide a second study which focuses on the underlying meta-gradient estimator. Specifically, we compare the LVC, DiCE, MAML and E-MAML estimators while optimizing meta-learning objective with vanilla policy gradient (VPG) ascent. This can be viewed as an ablated version of the algorithms which tries to eliminate the influences of the outer optimizers on the

2 https://sites.google.com/view/pro-mp

learning performance (see Fig. 14).

These algorithms are benchmarked on six different locomotion tasks that require adaptation: the half-cheetah and walker must switch between running forward and backward, the high-dimensional agents ant and humanoid must learn to adapt to run in different directions in the 2D-plane, and the hopper and walker have to adapt to different configuration of their dynamics.

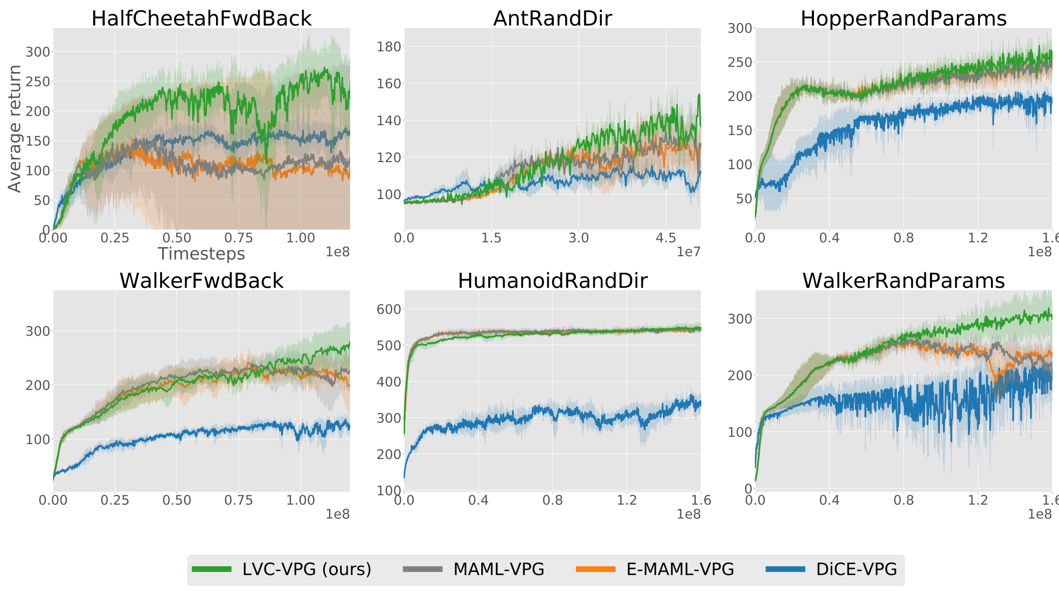


Figure 14: Meta-learning curves corresponding to different meta-gradient estimators in conjunction with VPG. The introduced LVC approach consistently outperforms the other estimators.

The results in Figure 13 highlight the strength of ProMP in terms of sample efficiency and asymptotic performance. In the meta-gradient estimator study in Fig. 14, we demonstrate the positive effect of the LVC objective, as it consistently outperforms the other estimators. In contrast, DiCE learns only slowly when compared to the other approaches. As we have motivated mathematically and substantiate empirically in the following experiment, the poor performance of DiCE may be ascribed to the high variance of its meta-gradient estimates. The fact that the results of MAML and E-MAML are comparable underpins the ineffectiveness of the naive pre-update credit assignment (i.e. formulation II), as discussed in section 6.3.

Results for four additional environments are displayed in Appendix C.4 along with hyperparameter settings, environment specifications and a wall-clock time comparison of the algorithms.

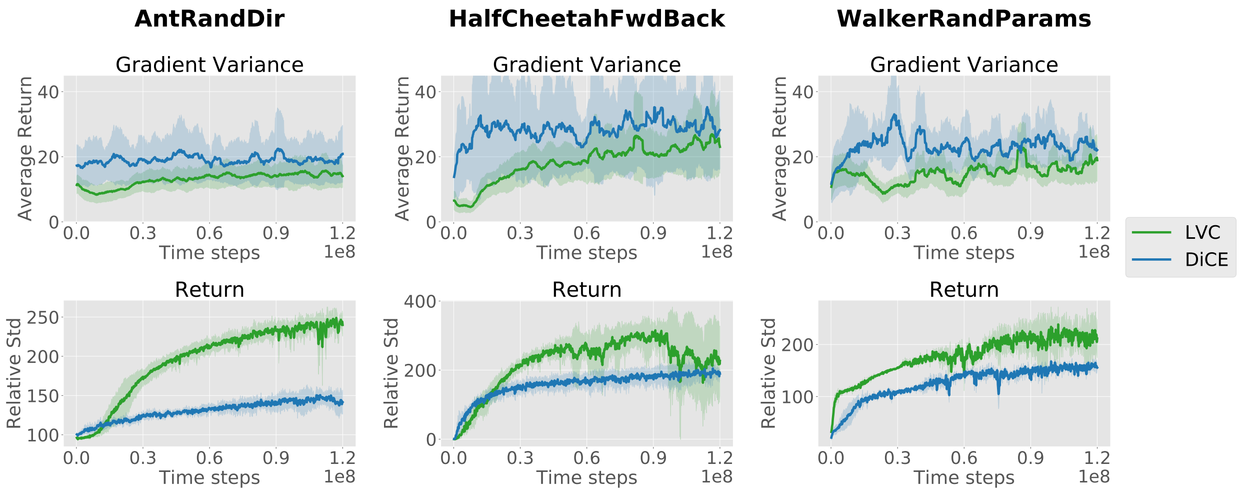


Figure 15: Top: Relative std of meta-policy gradients. Bottom: Returns in the respective environments throughout the learning process. LVC exhibits less variance in its meta-gradients which may explain its superior performance when compared to DiCE.

6.7.2 *Gradient Estimator Variance and Its Effect on Meta-Learning*

In Section 6.4 we discussed how the DiCE formulation yields unbiased but high variance estimates of the RL-objective hessian and served as motivation for the low variance curvature (LVC) estimator. Here we investigate the meta-gradient variance of both estimators as well as its implication on the learning performance. Specifically, we report the relative standard deviation of the meta-policy gradients as well as the average return throughout the learning process in three of the meta-environments.

The results, depicted in Figure 15, highlight the advantage of the low variance curvature estimate. The trajectory level dependencies inherent in the DiCE estimator leads to a meta-gradient standard deviation that is on average 60% higher when compared to LVC. As the learning curves indicate, the noisy gradients may be a driving factor for the poor performance of DiCE, impeding sample efficient meta-learning. Meta-policy search based on the LVC estimator leads to substantially better sample-efficiency and asymptotic performance.

In case of HalfCheetahFwdBack, we observe some unstable learning behavior of LVC-VPG which is most likely caused by the bias of LVC in combination with the naive VPG optimizer. However, the mechanisms in ProMP that ensure proximity w.r.t. to the policy's KL-divergence seem to counteract these instabilities during training, giving us a stable and efficient meta-learning algorithm.

6.7.3 *Comparison of Initial Sampling Distributions*

Here we evaluate the effect of the different objectives on the learned pre-update sampling distribution. We compare the low variance curvature (LVC) estimator with TRPO (LVC-TRPO) against MAML (Finn et al., 2017) and E-MAML-TRPO (Stadie et al., 2018) in a 2D environment on which the exploration behavior can be visualized. Each task of this environment corresponds to reaching a different corner location; however, the 2D agent only experiences reward when it is sufficiently close to the corner (translucent regions of Figure 16). Thus, to successfully identify the task, the agent must explore the different regions. We perform three inner adaptation steps on each task, allowing the agent to fully change its behavior from exploration to exploitation.

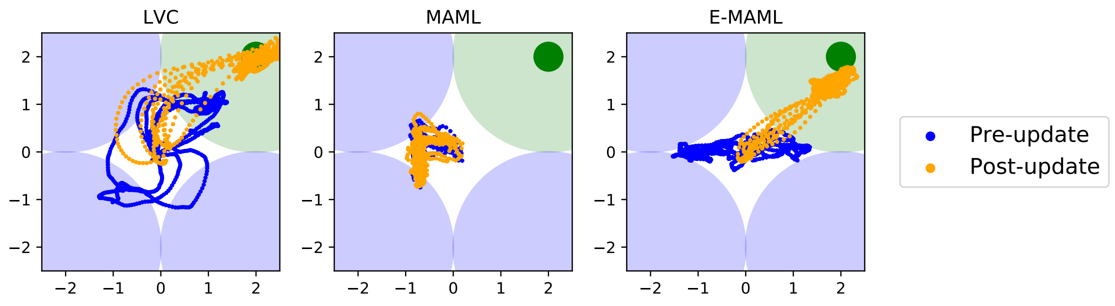


Figure 16: Exploration patterns of the pre-update policy and exploitation post-update with different update functions. Through its superior credit assignment, the LVC objective learns a pre-update policy that is able to identify the current task and respectively adapt its policy, successfully reaching the goal (dark green circle).

The different exploration-exploitation strategies are displayed in Figure 16. Since the MAML implementation does not assign credit to the pre-update sampling trajectory, it is unable to learn a sound exploration strategy for task identification and thus fails to accomplish the task. On the other hand, E-MAML, which corresponds to formulation II, learns to explore in long but random paths: because it can only assign credit to batches of pre-update trajectories, there is no notion of which actions in particular facilitate good task adaptation. As consequence the adapted policy slightly misses the task-specific target. The LVC estimator, instead, learns a consistent pattern of exploration, visiting each of the four regions, which it harnesses to fully solve the task.

6.7.4 *Gradient Update Directions of the Two Meta-RL Formulations*

To shed more light on the differences of the gradients of formulation I and formulation II, we evaluate the meta-gradient updates and the corresponding convergence to the optimum of both formulations in a simple 1D environment. In this environment, the agent starts in a random position in the real line and has to reach a goal located at the position 1 or -1. In order to visualize the convergence, we parameterize the policy with only two parameters θ_0 and θ_1. We employ formulation I by optimizing the DiCE objective with VPG, and formulation II by optimizing its (E-MAML) objective with VPG.

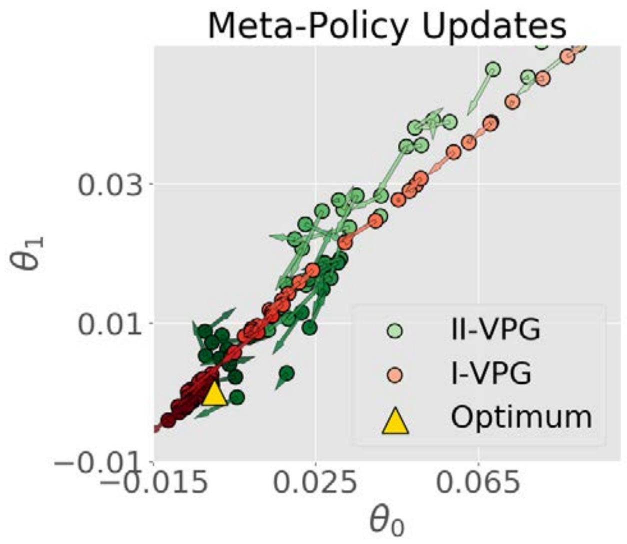


Figure 17: Meta-gradient updates of policy parameters θ_0 and θ_1 in a 1D environment w.r.t Formulation I (red) and Formulation II (green).

Figure 17 depicts meta-gradient updates of the parameters θ_i for both formulations. Formulation I (red) exploits the internal structure of the adaptation update yielding faster and steadier convergence to the optimum. Due to its inferior credit assignment, formulation II (green) produces noisier gradient estimates leading to worse convergence properties.

6.8 CONCLUSION

We propose a novel meta-RL algorithm, proximal meta-policy search (ProMP), which fully optimizes for the pre-update sampling distribution leading to effective task identification. Our method is the result of a theoretical analysis of gradient-based meta-RL formulations, based on which we develop the low variance curvature (LVC) surrogate objective that produces low variance meta-policy gradient estimates. Experimental results demonstrate that our approach surpasses previous meta-reinforcement learning approaches in a diverse set of continuous control tasks. Finally, we underpin our theoretical contributions with illustrative examples which further justify the soundness and effectiveness of our method.

7

IMPROVING MODEL-BASED REINFORCEMENT LEARNING VIA MODEL-AUGMENTED PATHWISE DERIVATIVE

While model-based via meta-policy adaptation (Chapter 5) is able to learn longer horizon tasks than ME-TRPO (Chapter 4), it is limited to trajectories around a thousand steps in simple domains. For longer horizons or more complex environments, learning becomes unstable. Long horizon predictions are challenging because of the compounding error effect. In this work, we specifically tackle the long horizon challenge by enhancing model-based with actor-critic methods (Lillicrap et al., 2015; Haarnoja et al., 2018a) with model-based. The model is just used to predict for a short horizon while the value function captures the long horizon predictions. This set-up was introduced by (Heess et al., 2015) and (Janner et al., 2019). In this work, we improve those methods by making use of the model derivatives through the pathwise derivative estimator (Mohamed et al., 2019; Kingma and Welling, 2013). We learn an ensemble of stochastic dynamics models, and the pathwise derivative estimator allows us to backpropagate through the stochasticity while at the same time have low variance gradients. Using the gradients of the learned models allows us to extract more information from them for the policy learning step, resulting in faster learning.

7.1 INTRODUCTION

Model-based reinforcement learning (RL) offers the potential to be a general-purpose tool for learning complex policies while being sample efficient. When learning in real-world physical systems, data collection can be an arduous process. Contrary to model-free methods, model-based approaches are appealing due to their comparatively fast learning. By first learning the dynamics of the system in a supervised learning way, it can exploit off-policy data. Then, model-based methods use the model to derive controllers from it either parametric controllers (Luo et al., 2019; Buckman et al., 2018; Janner et al.,

2019) or non-parametric controllers (Nagabandi et al., 2017; Chua et al., 2018).

Current model-based methods learn with an order of magnitude less data than their model-free counterparts while achieving the same asymptotic convergence. Tools like ensembles, probabilistic models, planning over shorter horizons, and meta-learning have been used to achieved such performance (Kurutach et al., 2018; Chua et al., 2018; Clavera et al., 2018). However, the model usage in all of these methods is the same: simple data augmentation. They use the learned model as a black-box simulator generating samples from it. In high-dimensional environments or environments that require longer planning, substantial sampling is needed to provide meaningful signal for the policy. *Can we further exploit our learned models?*

In this work, we propose to estimate the policy gradient by backpropagating its gradient through the model using the pathwise derivative estimator. Since the learned model is differentiable, one can link together the model, reward function, and policy to obtain an analytic expression for the gradient of the returns with respect to the policy. By computing the gradient in this manner, we obtain an expressive signal that allows rapid policy learning. We avoid the instabilities that often result from back-propagating through long horizons by using a terminal Q-function. This scheme fully exploits the learned model without harming the learning stability seen in previous approaches (Kurutach et al., 2018; Heess et al., 2015). The horizon at which we apply the terminal Q-function acts as a hyperparameter between model-free (when fully relying on the Q-function) and model-based (when using a longer horizon) of our algorithm.

The main contribution of this work is a model-based method that significantly reduces the sample complexity compared to state-of-the-art model-based algorithms (Janner et al., 2019; Buckman et al., 2018). For instance, we achieve a 10k return in the half-cheetah environment in just 50 trajectories. We theoretically justify our optimization objective and derive the monotonic improvement of our learned policy in terms of the Q-function and the model error. Furtermore, we experimentally analyze the theoretical derivations. Finally, we pinpoint the importance of our objective by ablating all the components of our algorithm. The results are reported in four model-based benchmarking environments (T. Wang et al., 2019; Todorov et al., 2012). The low sample complexity and high performance of our method carry high promise towards learning directly on real robots.

7.2 BACKGROUND

In this section, we present how actor critic methods work and a summary on Monte-Carlo gradient estimators.

7.2.1 *Actor-Critic Methods*

In actor-critic methods, we learn a function $\hat{Q}$ (critic) that approximates the expected return conditioned on a state s and action a, $\mathbb{E}[\sum_t \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)|\mathbf{s}_0 = s, \mathbf{a}_0 = a]$. Then, the learned Q-function is used to optimize a policy π (actor). Usually, the Q-function is learned by iteratively minimizing the Bellman residual:

$$J_Q = \mathbb{E}[(\hat{Q}(\mathbf{s}_t, \mathbf{a}_t) - (r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \hat{Q}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})))^2]$$

The above method is referred as one-step Q-learning, and while a naive implementation often results in unstable behaviour, recent methods have succeeded in stabilizing the Q-function training (Fujimoto et al., 2018). The actor then can be trained to maximize the learned $\hat{Q}$ function $J_\pi = \mathbb{E}\left[\hat{Q}(s, \pi(s))\right]$. The benefit of this form of actor-critic method is that it can be applied in an off-policy fashion, sampling random mini-batches of transitions from an experience replay buffer (Lin, 1992).

Model-Based RL. Model-based methods, contrary to model-free RL, learn the transition distribution from experience. Typically, this is carried out by learning a parametric function approximator $\hat{f}_\phi$, known as a dynamics model. We define the state predicted by the dynamics model as $\hat{s}_{t+1}$, i.e., $\hat{s}_{t+1} \sim \hat{f}_\phi(\mathbf{s}_t, \mathbf{a}_t)$. The models are trained via maximum likelihood: $\max_\phi J_f(\phi) = \max_\phi \mathbb{E}[\log p(\hat{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)]$

7.2.2 *Monte-Carlo Gradient Estimators*

In order to optimize the reinforcement learning objective, it is needed to take the gradient of an expectation. In general, it is not possible to compute the exact expectation so Monte-Carlo gradient estimators are used instead. These are mainly categorized into three classes: the pathwise, score function, and measure-valued gradient estimator (Mohamed et al., 2019). In this work, we use the pathwise gradient estimator, which is also known as the re-parameterization trick (Kingma and Welling, 2013). This estimator is derived from the law of the unconscious statistician (LOTUS) (Grimmett and Stirzaker, 2001)

$$\mathbb{E}_{p_\theta(x)}[f(x)] = \mathbb{E}_{p(\epsilon)}[f(g_\theta(\epsilon)]$$

Here, we have stated that we can compute the expectation of a random variable x without knowing its distribution, if we know its corresponding sampling path and base distribution. A common case, and the one used in this manuscript, θ parameterizes a Gaussian distribution: $x \sim p_\theta = \mathcal{N}(\mu_\theta, \sigma_\theta^2)$, which is equivalent to $x = \mu_\theta + \epsilon\sigma_\theta$ for $\epsilon \sim \mathcal{N}(0, 1)$.

In the following, we present our policy optimization scheme and describe the full algorithm. Exploiting the full capability of learned models has the potential to enable complex and high-dimensional real robotics tasks while maintaining low sample complexity. Our approach, model-augmented actor-critic (MAAC), exploits the learned model by computing the analytic gradient of the returns with respect to the policy. In contrast to sample-based methods, which one can think of as providing directional derivatives in trajectory space, MAAC computes the full gradient, providing a strong learning signal for policy learning, which further decreases the sample complexity.

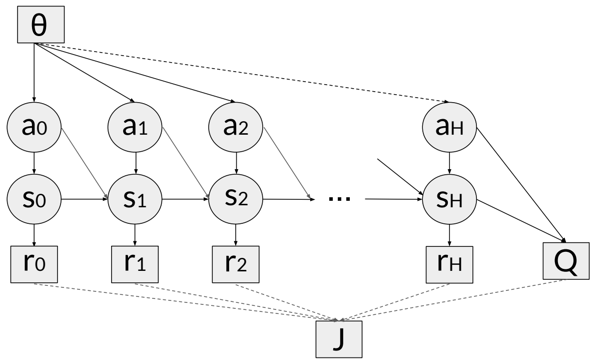


Figure 18: Stochastic computation graph of the proposed objective J_π. The stochastic nodes are represented by circles and the deterministic ones by squares.

7.3.1 *Model-Augmented Actor-Critic Objective*

Among model-free methods, actor-critic methods have shown superior performance in terms of sample efficiency and asymptotic performance (Haarnoja et al., 2018a). However, their sample efficiency remains worse than model-based approaches, and fully off-policy methods still show instabilities comparing to on-policy algorithms (Mnih et al., 2016). Here, we propose a modification of the Q-function parametrization by using the model predictions on the first time-steps after the action is taken. Specifically, we do policy optimization by maximizing the following objective:

$$J_\pi(\theta) = \mathbb{E}\left[\sum_{t=0}^{H-1} \gamma^t r(s_t) + \gamma^H \hat{Q}(s_H, a_H)\right]$$

whereby, $s_{t+1} \sim \hat{f}(s_t, a_t)$ and $a_t \sim \pi_\theta(s_t)$. Note that under the true dynamics and Q-function, this objective is the same as the RL objective. Contrary to previous reinforcement learning methods, we optimize this objective by back-propagation through time.

Since the learned dynamics model and policy are parameterized as Gaussian distributions, we can make use of the pathwise derivative estimator to compute the gradient, resulting in an objective that captures uncertainty while presenting low variance. The computational graph of the proposed objective is shown in Figure 18.

While the proposed objective resembles n-step bootstrap (Sutton and Barto, 2018), our model usage fundamentally differs from previous approaches. First, we do not compromise between being off-policy and stability. Typically, n-step bootstrap is either on-policy, which harms the sample complexity, or its gradient estimation uses likelihood ratios, which presents large variance and results in unstable learning (Heess et al., 2015). Second, we obtain a strong learning signal by backpropagating the gradient of the policy across multiple steps using the pathwise derivative estimator, instead of the REINFORCE estimator (Mohamed et al., 2019; Peters and Schaal, 2006). And finally, we prevent the exploding and vanishing gradients effect inherent to back-propagation through time by the means of the terminal Q-function (Kurutach et al., 2018).

The horizon H in our proposed objective allows us to trade off between the accuracy of our learned model and the accuracy of our learned Q-function. Hence, it controls the degree to which our algorithm is model-based or well model-free. If we were not to trust our model at all ($H = 0$), we would end up with a model-free update; for $H = \infty$, the objective results in a shooting objective. We will perform policy optimization by taking derivatives of the objective, hence we require accuracy on the derivatives of the objective. The following lemma provides a bound on the gradient error in terms of the error on the derivatives of the model, the Q-function, and the horizon H.

Lemma 1 (Gradient Error)**.** *Let $\hat{f}$ and $\hat{Q}$ be the learned approximation of the dynamics f and Q-function Q, respectively. Assume that Q and $\hat{Q}$ have $L_q/2$-Lipschitz continuous gradient and f and $\hat{f}$ have $L_f/2$-Lipschitz continuous gradient. Let $\epsilon_f = \max_t \|\nabla \hat{f}(\hat{s}_t, \hat{a}_t) - \nabla f(\mathbf{s}_t, \mathbf{a}_t)\|_2$ be the error on the model derivatives and $\epsilon_Q = \|\nabla \hat{Q}(\hat{s}_H, \hat{a}_H) - \nabla Q(\mathbf{s}_H, \mathbf{a}_H)\|_2$ the error on the Q-function derivative. Then the error on the gradient between the learned objective and the true objective can be bounded by:*

$$\mathbb{E}\left[\|\nabla_\theta J_\pi - \nabla_\theta \hat{J}_\pi\|_2\right] \leqslant c_1(H)\epsilon_f + c_2(H)\epsilon_Q$$

Proof. See Appendix. □

The result in Lemma 1 stipulates the error of the policy gradient in terms of the maximum error in the model derivatives and the error in the Q derivatives. The functions c_1 and c_2 are functions of the horizon and depend on the Lipschitz constants of the model and the Q-function. Note that we are just interested in the relation between both sources of error, since the gradient magnitude will be scaled by the learning rate, or by the optimizer, when applying it to the weights.

7.3.2 *Monotonic Improvement*

Previously, we presented our objective and the error it incurs in the policy gradient with respect to approximation error in the model and the Q function. However, the error on the gradient is not indicative of the effect of the desired metric: the average return. Here, we quantify the effect of the modeling error on the return. We will bound the KL-divergence between the policies resulting from taking the gradient with the true objective and the approximated one. Then we will bound the performance in terms of the KL.

Lemma 2 (Total Variation Bound)**.** *Under the assumptions of the Lemma 1, let* $\theta = \theta_0 + \alpha \nabla_\theta J_\pi$ *be the parameters resulting from taking a gradient step on the exact objective, and* $\hat{\theta} = \theta_0 + \alpha \nabla_\theta \hat{J}_\pi$ *the parameters resulting from taking a gradient step on approximated objective, where* $\alpha \in \mathbb{R}^+$*. Then the following bound on the total variation distance holds*

$$\max_s D_{TV}(\pi_\theta \| \pi_{\hat{\theta}}) \leqslant \alpha c_3(\epsilon_f c_1(H) + \epsilon_Q c_2(H))$$

Proof. See Appendix. □

The previous lemma results in a bound on the distance between the policies originated from taking a gradient step using the true dynamics and Q-function, and using its learned counterparts. Now, we can derive a similar result from (Kakade and Langford, 2002) to bound the difference in average returns.

Theorem 1 (Monotonic Improvement)**.** *Under the assumptions of the Lemma 1, be* θ' *and* $\hat{\theta}$ *as defined in Lemma 2, and assuming that the reward is bounded by* $r_{\max}$*. Then the average return of the* $\pi_{\hat{\theta}}$ *satisfies*

$$J_\pi(\hat{\theta}) \geqslant J_\pi(\theta) - \frac{2\alpha r_{\max}}{1-\gamma} \alpha c_3(\epsilon_f c_1(H) + \epsilon_Q c_2(H))$$

Proof. See Appendix. □

Algorithm 5 MAAC

```
 1: Initialize the policy $\pi_\theta$, model $\hat{f}_\phi$, $\hat{Q}_\psi$, $\mathcal{D}_{\text{env}} \leftarrow \emptyset$, and the model dataset $\mathcal{D}_{\text{model}} \leftarrow \emptyset$
 2: repeat
 3:     Sample trajectories from the real environment with policy $\pi_\theta$. Add them to $\mathcal{D}_{\text{env}}$.
 4:     for $i = 1 \dots G_1$ do
 5:         $\phi \leftarrow \phi - \beta_f \nabla_\phi J_f(\phi)$ using data from $\mathcal{D}_{\text{env}}$.
 6:     end for
 7:     Sample trajectories $\mathcal{T}$ from $\hat{f}_\phi$. Add them to $\mathcal{D}_{\text{model}}$.
 8:     $\mathcal{D} \leftarrow \mathcal{D}_{\text{model}} \cup \mathcal{D}_{\text{env}}$
 9:     for $i = 1 \dots G_2$ do
10:         Update $\theta \leftarrow \theta + \beta_\pi \nabla_\theta J_\pi(\theta)$ using data from $\mathcal{D}$
11:         Update $\psi \leftarrow \psi - \beta_Q \nabla_\psi J_Q(\psi)$ using data from $\mathcal{D}$
12:     end for
13: until the policy performs well in the real environment
14: return Optimal parameters $\theta^*$
```

Hence, we can provide explicit lower bounds of improvement in terms of model error and function error. Theorem 1 extends previous work of monotonic improvement for model-free policies (Schulman et al., 2015b; Kakade and Langford, 2002), to the model-based and actor critic set up by taking the error on the learned functions into account. From this bound one could, in principle, derive the optimal horizon H that minimizes the gradient error. However, in practice, approximation errors are hard to determine and we treat H as an extra hyper-parameter. In section 7.4.2, we experimentally analyze the error on the gradient for different estimators and values of H.

7.3.3 *Algorithm*

Based on the previous sections, we develop a new algorithm that explicitly optimizes the model-augmented actor-critic (MAAC) objective. The overall algorithm is divided into three main steps: model learning, policy optimization, and Q-function learning.

Model learning. In order to prevent overfitting and overcome model-bias (Deisenroth and Rasmussen, 2011), we use a bootstrap ensemble of dynamics models $\{\hat{f}_{\phi_1}, ..., \hat{f}_{\phi_M}\}$. Each of the dynamics models parameterizes the mean and the covariance of a Gaussian distribution with diagonal covariance. The bootstrap ensemble captures the epistemic uncertainty, uncertainty due to the limited capacity or data, while the probabilistic models are able to capture the aleatoric uncertainty (Chua et al., 2018), inherent uncertainty of the environment. We denote by f_{ϕ} the transitions dynamics resulting from $\boldsymbol{\phi}_U$, where $U \sim \mathcal{U}[M]$ is uniform random variable on $\{1, ..., M\}$. The dynamics models are trained via maximum likelihood with early stopping on a validation set.

Policy Optimization. We extend the MAAC objective with an entropy bonus (Haarnoja et al., 2018c), and perform policy learning by maximizing

$$J_{\pi}(\theta) = \mathbb{E}\left[\sum_{t=0}^{H-1} \gamma^t r(\hat{s}_t) + \gamma^H Q_{\psi}(\hat{s}_H, \mathbf{a}_H)\right] + \beta \mathcal{H}(\pi_{\theta})$$

where $\hat{s}_{t+1} \sim f_{\phi}(\hat{s}_t, \mathbf{a}_t)$, $\hat{s}_0 \sim \mathcal{D}$, $a \sim \pi_{\theta}$. We learn the policy by using the pathwise derivative of the model through H steps and the Q-function by sampling multiple trajectories from the same $\hat{s}_0$. Hence, we learn a maximum entropy policy using pathwise derivative of the model through H steps and the Q-function. We compute the expectation by sampling multiple actions and states from the policy and learned dynamics, respectively.

Q-function Learning. In practice, we train two Q-functions (Fujimoto et al., 2018) since it has been experimentally proven to yield better results. We train both Q functions by minimizing the Bellman error (Section 7.2.1):

$$J_Q(\psi) = \mathbb{E}[(Q_{\psi}(\mathbf{s}_t, \mathbf{a}_t) - (r(\mathbf{s}_t, \mathbf{a}_t) + \gamma Q_{\psi}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1})))^2]$$

Similar to (Janner et al., 2019), we minimize the Bellman residual on states previously visited and imagined states obtained from unrolling the learned model. Finally, the value targets are obtained in the same fashion the Stochastic Ensemble Value Expansion (Buckman et al., 2018), using H as a horizon for the expansion. In doing so, we maximally make use of the model by not only using it for the policy gradient step, but also for training the Q-function.

Our method, MAAC, iterates between collecting samples from the environment, model training, policy optimization, and Q-function learning. A practical implementation of our method is described in Algorithm 5. First, we obtain trajectories from the real environment using the latest policy available. Those samples are appended to a replay buffer $\mathcal{D}_{\text{env}}$, on which the dynamics models are trained until convergence. The third step is to collect imaginary data from the models: we collect k-step transitions by unrolling the latest policy from a randomly sampled state on the replay buffer. The imaginary data constitutes the $\mathcal{D}_{\text{model}}$, which together with the replay buffer, is used to learn the Q-function and train the policy.

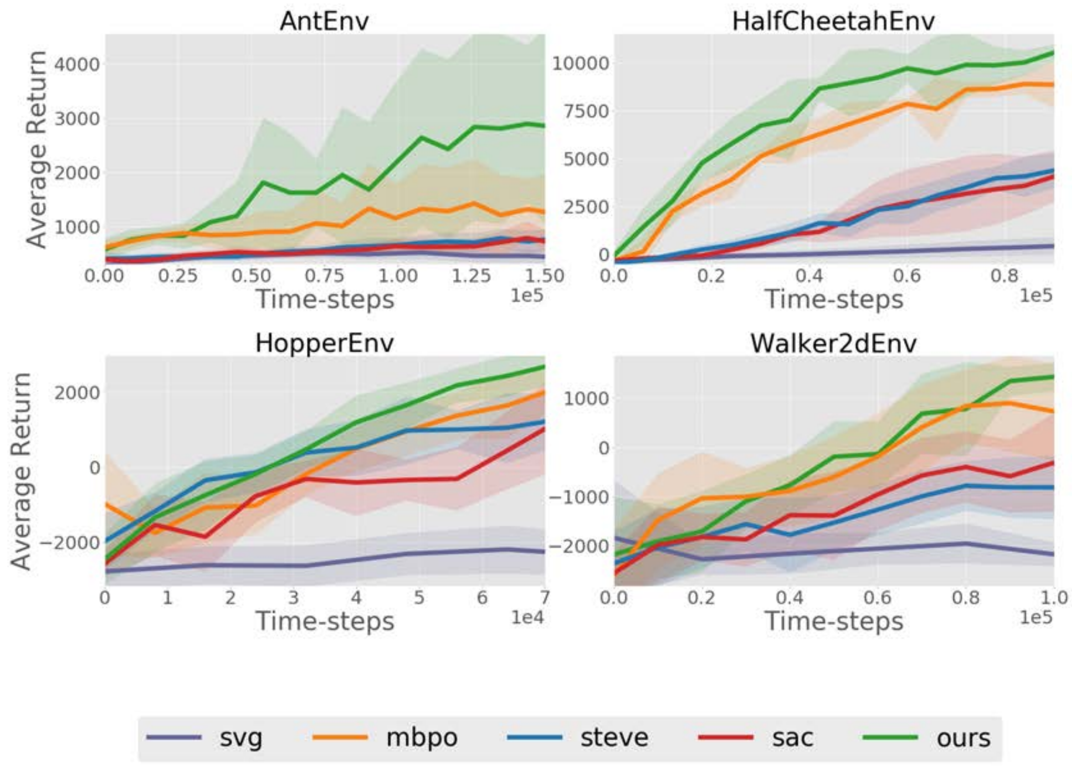


Figure 19: Comparison against state-of-the-art model-free and model-based baselines in four different MuJoCo environments. Our method, MAAC, attains better sample efficiency and asymptotic performance than previous approaches. The gap in performance between MAAC and previous work increases as the environments increase in complexity.

Our algorithm consolidates the insights built through the course of this paper, while at the same time making maximal use of recently developed actor-critic and model-based methods. All in all, it consistently outperforms previous model-based and actor-critic methods.

7.4 RESULTS

Our experimental evaluation aims to examine the following questions: 1) How does MAAC compares against state-of-the-art model-based and model-free methods? 2) Does the gradient error correlate with the derived bound?, 3) Which are the key components of its performance?, and 4) Does it benefit from planning at test time?

In order to answer the posed questions, we evaluate our approach on model-based continuous control benchmark tasks in the MuJoCo simulator (Todorov et al., 2012; T. Wang et al., 2019).

7.4.1 *Comparison Against State-of-the-Art*

We compare our method on sample complexity and asymptotic performance against state-of-the-art model-free (MF) and model-based (MB) baselines. Specifically, we compare against the model-free soft actor-critic (SAC) (Haarnoja et al., 2018a), which is an off-policy algorithm that has been proven to be sample efficient and performant; as well as two state-of-the-art model-based baselines: model-based policy-optimization (MBPO) (Janner et al., 2019) and stochastic ensemble value expansion (STEVE) (Buckman et al., 2018). The original STEVE algorithm builds on top of the model-free algorithm DDPG (Lillicrap et al., 2015), however this algorithm is outperformed by SAC. In order to remove confounding effects of the underlying model-free algorithm, we have implemented the STEVE algorithm on top of SAC. We also add SVG(1) (Heess et al., 2015) to comparison, which similar to our method uses the derivative of dynamic models to learn the policy.

The results, shown in Fig. 19, highlight the strength of MAAC in terms of performance and sample complexity. MAAC scales to higher dimensional tasks while maintaining its sample efficiency and asymptotic performance. In all the four environments, our method learns faster than previous MB and MF methods. We are able to learn near-optimal policies in the half-cheetah environment in just over 50 rollouts, while previous model-based methods need at least the double amount of data. Furthermore, in complex environments, such as ant, MAAC achieves near-optimal performance within 150 rollouts while other take orders of magnitudes more data.

7.4.2 *Gradient Error*

Here, we investigate how the bounds obtained relate to the empirical performance. In particular, we study the effect of the horizon of the model predictions on the gradient error. In order to do so, we construct a double integrator environment; since the transitions are linear and the cost is quadratic for a linear policy, we can obtain the analytic gradient of the expect return.

Figure 20 depicts the L1 error of the MAAC objective for different values of the horizon H as well as what would be the error using the true dynamics. As expected, using the true dynamics yields to lower gradient error since the only source comes from the learned Q-function that is weighted down by γ^H.

The results using learned dynamics correlate with our assumptions and the derived bounds: the error from the learned dynamics is lower than the one in the Q-function, but it scales poorly with the horizon. For short horizons the error decreases as we increase the horizon. However, large horizons is detrimental since it magnifies the error on the models.

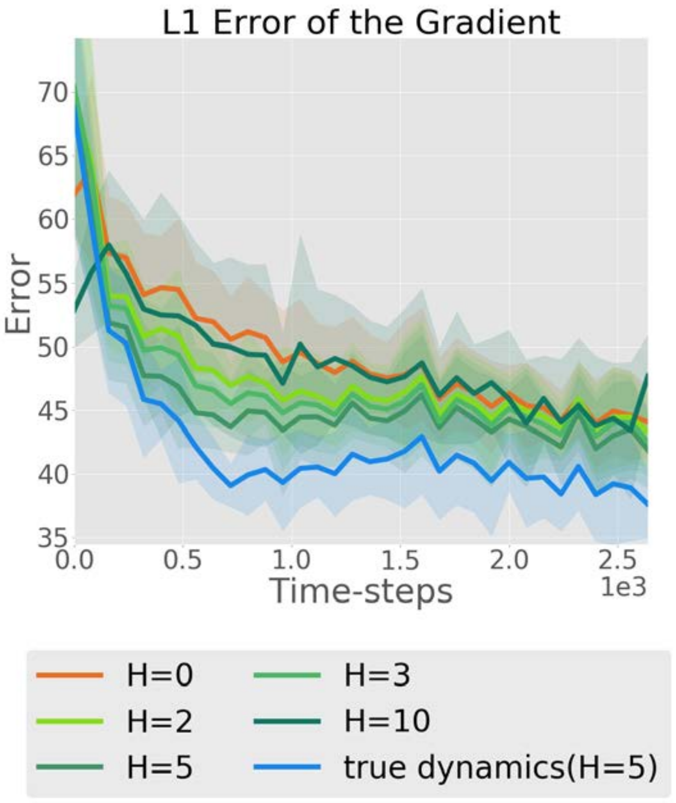


Figure 20: L1 error on the policy gradient when using the proposed objective for different values of the horizon H as well as the error obtained when using the true dynamics. The results correlate with the assumption that the error in the learned dynamics is lower than the error in the Q-function, as well as they correlate with the derived bounds.

7.4.3 *Ablations*

In order to investigate the importance of each of the components of our overall algorithm, we carry out an ablation test. Specifically, we test three different components: 1) not using the model to train the policy, i.e., set $H = 0$, 2) not using the STEVE targets for training the critic, and 3) using a single sample estimate of the path-wise derivative.

The ablation test is shown in Figure 21. The test underpins the importance of backpropagating through the model: setting H to be 0 inflicts a severe drop in the algorithm performance. On the other hand, using the STEVE targets results in slightly more stable training, but it does not have a significant effect. Finally, while single sample estimates can be used in simple environments, they are not accurate enough in higher dimensional environments such as ant.

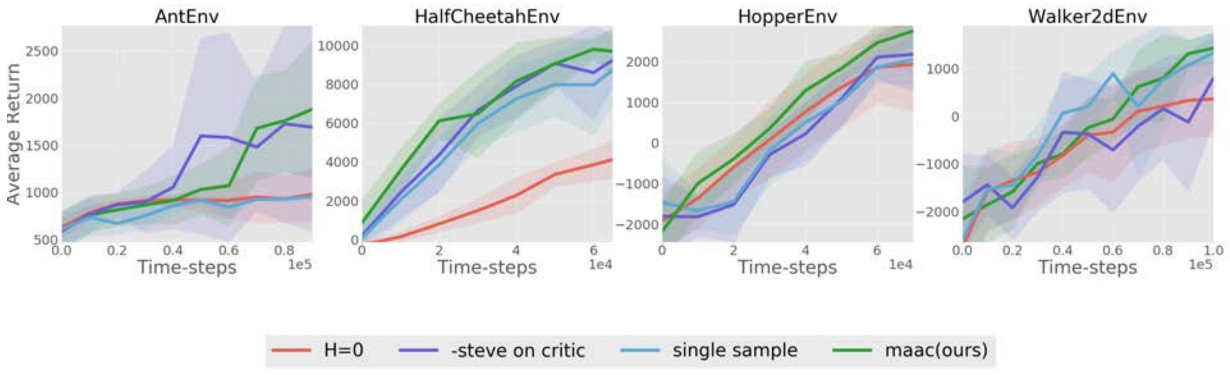


Figure 21: Ablation test of our method. We test the importance of several components of our method: not using the model to train the policy ($H = 0$), not using the STEVE targets for training the Q-function (-STEVE), and using a single sample estimate of the pathwise derivative. Using the model is the component that affects the most the performance, highlighting the importance of our derived estimator.

7.4.4 *Model Predictive Control*

One of the key benefits of methods that combine model-based reinforcement learning and actor-critic methods is that the optimization procedure results in a stochastic policy, a dynamics model and a Q-function. Hence, we have all the components for, at test time, refine the action selection by the means of model predictive control (MPC). Here, we investigate the improvement in performance of planning at test time. Specifically, we use the cross-entropy method with our stochastic policy as our initial distributions. The results, shown in Table 1, show benefits in online planning in complex domains; however, its improvement gains are more timid in easier domains, showing that the learned policy has already interiorized the optimal behaviour.

7.5 RELATED WORK

Differentable Planning. Previous work has used backpropagate through learned models to obtain the optimal sequences of actions. For instance, Levine and Abbeel, 2014 learn linear local models and obtain the optimal sequences of actions, which is then distilled into a neural network policy. The planning can be incorporated into the neural

	AntEnv	HalfCheetahEnv	HopperEnv	Walker2dEnv
MAAC+MPC	$3.97e3 \pm 1.48e3$	$1.09e4 \pm 94.5$	$2.8e3 \pm 11$	$1.76e3 \pm 78$
MAAC	$3.06e3 \pm 1.45e3$	$1.07e4 \pm 253$	$2.77e3 \pm 3.31$	$1.61e3 \pm 404$

Table 1: Performance at test time with (maac+mpc) and without (maac) planning of the converged policy using the MAAC objective.

network architecture (Okada et al., 2017; Tamar et al., 2016; Srinivas et al., 2018; Karkus et al., 2019) or formulated as a differentiable function (Pereira et al., 2018; Amos et al., 2018). Planning sequences of actions, even when doing model-predictive control (MPC), does not scale well to high-dimensional, complex domains Janner et al., 2019. Our method, instead learns a neural network policy in an actor-critic fashion aided with a learned model. In our study, we evaluate the benefit of carrying out MPC on top of our learned policy at test time, Section 7.4.4. The results suggest that the policy captures the optimal sequence of action, and re-planning does not result in significant benefits.

Policy Gradient Estimation. The reinforcement learning objective involves computing the gradient of an expectation (Schulman et al., 2015a). By using Gaussian processes (Deisenroth and Rasmussen, 2011), it is possible to compute the expectation analytically. However, when learning expressive parametric non-linear dynamical models and policies, such closed form solutions do not exist. The gradient is then estimated using Monte-Carlo methods (Mohamed et al., 2019). In the context of model-based RL, previous approaches mostly made use of the score-function, or REINFORCE estimator (Peters and Schaal, 2006; Kurutach et al., 2018). However, this estimator has high variance and extensive sampling is needed, which hampers its applicability in high-dimensional environments. In this work, we make use of the pathwise derivative estimator (Mohamed et al., 2019). Similar to our approach, (Heess et al., 2015) uses this estimator in the context of model-based RL. However, they just make use of real-world trajectories that introduces the need of a likelihood ratio term for the model predictions, which in turn increases the variance of the gradient estimate. Instead, we entirely rely on the predictions of the model, removing the need of likelihood ratio terms.

Actor-Critic Methods. Actor-critic methods alternate between policy evaluation, computing the value function for the policy; and policy improvement using such value function (Sutton and Barto, 2018; Barto et al., 1983). Actor-critic methods can be classified between on-policy and off-policy. On-policy methods tend to be more stable, but at the cost of sample efficiency (Sutton, 1991b; Mnih et al., 2016). On the other hand, off-policy methods offer better sample complexity (Lillicrap et al., 2015). Recent work has significantly stabilized and improved the performance of off-policy methods using maximum-entropy objectives (Haarnoja et al., 2018a) and multiple value functions (Fujimoto et al., 2018). Our method combines the benefit of both. By using the learned model we can have a learning that resembles an on-policy method while still being off-policy.

7.6 CONCLUSION

In this work, we present model-augmented actor-critic, MAAC, a reinforcement learning algorithm that makes use of a learned model by using the pathwise derivative across future timesteps. We prevent instabilities arisen from backpropagation through time by the means of a terminal value function. The objective is theoretically analyzed in terms of the model and value error, and we derive a policy improvement expression with respect to those terms. Our algorithm that builds on top of MAAC is able to achieve superior performance and sample efficiency than state-of-the-art model-based and model-free reinforcement learning algorithms. For future work, it would be enticing to deploy the presented algorithm on a real-robotic agent.

8

ASYNCHRONOUS METHODS FOR MODEL-BASED REINFORCEMENT LEARNING

Previous chapters have focused on tackling the algorithmic difficulties of model-based reinforcement learning. Specifically, we have aimed to answer the question: how can we combat model-bias? ME-TRPO (Chapter 4) offers a solution in terms of uncertainty characterization while MB-MPO (Chapter 5) tackles it from the perspective of policy adaptation. In this chapter, we are concerned of how to effectively deploy these algorithms in the real-world.

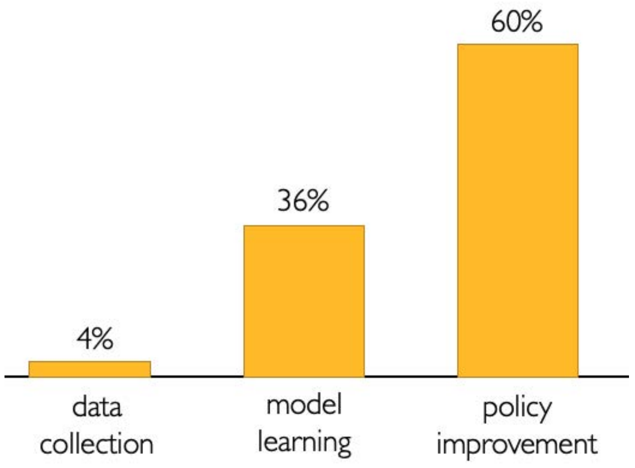


Figure 22: Profiling of one iteration of model-based via meta-policy optimization (MB-MPO). Each iteration takes around 25 min. The exact time depends on the environment.

Current model-based methods are computationally more expensive than traditional approaches, for instance the total training time of MB-MPO for learning a quadrupedal locomotion policy is of 2.2 days (T. Wang et al., 2019). Figure 22 shows the percentage of time spent in each step of the model-based learning process. This coupled by the sensitivity of model-based algorithms to multiple hyperparameters makes experimentation and learning of new policies extremely slow. In the following, we present an asynchronous framework for model-based methods that allows fast, wall-clock time, learning.

8.1 OVERVIEW

Autonomous skill acquisition has the potential to dramatically expand the tasks robots can perform ranging from manufacturing to household robotics. In real robotic agents, where data gathering is typically expensive, low sample complexity algorithms are required. Model-based reinforcement learning (RL) (Kaelbling et al., 1996) offers the potential to be data-efficient while achieving the same learning capabilities as model-free RL by first learning a predictive model of the environment and then deriving a controller from it.

In recent years, significant advances have been made in deep model-based reinforcement learning. Model-based algorithms presented in (Janner et al., 2019; Chua et al., 2018; Clavera et al., 2018; Buckman et al., 2018) achieve the same asymptotic performance as model-free algorithms while requiring an order of magnitude less data. However, these impressive results have been achieved at the cost of increasing the computational burden of model-based algorithms. Tools such as ensembles and probabilistic models have been key ingredients, preventing the policy from overfitting to the deficiencies of the learned model. As a result, while state-of-the-art model-based methods require just a few hours of agent interaction to learn complex tasks, they can nevertheless take days to train. For instance, the algorithm presented in (Clavera et al., 2018) takes less than three hours of real-world interaction to learn a locomotion behaviour, but the total training time is of 2.2 days (T. Wang et al., 2019). A need for algorithms that are both sample efficient and computationally fast is even more pressing when considering that these algorithms present a large number of sensitive hyperparameters. Altogether, slow experimentation and extensive hyperparameter search constitute a major barrier for the applicability of model-based methods to real-world robotics.

To bring down the wall-clock time of current model-based RL algorithms, we propose an asynchronous strategy where data collection, model learning, and policy improvement take place in parallel and asynchronously. Aside from speed, our method has two further benefits: First, learning the model while training the policy prevents the policy from overfitting to the deficiencies of the model, effectively regularizing the policy learning step (Luo et al., 2019). Second, collecting each rollout using the latest policy trained during the policy improvement process diversifies the data collected, which results in better predictive models.

The main contribution of our work is a general asynchronous framework for model-based reinforcement learning that reduces the run time of current model-based algorithms to be just the sampling time. It achieves better sample complexity than the classical sequential versions, and removes some hard-to-tune hyperparameters in current model-based approaches, such as the number of trajectories to collect or the number of gradients steps to take. Our experimental evaluation illustrates the strengths of our framework on four standard MuJoCo (Todorov et al., 2012) locomotion tasks. For instance, we were able to learn an optimal policy in high dimensional and complex quadrupedal locomotion within 60 minutes, while the classic sequential version takes more than 10 times longer. Finally, we showcase the effectiveness of our approach in real robotic manipulation skills that include block stacking and shape matching. In these cases, our asynchronous framework was able to succeed at each of the tasks within 10 minutes of wall-clock time. On the real robotic tasks our approach closely matches the performance of prior specialized work for such complex contact manipulation (Levine et al., 2015), while being more general. Code of parallel and sequential implementation of model-based algorithms, as well as videos of our method on the real robot environment, can be found at our website.[1]

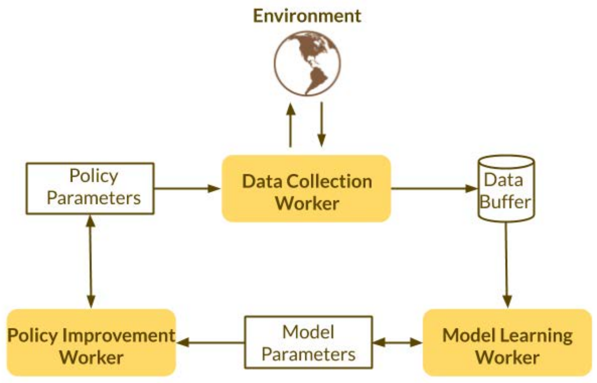
8.2 METHOD

Typically, model-based algorithms iterate through three phases till convergence: gathering data by interacting with the environment, learning a dynamics model using the gathered data, and improving policy using the learned dynamics model. Previous model-based RL work has made significant strides in decreasing sample complexity. It reduces interaction time with the environment, but shifts more computational load into learning distribution with models to capture uncertainty (Chua et al., 2018), and also into learning robust policies (Luo et al., 2019) or adaptive ones (Clavera et al., 2018). As a result, the wall-clock time of running such methods has significantly increased; for instance, training for 200k timesteps in the Ant environment takes 55 hours for MB-MPO (T. Wang et al., 2019).

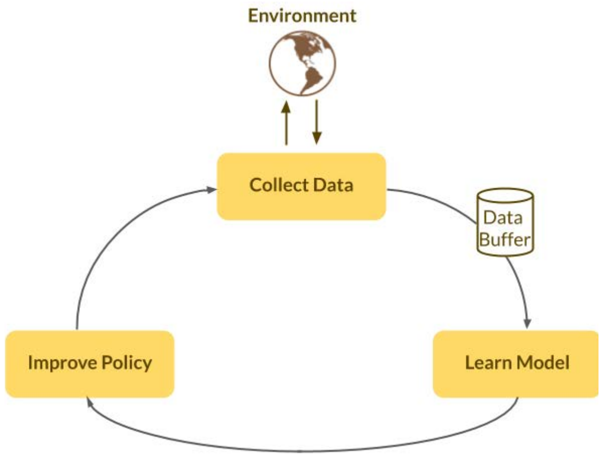
Our asynchronous framework, shown in Figure 23a, overcomes this deficiency and further improves sample efficiency of current model-based methods. In the following, we present the general recipe for asynchronous model-based reinforcement learning.

Within the framework, three main tasks of model-based algorithms are assigned to three parallel, independent workers that are respectively dedicated to data collection, model learning and policy improvement. The main task for each worker contains only the minimum amount of work (e.g. collecting one rollout, updating for one epoch or one gradient step). As a result, each worker fetches updates from servers with high frequency

1 https://sites.google.com/view/asynch-mb-rl/home



(a) Our proposed asynchronous model-based framework with three workers, communicating exclusively through three servers. The workers do not proceed in a specified order nor wait for the others to complete to execute their own function.



(b) Classic synchronous model-based methods, where three main steps proceed in well-defined order. Each of the steps does not start running until the preceding one has finalized.

and acts in fully asynchronous behavior. Each worker executes three operations:

- **Pull.** Worker gets an update from one of the three servers. For example, for the data collection worker, it pulls the latest policy parameters from the corresponding server.
- **Step.** The step operation corresponds to the main function of the worker. For the data collection worker, Step corresponds to collecting one rollout under its local copy of the policy. In the following subsections we explain in detail Step operation for each of the workers.
- **Push.** This operation sends the latest parameters or data to one of the three servers. Again, in the case the data collection worker, Push corresponds to pushing the collected rollout to the data buffer.

Each worker first checks one specific server either to fetch the latest parameters or to move all data from the remote server to its local buffer. Then it carries out its own step operation, and finally pushes the local change onto another specific server. Each worker loops through this process until a global stopping criterion is met. In the experiments, Section 8.4, the stopping criterion is set to be a total number of collected trajectories.

DATA COLLECTION. The data collection worker first pulls policy parameters θ from the server. With the latest policy it proceeds to the step operation, namely collecting one trajectory $\tau = (s_0, a_0, ..., s_{H-1}, a_{H-1}, s_H)$. Finally, it pushes the trajectory onto the data buffer and starts over from pulling again.

Algorithm 6 Data Collection

```
1: for $i = 1, ...$ do
2:     Pull policy parameters $\theta$
3:     Collect one trajectory $(s_t, a_t, s_{t+1})$ with $\pi_\theta$ in the real environment
4:     Push data $\{(s_t, a_t, s_{t+1})\}$
5: end for
```

MODEL LEARNING. In each iteration, this worker moves all trajectories from the remote data buffer, if it is not empty, to its local buffer. The local buffer is of fixed size and first-in-first-out. Then, it fits the model for one epoch on the local data buffer. Lastly it pushes model parameter ϕ to the model parameter server. Since in practice the data collection worker obtains samples at a slower pace than model training, we apply early stopping via computing validation loss on held-out samples. The training of the model stops if the an exponentially moving average of the validation loss increases after an epoch. When new samples are available, the worker resets the rolling average and starts training again. For long-horizon or low-data-frequency tasks where data collection is slow, early stopping is crucial to prevent overfitting.

Algorithm 7 Model Learning

```
1: $\mathcal{D} = \emptyset$
2: for $i = 1, ...$ do
3:     Pull samples $\{s_t, a_t\}$
4:     $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, s_{t+1})\}$
5:     Train dynamics model $\hat{p}_\phi$ for one epoch on $\mathcal{D}$
6:     Push dynamcis model parameters $\phi$
7: end for
```

POLICY IMPROVEMENT. In each iteration, the policy improvement worker first pulls from the model parameter server. Then it carries out the specific policy improvement step specified by a model-based algorithm. For instance, in the case of model-ensemble trust-region policy optimization (ME-TRPO) (Kurutach et al., 2018), this step corresponds to sampling a batch of imaginary trajectories followed by a TRPO update. Finally the worker pushes the improved policy weights θ to the policy parameter server.

Algorithm 8 Policy Improvement

```
1: Randomly initialize $\theta$
2: for $i = 1, \ldots$ do
3:     Pull model parameters $\boldsymbol{\phi}$
4:     Collect imagined samples using $\pi_\theta$
5:     Train policy for one gradient step
6:     Push policy
       parameters $\theta$
7: end for
```

Asynchronous learning offers several advantages over sequential learning. First, since the three main processes run in parallel, the running time of the algorithms is reduced to be the total sampling time. Second, since the policy is being learned while collecting data, at the beginning of each rollout a new policy is usually available to the data collection worker, resulting in more diverse data. Third, since the model and policy are learning concurrently, at each policy improvement step a new model is readily available for the policy to fit on. It prevents the policy from overfitting to the model deficiencies, similarly observed in (Luo et al., 2019). Finally, there is no need to specify crucial hyper-parameters for proper learning: number of environment rollouts, number of model epochs, or number of policy gradient steps per iteration.

8.3 RELATED WORK

In this section, we discuss related work, including asynchronous learning in the context of RL, and finally real robotic learning with RL.

Asynchronous learning. The Hogwild! algorithm (Recht et al., 2011) popularized asynchronous learning by showing that lock-free asynchronous stochastic gradient descent (SGD) is able to out perform its synchronous version. Later on, (Dean et al., 2012) demonstrated its benefits when training deep neural networks. Inspired by this, (Arun Nair et al., 2015) were the first to apply asynchronous training to deep reinforcement learning. And further work has extended these results to more efficient algorithms (Babaeizadeh et al., 2016; Espeholt et al., 2018; Heess et al., 2017; Mnih et al., 2016; Stooke and Abbeel, 2018). However, previous work has focused on model-free algorithms (Mnih et al., 2015; Schulman et al., 2017; Precup et al., 2000; Precup et al., 2001), where large amounts of data are required, and distributed data collection is crucial for fast learning. In the case of real robotics agents, however, parallel data collection requires having multiple robots, which can easily be prohibitively expensive (Gu et al., 2016a). In our asynchronous framework parallelization occurs across the different phases of model-based RL algorithms, rather than across multiple agents collaborating on one single phase such as collecting experience.

Real Robot Learning. Prior work on model-based reinforcement learning on real robotic agents has explored a diversity of schemes for dynamics learning, including Gaussian Processes (Deisenroth and Rasmussen, 2011), mixture models (Moldovan et al., 2015), and local linear models (Lioutikov et al., 2014). In this work we focus on learning dynamics model parametrized by deep neural networks, which offer the potential to scale up to higher dimensional domains and more complex tasks. While deep dynamics models has been previously used on real robots (Nagabandi et al., 2018a), it has been done using MPC type approaches. Another line of work has applied pure model-free RL (Haarnoja et al., 2018c; Haarnoja et al., 2018b; R. Hafner and M. Riedmiller, 2007; Gullapalli et al., 1992). For instance, (Gu et al., 2016a) used an asynchronous data collection method for door opening. However, model-free RL is still significantly more data inefficient than model-based methods, which hinders its applicability to general real robotics learning. The real robotic tasks attempted in this work are similar to the ones proposed by (Levine et al., 2015). In that work, however, they use a specialized method for contact rich manipulation tasks.

8.4 EXPERIMENTS

Here, we will empirically corroborate the claims in the previous sections. Specifically, the experiments are designed to address the following questions: (1) How does the learning speed of our asynchronous framework compare against sequential and model-free baselines? (2) Does asynchronous learning effectively prevent model-bias by regularizing the policy improvement step? (3) Is asynchronous data collection more effective than batch data collection? (4) Is our framework able to rapidly learn complex, real-world manipulation tasks? (5) Is our asynchronous framework brittle to data collection frequency?

To answer the posed questions, we will first evaluate our framework on four continuous control benchmark tasks in the Mujoco simulator (Todorov et al., 2012; Brockman et al., 2016). Then, we will analyze its benefits in further depth on a subset of those tasks. And finally, we will showcase its performance on several contact rich object manipulation tasks on the PR2 robot, Figure 28. The performance on all the simulated results is averaged over 4 random seeds.

8.4.1 *Wall-Clock Time Speed-up and Sample Efficiency*

We adapt our asynchronous framework to three different model-based algorithms, namely model-ensemble trust-region policy optimization (ME-TRPO) (Kurutach et al., 2018), a variant of it using proximal policy optimization (PPO) (Schulman et al., 2017) which will refer to as ME-PPO, and model-based meta-policy optimization (MB-MPO) (Clavera et al., 2018). We directly compare the performance of the sequential and the asynchronous version of each method, as well as two model-free methods TRPO (Schulman et al., 2015b) and PPO (Schulman et al., 2017). In order to simulate real-world robot experiments, where data-collection is typically the time bottleneck for RL algorithms, we report the wall-clock time that those algorithms would have taken if they were to be run in the real-world. All the experiments have a maximum path length of 200 timesteps. Hence, the time T to collect one trajectory corresponds to 200 times the control frequency, which is an attribute of the environment (Brockman et al., 2016). In the asynchronous case, since data simulation is typically much faster than real-time, the worker responsible for data collection sleeps until the time T elapses, and then starts the next step.

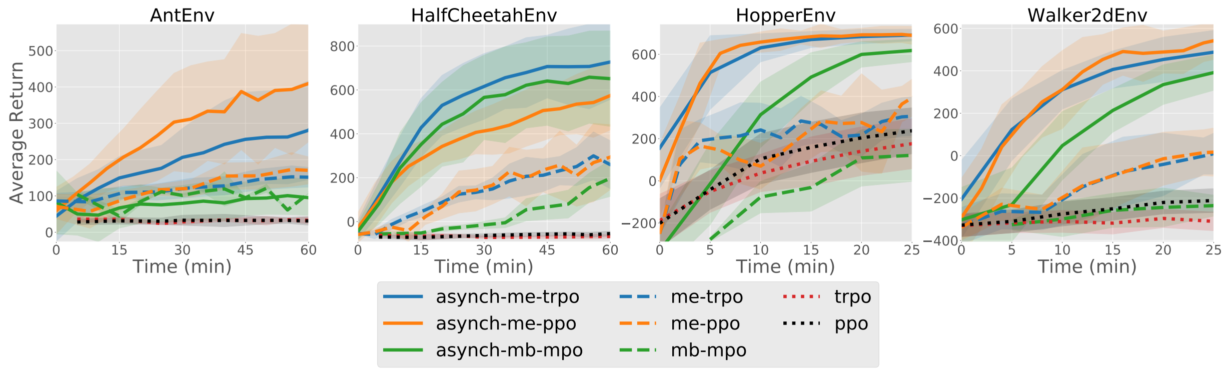


Figure 24: Wall-clock time comparison between asynchronous model-based (solid), synchronous model-based (dashed), and model-free (dotted) methods. Solid lines refer to algorithms within our asynchronous framework, and dashed to its corresponding sequential version. Asynchronous learning significantly speeds up the training time of current model-based algorithms. Best viewed in color.

Figure 24 shows the performance of the different algorithms in terms of wall-clock time. Here, we see that the asynchronous adaptations significantly speed up the training process. In general, asynch-ME-TRPO and asynch-ME-PPO converge faster than asynch-MB-MPO, and similar relative convergence speed is observed in their synchronous versions.

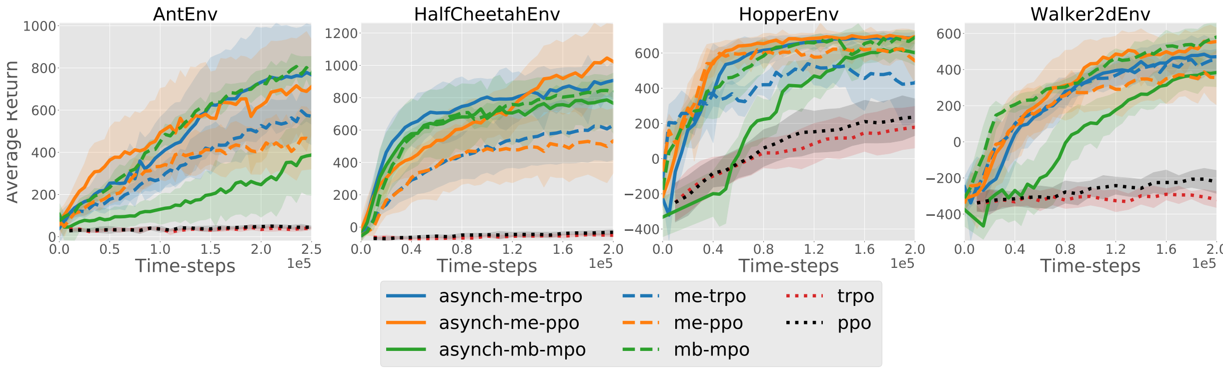


Figure 25: Sample complexity comparison between asynchronous model-based (solid), synchronous model-based (dashed), and model-free (dotted) methods. Solid lines refer to algorithms within our asynchronous framework, and dashed lines refer to corresponding sequential versions. Asynchronous learning in general offers better sample complexity than sequential synchronous learning. Best viewed in color.

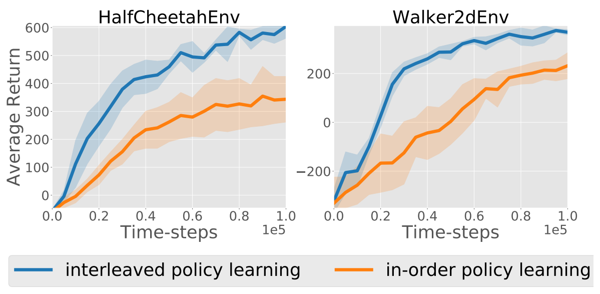
Figure 25 shows the performance of the different algorithms in terms of sample complexity. These results show that in general, asynchronous learning converges to its optimal solution faster than their corresponding synchronous methods. They suggest that the asynchronous framework enhances current model-based methods by further reducing the sample complexity of these already data-efficient algorithms.

8.4.2 *Interleaved Policy Learning and Model Learning*

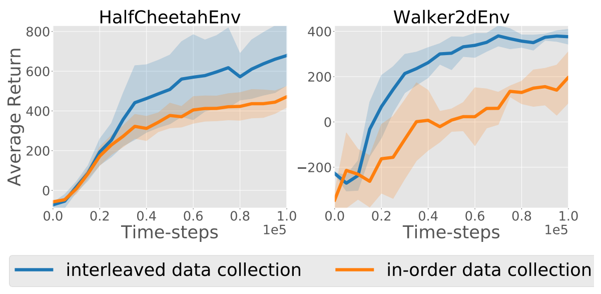
One aspect that differentiates the asynchronous framework from the synchronous one is that policy updates are interleaved with model updates, whereas in the latter case, the policy does not start taking gradient steps until all the models in the ensemble have either early stopped or reached a pre-determined maximum number of epochs. This section aims to show that such difference benefits the overall learning through policy regularization.

To remove confounding effects, we implemented a partially-asynchronous ME-TRPO , with each iteration containing two phases: First, collect N rollouts from environment; and second, alternatively fit the model ensemble for E epochs on current dataset and train the policy for G gradient steps with the updated models. The first phase inherits the implementation of synch-ME-TRPO, while the second phase mimics the asynchronous effect by updating the policy with the model parameters before the models are fully trained on the available dataset.

To test our hypothesis, we compare the aforementioned methods in two Mujoco environments, HalfCheetah and Walker2d. Figure 26a shows that the partially-asynchronous method achieves better sample-efficiency than the synchronous one. It suggests that interleaving model and policy updates, as is the case with the asynchronous framework, helps prevent the policy from overfitting to the model deficiencies.



(a) Comparison between interleaved and in-order model and policy updates. Interleaved model and policy updates, which mimics the asynchronous training, leads to better sample complexity by effectively regularizing the policy improvement steps.



(b) Comparison between interleaved and in-order policy updates and sample collection. Interleaved policy updates and data collections, which mimics the set up of asynchronous training, reduces sample complexity by collecting more diverse data.

Figure 26: Effects of asynchronous training in learning performance and sample efficiency.

8.4.3 *Interleaved Policy Learning and Data Collection*

A second aspect that distinguishes asynchronous methods is that policy learning and data collection are interleaved. That is, environment trajectories are potentially collected under policy even before the policy learner has taken sufficient gradient steps to fit to the current model.

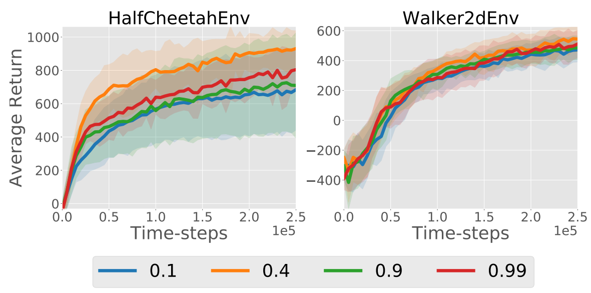
To investigate whether such a difference improves exploration for data collection, we implemented a second partially-asynchronous ME-TRPO. After acquiring an initial dataset, the trainer loops with two phases: First, fit the model to the obtained dataset; second, alternatively take G policy gradient steps and append a new sampled rollout to the dataset, for a total of N times.

This implementation is compared with synchronous ME-TRPO on HalfCheetah and Walker2d environments, as shown in Figure 26b. The result shows the advantage of asynchronous training in terms of sample-efficiency. It suggests that an asynchronous framework effectively encourages data exploration which results in learning benefits.

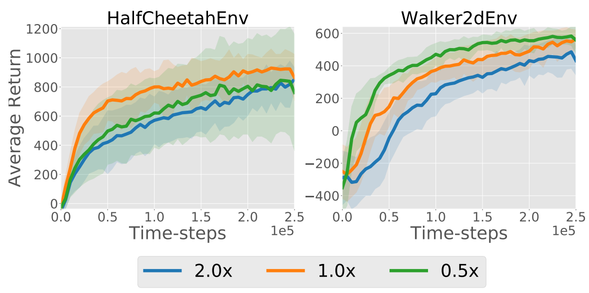
8.4.4 *Early Stopping & Sampling Speed Effect*

In this section, we first investigate the importance of integrating an early-stopping mechanism into our framework. In our framework, we stop training the model if the current validation loss is larger than the exponentially moving average of it. Second, we analyze the effect of the sampling frequency in the data efficiency of asynchronous model-based RL. We analyze these two aspects in asynchronous ME-TRPO in the environments of HalfCheetah and Walker2d.

In Figure 27a we show how different values of the weight in the exponentially moving average affect performance. In Walker2d, the framework is robust to different degree of early stopping. However, in the HalfCheetah environment, an appropriate value of early stopping leads to faster learning.



(a) Effect of the early stopping. Lower values weight values in the exponential moving average result in a more aggressive early stopping.



(b) Effect of the sampling speed. Slower data collection benefits the performance in the asynchronous framework by allowing more training on the model and the policy.

Figure 27: Early stopping and sampling speed effect on the average returns.

The length of a rollout is typically a characteristic of the problem, and thus not tunnable. Here, we compare the performance of our asynchronous training when data collection is carried out at different speeds; specifically, twice the speed and half of it, Figure 27b. The result shows that slower data collection, rather than faster, typically leads to better results. We attribute this to the fact that in the asynchronous framework, data collection speed determines the number of gradient steps taken on the model and policy. Slower data collection allows for more model and policy training. Hence in algorithms where model or policy training is particularly slow, the asynchronous framework would benefit from preventing the data collection worker to gather an excessive number of samples. A particular instance of this effect is the increase on sample complexity of asynch-MB-MPO in Section 8.4.1.

8.4.5 *Real-World Experiments*

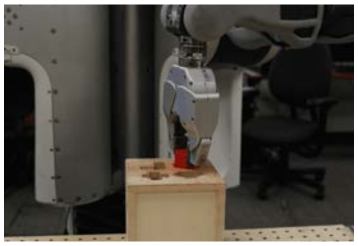
To best evaluate the real-life efficacy of our proposed asynchronous methods, we expanded our framework's domain of application to include a physical PR2 robot. In particular, we evaluated asynch-MB-MPO in three tasks: reaching a position, inserting a unique shape into its matching hole in a box, and stacking a modular block onto a fixed base. In the latter two experiments, the manipulated objects were assumed to be fixed extensions of the end-effector, allowing us to use forward kinematics to compute the object's current position. These tasks can be observed in Figures 28a-c.

The robot itself has a 23-dimensional state space composed of measurements from the active left arm: seven joint angles, seven joint velocities, and nine Cartesian points of the end-effector that determine the pose of the object. The action space was directly the torque commands for the 7-DOF arm. The actions were applied at a frequency of 10 Hz.

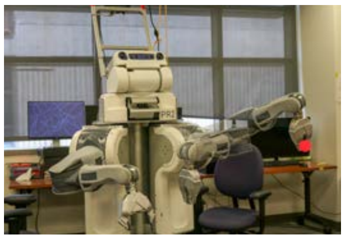
The reward function for each task was applied on the distance, d, between the current position of the end-effector and a fixed target position. As in (Levine et al., 2015), we formulated reward as a mixture of a quadratic penalty and a Lorentzian ρ-functions, i.e., $r(d) = -\omega d^2 - \nu \log(d^2 + \alpha)$, where we set $\omega = 1.0$, $\nu = 1.0$, and $\alpha = 10^{-5}$. The shape of this cost function ensures quick and precise execution of tasks. Based on this reward, we also introduced two scaled quadratic penalties on the magnitude of the joint velocities and applied torques controls to best secure a smooth performance in task completion.

The results, shown in Figure 29, show that asynchronous MB-MPO achives contact rich object manipulated tasks, such as lego stacking and shape matching, within 100 timesteps. This corresponds to 10 minutes of run time, matching similar speed performance attained in (Levine et al., 2015). Videos of our method on the real robot environment, can be found at our website.[2]

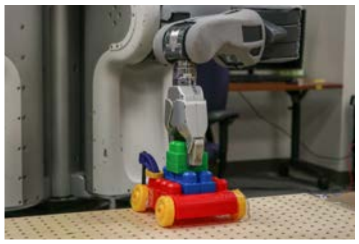
2 https://sites.google.com/view/asynch-mb-rl/home



(a) Shape matching, the PR2 robot insert shape into its matching hole.



(b) Reaching, the PR2 moves its end-effector to an pre-specified goal.



(c) Lego stacking, the PR2 assembles one lego block on top of another.

Figure 28: Tasks in our PR2 experiments.

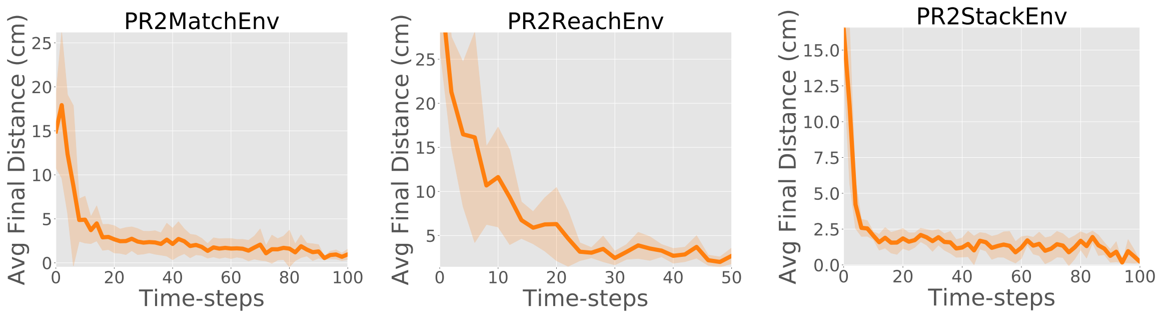


Figure 29: Final distance attained in each of the tasks evaluated on the PR2 robot with asynch-MB-MPO.

8.5 CONCLUSION

In this work we proposed an asynchronous framework for model-based reinforcement learning. Our empirical investigation shows that asynchronous model-based RL learns substantially faster than prior approaches. We characterized the key traits of asynchronous training that improves sample efficiency: policy regularization by interleaving policy learning and model learning, and better data collection by interleaving policy learning and data collection. Finally, we showcase the performance of asynchronous learning in real robotic manipulation, achieving to learn contact rich tasks within 10 min of run time.

9

LEARNING TO ADAPT IN DYNAMIC, REAL-WORLD ENVIRONMENTS THROUGH META-REINFORCEMENT LEARNING

In the previous chapters, we have developed a set of algorithms that allow for efficient, and fast automatic skill acquisition. The tasks considered so far are static, that is the environment does not change except because of the actions of the policy. While this formulation encompasses a wide range of useful applications, such as warehouse robotics, it is not sufficient for all real-world robotics: the real-world is stochastic and changing. As a result, the classical Markov models are not suitable.

In this last chapter, we present an approach for adaptive models. Our approach, allows for models that update themselves based on past timesteps. We stretch the notion of meta-learning to formulate online adaptation into its framework and learn models that can predict accurately in varying conditions. As in MB-MPO, Chapter 5, we rely on the generalization of the adaption step instead of the generalization of the model, which has been proven unsuccessful. We showcase our method on a real 6-legged robot on a varying of terrains and tasks not seen during training.

9.1 OVERVIEW

Both model-based and model-free reinforcement learning (RL) methods generally operate in one of two regimes: all training is performed in advance, producing a model or policy that can be used at test-time to make decisions in settings that approximately match those seen during training; or, training is performed online (e.g., as in the case of online temporal-difference learning), in which case the agent can slowly modify its behavior as it interacts with the environment. However, in both of these cases, dynamic changes such as failure of a robot's components, encountering a new terrain, environmental factors such as lighting and wind, or other unexpected perturbations, can cause the agent to fail. In contrast, humans can rapidly adapt their behavior to unseen physical perturbations and changes in their dynamics (Braun et al., 2009): adults can learn to walk on crutches in just a few seconds, people can adapt almost instantaneously to picking up an object that is unexpectedly heavy, and children that can walk on carpet and grass can quickly figure out how to walk on ice without having to relearn how to walk. How is this possible? If an agent has encountered a large number of perturbations in the past, it can in principle use that experience to *learn how to adapt*. In this work, we propose a meta-learning approach for learning online adaptation.

Motivated by the ability to tackle real-world applications, we specifically develop a model-based meta-reinforcement learning algorithm. In this setting, data for updating the model is readily available at every timestep in the form of recent experiences. But more crucially, the meta-training process for training such an adaptive model can be much more sample efficient than model-free meta-RL approaches (Duan et al., 2016b; J. Wang et al., 2016; Finn et al., 2017). Further, our approach foregoes the episodic framework on which model-free meta-RL approaches rely on, where tasks are pre-defined to be different rewards or environments, and tasks exist at the trajectory level only. Instead, our method considers each timestep to potentially be a new "task, " where any detail or setting could have changed at any timestep. This view induces a more general meta-RL problem setting by allowing the notion of a task to represent anything from existing in a different part of the state space, to experiencing disturbances, or attempting to achieve a new goal.

Learning to adapt a model alleviates a central challenge of model-based reinforcement learning: the problem of acquiring a global model that is accurate throughout the entire state space. Furthermore, even if it were practical to train a globally accurate dynamics model, the dynamics inherently change as a function of uncontrollable and often unobservable environmental factors, such as those mentioned above. If we have a model that can adapt online, it need not be perfect everywhere a priori. This property has previously been exploited by adaptive control methods (Åström and Wittenmark, 2013; Sastry and Isidori, 1989; Pastor et al., 2011; Meier et al., 2016); but, scaling such methods to complex tasks and nonlinear systems is exceptionally difficult. Even when working with deep neural networks, which have been used to model complex nonlinear systems (Kurutach et al., 2018), it is exceptionally difficult to enable adaptation, since such models typically require large amounts of data and many gradient steps to learn effectively. By specifically training a neural network model to require only a small amount of experience to adapt, we can enable effective online adaptation in complex environments while putting less pressure on needing a perfect global model.

The primary contribution of our work is an efficient meta reinforcement learning approach that achieves online adaptation in dynamic environments. To the best knowledge of the authors, this is the first meta-reinforcement learning algorithm to be applied in a real robotic system. Our algorithm efficiently trains a global model that is capable to use its recent experiences to quickly adapt, achieving fast online adaptation in dynamic environments. We evaluate two versions of our approach, recurrence-based adaptive learner (ReBAL) and gradient-based adaptive learner (GrBAL) on stochastic and simulated continuous control tasks with complex contact dynamics (Fig. 30). In our experiments, we show a quadrupedal "ant" adapting to the failure of different legs, as well as a "half-cheetah" robot adapting to the failure off different joints, navigating terrains with different slopes, and walking on floating platforms of varying buoyancy. Our model-based meta RL method attains substantial improvement over prior approaches, including standard model-based methods, online model-adaptive methods, model-free methods, and prior meta-reinforcement learning methods, when trained with similar amounts of data. In all experiments, meta-training across multiple tasks is sample efficient, using only the equivalent of $1.5-3$ hours of real-world experience, roughly $10\times$ less than what model-free methods require to learn a single task. Finally, we demonstrate GrBAL on a real dynamic legged millirobot (see Fig 30). To highlight not only the sample efficiency of our meta model-based reinforcement learning approach, but also the importance of fast online adaptation in the real world, we show the agent's learned ability to adapt online to tasks such as a missing leg, novel terrains and slopes, miscalibration or errors in pose estimation, and new payloads to be pulled.

9.2 METHOD

9.2.1 *Meta-Learning for Online Model Adaptation*

In this section, we present our approach for meta-learning for online model adaptation. As explained in Section 2.3, standard meta-learning formulations require the learned model $\boldsymbol{\phi}_*, \boldsymbol{\psi}_*$ to learn using M data points from some new "task." In prior gradient-based and model-based meta-RL approaches (Finn et al., 2017; Sæmundsson et al., 2018), the M has corresponded to M trajectories, leading to episodic adaptation.

Our notion of task is slightly more fluid, where every segment of a trajectory can be considered to be a different "task," and observations from the past T_1 *timesteps* (rather than the past episodes) can be considered as providing information about the current task setting. Since changes in system dynamics, terrain details, or other environmental changes can occur at any time, we consider (at every time step) the problem of adapting the model using the T_1 past time steps to predict the next T_2 timesteps. In this setting, T_1 and T_2 are pre-specified hyperparameters; see appendix for a sensitivity analysis of these parameters.

In this work, we use the notion of environment $\mathcal{M}$ to denote different settings or configurations of a particular problem, ranging from malfunctions in the system's joints to the state of external disturbances. We assume a distribution of environments $\rho(\mathcal{M})$ that share some common structure, such as the same observation and action space, but may differ in their dynamics $p_{\mathcal{M}}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$. We denote a trajectory segment by $\boldsymbol{\tau}_{\mathcal{M}}[i : j]$, which represents a sequence of states and actions $(\mathbf{s}_i, \mathbf{a}_i, ..., \mathbf{s}_j, \mathbf{a}_j, \mathbf{s}_{j+1})$ sampled within an environment $\mathcal{M}$. Our algorithm assumes that the environment is locally consistent, in that every segment of length $j - i$ has the same environment. Even though this assumption is not always correct, it allows us to learn to adapt from data without knowing when the environment has changed. Due to the fast nature of our adaptation (less than a second), this assumption is seldom violated.

We pose the meta-RL problem in this setting as an optimization over $(\boldsymbol{\phi}, \boldsymbol{\psi})$ with respect to a maximum likelihood meta-objective. The meta-objective is the likelihood of the data under a predictive model $\hat{p}_{\boldsymbol{\phi}'}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ with parameters $\boldsymbol{\phi}'$, where $\boldsymbol{\phi}' = u_{\boldsymbol{\psi}}(\boldsymbol{\tau}_{\mathcal{M}}[t - T_1, t], \boldsymbol{\phi})$ corresponds to model parameters that were updated using the past T_1 data points. Concretely, this corresponds to the following optimization:

$$\min_{\boldsymbol{\phi},\boldsymbol{\psi}} \mathbb{E}_{\boldsymbol{\tau}_{\mathcal{M}}[]t-T_1:t+T_2]\sim\mathcal{D}}\left[L(\boldsymbol{\tau}_{\mathcal{M}}[t : t + T_2], \boldsymbol{\phi'}_{\mathcal{M}})\right] \quad \text{s.t.:} \quad \boldsymbol{\phi'}_{\mathcal{M}} = u_{\boldsymbol{\psi}}(\boldsymbol{\tau}_{\mathcal{M}}[t - T_1 : t], \boldsymbol{\phi}), \tag{22}$$

In that $\boldsymbol{\tau}_{\mathcal{M}}[t - T_1, t + T_2) \sim \mathcal{D}$ corresponds to trajectory segments sampled from our previous experience, and the loss L corresponds to the negative log likelihood of the data under the model:

$$\mathcal{L}(\boldsymbol{\tau}_{\mathcal{M}}[t : t + T_2], \boldsymbol{\phi'}_{\mathcal{M}}) \triangleq -\frac{1}{T_2} \sum_{t'=t}^{t+T_2} \log \hat{p}_{\boldsymbol{\phi'}_{\mathcal{M}}}(\mathbf{s}_{t'+1}|\mathbf{s}_{t'}, \mathbf{a}_{t'}). \tag{23}$$

In the meta-objective in Equation 22, note that the past T_1 points are used to adapt $\boldsymbol{\phi}$ into $\boldsymbol{\phi}'$, and the loss of this $\boldsymbol{\phi}'$ is evaluated on the future T_2 points. Thus, we use the past T_1 timesteps to provide insight into how to adapt our model to perform well for nearby future timesteps. As outlined in Algorithm 9, the update rule $u_{\boldsymbol{\psi}}$ for the inner update and a gradient step on $\boldsymbol{\phi}$ for the outer update allow us to optimize this meta-objective of adaptation. Thus, we achieve fast adaptation at test time by being able to fine-tune the model using just T_1 data points.

While we focus on reinforcement learning problems in our experiments, this meta-learning approach could be used for a learning to adapt online in a variety of sequence modeling domains. We present our algorithm using both a recurrence and a gradient-based meta-learner, as we discuss next.

Gradient-Based Adaptive Learner (GrBAL). GrBAL uses a gradient-based meta-learning to perform online adaptation; in particular, we use MAML (Finn et al., 2017). In this case, our update rule is prescribed by gradient descent (24.)

$$\phi'_{\mathcal{M}} = u_{\psi}(\tau_{\mathcal{M}}[t - T_1 : t], \phi) = \phi_{\mathcal{M}} + \psi \nabla_{\phi} \frac{1}{T_1} \sum_{t'=t-T_1}^{t-1} \log \hat{p}_{\phi_{\mathcal{M}}}(s_{t'+1} | s_{t'}, a_{t'}) \tag{24}$$

Recurrence-Based Adaptive Learner (ReBAL). ReBAL, instead, utilizes a recurrent model, which learns its own update rule (i.e., through its internal gating structure). In this case, ψ and u_{ψ} correspond to the weights of the recurrent model that update its hidden state.

Algorithm 9 Model-Based Meta-Reinforcement Learning (train time)

```
Require: Distribution $\rho_{\mathcal{M}}$ over tasks
Require: Learning rate $\beta \in \mathbb{R}^+$
Require: Number of sampled tasks N, dataset $\mathcal{D}$
Require: Task sampling frequency $n_S \in \mathbb{Z}^+$
 1: Randomly initialize $\phi$
 2: for $i = 1, \ldots$ do
 3:     if $i \bmod n_S = 0$ then
 4:         Sample $\mathcal{M} \sim \rho(\mathcal{M})$
 5:         Collect $\tau_{\mathcal{M}}$ using Alg. 10
 6:         $\mathcal{D} \leftarrow \mathcal{D} \cup \{\tau_{\mathcal{M}}\}$
 7:     end if
 8:     for $j = 1 \ldots N$ do
 9:         $\tau_{\mathcal{M}}[t - T_1 : t - 1], \tau_{\mathcal{M}}[t : t + T_2] \sim \mathcal{D}$
10:         $\phi'_{\mathcal{M}} \leftarrow u_{\psi}(\tau_{\mathcal{M}}[t - T_1 : t], \phi)$
11:         $L_j \leftarrow L(\tau_{\mathcal{M}}[t : t + T_2], \phi'_{\mathcal{M}})$
12:     end for
13:     $\phi \leftarrow \phi - \beta \nabla_{\phi} \frac{1}{N} \sum_{j=1}^{N} L_j$
14:     $\psi \leftarrow \psi - \eta \nabla_{\psi} \frac{1}{N} \sum_{j=1}^{N} L_j$
15: end for
16: Return $(\phi, \psi)$ as $(\phi_*, \psi_*)$
```

9.2.2 *Meta-Online Adaptation for Control*

Now that we have discussed our approach for enabling online adaptation, we next propose how to build upon this idea to develop a model-based meta-reinforcement learning algorithm. First, we explain how the agent can use the adapted model to perform a task, given parameters $\boldsymbol{\phi}_*$ and $\boldsymbol{\psi}_*$ from optimizing the meta-learning objective.

Given $\boldsymbol{\phi}_*$ and $\boldsymbol{\psi}_*$, we use the agent's recent experience to adapt the model parameters: $\boldsymbol{\phi}'_* = u_{\boldsymbol{\psi}_*}(\boldsymbol{\tau}[t - T_1 : t], \boldsymbol{\phi}_*)$. This results in a model $\hat{p}_{\theta'_*}$ that better captures the local dynamics in the current setting, task, or environment. This adapted model is then passed to our controller, along with the reward function r and a planning horizon H. We use a planning H that is smaller than the adaptation horizon T_2, since the adapted model is only valid within the current context. We use model predictive path integral control (MPPI) (Williams et al., 2015), but, in principle, our model adaptation approach is agnostic to the model predictive control (MPC) method used.

Algorithm 10 Online Model Adaptation
(test time)

```
Require: Meta-learned parameters $\boldsymbol{\phi}_*, \boldsymbol{\psi}_*$
Require: controller(), $H$, $r$, $n_A$
1: $\mathcal{D} \leftarrow \emptyset$
2: for each timestep $t$ do
3:     $\boldsymbol{\phi}'_* \leftarrow u_{\boldsymbol{\psi}_*}(\mathcal{D}[t - T_1 : t], \boldsymbol{\phi}_*)$
4:     $\mathbf{a} \leftarrow$ controller($\boldsymbol{\phi}'_*, r, H, n_A$)
5:     Execute $\mathbf{a}$, add result to $\mathcal{D}$
6: end for
7: Return rollout $\mathcal{D}$
```

The use of MPC compensates for model inaccuracies by preventing accumulating errors, since we replan at each time step using updated state information. MPC also allows for further benefits in this setting of online adaptation, because the model $\hat{p}_{\boldsymbol{\phi}'_{\mathcal{M}}}$ itself will also improve by the next time step. After taking each step, we append the resulting state transition onto our dataset, reset the model parameters back to $\boldsymbol{\phi}_*$, and repeat the entire planning process for each timestep. See Algorithm 10 for this adaptation procedure. Finally, in addition to test-time, we also perform this online adaptation procedure during the meta-training phase itself, to provide on-policy rollouts for meta-training. For the complete meta-RL algorithm, see Algorithm 9.

9.3 RELATED WORK

Prior online adaptation approaches (Tanaskovic et al., 2013; Aswani et al., 2012) have aimed to learn an approximate global model and then adapt it at test time. Dynamic evaluation algorithms (Rei, 2015; Krause et al., 2017; Krause et al., 2016; Fortunato et al., 2017), for example, learn an approximate global distribution at training time and adapt those model parameters at test time to fit the current local distribution via gradient descent. There exists extensive prior work on online adaptation in model-based reinforcement learning and adaptive control (Sastry and Isidori, 1989). In contrast from inverse model adaptation (Kelouwani et al., 2012; Underwood and Husain, 2010; Pastor et al., 2011; Meier et al., 2016; Meier and Schaal, 2016; Rai et al., 2017), we are concerned in the problem of adapting the forward model, closely related to online system identification (Manganiello et al., 2014). Work in model adaptation (Levine and Koltun, 2013; Gu et al., 2016b; Fu et al., 2016; Weinstein and Botvinick, 2017) has shown that a perfect global model is not necessary, and prior knowledge can be fine-tuned to handle small changes. These methods, however, face a mismatch between what the model is trained for and how it is used at test time. In this paper, we bridge this gap by explicitly training a model for fast and effective adaptation. As a result, our model achieves more effective adaptation compared to these prior works, as validated in our experiments.

Our problem setting relates to meta-learning, a long-standing problem of interest in machine learning that is concerned with enabling artificial agents to efficiently learn new tasks by learning to learn (Thrun and Pratt, 1998; Schmidhuber and Huber, 1991; Naik and Mammone, 1992; Lake et al., 2015). A meta-learner can control learning through approaches such as deciding the learner's architecture (B. Baker et al., 2016), or by prescribing an optimization algorithm or update rule for the learner (Bengio et al., 1990; Schmidhuber, 1992; Younger et al., 2001; Andrychowicz et al., 2016; K. Li and Malik, 2016; Ravi and Larochelle, 2018). Another popular meta-learning approach involves simply unrolling a recurrent neural network (RNN) that ingests the data (Santoro et al., 2016; Munkhdalai and Yu, 2017; Munkhdalai et al., 2017; Mishra et al., 2018) and learns internal representations of the algorithms themselves, one instantiation of our approach (ReBAL) builds on top of these methods. On the other hand, the other instantiation of our method (GrBAL) builds on top of MAML (Finn et al., 2017). GrBAL differs from the supervised version of MAML in that MAML assumes access to a hand-designed distribution of tasks. Instead, one of our primary contributions is the online formulation of meta-learning, where tasks correspond to temporal segments, enabling "tasks" to be constructed automatically from the experience in the environment.

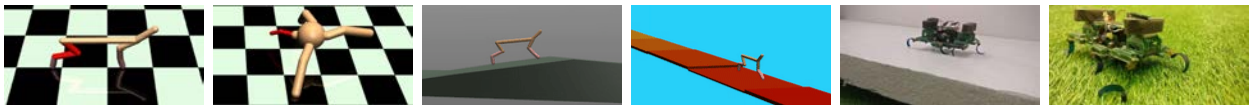


Figure 30: Two real-world and four simulated environments on which our method is evaluated and adaptation is crucial for success (e.g., adapting to different slopes and leg failures)

Meta-learning in the context of reinforcement learning has largely focused on model-free approaches (Duan et al., 2016b; J. Wang et al., 2016; Sung et al., 2017; Al-Shedivat et al., 2017). However, these algorithms present even more (meta-)training sample complexity than non-meta model-free RL methods, which precludes them from real-world applications. Recent work (Sæmundsson et al., 2018) has developed a model-based meta RL algorithm, framing meta-learning as a hierarchical latent variable model, training for episodic adaptation to dynamics changes; the modeling is done with GPs, and results are shown on the cart-pole and double-pendulum agents. In contrast, we propose an approach for learning online adaptation of high-capacity neural network dynamics models; we present two instantiations of this general approach and show results on both simulated agents and a real legged robot.

9.4 EXPERIMENTS

Our evaluation aims to answer the following questions: (1) Does our approach enable fast adaptation to varying dynamics, tasks, and environments, both inside and outside of the training distribution? (2) How does our method's performance compare to that of other methods? (3) How do GrBAL and ReBAL compare? (4) How does meta model-based RL compare to meta model-free RL in terms of sample efficiency and performance for these experiments? (5) Can our method learn to adapt online on a real robot, and if so, how does it perform? We next present our set-up and results, motivated by these questions. Videos are available online[1], and further analysis is provided in the appendix.

We first conduct a comparative evaluation of our algorithm, on a variety of simulated robots using the MuJoCo physics engine (Todorov et al., 2012). For all of our environments, we model the transition probabilities as Gaussian random variables with mean parameterized by a neural network model (3 hidden layers of 512 units each and ReLU activations) and fixed variance. In this case, maximum likelihood estimation corresponds to minimizing the mean squared error. We now describe the setup of our environments (Fig. 30), where each agent requires different types of adaptation to succeed at run-time:

1 Videos available at: https://sites.google.com/berkeley.edu/metaadaptivecontrol

Half-cheetah (HC): disabled joint. For each rollout during meta-training, we randomly sample a joint to be disabled (i.e., the agent cannot apply torques to that joint). At test time, we evaluate performance in two different situations: disabling a joint unseen during training, and switching between disabled joints during a rollout. The former examines extrapolation to out-of-distribution environments, and the latter tests fast adaptation to changing dynamics.

HC: sloped terrain. For each rollout during meta-training, we randomly select an upward or downward slope of low steepness. At test time, we evaluate performance on unseen settings including a gentle upward slope, a steep upward slope, and a steep hill that first goes up and then down.

HC: pier. In this experiment, the cheetah runs over a series of blocks that are floating on water. Each block moves up and down when stepped on, and the changes in the dynamics are rapidly changing due to each block having different damping and friction properties. The HC is meta-trained by varying these block properties, and tested on a specific (randomly-selected) configuration of properties.

Ant: crippled leg. For each meta-training rollout, we randomly sample a leg to cripple on this quadrupedal robot. This causes unexpected and drastic changes to the underlying dynamics. We evaluate this agent at test time by crippling a leg from outside of the training distribution, as well as transitioning within a rollout from normal operation to having a crippled leg.

In the following sections, we evaluate our model-based meta-RL methods (GrBAL and ReBAL) in comparison to several prior methods:

- **Model-free RL (TRPO)**: To evaluate the importance of adaptation, we compare to a model-free RL agent that is trained across environments $\mathcal{E} \sim \rho(\mathcal{E})$ using TRPO (Schulman et al., 2015b).
- **Model-free meta-RL (MAML-RL)**: We compare to a state-of-the-art model-free meta-RL method, MAML-RL (Finn et al., 2017).
- **Model-based RL (MB)**: Similar to the model-free agent, we also compare to a single model-based RL agent, to evaluate the importance of adaptation. This model is trained using supervised model-error and iterative model bootstrapping.
- **Model-based RL with dynamic evaluation (MB+DE)**: We compare to an agent trained with model-based RL, as above. However, at test time, the model is adapted by taking a gradient step at each timestep using the past T_1 observations, akin to dynamic evaluation (Krause et al., 2017). This final comparison evaluates the benefit of explicitly training for adaptability.

All model-based approaches (MB, MB+DE, GrBAL, and ReBAL) use model bootstrapping, use the same neural network architecture, and use the same planner within experiments: MPPI (Williams et al., 2015) for the simulated experiments and random shooting (RS) (Nagabandi et al., 2017) for the real-world experiments.

9.4.1 *Effect of Adaptation*

First, we analyze the effect of the model adaptation, and show results from test-time runs on three environments : HC pier, HC sloped terrain with a steep up/down hill, and ant crippled leg with the chosen leg not seen as crippled during training.

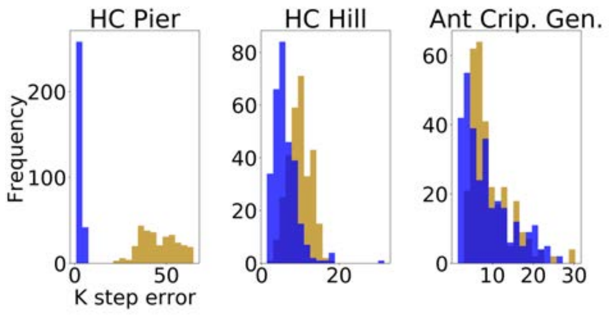


Figure 31: Histogram of normalized T_2-step model prediction errors of GrBAL, showing the improvement of the post-update model's predictions over the pre-update ones.

Figure 31 displays the distribution shift between the pre-update and post-update model prediction errors of three GrBAL runs, showing that using the past T_1 timesteps to update $\boldsymbol{\phi}_*$ (pre) into $\boldsymbol{\phi}'_*$ (post) does indeed reduce model error on predicting the following T_2 timesteps.

9.4.2 *Performance and Meta-training Sample Efficiency*

We first study the sample efficiency of the meta-training process. Figure 32 shows the average return across test environments w.r.t. the amount of data used for meta-training. We (meta-)train the model-free methods (TRPO and MAML-RL) until convergence, using the equivalent of about two days of real-world experience. In contrast, we meta-train the model-based methods (including our approach) using the equivalent of 1.5-3 hours of real-world experience. Our methods result in superior or equivalent performance to the model-free agent that is trained with 1000 times more data. Our methods also surpass the performance of the non-meta-learned model-based approaches. Finally, our performance closely matches the high asymptotic performance of the model-free meta-RL method for half-cheetah disabled, and achieves a suboptimal performance for ant crippled but, again, it does so with the equivalent of 1000 times less data. Note that this suboptimality in asymptotic performance is a known issue with model-based methods, and thus an interesting direction for future efforts. The improvement in sample efficiency from using model-based methods matches prior findings (Deisenroth and Rasmussen, 2011; Nagabandi et al., 2017; Kurutach et al., 2018); the most important evaluation, which we discuss in more detail next, is the ability for our method to adapt online to drastic dynamics changes in only a handful of timesteps.

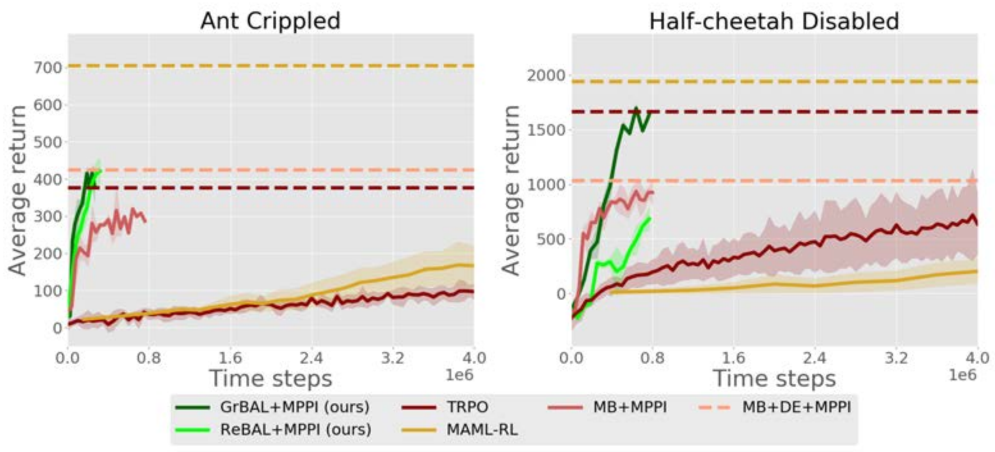


Figure 32: Compared to model-free RL, model-free meta-RL, and model-based RL methods, our model-based meta-RL methods achieve good performance with 1000× less data. Dotted lines indicate performance at convergence. For MB+DE+MPPI, we perform dynamic evaluation at test time on the final MB+MPPI model.

9.4.3 *Test-time Performance: Online Adaptation & Generalization*

In our second comparative evaluation, we evaluate final test time performance both GrBAL and ReBAL in comparison to the aforementioned methods. In the interest of developing efficient algorithms for real-world applications, we operate all methods in the low data regime for all experiments: the amount of data available (meta-)training is fixed across methods, and roughly corresponds to 1.5-3 hours of real-world experience depending on the domain. We also provide the performance of a MB oracle, which is trained using unlimited data from only the given test environment (rather than needing to generalize to various training environments).

In these experiments, note that all agents were meta-trained on a distribution of tasks/environments (as detailed above), but we then evaluate their adaptation ability on unseen environments at test time. We test the ability of each approach to adapt to sudden changes in the environment, as well as to generalize beyond the training environments. We evaluate the fast adaptation (F.A.) component on the HC disabled joint, ant crippled leg, and the HC pier. On the first two, we cause a joint/leg of the robot to malfunction in the middle of a rollout. We evaluate the generalization component also on the tasks of HC disabled joint and ant crippled leg, but this time, the leg/joint that malfunctions has not been seen as crippled during training. The last environment that we test generalization on is the HC slopped terrain for a hill, where the agent has to run up and down a steep slope, which is outside of the gentle slopes that it experienced during training. The results, shown in Fig. 33, show returns that are normalized such that the MB oracle achieves a return of 1.

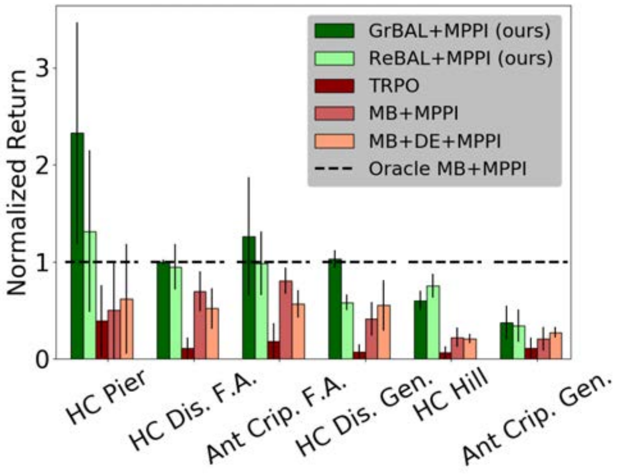


Figure 33: Simulated results in a variety of dynamic test environments. GrBAL outperforms other methods, even the MB oracle, in all experiments where fast adaptation is necessary. These results highlight the difficulty of training a global model, and the importance of adaptation.

In all experiments, due to low quantity of training data, TRPO performs poorly. Although MB+DE achieves better generalization than MB, the slow nature of its adaptation causes it to fall behind MB in the environments that require fast adaptation. On the other hand, our approach surpasses the other approaches in all of the experiments. In fact, in the HC pier and the fast adaptation of ant environments, our approach surpasses the model-based oracle. This result showcases the importance of adaptation in stochastic environments, where even a model trained with a lot of data cannot be robust to unexpected occurrences or disturbances. ReBAL displays its strengths on scenarios where longer sequential inputs allow it to better asses current environment settings, but overall, GrBAL seems to perform better for both generalization and fast adaptation.

9.4.4 *Real-World Results*

To test our meta model-based RL method's sample efficiency, as well as its ability to perform fast and effective online adaptation, we applied GrBAL to a real legged millirobot, comparing it to model-based RL (MB) and model-based RL with dynamic evaluation (MB+DE). Due to the cost of running real robot experiments, we chose the better performing method (i.e., GrBAL) to evaluate on the real robot.

This small 6-legged robot, as shown in Fig. 1 and Fig. 30, presents a modeling and control challenge in the form of highly stochastic and dynamic movement. This robot is an excellent candidate for online adaptation for many reasons: the rapid manufacturing techniques and numerous custom-design steps used to construct this robot make it impossible to reproduce the same dynamics each time, its linkages and other body parts deteriorate over time, and it moves very quickly and dynamically with The state space of the robot is a 24-dimensional vector, including center of mass positions and velocities, center of mass pose and angular velocities, back-EMF readings of motors, encoder readings of leg motor angles and velocities, and battery voltage. We define the action space to be velocity setpoints of the

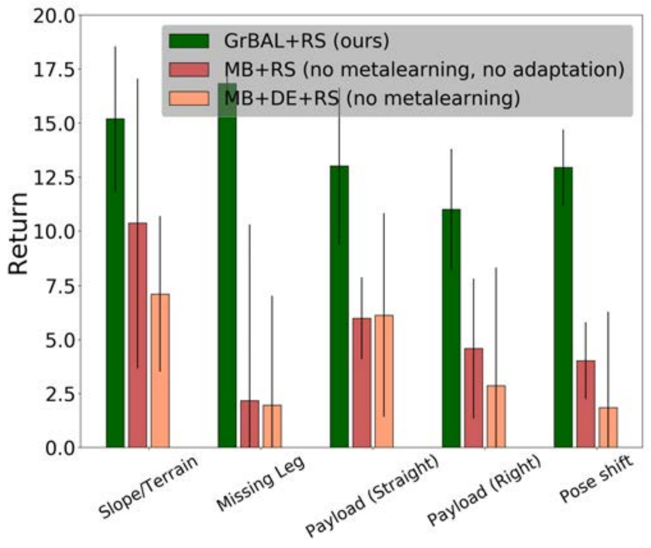


Figure 34: GrBAL clearly outperforms both MB and MB+DE, when tested on environments that (1) require online adaptation, and/or (2) were never seen during training.

		Left	Str	Z-z	F-8
Carpet	GrBAL	4.07	3.26	7.08	5.28
	MB	3.94	3.26	6.56	5.21
Styrofoam	GrBAL	3.90	3.75	7.55	6.01
	MB	4.09	4.06	7.48	6.54
Turf	GrBAL	1.99	1.65	2.79	3.40
	MB	1.87	1.69	3.52	2.61

Table 2: Trajectory following costs for real-world GrBAL and MB results when tested on three terrains that were seen during training. Tested here for left turn (Left), straight line (Str), zig-zag (Z-z), and figure-8 shapes (F-8). The methods perform comparably, indicating that online adaptation is not needed in the training terrains, but including it is not detrimental.

rotating legs. The action space has a dimension of two, since one motor on each side is coupled to all three of the legs on that side. All experiments are conducted in a motion capture room. Computation is done on an external computer, and the velocity setpoints are streamed over radio at 10 Hz to be executed by a PID controller on the microcontroller on-board of the robot.

We meta-train a dynamics model for this robot using the meta-objective described in Equation 22, and we train it to adapt on entirely real-world data from three different training terrains: carpet, styrofoam, and turf. We collect approximately 30 minutes of data from each of the three training terrains. This data was entirely collected using a random policy, in conjunction with a safety policy, whose sole purpose was to prevent the robot from exiting the area of interest.

Our first group of results (Table 2) show that, when data from a random policy is used to train a dynamics model, both a model trained with a standard supervised learning objective (MB) and a GrBAL model achieve comparable performance for executing desired trajectories on terrains from the training distribution.

Next, we test the performance of our method on what it is intended for: fast online adaptation of the learned model to enable successful execution of new, changing, or out-of-distribution environments at test time. Similar to the comparisons above, we compare GrBAL to a model-based method (MB) that involves neither meta-training nor online adaptation, as well as a dynamic evaluation method that involves online adaptation of that MB model (MB+DE). Our results (Fig. 6) demonstrate that GrBAL substantially outperforms MB and MB+DE, and, unlike MB and MB+DE, and that GrBAL can quickly

1) adapt online to a missing leg, 2) adjust to novel terrains and slopes, 3) account for miscalibration or errors in pose estimation, and 4) compensate for pulling payloads.

None of these environments were seen during training time, but the agent's ability to learn how to learn enables it to quickly leverage its prior knowledge and fine-tune to adapt to new environments online. Furthermore, the poor performance of the MB and MB+DE baselines demonstrate not only the need for adaptation, but also the importance of good initial parameters to adapt from (in this case, meta-learned parameters). The qualitative results of these experiments in Fig. 35 show that the robot is able to use our method to adapt online and effectively follow the target trajectories, even in the presence of new environments and unexpected perturbations at test time.

bounding-style gaits; hence, its dynamics are strongly dependent on the terrain or environment at hand.

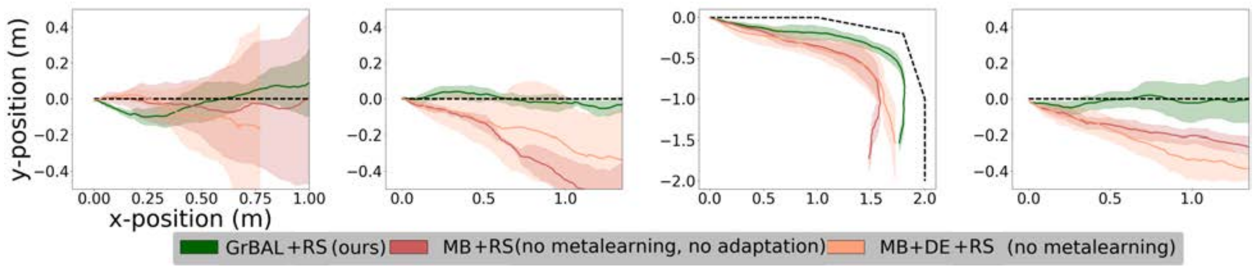


Figure 35: The dotted black line indicates the desired trajectory in the xy plane. By effectively adapting online, our method prevents drift from a missing leg, prevents sliding sideways down a slope, accounts for pose miscalibration errors, and adjusts to pulling payloads (left to right). Note that none of these tasks/environments were seen during training time, and they require fast and effective online adaptation for success.

9.5 CONCLUSION

In this work, we present an approach for model-based meta-RL that enables fast, online adaptation of large and expressive models in dynamic environments. We show that meta-learning a model for online adaptation results in a method that is able to adapt to unseen situations or sudden and drastic changes in the environment, and is also sample efficient to train. We provide two instantiations of our approach (ReBAL and GrBAL), and we provide a comparison with other prior methods on a range of continuous control tasks. Finally, we show that (compared to model-free meta-RL approaches), our approach is practical for real-world applications, and that this capability to adapt quickly is particularly important under complex real-world dynamics.

10

CONCLUSION

In this thesis, we have considered the problem of efficient autonomous skill acquisition, aiming to find methods that can make a maximal use of the data available to obtain optimal policies. We have focused our effort in model-based methods, which have the potential to solve such problem. First, we defined the model-based paradigm and dissected the problem of “model-bias," which explains the low performance of prior model-based methods. Then, we presented model-ensemble trust-region policy optimization, or ME-TRPO, that solves the issue of over-optimism in model-bias, allowing to train performant and sample efficient policies in simple domains by learning an ensemble of models that characterize the epistemic uncertainty. In order to scale ME-TRPO to more complex domains, we had to tackle the approximation and compounding error present in the model-bias problem. However, these errors are intrinsic of any machine learning system. We present a method that exploits such errors to learn an adaptive policy to different dynamics, more errors then correspond to more adaptability. At test time, the policy is adapted to the real-world dynamics. This is the idea behind model-based via meta-policy optimization (MB-MPO), a method that maintains the sample complexity of ME-TRPO while scaling to complex domains. We further enhance MB-MPO by developing a meta-learning algorithm that is able to learn the exploration strategy needed to identify the task, or dynamics, in order to adapt successfully. Finally, we present a method that efficiently tackles the compounding error problem in model-bias by truncating the predictions of the model with a value function. This last algorithm, model-augmented actor-critic, is able to efficiently scale to complex environment by making use of the derivatives of the learned model.

After presenting these two methods that tackle model-bias, we develop an asynchronous framework for fast policy learning. Enabling not only efficient, but also fast skill learning will expand the applicability of robotic systems in our daily lives. The asynchronous framework presented is tailored to model-based methods, reducing the wall-clock training of these algorithm by a factor of 10. We use this framework to learn a block stacking policy within 10 min, and a shape matching policy within 3 min in the real PR2 robot. Finally, we emphasize the difference between the studied environments and the real-world. Contrary to the environments studied so far, the real-world is stochastic, partially observable and changing. In the last chapter, we present a method that enables model-based algorithms to adapt online when faced to changes in the environment. This is accomplished by the means of meta-reinforcement learning, when each task is considered to be a sequence of state and actions. Our algorithm presents out-of-distribution adaptation, as we show in a real 6-legged robot that has been trained in different terrains, but tested in a variety of task such as different payload, leg removal, or walking different inclinations of the terrain.

10.1 RELATED WORK SINCE ORIGINAL PUBLICATION OF THIS WORK

In here, we provide the related work on model-based reinforcement learning since the publication of the first work in this thesis: model-ensemble trust-region policy optimization. Specifically, we describe the areas of (1) tackling model-bias, and (2) online adaptation in model-based RL.

Tackling model-bias. Beyond the works presented in this thesis, there has been several of work since the publication of model-ensemble trust-region policy optimization (Chapter 4). For instance, (Luo et al., 2019) uses a theoretically analyzes the ME-TRPO set up and develops a more sample efficient algorithm from the findings. In the context, of combining model-based methods and model-free actor critic algorithms, (Feinberg et al., 2018b; Buckman et al., 2018) use the model to learn better estimates for the value function. The method presented in Chapter 7 resembles to (Feinberg et al., 2018b); however, they do not make use the gradients provided by the model to train the policy. (Kalweit and Boedecker, 2017) tackles the model-bias problem by interpolating between model and real data. Finally, exciting results have been obtained when planning with the model; specifically, when using model predictive control (MPC). The first of those results was presented in (Chua et al., 2018), which learned an ensemble of stochastic model to combat the over-optimism challenge. (Lowrey et al., 2018; Hong et al., 2019; T. Wang and Ba, 2019) have followed that line of work by using value functions and distillation of the MPC controller.

Online adaptation. Nagabandi et al. (2018b) formulated the online learning set up presented in Chapter 9 as an expectation maximization algorithm with a Chinese restaurant process, allowing the models to adapt as necessary while retaining old models in case a previously task is encountered again. In order to over come the limited expressivity that model-agnostic meta-learning presents in practice and to capture the uncertainty on the dynamics (M. Xu et al., 2020), represents each type of dynamics with a Gaussian process. Guassian processes combined with latenet variables have been used by (Sæmundsson et al., 2018) for online adaptation. Similarly, (K. Lee et al., 2020) uses a latent vector, which extracts tasks specific information, that acts as a conditional variable for a neural network dynamic model. Finally, (Song et al., 2020) draws inspiration from no-regret online learning (Shalev-Shwartz, 2012) to develop a provably efficient algorithm.

10.2 FUTURE WORK

While these methods are a stepping-stone towards enabling general autonomous robotic learning, there is a lot more that can be done. Below, we speculate on important directions for future work:

Long horizon predictions. While model-based via meta-policy adaptation (Chapter 5) is able to learn longer horizon tasks than ME-TRPO (Chapter 4), it is limited to trajectories around a thousand steps in simple domains. For longer horizons or more complex environments, learning becomes unstable. To carry out real-world complex tasks, long horizon planning or reasoning is needed. In such cases, a promising avenue of work is learning models at different temporal-scales, or hierarchical models (G. Lee et al., 2015; Kaelbling et al., 1996; Stulp and Schaal, 2011). Another successful alternative for long horizon tasks in the recent years has been to combine model-based methods with actor critic ones (Janner et al., 2019; Clavera et al., 2020)

Learning from other sensory inputs. The work presented in this thesis learns models from positions and velocities, and exciting area of research is to successfully learn model from other sensor modalities. Recent work has aimed to extend model-based RL to raw sensory high-dimensional input spaces, such as images. In such cases, learning a compact and accurate latent space is crucial to relieve the dynamics model from directly modeling the raw sensor input space. Recent work on high-dimensional observations on model-based RL can be categorized in two main classes: 1) video prediction models (Ebert et al., 2018; Jayaraman et al., 2018), and 2) latent space learning with reconstruction (D. Hafner et al., 2018; Ha and Schmidhuber, 2018; Watter et al., 2015; Wahlström et al., 2015; M. Zhang et al., 2018). While progress has been made, current MB methods from images still do not achieve similar levels of sample efficiency or performance than state-based methods. Successfully deploying model-based algorithms from high-dimensional observations will release the burden of running pose estimation.

Life long learning. Being able to continually learn and adapt to new situations is a hallmark of human intelligence. The world is too stochastic and variable for us to describe or train an agent on all the possible situations it might encounter. We need algorithms that enable our robotic agents to keep learning during deployment (Parisi et al., 2018). Chapter 9 offers an approach for it. However, there is still a long way to go. The algorithm presented optimizes at every step for adaptation, which in most of the scenarios is not needed. Furthermore, the performance of the algorithm in static tasks is well bellow MB-MPO.

BIBLIOGRAPHY

Abbeel, Pieter, Adam Coates, and Andrew Y. Ng (Nov. 2010). "Autonomous Helicopter Aerobatics through Apprenticeship Learning." In: *International Journal of Robotics Research (IJRR)* 29 (13) (cit. on p. 1).

Abbeel, Pieter, Morgan Quigley, and Andrew Y Ng (2006). "Using inaccurate models in reinforcement learning." In: *ICML* (cit. on pp. 7, 31).

Achiam, Joshua, David Held, Aviv Tamar, and Pieter Abbeel (2017). "Constrained Policy Optimization." In: *CoRR* abs/1705.10528. arXiv: 1705.10528 (cit. on p. 142).

Agrawal, Pulkit, Ashvin Nair, Pieter Abbeel, Jitendra Malik, and Sergey Levine (2016). "Learning to poke by poking: Experiential learning of intuitive physics." In: *Advances In Neural Information Processing Systems* (cit. on p. 7).

Al-Shedivat, Maruan, Trapit Bansal, Yuri Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel (2017). "Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments." In: *CoRR* abs/1710.03641. arXiv: 1710.03641 (cit. on p. 86).

Alet, Ferran, Tomás Lozano-Pérez, and Leslie Pack Kaelbling (2018). "Modular meta-learning." In: *CoRR* abs/1806.10166. arXiv: 1806.10166. URL: http://arxiv.org/abs/1806.10166 (cit. on p. 46).

Amos, Brandon, Ivan Dario Jimenez Rodriguez, Jacob Sacks, Byron Boots, and J. Zico Kolter (2018). *Differentiable MPC for End-to-end Planning and Control*. arXiv: 1810.13400 [cs.LG] (cit. on p. 64).

Andrychowicz, Marcin, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas (2016). "Learning to learn by gradient descent by gradient descent." In: *CoRR* abs/1606.04474. arXiv: 1606.04474 (cit. on pp. 31, 45, 85).

Asadi, Kavosh, Dipendra Misra, Seungchan Kim, and Michael L. Littman (2019). "Combating the Compounding-Error Problem with a Multi-step Model." In: *CoRR* abs/1905.13320. arXiv: 1905.13320 (cit. on p. 13).

Åström, Karl J and Björn Wittenmark (2013). *Adaptive control*. Courier Corporation (cit. on p. 80).

Aswani, Anil, Patrick Bouffard, and Claire Tomlin (2012). “Extensions of learning-based model predictive control for real-time application to a quadrotor helicopter.” In: *American Control Conference (ACC), 2012*. IEEE (cit. on p. 85).

Atkeson, Christopher G and Juan Carlos Santamaria (1997). “A comparison of direct and model-based reinforcement learning.” In: *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*. Vol. 4. IEEE, pp. 3557–3564 (cit. on pp. 2, 12, 17).

Babaeizadeh, Mohammad, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz (2016). “GA3C: GPU-based A3C for Deep Reinforcement Learning.” In: *CoRR* abs/1611.06256. arXiv: 1611.06256 (cit. on p. 71).

Bagnell, J Andrew and Jeff G Schneider (2001). “Autonomous helicopter control using reinforcement learning policy search methods.” In: *ICRA*. Vol. 2. IEEE, pp. 1615–1620 (cit. on pp. 7, 31).

Baker, Bowen, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar (2016). “Designing neural network architectures using reinforcement learning.” In: *arXiv preprint arXiv:1611.02167* (cit. on p. 85).

Bartlett, Peter L. and Jonathan Baxter (2011). “Infinite-Horizon Policy-Gradient Estimation.” In: *CoRR* abs/1106.0665. arXiv: 1106.0665. URL: http://arxiv.org/abs/1106.0665 (cit. on pp. 42, 132).

Barto, Andrew G, R. S. Sutton, and C. W. Anderson (Sept. 1983). “Neuronlike adaptive elements that can solve difficult learning control problems.” In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-13.5, pp. 834–846. DOI: 10.1109/TSMC.1983.6313077 (cit. on p. 64).

Bengio, Yoshua, Samy Bengio, and Jocelyn Cloutier (1990). *Learning a synaptic learning rule*. Université de Montréal, Département d’informatique et de recherche opérationnelle (cit. on p. 85).

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi (1994). “Learning long-term dependencies with gradient descent is difficult.” In: *IEEE transactions on neural networks* 5.2, pp. 157–166 (cit. on pp. 17, 18).

Braun, Daniel A, Ad Aertsen, Daniel M Wolpert, and Carsten Mehring (2009). “Learning optimal adaptation strategies in unpredictable motor tasks.” In: *Journal of Neuroscience* (cit. on p. 80).

Brockman, Greg et al. (2016). “OpenAI Gym.” In: *CoRR* abs/1606.01540 (cit. on pp. 72, 73, 122).

Buckman, Jacob, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee (2018). “Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion.” In: *CoRR* abs/1807.01675. arXiv: 1807.01675 (cit. on pp. 52, 53, 59, 61, 67, 95).

Calinon, Sylvain (2009). *Robot Programming by Demonstration*. 1st. USA: CRC Press, Inc. ISBN: 1439808678 (cit. on p. 1).

Chen, Yutian, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando De Freitas (2017). "Learning to Learn without Gradient Descent by Gradient Descent." In: *ICML* (cit. on p. 45).

Chua, Kurtland, Roberto Calandra, Rowan McAllister, and Sergey Levine (2018). "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models." In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., pp. 4754–4765 (cit. on pp. 30, 31, 53, 59, 67, 68, 95).

Clavera, Ignasi, Violet Fu, and Pieter Abbeel (2020). *Model-Augmented Actor-Critic: Backpropagating through Paths*. arXiv: 2005.08068 (cit. on pp. 4, 96).

Clavera, Ignasi, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel (2018). "Model-Based Reinforcement Learning via Meta-Policy Optimization." In: *CoRL* (cit. on pp. 4, 46, 53, 67, 68, 73).

Coates, Adam, Pieter Abbeel, and Andrew Y. Ng (2008). "Learning for Control from Muliple Demonstrations." In: *Proceedings of the 25th International Conference on Machine Learning (ICML)* (cit. on p. 1).

Dean, Jeffrey, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Quoc V. Le, and Andrew Y. Ng (2012). "Large Scale Distributed Deep Networks." In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., pp. 1223–1231 (cit. on p. 71).

Deisenroth, Marc Peter, Gerhard Neumann, and Jan Peters (2013). "A Survey on Policy Search for Robotics." In: *Found. Trends Robot* 2 (cit. on p. 25).

Deisenroth, Marc Peter and Carl E Rasmussen (2011). "PILCO: A model-based and data-efficient approach to policy search." In: *ICML*, pp. 465–472 (cit. on pp. 2, 6, 7, 9, 12, 13, 17, 25, 27, 31, 58, 64, 72, 89).

Depeweg, Stefan, Finale Doshi-velez, and Steffen Udluft (2017a). "Learning and Policy Search in Stochastic Dynamical Systems with Bayesian Neural Networks." In: *ICML* (cit. on p. 31).

Depeweg, Stefan, José Miguel Hernández-Lobato, Finale Doshi-Velez, and Steffen Udluft (2017b). "Learning and policy search in stochastic dynamical systems with bayesian neural networks." In: *ICLR* (cit. on p. 6).

Dhariwal, Prafulla, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu (2017). *OpenAI Baselines*. `https://github.com/openai/baselines` (cit. on pp. 20, 46, 117).

Duan, Yan (Dec. 2017). "Meta Learning for Control." Doctoral dissertation. EECS Department, University of California, Berkeley (cit. on p. 10).

Duan, Yan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel (2016a). "Benchmarking deep reinforcement learning for continuous control." In: *International Conference on Machine Learning*, pp. 1329–1338 (cit. on pp. 20, 117, 124).

Duan, Yan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel (Nov. 2016b). "RL2: Fast Reinforcement Learning via Slow Reinforcement Learning." In: (cit. on pp. 11, 24, 25, 28, 32, 38, 45, 80, 86).

Ebert, Frederik, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex X. Lee, and Sergey Levine (2018). "Visual Foresight: Model-Based Deep Reinforcement Learning for Vision-Based Robotic Control." In: *CoRR* abs/1812.00568. arXiv: `1812.00568` (cit. on p. 96).

Espeholt, Lasse, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu (2018). "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures." In: *CoRR* abs/1802.01561. arXiv: `1802.01561` (cit. on p. 71).

Feinberg, Vladimir, Alvin Wan, Ion Stoica, Michael I. Jordan, Joseph E. Gonzalez, and Sergey Levine (2018a). "Model-Based Value Expansion for Efficient Model-Free Reinforcement Learning." In: (cit. on p. 30).

Feinberg, Vladimir, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine (2018b). "Model-based value estimation for efficient model-free reinforcement learning." In: *arXiv preprint arXiv:1803.00101* (cit. on p. 95).

Fikes, Richard E, Peter E Hart, and Nils J Nilsson (1972). "Learning and executing generalized robot plans." In: *Artificial Intelligence* 3, pp. 251–288. ISSN: 0004-3702. DOI: `https://doi.org/10.1016/0004-3702(72)90051-3` (cit. on p. 1).

Finn, Chelsea (2018). "Learning to Learn with Gradients." Doctoral dissertation. EECS Department, University of California, Berkeley (cit. on p. 10).

Finn, Chelsea, Pieter Abbeel, and Sergey Levine (2017). "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks." In: *ICML* (cit. on pp. 10, 11, 24, 25, 28, 30, 32, 37–39, 41, 46, 47, 50, 80, 81, 83, 85, 87, 124, 126, 135).

Finn, Chelsea and Sergey Levine (2017a). "Deep visual foresight for planning robot motion." In: *IEEE International Conference on Robotics and Automation (ICRA)* (cit. on p. 7).

Finn, Chelsea and Sergey Levine (2017b). "Meta-Learning and Universality: Deep Representations and Gradient Descent can Approximate any Learning Algorithm." In: *CoRR* abs/1710.11622. arXiv: 1710.11622 (cit. on p. 10).

Foerster, Jakob, Gregory Farquhar, Maruan Al-Shedivat, Tim Rocktaschel, Eric P Xing, and Shimon Whiteson (2018). "DiCE: The Infinitely Differentiable Monte Carlo Estimator." In: *ICML* (cit. on pp. 41, 42, 132, 135).

Fortunato, Meire, Charles Blundell, and Oriol Vinyals (2017). "Bayesian recurrent neural networks." In: *arXiv preprint arXiv:1704.02798* (cit. on p. 85).

Frans, Kevin, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman (Oct. 2018). "Meta Learning Shared Hierarchies." In: *ICLR* (cit. on p. 46).

Fu, Justin, Sergey Levine, and Pieter Abbeel (2016). "One-shot learning of manipulation skills with online dynamics adaptation and neural network priors." In: *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, pp. 4019–4026 (cit. on pp. 9, 31, 85).

Fujimoto, Scott, Herke van Hoof, and David Meger (2018). "Addressing function approximation error in actor-critic methods." In: *arXiv preprint arXiv:1802.09477* (cit. on pp. 54, 59, 65).

Furmston, Thomas, Guy Lever, and David Barber (2016). "Approximate Newton Methods for Policy Search in Markov Decision Processes." In: *J. Mach. Learn. Res.* 17, 227:1–227:51 (cit. on pp. 42, 43, 132, 137, 140).

Gal, Yarin, Rowan Thomas McAllister, and Carl Edward Rasmussen (2016). "Improving PILCO with Bayesian Neural Network Dynamics Models." In: *Data-Efficient Machine Learning workshop*. Vol. 951, p. 2016 (cit. on p. 7).

Grimmett, G.R. and D.R. Stirzaker (2001). *Probability and random processes*. Vol. 80. Oxford university press (cit. on p. 54).

Gu, Shixiang, Ethan Holly, Timothy P. Lillicrap, and Sergey Levine (2016a). "Deep Reinforcement Learning for Robotic Manipulation." In: *CoRR* abs/1610.00633. arXiv: 1610.00633 (cit. on p. 72).

Gu, Shixiang, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine (2016b). "Continuous deep Q-learning with model-based acceleration." In: *ICML*. JMLR.org (cit. on pp. 31, 85).

Gullapalli, Vijaykumar, Roderic A Grupen, and Andrew G Barto (1992). "Learning reactive admittance control." In: *Proceedings 1992 IEEE International Conference on Robotics and Automation*. IEEE, pp. 1475–1480 (cit. on p. 72).

Gupta, Abhishek, Benjamin Eysenbach, Chelsea Finn, and Sergey Levine (2018a). "Unsupervised Meta-Learning for Reinforcement Learning." In: *ICML* (cit. on p. 46).

Gupta, Abhishek, Russell Mendonca, Yuxuan Liu, Pieter Abbeel, and Sergey Levine (2018b). “Meta-Reinforcement Learning of Structured Exploration Strategies.” In: *ICML* (cit. on pp. 38, 46).

Ha, David and Jürgen Schmidhuber (2018). “World Models.” In: *CoRR* abs/1803.10122. arXiv: 1803.10122 (cit. on p. 96).

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine (2018a). “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.” In: *arXiv preprint arXiv:1801.01290* (cit. on pp. 52, 55, 61, 65).

Haarnoja, Tuomas, Aurick Zhou, Sehoon Ha, Jie Tan, George Tucker, and Sergey Levine (2018b). “Learning to Walk via Deep Reinforcement Learning.” In: *CoRR* abs/1812.11103. arXiv: 1812.11103 (cit. on p. 72).

Haarnoja, Tuomas, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine (2018c). “Soft Actor-Critic Algorithms and Applications.” In: *CoRR* abs/1812.05905. arXiv: 1812.05905 (cit. on pp. 59, 72).

Hafner, Danijar, Timothy P. Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson (2018). “Learning Latent Dynamics for Planning from Pixels.” In: *CoRR* abs/1811.04551. arXiv: 1811.04551 (cit. on p. 96).

Hafner, R. and M. Riedmiller (Apr. 2007). “Neural Reinforcement Learning Controllers for a Real Robot Application.” In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 2098–2103 (cit. on p. 72).

Heess, Nicolas, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin A. Riedmiller, and David Silver (2017). “Emergence of Locomotion Behaviours in Rich Environments.” In: *CoRR* abs/1707.02286. arXiv: 1707.02286 (cit. on p. 71).

Heess, Nicolas, Gregory Wayne, David Silver, Tim Lillicrap, Tom Erez, and Yuval Tassa (2015). “Learning continuous control policies by stochastic value gradients.” In: *Advances in Neural Information Processing Systems*, pp. 2944–2952 (cit. on pp. 7, 21, 52, 53, 56, 61, 64, 117).

Hochreiter, Sepp, A. Steven Younger, and Peter R. Conwell (2001). “Learning To Learn Using Gradient Descent.” In: *ICANN*, pp. 87–94 (cit. on p. 45).

Hong, Zhang-Wei, Joni Pajarinen, and Jan Peters (2019). “Model-based Lookahead Reinforcement Learning.” In: *ArXiv* abs/1908.06012 (cit. on p. 95).

Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman (2007). *Introduction to Automata Theory, Languages and Computation*. 3rd ed. Boston, MA: Pearson Addison-Wesley. ISBN: 978-0-321-51448-6 (cit. on p. 1).

Hüsken, Michael and Christian Goerick (2000). "Fast Learning for Problem Classes using a Knowledge Based Network Initialization." In: *IJCNN* (cit. on pp. 32, 46).

Janner, Michael, Justin Fu, Marvin Zhang, and Sergey Levine (2019). "When to Trust Your Model: Model-Based Policy Optimization." In: *CoRR* abs/1906.08253. arXiv: 1906.08253 (cit. on pp. 52, 53, 59, 61, 64, 67, 96).

Jayaraman, Dinesh, Frederik Ebert, Alexei A. Efros, and Sergey Levine (2018). "Time-Agnostic Prediction: Predicting Predictable Video Frames." In: *CoRR* abs/1808.07784. arXiv: 1808.07784. URL: http://arxiv.org/abs/1808.07784 (cit. on p. 96).

Jordan, Michael I and David E Rumelhart (1992). "Forward models: Supervised learning with a distal teacher." In: *Cognitive science* 16.3, pp. 307–354 (cit. on p. 7).

Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore (1996). "Reinforcement Learning: A Survey." In: *CoRR* cs.AI/9605103 (cit. on pp. 67, 96).

Kakade, Sham and John Langford (2002). "Approximately Optimal Approximate Reinforcement Learning." In: *ICML* (cit. on pp. 44, 57, 58, 138, 141).

Kalweit, Gabriel and Joschka Boedecker (Nov. 2017). "Uncertainty-driven Imagination for Continuous Deep Reinforcement Learning." In: *Proceedings of the 1st Annual Conference on Robot Learning*. Ed. by Sergey Levine, Vincent Vanhoucke, and Ken Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, pp. 195–206 (cit. on p. 95).

Kamthe, Sanket and Marc Peter Deisenroth (2017). "Data-Efficient Reinforcement Learning with Probabilistic Model Predictive Control." In: *CoRR* abs/1706.06491. arXiv: 1706.06491 (cit. on p. 31).

Karkus, Péter, Xiao Ma, David Hsu, Leslie Pack Kaelbling, Wee Sun Lee, and Tomás Lozano-Pérez (2019). "Differentiable Algorithm Networks for Composable Robot Learning." In: *CoRR* abs/1905.11602. arXiv: 1905.11602 (cit. on p. 64).

Kelouwani, Sousso, Kokou Adegnon, Kodjo Agbossou, and Yves Dube (2012). "Online system identification and adaptive control for PEM fuel cell maximum efficiency tracking." In: *IEEE Transactions on Energy Conversion* 27.3, pp. 580–592 (cit. on p. 85).

Khatib, Oussama (1986). "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots." In: *The International Journal of Robotics Research* 5.1, pp. 90–98. DOI: 10.1177/027836498600500106. eprint: https://doi.org/10.1177/027836498600500106 (cit. on p. 1).

Kingma, Diederik P and Max Welling (2013). "Auto-encoding variational bayes." In: *arXiv preprint arXiv:1312.6114* (cit. on pp. 52, 54).

Ko, Jonathan, Daniel J Klein, Dieter Fox, and Dirk Haehnel (2007). "Gaussian processes and reinforcement learning for identification and control of an autonomous blimp."

In: *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, pp. 742–747 (cit. on p. 7).

Krause, Ben, Emmanuel Kahembwe, Iain Murray, and Steve Renals (2017). "Dynamic Evaluation of Neural Sequence Models." In: *CoRR* abs/1709.07432. arXiv: 1709.07432 (cit. on pp. 85, 87).

Krause, Ben, Liang Lu, Iain Murray, and Steve Renals (2016). "Multiplicative LSTM for sequence modelling." In: *arXiv preprint arXiv:1609.07959* (cit. on p. 85).

Kukacka, Jan, Vladimir Golkov, and Daniel Cremers (2017). "Regularization for Deep Learning: A Taxonomy." In: *CoRR* abs/1710.10686. arXiv: 1710.10686 (cit. on p. 15).

Kumar, Vikash, Emanuel Todorov, and Sergey Levine (2016). "Optimal control with learned local models: Application to dexterous manipulation." In: *ICRA*. IEEE, pp. 378–383 (cit. on p. 7).

Kurutach, Thanard, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel (2018). "Model-ensemble trust-region policy optimization." In: *arXiv preprint arXiv:1802.10592* (cit. on pp. 4, 13, 25, 30, 31, 33, 53, 56, 64, 70, 73, 80, 89).

Lake, Brenden M, Ruslan Salakhutdinov, and Joshua B Tenenbaum (2015). "Human-level concept learning through probabilistic program induction." In: *Science* (cit. on p. 85).

Lee, Gilwoo, Tomás Lozano-Pérez, and Leslie Pack Kaelbling (2015). "Hierarchical planning for multi-contact non-prehensile manipulation." In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015*. IEEE, pp. 264–271. DOI: 10.1109/IROS.2015.7353384. URL: https://doi.org/10.1109/IROS.2015.7353384 (cit. on p. 96).

Lee, Kimin, Younggyo Seo, Seunghyun Lee, Honglak Lee, and Jinwoo Shin (2020). *Context-aware Dynamics Model for Generalization in Model-Based Reinforcement Learning*. arXiv: 2005.06800 [cs.LG] (cit. on p. 96).

Lenz, Ian, Ross A Knepper, and Ashutosh Saxena (2015). "DeepMPC: Learning Deep Latent Features for Model Predictive Control." In: *Robotics: Science and Systems* (cit. on pp. 7, 31).

Levine, Sergey and Pieter Abbeel (2014). "Learning neural network policies with guided policy search under unknown dynamics." In: *NIPS*, pp. 1071–1079 (cit. on pp. 7, 31, 63).

Levine, Sergey, Chelsea Finn, Trevor Darrell, and Pieter Abbeel (2016a). "End-to-end training of deep visuomotor policies." In: *Journal of Machine Learning Research* 17.39, pp. 1–40 (cit. on pp. 7, 31).

Levine, Sergey and Vladlen Koltun (2013). "Guided policy search." In: *International Conference on Machine Learning*, pp. 1–9 (cit. on p. 85).

Levine, Sergey, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen (2016b). "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection." In: *arXiv preprint arXiv:1603.02199* (cit. on p. 7).

Levine, Sergey, Nolan Wagener, and Pieter Abbeel (2015). "Learning contact-rich manipulation skills with guided policy search." In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 156–163 (cit. on pp. 68, 72, 77).

Li, Ke and Jitendra Malik (2016). "Learning to optimize." In: *arXiv preprint arXiv:1606.01885* (cit. on p. 85).

Li, Zhenguo, Fengwei Zhou, Fei Chen, and Hang Li (2017). "Meta-SGD: Learning to Learn Quickly for Few Shot Learning." In: *CoRR* abs/1707.09835. arXiv: 1707.09835. URL: http://arxiv.org/abs/1707.09835 (cit. on pp. 41, 130).

Lillicrap, Timothy P. et al. (2015). "Continuous control with deep reinforcement learning." In: *CoRR* abs/1509.02971. arXiv: 1509.02971 (cit. on pp. 21, 25, 32, 52, 61, 65, 117).

Lim, Shiau Hong, Huan Xu, and Shie Mannor (2013). "Reinforcement Learning in Robust Markov Decision Processes." In: *NIPS* (cit. on pp. 25, 31).

Lin, Long-Ji (May 1992). "Self-improving reactive agents based on reinforcement learning, planning and teaching." In: *Machine Learning* 8.3, pp. 293–321. ISSN: 1573-0565. DOI: 10.1007/BF00992699 (cit. on p. 54).

Lioutikov, Rudolf, Alexandros Paraschos, Jan Peters, and Gerhard Neumann (2014). "Sample-based informationl-theoretic stochastic optimal control." In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 3896–3902 (cit. on p. 72).

Lowrey, Kendall, Aravind Rajeswaran, Sham M. Kakade, Emanuel Todorov, and Igor Mordatch (2018). "Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control." In: *CoRR* abs/1811.01848. arXiv: 1811.01848 (cit. on p. 95).

Luo, Yuping, Huazhe Xu, Yuanzhi Li, Yuandong Tian, Trevor Darrell, and Tengyu Ma (2019). "Algorithmic Framework for Model-based Deep Reinforcement Learning with Theoretical Guarantees." In: *ICLR* (cit. on pp. 52, 67, 68, 71, 95).

Maitin-Shepard, Jeremy, Marco Cusumano-Towner, Jinna Lei, and Pieter Abbeel (2010). "Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding." In: *In International Conference on Robotics and Automation (ICRA* (cit. on p. 1).

Manganiello, Patrizio, Mattia Ricco, Giovanni Petrone, Eric Monmasson, and Giovanni Spagnuolo (2014). "Optimization of Perturbative PV MPPT Methods Through Online System Identification." In: *IEEE Trans. Industrial Electronics* 61.12, pp. 6812–6821 (cit. on p. 85).

Meier, Franziska, Daniel Kappler, Nathan Ratliff, and Stefan Schaal (2016). "Towards Robust Online Inverse Dynamics Learning." In: *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*. IEEE (cit. on pp. 80, 85).

Meier, Franziska and Stefan Schaal (May 2016). "Drifting Gaussian Processes with Varying Neighborhood Sizes for Online Model Learning." In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) 2016*. IEEE (cit. on p. 85).

Miconi, Thomas, Jeff Clune, and Kenneth O. Stanley (Apr. 2018). "Differentiable plasticity: training plastic neural networks with backpropagation." In: *ICML* (cit. on p. 46).

Mishra, Nikhil, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel (July 2018). "A Simple Neural Attentive Meta-Learner." In: *ICLR* (cit. on pp. 26, 28, 45, 85).

Mnih, Volodymyr, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu (2016). "Asynchronous methods for deep reinforcement learning." In: *International Conference on Machine Learning*, pp. 1928–1937 (cit. on pp. 55, 64, 71).

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. (2015). "Human-level control through deep reinforcement learning." In: *Nature* 518.7540, pp. 529–533 (cit. on pp. 2, 6, 7, 71).

Mohamed, Shakir, Mihaela Rosca, Michael Figurnov, and Andriy Mnih (2019). *Monte Carlo Gradient Estimation in Machine Learning*. arXiv: 1906.10652 [stat.ML] (cit. on pp. 52, 54, 56, 64).

Moldovan, T. M., S. Levine, M. I. Jordan, and P. Abbeel (May 2015). "Optimism-driven exploration for nonlinear systems." In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3239–3246. DOI: 10.1109/ICRA.2015.7139645 (cit. on p. 72).

Moudgal, V. G., K. M. Passino, and S. Yurkovich (1994). "Rule-based control for a flexible-link robot." In: *IEEE Transactions on Control Systems Technology* 2.4, pp. 392–405 (cit. on p. 1).

Munkhdalai, Tsendsuren and Hong Yu (2017). "Meta networks." In: *arXiv preprint arXiv:1703.00837* (cit. on p. 85).

Munkhdalai, Tsendsuren, Xingdi Yuan, Soroush Mehri, Tong Wang, and Adam Trischler (2017). "Learning Rapid-Temporal Adaptations." In: *arXiv preprint arXiv:1712.09926* (cit. on p. 85).

Nagabandi, Anusha, Ignasi Clavera, Simin Liu, Ronald S. Fearing, Pieter Abbeel, Sergey Levine, and Chelsea Finn (2018a). *Learning to Adapt in Dynamic, Real-World Environ-*

ments Through Meta-Reinforcement Learning. arXiv: 1803.11347 [cs.LG] (cit. on pp. 4, 31, 72).

Nagabandi, Anusha, Chelsea Finn, and Sergey Levine (2018b). "Deep Online Learning via Meta-Learning: Continual Adaptation for Model-Based RL." In: *CoRR* abs/1812.07671. arXiv: 1812.07671 (cit. on p. 95).

Nagabandi, Anusha, Gregory Kahn, Ronald S. Fearing, and Sergey Levine (2017). "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning." In: *CoRR* abs/1708.02596 (cit. on pp. 7, 9, 13, 27, 31, 33, 53, 88, 89).

Naik, Devang K and RJ Mammone (1992). "Meta-neural networks that learn by learning." In: *Neural Networks, 1992. IJCNN., International Joint Conference on*. Vol. 1. IEEE, pp. 437–442 (cit. on p. 85).

Nair, Arun, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, Shane Legg, Volodymyr Mnih, Koray Kavukcuoglu, and David Silver (2015). "Massively Parallel Methods for Deep Reinforcement Learning." In: *CoRR* abs/1507.04296. arXiv: 1507.04296 (cit. on p. 71).

Nair, Ashvin, Dian Chen, Pulkit Agrawal, Phillip Isola, Pieter Abbeel, Jitendra Malik, and Sergey Levine (2017). "Combining Self-Supervised Learning and Imitation for Vision-Based Rope Manipulation." In: *arXiv preprint arXiv:1703.02018* (cit. on p. 7).

Nguyen-Tuong, D., M. Seeger, and Jan Peters (2009). "Local Gaussian Process Regression for Real Time Online Model Learning and Control." In: *NIPS*, pp. 1193–1200 (cit. on p. 31).

Nguyen, Derrick H and Bernard Widrow (1990). "Neural networks for self-learning control systems." In: *IEEE Control systems magazine* 10.3, pp. 18–23 (cit. on pp. 7, 13).

Nichol, Alex, Joshua Achiam, and John Schulman (2018). "On First-Order Meta-Learning Algorithms." In: *CoRR* abs/1803.02999. arXiv: 1803.02999 (cit. on pp. 30, 41, 46, 130).

Oh, Junhyuk, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh (2015). "Action-conditional video prediction using deep networks in atari games." In: *Advances in Neural Information Processing Systems*, pp. 2863–2871 (cit. on p. 7).

Okada, Masashi, Luca Rigazio, and Takenobu Aoshima (2017). *Path Integral Networks: End-to-End Differentiable Optimal Control*. arXiv: 1706.09597 [cs.AI] (cit. on p. 64).

Parisi, German Ignacio, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter (2018). "Continual Lifelong Learning with Neural Networks: A Review." In: *CoRR* abs/1802.07569. arXiv: 1802.07569 (cit. on p. 97).

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2013). “On the difficulty of training recurrent neural networks.” In: *International Conference on Machine Learning*, pp. 1310–1318 (cit. on p. 18).

Pastor, P., Ludovic Righetti, M. Kalakrishnan, and Stefan Schaal (Sept. 2011). “Online movement adaptation based on previous sensor experiences.” English (US). In: *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pp. 365–371 (cit. on pp. 80, 85).

Peng, Xue Bin, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel (Oct. 2017). “Sim-to-Real Transfer of Robotic Control with Dynamics Randomization.” In: (cit. on p. 24).

Pereira, Marcus, David D. Fan, Gabriel Nakajima An, and Evangelos A. Theodorou (2018). “MPC-Inspired Neural Network Policies for Sequential Decision Making.” In: *CoRR* abs/1802.05803. arXiv: 1802.05803. URL: http://arxiv.org/abs/1802.05803 (cit. on p. 64).

Peters, Jan and Stefan Schaal (Oct. 2006). “Policy Gradient Methods for Robotics.” In: *IROS*, pp. 2219–2225 (cit. on pp. 11, 18, 28, 56, 64, 132).

Pong, Vitchyr, Shixiang Gu, Murtaza Dalal, and Sergey Levine (2018). “Temporal Difference Models: Model-Free Deep RL for Model-Based Control.” In: *ICLR* (cit. on p. 25).

Precup, Doina, Richard S Sutton, and Sanjoy Dasgupta (2001). “Off-Policy Temporal Difference Learning with Function Approximation.” In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML ’01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 417–424. ISBN: 1-55860-778-1 (cit. on p. 71).

Precup, Doina, Richard S Sutton, and Satinder P Singh (2000). “Eligibility Traces for Off-Policy Policy Evaluation.” In: *Proceedings of the Seventeenth International Conference on Machine Learning*. ICML ’00. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 759–766. ISBN: 1-55860-707-2 (cit. on p. 71).

Punjani, Ali and Pieter Abbeel (2015). “Deep learning helicopter dynamics models.” In: *ICRA*, pp. 3223–3230 (cit. on p. 31).

Rai, Akshara, Giovanni Sutanto, Stefan Schaal, and Franziska Meier (May 2017). “Learning Feedback Terms for Reactive Planning and Control.” In: *Proceedings 2017 IEEE International Conference on Robotics and Automation (ICRA)*. Piscataway, NJ, USA: IEEE (cit. on p. 85).

Rajeswaran, Aravind, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine (Oct. 2016). “EPOpt: Learning Robust Neural Network Policies Using Model Ensembles.” In: (cit. on pp. 25, 31).

Rasmussen, Carl E, Malte Kuss, et al. (2003). “Gaussian Processes in Reinforcement Learning.” In: *NIPS*. Vol. 4, p. 1 (cit. on p. 7).

Ravi, Sachin and Hugo Larochelle (2018). “Optimization as a model for few-shot learning.” In: *ICLR* (cit. on pp. 32, 45, 85).

Recht, Benjamin, Christopher Re, Stephen Wright, and Feng Niu (2011). “Hogwild: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent.” In: *Advances in Neural Information Processing Systems 24*. Ed. by J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger. Curran Associates, Inc., pp. 693–701 (cit. on p. 71).

Rei, Marek (2015). “Online Representation Learning in Recurrent Neural Language Models.” In: *CoRR* abs/1508.03854. arXiv: 1508.03854 (cit. on p. 85).

Ross, Stephane, Geoffrey J. Gordon, and J. Andrew Bagnell (2010). “No-Regret Reductions for Imitation Learning and Structured Prediction.” In: *CoRR* abs/1011.0686. arXiv: 1011.0686. URL: http://arxiv.org/abs/1011.0686 (cit. on pp. 1, 121).

Rothfuss, Jonas, Dennis Lee, Ignasi Clavera, Tamim Asfour, and Pieter Abbeel (2018). “ProMP: Proximal Meta-Policy Search.” In: *arXiv preprint arXiv:1810.06784* (cit. on p. 4).

Saemundsson, Steindor, Katja Hofmann, and Marc Peter Deisenroth (2018). “Meta Reinforcement Learning with Latent Variable Gaussian Processes.” In: *UAI* (cit. on p. 46).

Sæmundsson, Steindór, Katja Hofmann, and Marc Peter Deisenroth (2018). “Meta Reinforcement Learning with Latent Variable Gaussian Processes.” In: *arXiv preprint arXiv:1803.07551* (cit. on pp. 38, 81, 86, 96).

Salimans, Tim and Diederik P Kingma (Feb. 2016). “Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks.” In: *NIPS* (cit. on p. 27).

Santoro, Adam, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap (2016). “One-shot learning with memory-augmented neural networks.” In: *arXiv preprint arXiv:1605.06065* (cit. on pp. 32, 45, 85).

Sastry, Sosale Shankara and Alberto Isidori (1989). “Adaptive control of linearizable systems.” In: *IEEE Transactions on Automatic Control* (cit. on pp. 80, 85).

Schaal, Stefan (1997). “Learning from Demonstration.” In: *Advances in Neural Information Processing Systems 9*. Ed. by M. C. Mozer, M. I. Jordan, and T. Petsche. MIT Press, pp. 1040–1046 (cit. on p. 12).

Schaal, Stefan (1999). *Is imitation learning the route to humanoid robots?* (Cit. on p. 1).

Schmidhuber, Jürgen (1987). "Evolutionary principles in self-referential learning. On learning how to learn: The meta-meta-... hook." Doctoral dissertation. Technische Universitaet Munchen (cit. on pp. 10, 31, 37, 45).

Schmidhuber, Jürgen (1992). "Learning to control fast-weight memories: An alternative to dynamic recurrent networks." In: *Neural Computation* (cit. on p. 85).

Schmidhuber, Jürgen and Rudolf Huber (1991). "Learning to generate artificial fovea trajectories for target detection." In: *International Journal of Neural Systems* 2.01n02, pp. 125–134 (cit. on pp. 7, 13, 85).

Schmidhuber, Jürgen, Jieyu Zhao, and Marco Wiering (1997). "Shifting Inductive Bias with Success-Story Algorithm, Adaptive Levin Search, and Incremental Self-Improvement." In: *Machine Learning* 28.1, pp. 105–130. ISSN: 08856125. DOI: 10.1023/A:1007383707642 (cit. on p. 45).

Schneider, Jeff G (1997). "Exploiting model uncertainty estimates for safe dynamic control learning." In: *Advances in neural information processing systems*, pp. 1047–1053 (cit. on pp. 2, 12, 17).

Schulman, John, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel (2014). "Motion planning with sequential convex optimization and convex collision checking." In: *The International Journal of Robotics Research* 33.9, pp. 1251–1270. DOI: 10.1177/0278364914528132. eprint: https://doi.org/10.1177/0278364914528132 (cit. on p. 1).

Schulman, John, Nicolas Heess, Theophane Weber, and Pieter Abbeel (2015a). "Gradient Estimation Using Stochastic Computation Graphs." In: *NIPS* (cit. on pp. 39, 42, 64, 132).

Schulman, John, Sergey Levine, Philipp Moritz, Michael I Jordan, and Pieter Abbeel (2015b). "Trust region policy optimization." In: *CoRR, abs/1502.05477* (cit. on pp. 2, 6, 7, 17, 18, 21, 25, 28, 32, 43, 58, 73, 87, 117, 124, 138, 141, 142).

Schulman, John, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel (2016a). "High-Dimensional Continuous Control Using Generalized Advantage Estimation." In: *ICLR* (cit. on p. 124).

Schulman, John, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel (2016b). "High-dimensional continuous control using generalized advantage estimation." In: *International Conference on Learning Representations (ICLR2016)* (cit. on p. 7).

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). "Proximal Policy Optimization Algorithms." In: *CoRR*. arXiv: 1707.06347 (cit. on pp. 18, 21, 32, 44, 71, 73, 117, 143).

Shalev-Shwartz, Shai (2012). “Online Learning and Online Convex Optimization.” In: *Foundations and Trends in Machine Learning* 4.2, pp. 107–194. ISSN: 1935-8237. DOI: 10.1561/2200000018 (cit. on p. 96).

Al-Shedivat, Maruan, Trapit Bansal, Umass Amherst, Yura Burda, Openai Ilya, Sutskever Openai, Igor Mordatch Openai, and Pieter Abbeel (2018). “Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments.” In: *ICLR* (cit. on pp. 38, 39, 46, 127, 135).

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. (2016). “Mastering the game of Go with deep neural networks and tree search.” In: *Nature* 529.7587, pp. 484–489 (cit. on pp. 2, 6, 7, 25).

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. (2017). “Mastering the game of Go without human knowledge.” In: *Nature* 550.7676, pp. 354–359 (cit. on pp. 2, 6).

Song, Yuda, Aditi Mavalankar, Wen Sun, and Sicun Gao (2020). *Provably Efficient Model-based Policy Adaptation*. arXiv: 2006.08051 [cs.LG] (cit. on p. 96).

Srinivas, Aravind, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn (2018). “Universal planning networks.” In: *arXiv preprint arXiv:1804.00645* (cit. on p. 64).

Stadie, Bradly C., Ge Yang, Rein Houthooft, Xi Chen, Yan Duan, Yuhuai Wu, Pieter Abbeel, and Ilya Sutskever (2018). “Some Considerations on Learning to Explore via Meta-Reinforcement Learning.” In: *CoRR* abs/1803.01118. arXiv: 1803.01118 (cit. on pp. 38, 39, 46, 50, 127, 135).

Stooke, Adam and Pieter Abbeel (2018). “Accelerated Methods for Deep Reinforcement Learning.” In: *CoRR* abs/1803.02811. arXiv: 1803.02811 (cit. on p. 71).

Stulp, F. and Stefan Schaal (2011). “Hierarchical reinforcement learning with movement primitives.” In: *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pp. 231–238 (cit. on p. 96).

Sung, Flood, Li Zhang, Tao Xiang, Timothy M. Hospedales, and Yongxin Yang (2017). “Learning to Learn: Meta-Critic Networks for Sample Efficient Learning.” In: *CoRR* abs/1706.09529 (cit. on pp. 26, 32, 46, 86).

Sutton, Richard S (1990). “Integrated architectures for learning, planning, and reacting based on approximating dynamic programming.” In: *Proceedings of the seventh international conference on machine learning*, pp. 216–224 (cit. on p. 6).

Sutton, Richard S (1991a). “Dyna, an integrated architecture for learning, planning, and reacting.” In: *ACM SIGART Bulletin* 2.4, pp. 160–163 (cit. on pp. 6, 13).

Sutton, Richard S (1991b). "Planning by incremental dynamic programming." In: *Machine Learning Proceedings 1991*. Elsevier, pp. 353–357 (cit. on p. 64).

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement Learning: An Introduction*. Second. The MIT Press (cit. on pp. 6, 56, 64).

Sutton, Richard S, David Mcallester, Satinder Singh, and Yishay Mansour (2000). "Policy Gradient Methods for Reinforcement Learning with Function Approximation." In: *NIPS* (cit. on pp. 131, 137).

Tamar, Aviv, Sergey Levine, and Pieter Abbeel (2016). "Value Iteration Networks." In: *CoRR* abs/1602.02867. arXiv: 1602.02867. URL: http://arxiv.org/abs/1602.02867 (cit. on p. 64).

Tanaskovic, Marko, Lorenzo Fagiano, Roy Smith, Paul Goulart, and Manfred Morari (2013). "Adaptive model predictive control for constrained linear systems." In: *Control Conference (ECC), 2013 European*. IEEE (cit. on p. 85).

Thrun, Sebastian and Lorien Pratt (1998). *Learning to learn*, p. 354 (cit. on pp. 37, 45, 85).

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). "Mujoco: A physics engine for model-based control." In: *IROS* (cit. on pp. 20, 32, 46, 53, 61, 68, 72, 86, 122).

Underwood, Samuel J and Iqbal Husain (2010). "Online parameter estimation and adaptive control of permanent-magnet synchronous machines." In: *IEEE Transactions on Industrial Electronics* 57.7, pp. 2435–2443 (cit. on p. 85).

Wahlström, Niklas, Thomas B. Schön, and Marc Peter Deisenroth (2015). "From Pixels to Torques: Policy Learning with Deep Dynamical Models." In: *CoRR* abs/1502.02251 (cit. on pp. 31, 96).

Wang, Jane, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick (2016). "Learning to reinforcement learn." In: *CoRR* abs/1611.0 (cit. on pp. 11, 26, 32, 80, 86).

Wang, Tingwu and Jimmy Ba (2019). "Exploring Model-based Planning with Policy Networks." In: *CoRR* abs/1906.08649. arXiv: 1906.08649 (cit. on p. 95).

Wang, Tingwu, 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." In: *CoRR* abs/1907.02057. arXiv: 1907.02057 (cit. on pp. 53, 61, 66–68).

Warren, C. W. (1989). "Global path planning using artificial potential fields." In: *Proceedings, 1989 International Conference on Robotics and Automation*, 316–321 vol.1 (cit. on p. 1).

Watter, Manuel, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller (2015). "Embed to control: A locally linear latent dynamics model for control from raw images." In: *Advances in Neural Information Processing Systems*, pp. 2746–2754 (cit. on pp. 7, 96).

Weinstein, Ari and Matthew Botvinick (2017). "Structure Learning in Motor Control: A Deep Reinforcement Learning Model." In: *CoRR* abs/1706.06827. arXiv: 1706.06827 (cit. on p. 85).

Williams, Grady, Andrew Aldrich, and Evangelos Theodorou (2015). "Model Predictive Path Integral Control using Covariance Variable Importance Sampling." In: *CoRR* abs/1509.01149. arXiv: 1509.01149 (cit. on pp. 84, 88).

Wu, Yuhuai, Elman Mansimov, Shun Liao, Roger B. Grosse, and Jimmy Ba (2017). "Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation." In: *CoRR* abs/1708.05144. arXiv: 1708.05144 (cit. on p. 32).

Xu, Mengdi, Wenhao Ding, Jiacheng Zhu, Zuxin Liu, Baiming Chen, and Ding Zhao (2020). *Task-Agnostic Online Reinforcement Learning with an Infinite Mixture of Gaussian Processes*. arXiv: 2006.11441 [cs.LG] (cit. on p. 96).

Xu, Zhongwen, Hado P van Hasselt, and David Silver (2018). "Meta-Gradient Reinforcement Learning." In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., pp. 2396–2407 (cit. on pp. 11, 38, 46).

Younger, A Steven, Sepp Hochreiter, and Peter R Conwell (2001). "Meta-learning with backpropagation." In: *International Joint Conference on Neural Networks*. IEEE (cit. on p. 85).

Zhang, Marvin, Sharad Vikram, Laura Smith, Pieter Abbeel, Matthew J Johnson, and Sergey Levine (2018). "Solar: Deep structured latent representations for model-based reinforcement learning." In: *arXiv preprint arXiv:1808.09105* (cit. on p. 96).

Zhang, Tianhao, Zoe McCarthy, Owen Jow, Dennis Lee, Xi Chen, Ken Goldberg, and Pieter Abbeel (May 2018). "Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation." In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. DOI: 10.1109/icra.2018.8461249. URL: http://dx.doi.org/10.1109/ICRA.2018.8461249 (cit. on pp. 1, 2).

Zhang, Yunzhi, Ignasi Clavera, Boren Tsai, and Pieter Abbeel (2019). *Asynchronous Methods for Model-Based Reinforcement Learning*. arXiv: 1910.12453 [cs.LG] (cit. on p. 4).

Zhou, Kemin, John C. Doyle, and Keith Glover (1996). *Robust and Optimal Control*. Prentice-Hall, Inc. (cit. on pp. 25, 31).

A

APPENDIX: MODEL-ENSEMBLE TRUST-REGION POLICY OPTIMIZATION

A.1 MODEL-ENSEMBLE TRUST-REGION POLICY OPTIMIZATION

Our algorithm can be broken down into three parts: data collection, model learning, and policy learning. We describe the numerical details for each part below.

A.1.1 *Data collection*

In each outer iteration, we use the stochastic policy to collect 3000 timesteps of real world data for every environment, except Humanoid in which we collect 6000 timesteps. At the beginning of every roll-out we sample the policy standard deviation randomly from $\mathcal{U}[0.0, 3.0]$, and we keep the value fixed throughout the episode. Furthermore, we perturb the policy's parameters by adding white Gaussian noise with standard deviation proportional to the absolute difference between the current parameters and the previous one. Finally, we split the collected data using a 2-to-1 ratio for training and validation datasets.

A.1.2 *Model Learning*

We represent the dynamics model with a 2-hidden-layer feed-forward neural network with hidden sizes 1024-1024 and ReLU nonlinearities. We train the model with the Adam optimizer with learning rate 0.001 using a batch size of 1000. The model is trained until the validation loss has not decreased for 25 passes over the entire training dataset (we validate the training every 5 passes).

A.1.3 *Policy Learning*

We represent the policy with a 2-hidden-layer feed-forward neural network with hidden sizes 32-32 and tanh nonlinearities for all the environments, except Humanoid, in which we use the hidden sizes 100-50-25. The policy is trained with TRPO on the learned models using initial standard deviation 1.0, step size δ_{KL} 0.01, and batch size 50000. If the policy fails the validation for 25 updates (we do the validation every 5 updates), we stop the learning and repeat the overall process.

A.2 ENVIRONMENT DETAILS

The reward functions $r_t(s_t, a_t)$ and optimization horizons are described below:

Environments	Reward functions	Horizon
Swimmer	s_t^{vel} - $0.005\|a_t\|_2^2$	200
Snake	s_t^{vel} - $0.005\|a_t\|_2^2$	200
Hopper	s_t^{vel} - 0.005 $\|a_t\|_2^2$ $-10\max(0.45 - s_t^{height}, 0)$ - $10\sum(\max(s_t - 100, 0))$	100
Half Cheetah	s_t^{vel} - $0.05\|a_t\|_2^2$	100
Ant	s_t^{vel} - $0.005\|a_t\|_2^2$ + 0.05	100
Humanoid	$(s_t^{head} - 1.5)^2$+ $\|a_t\|_2^2$	100

Note that in Hopper we relax the early stopping criterion to a soft constraint in reward function, whereas in Ant we early stop when the center of mass long z-axis is outside [0.2, 1.0] and have a survival reward when alive.

The state space of every environment is composed by the joint angles, joint velocities, and the cartesian position of the center of mass of a part of the simulated robot. We are not using the contact information, which make the environments effectively POMDPs in Half Cheetah, Ant, Hopper and Humanoid. We also eliminate the redundancies in the state space in order to avoid infeasible states in the prediction.

A.2.1 *Baselines*

In Section 4.3.2 we compare our method against TRPO, PPO, DDPG, and SVG. For every environment we represent the policy with a feed-forward neural network of the same size, horizon, and discount factor as the ones specified in the Appendix A.1.3. In the following we provide the hyper-parameters details:

Trust Region Policy Optimization. (Schulman et al., 2015b) We used the implementation of Duan et al., 2016a with a batch size of 500000, and we train the policies for 1000 iterations. The step size δ_{KL} that we used in all the experiments was of 0.05.

Proximal Policy Optimization (Schulman et al., 2017). We referred to the implementation of Dhariwal et al., 2017. The policies were trained for 10^7 steps using the default hyper-parameters across all tasks.

Deep Deterministic Policy Gradient. (Lillicrap et al., 2015) We also use the implementation of Dhariwal et al., 2017 using a number epochs of 2000, the rest of the hyper-parameters used were the default ones.

Stochastic Value Gradient. (Heess et al., 2015) We parametrized the dynamics model as a feed-forward neural network of two hidden layers of 512 units each and ReLU non-linearities. The model was trained after every episode with the data available in the replay buffer, using the Adam optimizer with a learning rate of 10^{-4}, and batch size of 128. We additionally clipped the gradient we the norm was larger than 10.

A.3 OVERFITTING

We show that replacing the ensemble with just one model leads to the policy overoptimizing to the one fitted dynamics model. At each outer iteration, we see that at the end of the policy optimization step the estimated performance increases while the real performance is in fact decreasing.

A.4 ABLATION STUDY

We further provide a series of ablation experiments to characterize the importance of the two main regularization components of our algorithm: the ensemble validation and the ensemble sampling techniques. In these experiments, we make only one change at a time to ME-TRPO with 5 models.

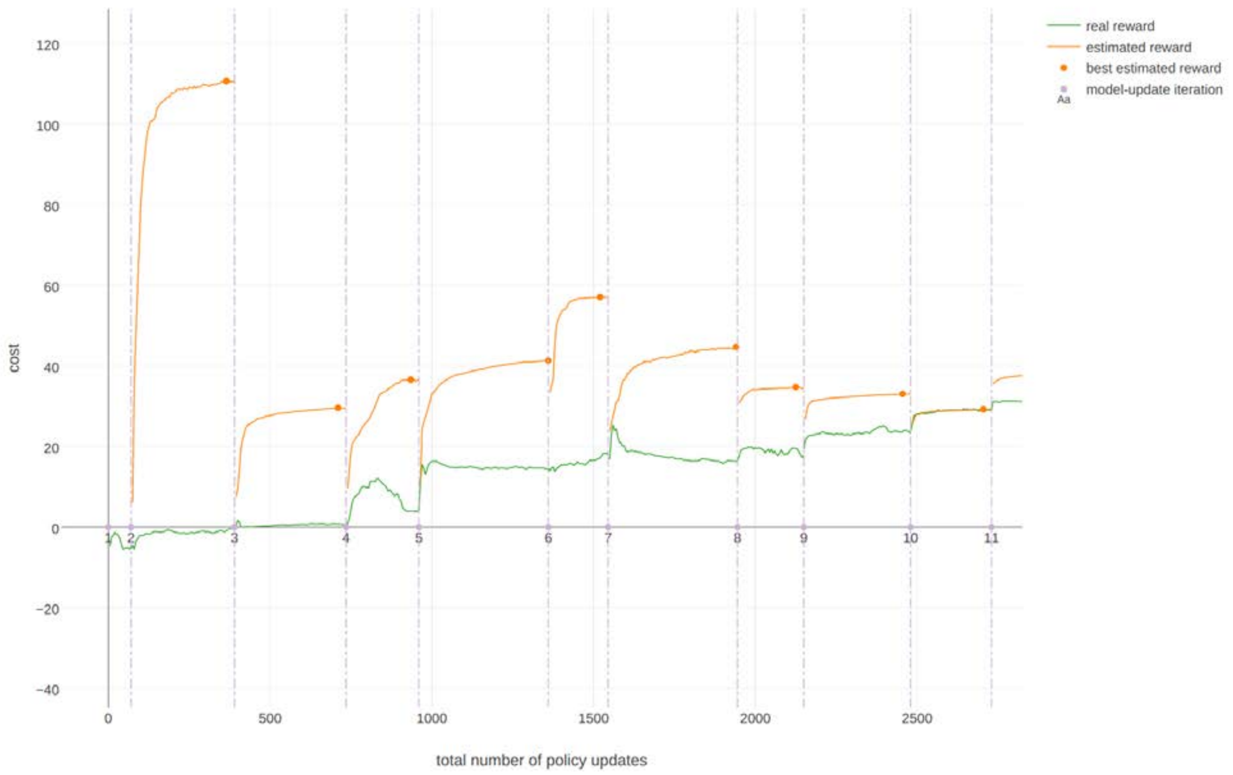


Figure 36: Predicted and real performances during the training process using our approach with one model instead of an ensemble. The policy overfits to dynamics model which degrades the real performance.

A.4.1 *Ensemble Sampling Methods*

We explore several ways to simulate the trajectories from the model ensemble. At a current state and action, we study the effect of simulating the next step given by: (1) sampling randomly from the different models (step_rand), (2) a normal distribution fitted from the predictions (model_mean_std), (3) the mean of the predictions (model_mean), (4) the median of the predictions (model_med), (5) the prediction of a fixed model over the entire episode (i.e., equivalent to averaging the gradient across all simulations) (eps_rand), and (6) sampling from one model (one_model).

The results in Figure 37 provide evidence that using the next step as the prediction of a randomly sampled model from our ensemble is the most robust method across environments. In fact, using the median or the mean of the predictions does not prevent overfitting; this effect is shown in the HalfCheetah environment where we see a decrease of the performance in latter iteration of the optimization process. Using the gradient average (5) also provides room for the policy to overfit to one or more models. This supports that having an estimate of the model uncertainty, such as in (1) and (2), is the principled way to avoid overfitting the learned models.

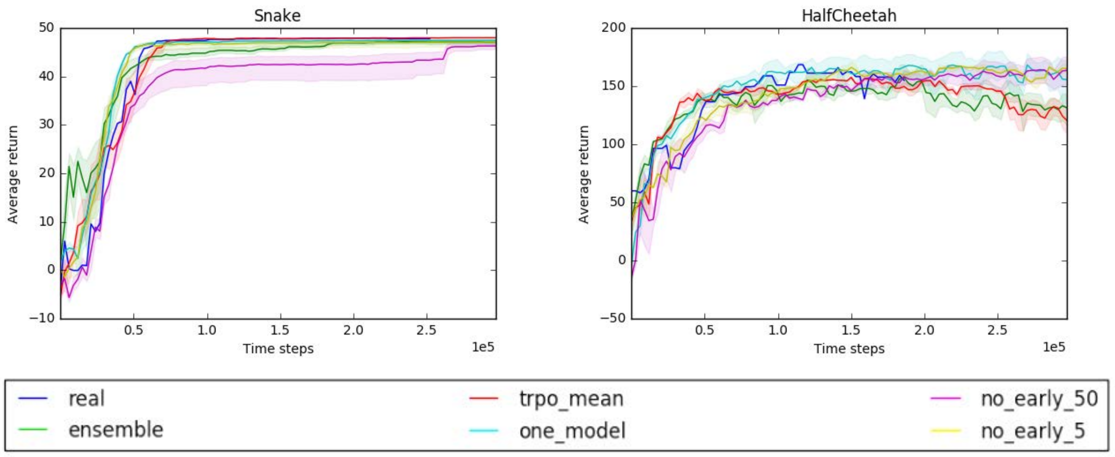


Figure 37: Comparison among different sampling techniques for simulating roll-outs. By sampling each step from a different model, we prevent overfitting and enhance the learning performance (Best viewed in color).

A.4.2 *Ensemble Validation*

Finally, we provide a study of the different ways for validating the policy. We compare the following techniques: (1) using the real performance (i.e., using an oracle) (real), (2) using the average return in the trpo roll-outs (trpo_mean), (3) stopping the policy after 50 gradient updates (no_early_50), (4) or after 5 gradient updates (no_early_5), (5) using one model to predict the performances (one_model), and (6) using an ensemble of models (ensemble). The experiments are designed to use the same number of models and hyper-parameters for the other components of the algorithm.

In Figure 38 we can see the effectiveness of each approach. It is noteworthy that having an oracle of the real performance is not the best approach. Such validation is overcautious, and does not give room for exploration resulting in a poor trained dynamics model. Stopping the gradient after a fixed number of updates results in good performance if the right number of updates is set. This burdens the hyper-parameter search with one more hyper-parameter. On the other hand, using the ensemble of models has good performance across environments without adding extra hyper-parameters.

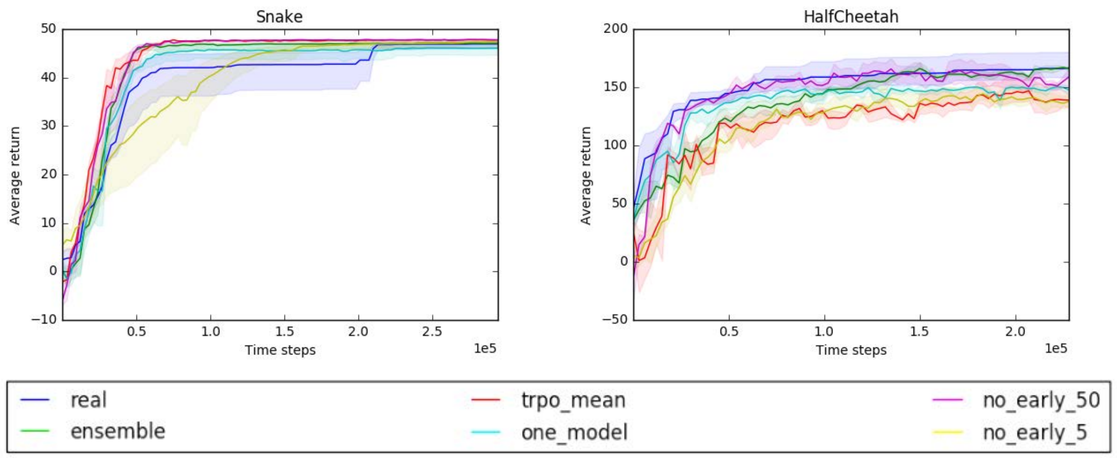


Figure 38: Comparison among validation techniques. The ensemble of models yields to good performance across environments (Best viewed in color).

B

APPENDIX: MODEL-BASED REINFORCEMENT LEARNING VIA META-POLICY OPTIMIZATION

B.1 TAILORED DATA COLLECTION

We present the effects of collecting data using tailored exploration. We refer to tailored exploration as the effect of collecting data using the post-update policies – the policies adapted to each specific model. When training policies on learned models they tend to exploit the deficiencies of the model, and thus overfitting to it. Using the post-update policies to collect data results in exploring the regions of the state space where these policies overfit and the model is inaccurate. Iteratively collecting data in the regions where the models are innacurate has been shown to greatly improve the performance (Ross et al., 2010).

The effect of using tailored exploration is shown in Figure 39. In the half-cheetah and the walker we get an improvement of 12% and 11%, respectively. The tailored exploration effect cannot be accomplished by robust optimization algorithms, such as ME-TRPO. Those algorithms learn a single policy that is robust across models. The data collection using such policy will not exploit the regions in which each model fails resulting in less accurate models.

B.2 HYPERPARAMETER STUDY

We perform a hyperparameter study (see Figure 40) to assess the sensitivity of MB-MPO to its parameters. Specifically, we vary the inner learning rate α, the size of the ensemble, and the number of meta gradient steps before collecting further real environment samples. Consistent with the results in Figure 9, we find that adaptation significantly improves the performance when compared to the non-adaptive case of $\alpha = 0$. Increasing the number of models and meta gradient steps per iteration results in higher perfor-

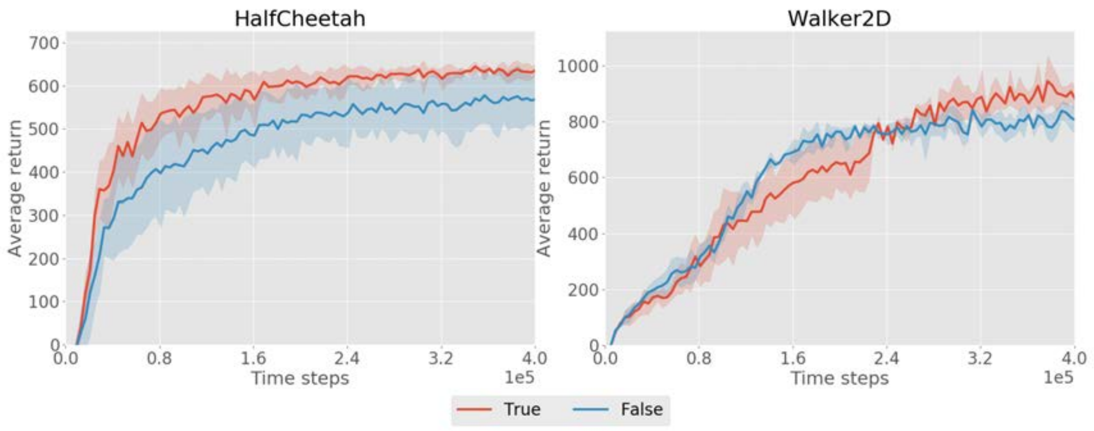


Figure 39: Tailored exploration study in the half-cheetah and walker2D environment. “True" means the data is collected by using tailored exploration, and “False" is the result of not using it, i.e., using the pre-update policy to collect data.

mance at a computational cost. However, as the computational burden is increased the performance gains diminish.

Up to a certain level, increasing the number of meta gradient steps per iteration improves performance. Though, too many meta gradients steps (i.e. 60) can lead to early convergence to a suboptimal policy. This may be due to the fact that the variance of the Gaussian policy distribution is also learned. Usually, the policies variance decreases during the training. If the number of meta-gradient steps is too large, the policy loses its exploration capabilities too early and can hardly improve once the models are more accurate. This problem can be alleviated using a fixed policy variance, or by adding an entropy bonus the learning objective.

B.3 EXPERIMENT SETUP

In the following we provide a detailed description of the setup used in the experiments presented in section 5.5:

Environments:

We benchmark MB-MPO on six continuous control benchmark tasks in the Mujoco simulator (Todorov et al., 2012), shown in Fig. 41. Five of these tasks, namely swimmer, half-cheetah, walker2D, hopper and ant, involve robotic locomotion and are provided trough the OpenAI gym (Brockman et al., 2016).

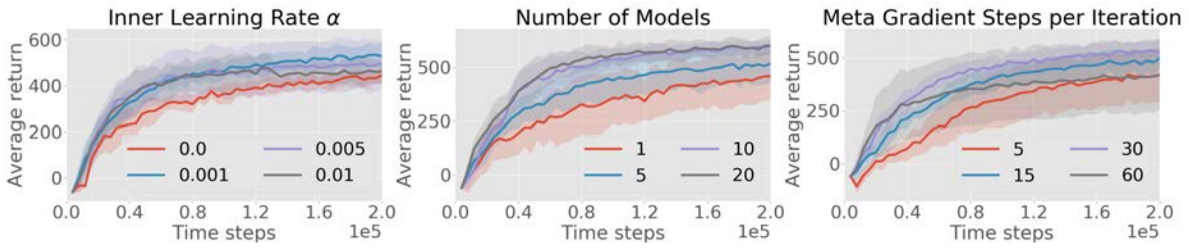


Figure 40: Hyper-parameter study in the the half-cheetah environment of a) the inner learning rate α, b) the number of dynamic models in the ensemble, and c) the number of meta gradient steps before collecting real environment samples and refitting the dynamic models.

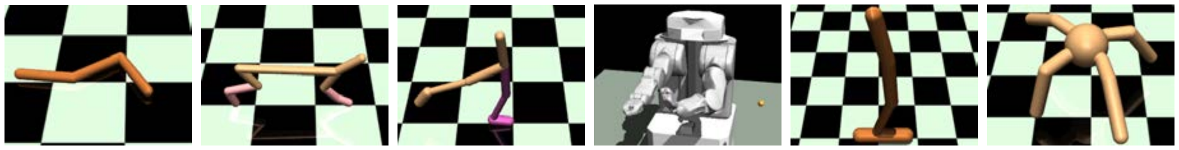


Figure 41: Mujoco environments used in our experiments. Form left to right: swimmer, half-cheetah, walker2D, PR2, hopper, and ant.

The sixth, the 7-DoF arm of the PR2 robot, has to reach arbitrary end-effector positions. Thereby, the PR2 robot is torque controlled. The reward function is comprised of the squared distance of the end-effector (TCP) to the goal and energy / control costs:

$$r(s, a) = -||s_{TCP} - x_{goal}||_2^2 - 0.05 * ||a||_2^2$$

In section 5.5.4 we use the simple 2D-Point environment to analyze the connection between policy plasticity and model uncertainty. The corresponding MDP is defined as follows:

$$\begin{aligned} \mathcal{S} &= \mathbb{R}^2 \\ \mathcal{A} &= [-0.1, 0.1]^2 \\ p_0(s_0) &= \mathcal{U}_{[-2,2]^2}(s_0) \quad \text{(uniform distribution over } [-2, 2]^2) \\ p(s_{t+1}|s_t, a_t) &= \delta(s_t + a_t) \\ r(s_t, a_t) &= -||s_t||_2^2 \\ H &= 30 \end{aligned}$$

Policy: We use a Gaussian policy $\pi_\theta(a|s) = \mathcal{N}(a|\mu(a)_{\theta_\mu}, \sigma_{\theta_\sigma})$ with diagonal covariance matrix. The mean $\mu(a)_{\theta_\mu}$ is computed by a neural network (2 hidden layers of size

32, tanh nonlinearity) which receives the current state s as an input. During the policy optimization, both the weights θ_μ of the neural network and the standard deviation vector σ_{θ_σ} are learned.

Advantage-Estimation: We use generalized advantage estimation (GAE) (Schulman et al., 2016a) with $\gamma = 0.99$ and $\lambda = 1$ in conjunction with a linear reward baseline as in (Duan et al., 2016a) to estimate advantages.

Dynamics Model Ensemble: In all experiments (except in Figure 40b) we use an ensemble of 5 fully connected neural networks. For the different environments the following hidden layer sizes were used:

- Ant, Walker: (512, 512, 512)
- PR2, Swimmer, Hopper, Half-Cheetah: (512, 512)
- 2D-Point-Env: (128, 128)

In all models, we used weight normalization and ReLu nonlinearities. For the minimization of the l^2 prediction error, the Adam optimizer with a batch-size of 500 was employed. In the first iteration all models are randomly initialized. In later iterations, the models are trained with warm starts using the parameters of the previous iteration. In each iteration and for each model in the ensemble the transition data buffer $\mathcal{D}$ is randomly split in a training (80%) and validation (20%) set. The latter split is used to compute the validation loss after each training epoch on the shuffled training split. A rolling average of the validation losses with a persistence of 0.95 is maintained throughout the epochs. Each model's training is stopped individually as soon as the rolling validation loss average decreases.

Meta-Policy Optimization: As described in section 5.2.2, the policy parameters θ are optimized using the gradient-based meta learning framework MAML. For the inner adaptation step we use a gradient step-size of $\alpha = 0.001$. For maximizing the meta-objective specified in equation 7 we use the policy gradient method TPRO (Schulman et al., 2015b) with KL-constraint $\delta = 0.01$. Since computing the gradients of the meta-objective involves second order terms such as the Hessian of the policy's log-likelihood, computing the necessary Hessian vector products for TRPO analytically is very compute intensive. Hence, we use a finite difference approximation of the vector product of the Fisher Information Matrix and the gradients as suggested in (Finn et al., 2017). If not denoted differently, 30 meta-optimization steps are performed before new trajectories are collected from the real environment.

Trajectory collection: In each algorithm iteration 4000 environment transitions (20 trajectories of 200 time steps) are collected. For the meta-optimization, 100000 imaginary environment transitions are sampled.

B.4 COMPUTATIONAL ANALYSIS

In this section we compare the computational complexity of MB-MPO against TRPO. Specifically, we report the wall clock time that it takes both algorithms to reach maximum performance on the half-cheetah environment when running the experiments on an Amazon Web Services EC2 c4.4xlarge compute instance. Our method only requires 20% more compute time than TRPO (7 hours instead of 5.5), while attaining $70\times$ reduction in sample complexity. The main time bottleneck of our method compared with the model-free algorithms is training the models.

Notice that when running real world experiment, our method will be significantly faster than model-free approaches since the bottleneck then would shift towards the data collection step.

C APPENDIX: PROMP: PROXIMAL META-POLICY SEARCH

C.1 TWO META-POLICY GRADIENT FORMULATIONS

In this section we discuss two different gradient-based meta-learning formulations, derive their gradients and analyze the differences between them.

C.1.1 *Meta-Policy Gradient Formulation I*

The first meta-learning formulation, known as MAML (Finn et al., 2017), views the inner update rule $U(\theta, \mathcal{M})$ as a mapping from the pre-update parameter θ and the task $\mathcal{M}$ to an adapted policy parameter θ'. The update function can be viewed as stand-alone procedure that encapsulates sampling from the task-specific trajectory distribution $P_{\mathcal{M}}(\boldsymbol{\tau}|\pi_\theta)$ and updating the policy parameters. Building on this concept, the meta-objective can be written as

$$J^I(\theta) = \mathbb{E}_{\mathcal{M}\sim\rho(\mathcal{M})}\left[\mathbb{E}_{\boldsymbol{\tau}'\sim P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')}\left[R(\boldsymbol{\tau}')\right]\right] \quad \text{with} \quad \theta' := U(\theta, \mathcal{M}) \tag{25}$$

The task-specific gradients follow as

$$\begin{aligned}
\nabla_\theta J^I_{\mathcal{M}}(\theta) &= \nabla_\theta E_{\boldsymbol{\tau}'\sim P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')}\left[R(\boldsymbol{\tau}')\right] & (26)\\
&= E_{\boldsymbol{\tau}'\sim P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')}\left[\nabla_\theta \log P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta') R(\boldsymbol{\tau}')\right] & (27)\\
&= E_{\boldsymbol{\tau}'\sim P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')}\left[\nabla_{\theta'} \log P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta') R(\boldsymbol{\tau}') \nabla_\theta \theta'\right] & (28)
\end{aligned}$$

In order to derive the gradients of the inner update $\nabla_\theta \theta' = \nabla_\theta U(\theta, \mathcal{M})$ it is necessary to know the structure of U. The main part of this paper assumes the inner update rule to be a policy gradient descent step

$$\begin{aligned}
\nabla_\theta U(\theta, \mathcal{M}) &= \nabla_\theta \left(\theta + \alpha\, \nabla_\theta \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\left[R(\boldsymbol{\tau})\right]\right) & (29)\\
&= I + \alpha \nabla^2_\theta\, \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\left[R(\boldsymbol{\tau})\right] & (30)
\end{aligned}$$

Thereby the second term in (30) is the local curvature (hessian) of the inner adaptation objective function. The correct hessian of the inner objective can be derived as follows:

$$\nabla_\theta^2 \, \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)} \left[R(\boldsymbol{\tau})\right] = \nabla_\theta \, \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)} \left[\nabla_\theta \log \pi_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau})\right] \tag{31}$$

$$= \nabla_\theta \int P_{\mathcal{M}}(\boldsymbol{\tau}|\theta) \nabla_\theta \log \pi_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau} \tag{32}$$

$$= \int P_{\mathcal{M}}(\boldsymbol{\tau}|\theta) \nabla_\theta \log \pi_\theta(\boldsymbol{\tau}) \nabla_\theta \log \pi_\theta(\boldsymbol{\tau})^\top R(\boldsymbol{\tau}) + \tag{33}$$

$$P_{\mathcal{M}}(\boldsymbol{\tau}|\theta) \nabla_\theta^2 \log \pi_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau}) d\boldsymbol{\tau} \tag{34}$$

$$= \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)} \left[R(\boldsymbol{\tau}) \left(\nabla_\theta^2 \log \pi_\theta(\boldsymbol{\tau}) + \nabla_\theta \log \pi_\theta(\boldsymbol{\tau}) \nabla_\theta \log \pi_\theta(\boldsymbol{\tau})^\top\right)\right] \tag{35}$$

c.1.2 *Meta-Policy Gradient Formulation II*

The second meta-reinforcement learning formulation views the the inner update $\theta' = U(\theta, \tau^{1:N})$ as a deterministic function of the pre-update policy parameters θ and N trajectories $\boldsymbol{\tau}^{1:N} \sim P_{\mathcal{M}}(\boldsymbol{\tau}^{1:N}|\theta)$ sampled from the pre-update trajectory distribution. This formulation was introduced in (Al-Shedivat et al., 2018) and further discussed with respect to its exploration properties in (Stadie et al., 2018).

Viewing U as a function that adapts the policy parameters θ to a specific task $\mathcal{M}$ given policy rollouts in this task, the corresponding meta-learning objective can be written as

$$J^{II}(\theta) = \mathbb{E}_{\mathcal{M}\sim\rho(\mathcal{M})}\left[\mathbb{E}_{\boldsymbol{\tau}^{1:N}\sim P_{\mathcal{M}}(\boldsymbol{\tau}^{1:N}|\theta)}\left[\mathbb{E}_{\boldsymbol{\tau}'\sim P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')}\left[R(\boldsymbol{\tau}')\right]\right]\right] \quad \text{with} \quad \theta' := U(\theta, \boldsymbol{\tau}^{1:N}) \tag{36}$$

Since the first part of the gradient derivation is agnostic to the inner update rule $U(\theta, \tau^{1:N})$, we only assume that the inner update function U is differentiable w.r.t. θ. First we rewrite the meta-objective $J(\theta)$ as expectation of task specific objectives $J^{II}_{\mathcal{M}}(\theta)$ under the task distribution. This allows us to express the meta-policy gradients as expectation of task-specific gradients:

$$\nabla_\theta J^{II}(\theta) = \mathbb{E}_{\mathcal{M}\sim\rho(\mathcal{M})}\left[\nabla_\theta J^{II}_{\mathcal{M}}(\theta)\right] \tag{37}$$

The task specific gradients can be calculated as follows

$$\begin{aligned}
\nabla_\theta J^{II}_{\mathcal{M}}(\theta) =& \quad \nabla_\theta \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}^{1:N}|\theta)}\left[\mathbb{E}_{\boldsymbol{\tau}'\sim P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')}\left[R(\boldsymbol{\tau}')\right]\right] \\
=& \quad \nabla_\theta \int\int R(\boldsymbol{\tau}')\ P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')\ P_{\mathcal{M}}(\boldsymbol{\tau}^{1:N}|\theta)\ d\boldsymbol{\tau}'\ d\boldsymbol{\tau} \\
=& \quad \int\int R(\boldsymbol{\tau}')\ P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')\ \nabla_\theta \log P_{\mathcal{M}}(\boldsymbol{\tau}^{1:N}|\theta) P_{\mathcal{M}}(\boldsymbol{\tau}^{1:N}|\theta) + \\
& \qquad R(\boldsymbol{\tau}')\ \nabla_\theta \log P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta') P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')\ P_{\mathcal{M}}(\boldsymbol{\tau}^{1:N}|\theta)\ d\boldsymbol{\tau}'\ d\boldsymbol{\tau} \\
=& \quad \mathbb{E}_{\substack{\boldsymbol{\tau}^{1:N}\sim P_{\mathcal{M}}(\boldsymbol{\tau}^{1:N}|\theta) \\ \boldsymbol{\tau}'\sim P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')}}\left[R(\boldsymbol{\tau}')\left(\nabla_\theta \log P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta') + \sum_{i=1}^{N} \nabla_\theta \log P_{\mathcal{M}}(\boldsymbol{\tau}^{(n)}|\theta)\right)\right] \\
=& \quad \mathbb{E}_{\substack{\boldsymbol{\tau}^{1:N}\sim P_{\mathcal{M}}(\boldsymbol{\tau}^{1:N}|\theta) \\ \boldsymbol{\tau}'\sim P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')}}\left[R(\boldsymbol{\tau}')\left(\nabla_{\theta'} \log P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')\nabla_\theta\theta' + \sum_{n=1}^{N} \nabla_\theta \log P_{\mathcal{M}}(\boldsymbol{\tau}^{(n)}|\theta)\right)\right]
\end{aligned}$$

As in C.1.1 the structure of $U(\theta, \boldsymbol{\tau}^{1:N})$ must be known in order to derive the gradient $\nabla_\theta\theta'$. Since we assume the inner update to be vanilla policy gradient, the respective gradient follows as

$$U(\theta, \tau^{1:N}) = \theta + \alpha\frac{1}{N}\sum_{n=1}^{N}\nabla_\theta \log \pi_\theta(\tau^{(n)}))R(\tau^{(n)}) \quad \text{with} \quad \nabla_\theta \log \pi_\theta(\tau) = \sum_{t=0}^{H-1}\nabla_\theta \log \pi_\theta(a_t|s_t)$$

The respective gradient of $U(\theta, \tau^{1:N})$ follows as

$$\nabla_\theta U(\theta, \tau^{1:N}) = \nabla_\theta\left(\theta + \alpha\frac{1}{N}\sum_{n=1}^{N}\nabla_\theta \log \pi_\theta(\tau^{(n)}))R(\tau^{(n)})\right) \tag{38}$$

$$= I + \alpha\frac{1}{N}\sum_{n=1}^{N}\nabla^2_\theta \log \pi_\theta(\tau^{(n)}))R(\tau^{(n)}) \tag{39}$$

c.1.3 *Comparing the Gradients of the Two Formulations*

In the following we analyze the differences between the gradients derived for the two formulations. To do so, we begin with $\nabla_\theta J^I_{\mathcal{M}}(\theta)$ by inserting the gradient of the inner adaptation step (30) into (28):

$$\nabla_\theta J^I_{\mathcal{M}}(\theta) = E_{\boldsymbol{\tau}'\sim P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')}\left[\nabla_{\theta'} \log P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')R(\boldsymbol{\tau}')\left(I + \alpha\nabla^2_\theta\ \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\left[R(\boldsymbol{\tau})\right]\right)\right] \tag{40}$$

We can substitute the hessian of the inner objective by its derived expression from (35) and then rearrange the terms. Also note that $\nabla_\theta \log P_{\mathcal{M}}(\boldsymbol{\tau}|\theta) = \nabla_\theta \log \pi_\theta(\boldsymbol{\tau}) = \sum_{t=1}^{H-1} \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ where H is the MDP horizon.

$$\nabla_\theta J^I_{\mathcal{M}}(\theta) = E_{\boldsymbol{\tau}' \sim P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')}\Big[\nabla_{\theta'} \log P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')R(\boldsymbol{\tau}')\Big(I + \alpha \mathbb{E}_{\boldsymbol{\tau} \sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\big[R(\boldsymbol{\tau}) \tag{41}$$

$$\big(\nabla^2_\theta \log \pi_\theta(\boldsymbol{\tau}) + \nabla_\theta \log \pi_\theta(\boldsymbol{\tau}) \nabla_\theta \log \pi_\theta(\boldsymbol{\tau})^\top\big)\big]\Big)\Big] \tag{42}$$

$$= \mathbb{E}_{\substack{\boldsymbol{\tau} \sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta) \\ \boldsymbol{\tau}' \sim P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')}}\Bigg[\underbrace{\nabla_{\theta'} \log \pi_{\theta'}(\boldsymbol{\tau}')R(\boldsymbol{\tau}')\Big(I + \alpha R(\boldsymbol{\tau})\nabla^2_\theta \log \pi_\theta(\boldsymbol{\tau})\Big)}_{\nabla_\theta J_{\text{post}}(\boldsymbol{\tau},\boldsymbol{\tau}')} \tag{43}$$

$$\underbrace{+\alpha \nabla_{\theta'} \log \pi_{\theta'}(\boldsymbol{\tau}')R(\boldsymbol{\tau}')R(\boldsymbol{\tau})\nabla_\theta \log \pi_\theta(\boldsymbol{\tau})\nabla_\theta \log \pi_\theta(\boldsymbol{\tau})^\top}_{\nabla_\theta J^I_{\text{pre}}(\boldsymbol{\tau},\boldsymbol{\tau}')}\Bigg] \tag{44}$$

Next, we rearrange the gradient of J^{II} into a similar form as $\nabla_\theta J^I_{\mathcal{M}}(\theta)$. For that, we start by inserting (39) for $\nabla_\theta \theta'$ and replacing the expectation over pre-update trajectories $\boldsymbol{\tau}^{1:N}$ by the expectation over a single trajectory $\boldsymbol{\tau}$.

$$\nabla_\theta J^I_{\mathcal{M}}(\theta) = \mathbb{E}_{\substack{\boldsymbol{\tau} \sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta) \\ \boldsymbol{\tau}' \sim P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')}}\Bigg[\underbrace{R(\boldsymbol{\tau}')\nabla_{\theta'} \log \pi_\theta(\boldsymbol{\tau}')\Big(I + \alpha R(\boldsymbol{\tau})\nabla^2_\theta \log \pi_\theta(\tau)\Big)}_{\nabla_\theta J_{\text{post}}(\boldsymbol{\tau},\boldsymbol{\tau}')} \tag{45}$$

$$\underbrace{+R(\boldsymbol{\tau}')\nabla_\theta \log \pi_\theta(\boldsymbol{\tau})}_{\nabla_\theta J^I_{\text{pre}}(\boldsymbol{\tau},\boldsymbol{\tau}')}\Bigg] \tag{46}$$

While the first part of the gradients match ((43) and (45)), the second part ((44) and (46)) differs. Since the second gradient term can be viewed as responsible for shifting the pre-update sampling distribution $P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)$ towards higher post-update returns, we refer to it as $\nabla_\theta J_{\text{pre}}(\boldsymbol{\tau}, \boldsymbol{\tau}')$. To further analyze the difference between $\nabla_\theta J^I_{\text{pre}}$ and $\nabla_\theta J^{II}_{\text{pre}}$ we slightly rearrange (44) and put both gradient terms next to each other:

$$\nabla_\theta J^{\mathrm{I}}_{\mathrm{pre}}(\boldsymbol{\tau}, \boldsymbol{\tau}') = \alpha \nabla_\theta \log \pi_\theta(\boldsymbol{\tau}) \left(\underbrace{(\nabla_\theta \log \pi_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau}))^\top}_{\nabla_\theta J^{\mathrm{inner}}} \underbrace{(\nabla_{\theta'} \log \pi_{\theta'}(\boldsymbol{\tau}') R(\boldsymbol{\tau}'))}_{\nabla_{\theta'} J^{\mathrm{outer}}} \right) \tag{47}$$

$$\nabla_\theta J^{\mathrm{II}}_{\mathrm{pre}}(\boldsymbol{\tau}, \boldsymbol{\tau}') = \alpha \nabla_\theta \log \pi_\theta(\boldsymbol{\tau}) R(\boldsymbol{\tau}') \tag{48}$$

In the following we interpret and and compare of the derived gradient terms, aiming to provide intuition for the differences between the formulations:

The first gradient term J_{post} that matches in both formulations corresponds to a policy gradient step on the post-update policy $\pi_{\theta'}$. Since θ' itself is a function of θ, the term $\left(I + \alpha R(\boldsymbol{\tau}) \nabla^2_\theta \log \pi_\theta(\tau))\right)$ can be seen as linear transformation of the policy gradient update $R(\boldsymbol{\tau}') \nabla_{\theta'} \log \pi_\theta(\boldsymbol{\tau}')$ from the post-update parameter θ' into θ. Although J_{post} takes into account the functional relationship between θ' and θ, it does not take into account the pre-update sampling distribution $P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)$.

This is where $\nabla_\theta J_{\mathrm{pre}}$ comes into play: $\nabla_\theta J^{\mathrm{I}}_{\mathrm{pre}}$ can be viewed as policy gradient update of the pre-update policy π_θ w.r.t. to the post-update return $R(\tau')$. Hence this gradient term aims to shift the pre-update sampling distribution so that higher post-update returns are achieved. However, $\nabla_\theta J^{\mathrm{II}}_{\mathrm{pre}}$ does not take into account the causal dependence of the post-update policy on the pre-update policy. Thus a change in θ due to $\nabla_\theta J^{\mathrm{II}}_{\mathrm{pre}}$ may counteract the change due to $\nabla_\theta J^{\mathrm{II}}_{\mathrm{post}}$. In contrast, $\nabla_\theta J^{\mathrm{I}}_{\mathrm{pre}}$ takes the dependence of the the post-update policy on the pre-update sampling distribution into account. Instead of simply weighting the gradients of the pre-update policy $\nabla_\theta \log \pi_\theta(\boldsymbol{\tau})$ with $R(\tau')$ as in $\nabla_\theta J^{\mathrm{I}}_{\mathrm{post}}$, $\nabla_\theta J^{\mathrm{I}}_{\mathrm{post}}$ weights the gradients with inner product of the pre-update and post-update policy gradients. This inner product can be written as

$$\nabla_\theta {J^{\mathrm{inner}}}^\top \nabla_{\theta'} J^{\mathrm{outer}} = \|\nabla_\theta J^{\mathrm{inner}}\|_2 \cdot \|\nabla_{\theta'} J^{\mathrm{outer}}\|_2 \cdot \cos(\delta) \tag{49}$$

wherein δ denotes the angle between the the inner and outer pre-update and post-update policy gradients. Hence, $\nabla_\theta J^{\mathrm{I}}_{\mathrm{post}}$ steers the pre-update policy towards not only towards larger post-updates returns but also towards larger adaptation steps $\alpha \nabla_\theta J^{\mathrm{inner}}$, and better alignment of pre- and post-update policy gradients. This directly optimizes for maximal improvement / adaptation for the respective task. See (Z. Li et al., 2017; Nichol et al., 2018) for a comparable analysis in case of domain generalization and supervised meta-learning. Also note that (49) allows formulation I to perform credit assignment on the trajectory level whereas formulation II can only assign credit to entire batches of N pre-update trajectories $\tau^{1:N}$.

As a result, we expect the first meta-policy gradient formulation to learn faster and more stably since the respective gradients take the dependence of the pre-update returns on the pre-update sampling distribution into account while this causal link is neglected in the second formulation.

C.2 ESTIMATING THE META-POLICY GRADIENTS

When employing formulation I for gradient-based meta-learning, we aim maximize the loss

$$J(\theta) = \mathbb{E}_{\mathcal{M} \sim \rho(\mathcal{M})} \left[\mathbb{E}_{\boldsymbol{\tau}' \sim P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')} \left[R(\boldsymbol{\tau}') \right] \right] \quad \text{with} \quad \theta' := \theta + \alpha \, \nabla_\theta \mathbb{E}_{\boldsymbol{\tau} \sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)} \left[R(\boldsymbol{\tau}) \right] \tag{50}$$

by performing a form of gradient-descent on $J(\theta)$. Note that we, from now on, assume $J := J^I$ and thus omit the superscript indicating the respective meta-learning formulation. As shown in C.1.2 the gradient can be derived as $\nabla_\theta J(\theta) = \mathbb{E}_{(T) \sim \rho(T)}[\nabla_\theta J_{\mathcal{M}}(\theta)]$ with

$$\nabla_\theta J_{\mathcal{M}}(\theta) = E_{\boldsymbol{\tau}' \sim P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta')} \left[\nabla_{\theta'} \log P_{\mathcal{M}}(\boldsymbol{\tau}'|\theta') R(\boldsymbol{\tau}') \left(I + \alpha \nabla_\theta^2 \, \mathbb{E}_{\boldsymbol{\tau} \sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)} \left[R(\boldsymbol{\tau}) \right] \right) \right] \tag{51}$$

where $\nabla_\theta^2 J_{\text{inner}}(\theta) := \nabla_\theta^2 \, \mathbb{E}_{\boldsymbol{\tau} \sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)} [R(\boldsymbol{\tau})]$ denotes hessian of the inner adaptation objective w.r.t. θ. This section concerns the question of how to properly estimate this hessian.

C.2.1 *Estimating Gradients of the RL Reward Objective*

Since the expectation over the trajectory distribution $P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)$ is in general intractable, the score function trick is typically used to used to produce a Monte Carlo estimate of the policy gradients. Although the gradient estimate can be directly defined, when using a automatic-differentiation toolbox it is usually more convenient to use an objective function whose gradients correspond to the policy gradient estimate. Due to the Policy Gradient Theorem (PGT) (Sutton et al., 2000) such a “surrogate" objective can be written as:

$$\hat{J}^{\text{PGT}} = \frac{1}{K} \sum_{\tau_k} \sum_{t=0}^{H-1} \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{H} r(s_{t'}, a_{t'}) \right) \quad \tau_k \sim P_{\mathcal{M}}(\tau) \tag{52}$$

$$= \frac{1}{K} \sum_{\tau_k} \sum_{t=0}^{H-1} \left(\sum_{t'=0}^{t} \log \pi_\theta(a_t|s_t) \right) r(s_{t'}, a_{t'}) \quad \tau_k \sim P_{\mathcal{M}}(\tau) \tag{53}$$

While (52) and (53) are equivalent (Peters and Schaal, 2006), the more popular formulation formulation (52) can be seen as forward looking credit assignment while (53) can be interpreted as backward looking credit assignment (Foerster et al., 2018). A generalized procedure for constructing "surrogate" objectives for arbitrary stochastic computation graphs can be found in (Schulman et al., 2015a).

C.2.2 *A decomposition of the hessian*

Estimating the the hessian of the reinforcement learning objective has been discussed in (Furmston et al., 2016) and (Bartlett and Baxter, 2011) with focus on second order policy gradient methods. In the infinite horizon MDP case, (Bartlett and Baxter, 2011) derive a decomposition of the hessian. In the following, we extend their finding to the finite horizon case.

Proposition. The hessian of the RL objective can be decomposed into four matrix terms:

$$\nabla_\theta^2 J_{\text{inner}}(\theta) = H_1 + H_2 + H_{12} + H_{12}^\top \tag{54}$$

where

$$H_1 = \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_t, \mathbf{s}_t) \nabla_\theta \log \pi_\theta(\mathbf{a}_t, \mathbf{s}_t)^\top \left(\sum_{t'=t}^{H-1} r(\mathbf{s}_{t'}, \mathbf{a}_{t'})\right)\right] \tag{55}$$

$$H_2 = \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\left[\sum_{t=0}^{H-1} \nabla_\theta^2 \log \pi_\theta(\mathbf{a}_t, \mathbf{s}_t) \left(\sum_{t'=t}^{H-1} r(\mathbf{s}_{t'}, \mathbf{a}_{t'})\right)\right] \tag{56}$$

$$H_{12} = \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(\mathbf{a}_t, \mathbf{s}_t) \nabla_\theta Q_t^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t)^\top\right] \tag{57}$$

Here $Q_t^{\pi_\theta}(\mathbf{s}_t, \mathbf{a}_t) = \mathbb{E}_{\boldsymbol{\tau}^{t+1:H-1}\sim P_{\mathcal{M}}(\cdot|\theta)}\left[\sum_{t'=t}^{H-1} r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | s_t, a_t\right]$ denotes the expected state-action value function under policy π_θ at time t.

Proof. As derived in (35), the hessian of $J_{\text{inner}}(\theta)$ follows as:

$$\nabla^2_\theta J_{\text{inner}} = \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\left[R(\boldsymbol{\tau})\left(\nabla^2_\theta \log\pi_\theta(\boldsymbol{\tau}) + \nabla_\theta \log\pi_\theta(\boldsymbol{\tau})\nabla_\theta \log\pi_\theta(\boldsymbol{\tau})^\top\right)\right] \tag{58}$$

$$= \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\left[\sum_{t=0}^{H-1}\left(\sum_{t'=0}^{t}\nabla^2_\theta \log\pi_\theta(\mathbf{a}_{t'},\mathbf{s}_{t'})\right) r(\mathbf{s}_t,\mathbf{a}_t)\right] \tag{59}$$

$$+ \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\left[\sum_{t=0}^{H-1}\left(\sum_{t'=0}^{t}\nabla_\theta \log\pi_\theta(\mathbf{a}_{t'},\mathbf{s}_{t'})\right)\left(\sum_{t'=0}^{t}\nabla_\theta \log\pi_\theta(\mathbf{a}_{t'},\mathbf{s}_{t'})\right)^\top r(\mathbf{s}_t,\mathbf{a}_t)\right] \tag{60}$$

$$= \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\left[\sum_{t=0}^{H-1}\nabla^2_\theta \log\pi_\theta(\mathbf{a}_t,\mathbf{s}_t)\left(\sum_{t'=t}^{H-1} r(\mathbf{s}_{t'},\mathbf{a}_{t'})\right)\right] \tag{61}$$

$$+ \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\left[\sum_{t=0}^{H-1}\left(\sum_{t'=0}^{t}\sum_{h=0}^{t}\nabla_\theta \log\pi_\theta(\mathbf{a}_{t'},\mathbf{s}_{t'})\nabla_\theta \log\pi_\theta(\mathbf{a}_h,\mathbf{s}_h)^\top\right) r(\mathbf{s}_t,\mathbf{a}_t)\right] \tag{62}$$

The term in (61) is equal to H_2. We continue by showing that the remaining term in (62) is equivalent to $H_1 + H_{12} + H_{12}^\top$. For that, we split the inner double sum in (62) into three components:

$$\mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\left[\sum_{t=0}^{H-1}\left(\sum_{t'=0}^{t}\sum_{h=0}^{t}\nabla_\theta \log\pi_\theta(\mathbf{a}_{t'},\mathbf{s}_{t'})\nabla_\theta \log\pi_\theta(\mathbf{a}_h,\mathbf{s}_h)^\top\right) r(\mathbf{s}_t,\mathbf{a}_t)\right] \tag{63}$$

$$= \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\left[\sum_{t=0}^{H-1}\left(\sum_{t'=0}^{t}\nabla_\theta \log\pi_\theta(\mathbf{a}_{t'},\mathbf{s}_{t'})\nabla_\theta \log\pi_\theta(\mathbf{a}_{t'},\mathbf{s}_{t'})^\top\right) r(\mathbf{s}_t,\mathbf{a}_t)\right] \tag{64}$$

$$+ \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\left[\sum_{t=0}^{H-1}\left(\sum_{t'=0}^{t}\sum_{h=0}^{t'-1}\nabla_\theta \log\pi_\theta(\mathbf{a}_{t'},\mathbf{s}_{t'})\nabla_\theta \log\pi_\theta(\mathbf{a}_h,\mathbf{s}_h)^\top\right) r(\mathbf{s}_t,\mathbf{a}_t)\right] \tag{65}$$

$$+ \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\left[\sum_{t=0}^{H-1}\left(\sum_{t'=0}^{t}\sum_{h=t'+1}^{t}\nabla_\theta \log\pi_\theta(\mathbf{a}_{t'},\mathbf{s}_{t'})\nabla_\theta \log\pi_\theta(\mathbf{a}_h,\mathbf{s}_h)^\top\right) r(\mathbf{s}_t,\mathbf{a}_t)\right] \tag{66}$$

By changing the backward looking summation over outer products into a forward looking summation of rewards, (64) can be shown to be equal to H_1:

$$\mathbb{E}_{\tau\sim P_{\mathcal{M}}(\tau|\theta)}\left[\sum_{t=0}^{H-1}\left(\sum_{t'=0}^{t}\nabla_\theta\log\pi_\theta(a_{t'},s_{t'})\nabla_\theta\log\pi_\theta(a_{t'},s_{t'})^\top\right)r(s_t,a_t)\right] \tag{67}$$

$$= \mathbb{E}_{\tau\sim P_{\mathcal{M}}(\tau|\theta)}\left[\sum_{t=0}^{H-1}\nabla_\theta\log\pi_\theta(a_t,s_t)\nabla_\theta\log\pi_\theta(a_t,s_t)^\top\left(\sum_{t'=t}^{H-1}r(s_{t'},a_{t'})\right)\right] \tag{68}$$

$$= H_1 \tag{69}$$

By simply exchanging the summation indices t' and h in (66) it is straightforward to show that (66) is the transpose of (65). Hence it is sufficient to show that (65) is equivalent to H_{12}. However, instead of following the direction of the previous proof we will now start with the definition of H_{12} and derive the expression in (65).

$$H_{12} = \mathbb{E}_{\tau\sim P_{\mathcal{M}}(\tau|\theta)}\left[\sum_{t=0}^{H-1}\nabla_\theta\log\pi_\theta(a_t,s_t)\nabla_\theta Q_t^{\pi_\theta}(s_t,a_t)^\top\right] \tag{70}$$

$$\tag{71}$$

The gradient of $Q_t^{\pi_\theta}$ can be expressed recursively:

$$\nabla_\theta Q_t^{\pi_\theta}(s_t,a_t) = \nabla_\theta\mathbb{E}_{\substack{s_{t+1}\\a_{t+1}}}\left[Q_{t+1}^{\pi_\theta}(s_{t+1},a_{t+1})\right] \tag{72}$$

$$= \mathbb{E}_{\substack{s_{t+1}\\a_{t+1}}}\left[\nabla_\theta\log\pi_\theta(a_{t+1},s_{t+1})Q_{t+1}^{\pi_\theta}(s_{t+1},a_{t+1}) + \nabla_\theta Q_{t+1}^{\pi_\theta}(s_{t+1},a_{t+1})\right] \tag{73}$$

By induction, it follows that

$$\nabla_\theta Q_t^{\pi_\theta}(s_t,a_t) = \mathbb{E}_{\tau^{t+1:H-1}\sim P_{\mathcal{M}}(\cdot|\theta)}\left[\sum_{t'=t+1}^{H-1}\nabla_\theta\log\pi_\theta(a_{t'},s_{t'})\left(\sum_{h=t'}^{H-1}r(s_h,a_h)\right)\right] \tag{74}$$

When inserting (74) into (70) and swapping the summation, we are able to show that H_{12} is equivalent to (65).

$$H_{12} = \mathbb{E}_{\tau\sim P_{\mathcal{M}}(\tau|\theta)}\left[\sum_{t=0}^{H-1}\sum_{t'=t+1}^{H-1}\nabla_\theta\log\pi_\theta(a_t,s_t)\nabla_\theta\log\pi_\theta(a_{t'},s_{t'})^\top\left(\sum_{h=t'}^{H-1}r(s_h,a_h)\right)\right] \tag{75}$$

$$= \mathbb{E}_{\tau\sim P_{\mathcal{M}}(\tau|\theta)}\left[\sum_{t=0}^{H-1}\left(\sum_{t'=0}^{t}\sum_{h=0}^{t'-1}\nabla_\theta\log\pi_\theta(a_{t'},s_{t'})\nabla_\theta\log\pi_\theta(a_h,s_h)^\top\right)r(s_t,a_t)\right] \tag{76}$$

This concludes the proof that the hessian of the expected sum of rewards under policy π_θ and an MDP with finite time horizon H can be decomposed into $H_1 + H_2 + H_{12} + H_{12}^\top$.

$\square$

C.2.3 *Estimating the Hessian of the RL Reward Objective*

As pointed out by (Al-Shedivat et al., 2018; Stadie et al., 2018) and (Foerster et al., 2018), simply differentiating through the gradient of surrogate objective J^{PGT} as done in the original MAML version (Finn et al., 2017) leads to biased hessian estimates. Specifically, when compared with the unbiased estimate, as derived in (35) and decomposed in Appendix C.2.2, both H_1 and $H_{12} + H_{12}^\top$ are missing. Thus, $\nabla_\theta J_{\text{pre}}$ does not appear in the gradients of the meta-objective (i.e. $\nabla_\theta J = \nabla_\theta J_{\text{post}}$). Only performing gradient descent with $\nabla_\theta J_{\text{post}}$ entirely neglects influences of the pre-update sampling distribution. This issue was overseen in the RL-MAML implementation of (Finn et al., 2017). As discussed in (Stadie et al., 2018) this leads to poor performance in meta-learning problems that require exploration during the pre-update sampling.

C.2.3.1 *The DICE Monte-Carlo Estimator*

Addressing the issue of incorrect higher-order derivatives of monte-carlo estimators, (Foerster et al., 2018) propose DICE which mainly builds upon an newly introduced MagicBox($\square$) operator. This operator allows to formulate monte-carlo estimators with correct higher-order derivatives. A DICE formulation of a policy gradient estimator reads as:

$$J^{\text{DICE}} = \sum_{t=0}^{H-1} \square_\theta(\{a^{t' \leqslant t}\}) r(s_t, a_t) \tag{77}$$

$$= \sum_{t=0}^{H-1} \exp\left(\sum_{t'=0}^{t} \log \pi_\theta(a_{t'}|s_{t'}) - \perp(\log \pi_\theta(a_{t'}|s_{t'}) \right) r(s_t, a_t) \tag{78}$$

In that, $\perp$ denotes a “stop_gradient" operator (i.e. $\perp(f_\theta(x)) \rightarrow f_\theta(x)$ but $\nabla_\theta \perp(f_\theta(x)) \rightarrow 0$). Note that $\rightarrow$ denotes a “evaluates to" and does not necessarily imply equality w.r.t. to gradients. Hence, $J^{\text{DICE}}(\theta)$ evaluates to the sum of rewards at 0th order but produces the unbiased gradients $\nabla_\theta^n J^{\text{DICE}}(\theta)$ when differentiated n-times (see (Foerster et al., 2018) for proof). To shed more light on the maverick DICE formulation, we rewrite (78) as follows:

$$J^{\mathrm{DICE}} = \sum_{t=0}^{H-1} \left(\prod_{t'=0}^{t} \frac{\pi_\theta(a_{t'}|s_{t'})}{\perp(\pi_\theta(a_{t'}|s_{t'}))} \right) r(s_t, a_t) \tag{79}$$

Interpreting this novel formulation, the MagicBox operator $\square_\theta(\{a^{t' \leqslant t}\})$ can be understood as "dry" importance sampling weight. At 0th order it evaluates to 1 and leaves the objective function unaffected, but when differentiated once it yields an estimator for the marginal rate of return due to a change in the policy-implied trajectory distribution.

In the following we show that on expectation 1) the gradients of (79) match standard policy gradients and 2) its hessian estimate is equal to the hessian of inner RL objective, derived in C.2.2.

$$\nabla_\theta J^{\mathrm{DICE}} = \sum_{t=0}^{H-1} \nabla_\theta \left(\prod_{t'=0}^{t} \frac{\pi_\theta(a_{t'}|s_{t'})}{\perp(\pi_\theta(a_{t'}|s_{t'}))} \right) r(s_t, a_t) \tag{80}$$

$$= \sum_{t=0}^{H-1} \left(\prod_{t'=0}^{t} \frac{\pi_\theta(a_{t'}|s_{t'})}{\perp(\pi_\theta(a_{t'}|s_{t'}))} \right) \left(\sum_{t'=0}^{t} \nabla_\theta \log \pi_\theta(a_{t'}|s_{t'}) \right) r(s_t, a_t) \tag{81}$$

$$\to \sum_{t=0}^{H-1} \left(\sum_{t'=0}^{t} \nabla_\theta \log \pi_\theta(a_{t'}|s_{t'}) \right) r(s_t, a_t) \tag{82}$$

Here, (82) corresponds to the backward looking credit assignment formulation of policy gradients $\nabla_\theta J^{\mathrm{PGT}}$ as discussed in C.2.1. Once again we take the derivative in order to obtain the Hessian of J^{DICE}:

$$\nabla_\theta^2 J^{\mathrm{DICE}} = \sum_{t=0}^{H-1} \nabla_\theta \left(\prod_{t'=0}^{t} \frac{\pi_\theta(a_{t'}|s_{t'})}{\perp(\pi_\theta(a_{t'}|s_{t'}))} \right) \left(\sum_{t'=0}^{t} \nabla_\theta \log \pi_\theta(a_{t'}|s_{t'}) \right) r(s_t, a_t) \tag{83}$$

$$+ \left(\prod_{t'=0}^{t} \frac{\pi_\theta(a_{t'}|s_{t'})}{\perp(\pi_\theta(a_{t'}|s_{t'}))} \right) \nabla_\theta \left(\sum_{t'=0}^{t} \nabla_\theta \log \pi_\theta(a_{t'}|s_{t'}) \right) r(s_t, a_t) \tag{84}$$

$$\to \sum_{t=0}^{H-1} \left(\sum_{t'=0}^{t} \nabla_\theta \log \pi_\theta(a_{t'}|s_{t'}) \right) \left(\sum_{t'=0}^{t} \nabla_\theta \log \pi_\theta(a_{t'}|s_{t'}) \right)^\top r(s_t, a_t) \tag{85}$$

$$+ \left(\sum_{t'=0}^{t} \nabla_\theta^2 \log \pi_\theta(a_{t'}|s_{t'}) \right) r(s_t, a_t) \tag{86}$$

In expectation, $\mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}[\nabla^2_\theta J^{\text{DICE}}]$ the DICE monte carlo estimate of the hessian is equivalent to the hessian of the inner objective. To show this, we use the expression of $\nabla^2_\theta J_{\text{inner}}$ (60):

$$\mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}[\nabla^2_\theta J^{\text{DICE}}] \tag{87}$$

$$= \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\Bigg[\sum_{t=0}^{H-1}\left(\sum_{t'=0}^{t}\nabla_\theta \log \pi_\theta(a_{t'}|s_{t'})\right)\left(\sum_{t'=0}^{t}\nabla_\theta \log \pi_\theta(a_{t'}|s_{t'})\right)^\top \tag{88}$$

$$r(s_t,a_t) + \left(\sum_{t'=0}^{t}\nabla^2_\theta \log \pi_\theta(a_{t'}|s_{t'})\right) r(s_t,a_t)\Bigg] \tag{89}$$

$$= H_1 + H_2 + H_{12} + H_{12}^\top \tag{90}$$

$$= \nabla^2_\theta J_{\text{inner}} \tag{91}$$

C.2.4 *Bias and variance of the curvature estimate*

As shown in the previous section, $\nabla^2_\theta J^{\text{DICE}}$ provides an unbiased estimate of the hessian of the inner objective $J_{\text{inner}} = \mathbb{E}_{\boldsymbol{\tau}\sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}[R(\boldsymbol{\tau})]$. However, recall the DICE objective involves a product of importance weights along the trajectory.

$$J^{\text{DICE}} = \sum_{t=0}^{H-1}\left(\prod_{t'=0}^{t}\frac{\pi_\theta(a_{t'}|s_{t'})}{\perp(\pi_\theta(a_{t'}|s_{t'}))}\right) r(s_t,a_t) \tag{92}$$

Taking the 2nd derivative of this product leads to the outer product of sums in (85) which is of high variance w.r.t to $\boldsymbol{\tau}$. Specifically, this outer product of sums can be decomposed into three terms $H_1 + H_{12} + H_{12}^\top$ (see Appendix C.2.2). As noted by (Furmston et al., 2016), $H_{12} + H_{12}^\top$ is particularly difficult to estimate. In section 6.7.2 we empirically show that the high variance curvature estimates obtained with the DICE objective require large batch sizes and impede sample efficient learning.

In the following we develop a low variance curvature (LVC) estimator J^{LVC} which matches J^{DICE} at the gradient level and yields lower-variance estimates of the hessian by neglecting $H_{12} + H_{12}^\top$. Before formally introducing J^{LVC}, we motivate such estimator starting with the policy gradient estimate that was originally derived in (Sutton et al., 2000), followed by marginalizing the trajectory level distribution $P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)$ over states s_t

and actions a_t. Note that we omit reward baselines for notational simplicity.

$$\nabla_\theta J_{\text{inner}} = \mathbb{E}_{\boldsymbol{\tau} \sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)} \left[\sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{H-1} r(s_{t'}, a_{t'}) \right) \right] \tag{93}$$

$$= \sum_{t=0}^{H-1} \mathbb{E}_{\substack{s_t \sim p_t^{\pi_\theta}(s_t) \\ a_t \sim \pi_\theta(a_t|s_t)}} \left[\nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{H-1} r(s_{t'}, a_{t'}) \right) \right] \tag{94}$$

In that, $p_t^{\pi_\theta}(s_t)$ denotes the state visitation frequency at time step t, i.e. the probability density of being in s_t after t steps under the policy π_θ. In the general case $p_t^{\pi_\theta}(s_t)$ is intractable but depends on the policy parameter θ. We make the simplifying assumption that $p_t^{\pi_\theta}(s_t)$ is fixed in a local region of θ. Since we make this assumption at the gradient level, this corresponds to a 1st order Taylor expansion of $p_t^{\pi_\theta}(s_t)$ in θ. Note that this assumption is also used in the Monotonic Policy Improvement Theory (Kakade and Langford, 2002; Schulman et al., 2015b). Based on this condition, the hessian follows as derivative of (94) whereby a "stop_gradient" expression around the state visitation frequency $p_t^{\pi_\theta}(s_t)$ resembles the 1st order Taylor approximation:

$$\mathbb{E}_\tau \left[\nabla_\theta^2 J^{\text{LVC}} \right] = \nabla_\theta \sum_{t=0}^{H-1} \mathbb{E}_{\substack{s_t \sim \perp(p_t^{\pi_\theta}(s_t)) \\ a_t \sim \pi_\theta(a_t|s_t)}} \left[\nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{H-1} r(s_{t'}, a_{t'}) \right) \right] \tag{95}$$

$$= \sum_{t=0}^{H-1} \mathbb{E}_{\substack{s_t \sim \perp(p_t^{\pi_\theta}(s_t)) \\ a_t \sim \pi_\theta(a_t|s_t)}} \left[\nabla_\theta \log \pi_\theta(a_t|s_t) \nabla_\theta \log \pi_\theta(a_t|s_t)^\top \left(\sum_{t'=t}^{H-1} r(s_{t'}, a_{t'}) \right) \right. \tag{96}$$

$$\left. + \nabla_\theta^2 \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{H-1} r(s_{t'}, a_{t'}) \right) \right] \tag{97}$$

Since the expectation in (95) is intractable it must be evaluated by a monte carlo estimate. However, simply replacing the expectation with an average of samples trajectories induces a wrong hessian that does not correspond to (97) since outer product of log-gradients would be missing when differentiated. To ensure that automatic differentiation still yields the correct hessian, we add a "dry" importance weight comparable to DICE:

$$\nabla_\theta J^{\text{LVC}} = \sum_{t=0}^{H-1} \frac{\pi_\theta(a_t|s_t)}{\perp(\pi_\theta(a_t|s_t))} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{H-1} r(s_{t'}, a_{t'}) \right) \qquad \boldsymbol{\tau} \sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta) \tag{98}$$

When integrated this resembles the LVC "surrogate" objective J^{LVC}.

$$J^{LVC} = \sum_{t=0}^{H-1} \frac{\pi_\theta(a_t|s_t)}{\perp(\pi_\theta(a_t|s_t))} \left(\sum_{t'=t}^{H-1} r(s_{t'}, a_{t'}) \right) \quad \boldsymbol{\tau} \sim P_{\mathcal{M}}(\boldsymbol{\tau}|\boldsymbol{\theta}) \tag{99}$$

The gradients of J^{LVC} match $\nabla_\theta J^{DICE}$ and resemble an unbiased policy gradient estimate:

$$\nabla_\theta J^{LVC} = \sum_{t=0}^{H-1} \frac{\nabla_\theta \pi_\theta(a_t|s_t)}{\perp(\pi_\theta(a_t|s_t))} \left(\sum_{t'=t}^{H-1} r(s_{t'}, a_{t'}) \right) \tag{100}$$

$$= \sum_{t=0}^{H-1} \frac{\pi_\theta(a_t|s_t)}{\perp(\pi_\theta(a_t|s_t))} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{H-1} r(s_{t'}, a_{t'}) \right) \tag{101}$$

$$\to \sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{H-1} r(s_{t'}, a_{t'}) \right) \tag{102}$$

The respective Hessian can be obtained by differentiating (101):

$$\nabla_\theta^2 J^{LVC} = \nabla_\theta \sum_{t=0}^{H-1} \frac{\pi_\theta(a_t|s_t)}{\perp(\pi_\theta(a_t|s_t))} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{H-1} r(s_{t'}, a_{t'}) \right) \tag{103}$$

$$= \sum_{t=0}^{H-1} \frac{\pi_\theta(a_t|s_t)}{\perp(\pi_\theta(a_t|s_t))} \nabla_\theta \log \pi_\theta(a_t|s_t) \nabla_\theta \log \pi_\theta(a_t|s_t)^\top \left(\sum_{t'=t}^{H-1} r(s_{t'}, a_{t'}) \right) \tag{104}$$

$$+ \frac{\pi_\theta(a_t|s_t)}{\perp(\pi_\theta(a_t|s_t))} \nabla_\theta^2 \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{H-1} r(s_{t'}, a_{t'}) \right) \tag{105}$$

$$\to \sum_{t=0}^{H-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \nabla_\theta \log \pi_\theta(a_t|s_t)^\top \left(\sum_{t'=t}^{H-1} r(s_{t'}, a_{t'}) \right) \tag{106}$$

$$+ \nabla_\theta^2 \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{H-1} r(s_{t'}, a_{t'}) \right) \tag{107}$$

$$= \sum_{t=0}^{H-1} \left(\sum_{t'=0}^{t} \nabla_\theta \log \pi_\theta(a_{t'}|s_{t'}) \nabla_\theta \log \pi_\theta(a_t|s_t)^\top \right) r(s_t, a_t) \tag{108}$$

$$+ \left(\sum_{t'=0}^{t} \nabla_\theta^2 \log \pi_\theta(a_{t'}|s_{t'}) \right) r(s_t, a_t) \tag{109}$$

In expectation $\nabla_\theta^2 J^{\mathrm{LVC}}$ is equivalent to $H_1 + H_2$:

$$\mathbb{E}_{\boldsymbol{\tau} \sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\left[J^{\mathrm{LVC}}\right] = \mathbb{E}_{\boldsymbol{\tau} \sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\left[\sum_{t=0}^{H-1}\left(\sum_{t'=0}^{t} \nabla_\theta \log \pi_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'}) \nabla_\theta \log \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)^\top\right) r(\mathbf{s}_t, \mathbf{a}_t)\right] \tag{110}$$

$$+ \mathbb{E}_{\boldsymbol{\tau} \sim P_{\mathcal{M}}(\boldsymbol{\tau}|\theta)}\left[\sum_{t=0}^{H-1}\left(\sum_{t'=0}^{t} \nabla_\theta^2 \log \pi_\theta(\mathbf{a}_{t'}|\mathbf{s}_{t'})\right) r(\mathbf{s}_t, \mathbf{a}_t)\right] \tag{111}$$

$$= H_1 + H_2 \tag{112}$$

The Hessian $\nabla_\theta^2 J^{\mathrm{LVC}}$ no longer provides an unbiased estimate of $\nabla_\theta^2 J_{\mathrm{inner}}$ since neglects the matrix term $H_{12} + H_{12}^\top$. This approximation is based on the assumption that the state visitation distribution is locally unaffected by marginal changes in θ and leads to a substantial reduction of variance in the hessian estimate. (Furmston et al., 2016) show that under certain conditions (i.e. infinite horizon MDP, sufficiently rich policy parameterisation) the term $H_{12} + H_{12}^\top$ vanishes around a local optimum θ^*. Given that the conditions hold, this implies that $\mathbb{E}_{\boldsymbol{\tau}}[\nabla_\theta^2 J^{\mathrm{LVC}}] \to \mathbb{E}_{\boldsymbol{\tau}}[\nabla_\theta^2 J^{\mathrm{DICE}}]$ as $\theta \to \theta^*$, i.e. the bias of the LCV estimator becomes negligible close to the local optimum. The experiments in section 6.7.2 confirm this theoretical argument empirically and show that using the low variance curvature estimates obtained through J^{LVC} improve the sample-efficiency of meta-learning by a significant margin.

C.3 PROXIMAL POLICY SEARCH METHODS

C.3.1 *Monotonic Policy Improvement Theory*

This section provides a brief introduction to policy performance bounds and the theory of monotonic policy improvement in the setting of reinforcement learning. While Section 6.5 discusses the extension of this theory to meta learning, the following explanations assume a standard RL setting where $\mathcal{M}$ is exogenously given. Hence, we will omit mentioning the dependence on $\mathcal{M}$ for notational brevity. Since the monotonic policy improvement frameworks relies on infinite-time horizon MDPs, we assume $H \to \infty$ for the remainder of this chapter.

In addition to the expected reward $J(\pi)$ under policy π, we will use the state value function V^π, the state-action value function Q^π as well as the advantage function A^π:

$$V^\pi(s) = \mathbb{E}_{a_0, s_1, \ldots}\left[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t) \middle| \mathbf{s}_t = s\right]$$

$$Q^\pi(s, a) = \mathbb{E}_{s_1, a_1, \ldots}\left[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t) \middle| \mathbf{s}_t = s, \mathbf{a}_0 = a\right] = r(s, a) + \gamma \mathbb{E}_{s' \sim p(s'|s,a)}\left[V_\pi(s')\right]$$

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$$

with $\mathbf{a}_t \sim \pi(a_t|s_t)$ and $\mathbf{s}_{t+1} \sim p(s_{t+1}|s_t, a_t)$.

The expected return under a policy $\tilde{\pi}$ can be expressed as the sum of the expected return of another policy π and the expected discounted advantage of $\tilde{\pi}$ over π (see (Schulman et al., 2015b) for proof).

$$J(\tilde{\pi}) = J(\pi) + \mathbb{E}_{\boldsymbol{\tau} \sim P(\boldsymbol{\tau}, \tilde{\pi})}\left[\sum_{t=0}^{\infty} \gamma^t A^\pi(\mathbf{s}_t, \mathbf{a}_t)\right]$$

Let d_π denote the discounted state visitation frequency:

$$d_\pi(\mathbf{s}) = \gamma_t \sum_{t=0}^{\infty} p(s_t = \mathbf{s}|\pi)$$

We can use d_π to express the expectation over trajectories $\boldsymbol{\tau} \sim p^\pi(\tau)$ in terms of states and actions:

$$J(\tilde{\pi}) = J(\pi) + \mathbb{E}_{\substack{\mathbf{s} \sim d_{\tilde{\pi}}(\mathbf{s}) \\ \mathbf{a} \sim \tilde{\pi}(\mathbf{a}|\mathbf{s})}}\left[A^\pi(\mathbf{s}, \mathbf{a})\right] \tag{113}$$

Local policy search aims to find a policy update $\pi \to \tilde{\pi}$ in the proximity of π so that $J(\tilde{\pi})$ is maximized. Since $J(\pi)$ is not affected by the policy update $\pi \to \tilde{\pi}$, it is sufficient to maximize the expected advantage under $\tilde{\pi}$. However, the complex dependence of $d_{\tilde{\pi}}(\mathbf{s})$ on $\tilde{\pi}$ makes it hard to directly maximize the objective in (113). Using a local approximation of (113) where it is assumed that the state visitation frequencies d_π and $d_{\tilde{\pi}}$ are identical, the optimization can be phrased as

$$\tilde{J}_\pi(\tilde{\pi}) = J(\pi) + \mathbb{E}_{\substack{\mathbf{s} \sim d_\pi(\mathbf{s}) \\ \mathbf{a} \sim \tilde{\pi}(\mathbf{a}|\mathbf{s})}}\left[A^\pi(\mathbf{s}, \mathbf{a})\right] = J(\pi) + \mathbb{E}_{\substack{\mathbf{s} \sim d_\pi(\mathbf{s}) \\ \mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})}}\left[\frac{\tilde{\pi}(\mathbf{a}|\mathbf{s})}{\pi(\mathbf{a}|\mathbf{s})} A^\pi(\mathbf{s}, \mathbf{a})\right] \tag{114}$$

In the following we refer to $\tilde{J}(\tilde{\pi})$ as surrogate objective. It can be shown that the surrogate objective $\tilde{J}$ matches J to first order when $\pi = \tilde{\pi}$ (see (Kakade and Langford, 2002)). If π_θ

is a parametric and differentiable function with parameter vector θ, this means that for any θ_o:

$$\tilde{J}_{\pi_{\theta_o}}(\pi_{\theta_o}) = J_{\pi_{\theta_o}}(\pi_{\theta_o}) \quad \text{and} \quad \nabla_\theta \tilde{J}_{\pi_{\theta_o}}(\pi_\theta)\big|_{\theta_o} = \nabla_\theta J_{\pi_{\theta_o}}(\pi_\theta)\big|_{\theta_o} \tag{115}$$

When $\pi \neq \tilde{\pi}$, an approximation error of the surrogate objective $\tilde{J}$ w.r.t. to the true objective J is introduced. (Achiam et al., 2017) derive a lower bound for the true expected return of $\tilde{\pi}$:

$$J(\tilde{\pi}) \geqslant J_\pi(\tilde{\pi}) - C\sqrt{\mathbb{E}_{s\sim d_\pi}\left[\mathcal{D}_{KL}[\tilde{\pi}(\cdot|\mathbf{s})||\pi(\cdot|\mathbf{s})]\right]} = J_\pi(\tilde{\pi}) - C\sqrt{\bar{\mathcal{D}}_{KL}[\tilde{\pi}||\pi]} \tag{116}$$

with $C = \frac{\sqrt{2}\gamma}{1-\gamma}\max_s |\mathbb{E}_{a\sim\tilde{\pi}(a,s)}[A^\pi(s,\mathbf{a})]|$

C.3.2 *Trust Region Policy Optimization (TRPO)*

Trust region policy optimization (TPRO) (Schulman et al., 2015b) attempts to approximate the bound in (116) by phrasing local policy search as a constrained optimization problem:

$$\underset{\theta}{\arg\max}\ \mathbb{E}_{\substack{s\sim d_{\pi_{\theta_o}}(s)\\ a\sim\pi_{\theta_o}(a|s)}}\left[\frac{\pi_\theta(\mathbf{a}|\mathbf{s})}{\pi_{\theta_o}(\mathbf{a}|\mathbf{s})}A^{\pi_{\theta_o}}(\mathbf{s},\mathbf{a})\right] \quad \text{s.t.} \quad \bar{\mathcal{D}}_{KL}[\pi_{\theta_o}||\pi_\theta] \leqslant \delta \tag{117}$$

Thereby the KL-constraint δ induces a local trust region around the current policy π_{θ_o}. A practical implementation of TPRO uses a quadratic approximation of the KL-constraint which leads to the following update rule:

$$\theta \leftarrow \theta + \sqrt{\frac{2\delta}{g^\top F g}}F^{-1}g \tag{118}$$

with $g := \nabla_\theta \mathbb{E}_{\substack{s\sim d_{\pi_{\theta_o}}(s)\\ a\sim\pi_{\theta_o}(a|s)}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_o}(a|s)}A^{\pi_{\theta_o}}(\mathbf{s},\mathbf{a})\right]$ being the gradient of the objective and $F = \nabla^2_\theta \bar{\mathcal{D}}_{KL}[\pi_{\theta_o}||\pi_\theta]$ the Fisher information matrix of the current policy π_{θ_o}. In order to avoid the cubic time complexity that arise when inverting F, the Conjugate Gradient (CG) algorithm is typically used to approximate the Hessian vector product $F^{-1}g$.

C.3.3 *Proximal Policy Optimization (PPO)*

While TPRO is framed as constrained optimization, the theory discussed in Appendix C.3.1 suggest to optimize the lower bound. Based on this insight, (Schulman et al., 2017) propose adding a KL penalty to the objective and solve the following unconstrained optimization problem:

$$\arg\max_{\theta} \; \mathbb{E}_{\substack{s\sim d_{\pi_{\theta_o}}(s) \\ a\sim\pi_{\theta_o}(a|s)}} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_o}(a|s)} A^{\pi_{\theta_o}}(s,a) - \beta \mathcal{D}_{KL}[\pi_{\theta_o}(\cdot|s) || \pi_\theta(\cdot|s)] \right] \tag{119}$$

However, they also show that it is not sufficient to set a fixed penalty coefficient β and propose two alternative methods, known as Proximal Policy Optimization (PPO) that aim towards alleviating this issue:

1) Adapting the KL coefficient β so that a desired target KL-divergence $\bar{\mathcal{D}}_{KL}[\pi_{\theta_o} || \pi_\theta]$ between the policy before and after the parameter update is achieved

2) Clipping the likelihood ratio so that the optimization has no incentive to move the policy π_θ too far away from the original policy π_{θ_o}. A corresponding optimization objective reads as:

$$J^{CLIP} = \mathbb{E}_{\substack{s\sim d_{\pi_{\theta_o}}(s) \\ a\sim\pi_{\theta_o}(a|s)}} \left[\min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_o}(a|s)} A^{\pi_{\theta_o}}(s,a) \; , \; \text{clip}_{1-\epsilon}^{1+\epsilon}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_o}(a|s)} \right) A^{\pi_{\theta_o}}(s,a) \right) \right] \tag{120}$$

Empirical results show that the latter approach leads to better learning performance (Schulman et al., 2017).

Since PPO objective keeps π_θ in proximity of π_{θ_o}, it allows to perform multiple gradient steps without re-sampling trajectories from the updated policy. This property substantially improves the data-efficiency of PPO over vanilla policy gradient methods which need to re-estimate the gradients after each step.

C.4 EXPERIMENTS

C.4.1 *Hyperparameter Choice*

The optimal hyperparameter for each algorithm was determined using parameter sweeps. Table 3 contains the hyperparameter settings used for the different algorithms. Any environment specific modifications are noted in the respective paragraph describing the environment.

All Algorithms	
Policy Hidden Layer Sizes	(64, 64)
	(128, 128) for Humanoid
Num Adapt Steps	1
Inner Step Size α	0.01
Tasks Per Iteration	40
Trajectories Per Task	20
ProMP	
Outer Learning Rate β	0.001
Grad Steps Per ProMP Iteration	5
Outer Clip Ratio ϵ	0.3
KL Penalty Coef. η	0.0005
MAML-TRPO	
Trust Region Size	0.01
MAML-VPG	
Outer Learning Rate β	0.001

Table 3: Hyperparameter settings used in each algorithm

C.4.2 *Environment Specifications*

PointEnv (used in the experiment in 6.7.3)

- Trajectory Length : 100
- Num Adapt Steps : 3

In this environment, each task corresponds to one corner of the area. The point mass must reach the goal by applying directional forces. The agent only experiences a reward when within a certain radius of the goal, and the magnitude of the reward is equal to the distance to the goal.

HalfCheetahFwdBack, AntFwdBack, WalkerFwdBack, HumanoidFwdBack

- Trajectory Length : 100 (HalfCheetah, Ant); 200 (Humanoid, Walker)
- Num Adapt Steps: 1

The task is chosen between two directions - forward and backward. Each agent must run along the goal direction as far as possible, with reward equal to average velocity minus control costs.

AntRandDirec, HumanoidRandDirec

- Trajectory Length : 100 (Ant); 200 (Humanoid)
- Num Adapt Steps: 1

Each task corresponds to a random direction in the XY plane. As above, each agent must learn to run in that direction as far as possible, with reward equal to average velocity minus control costs.

AntRandGoal

- Trajectory Length : 200
- Num Adapt Steps: 2

In this environment, each task is a location randomly chosen from a circle in the XY plane. The goal is not given to the agent - it must learn to locate, approach, and stop at the target. The agent receives a penalty equal to the distance from the goal.

HopperRandParams, WalkerRandParams

- Trajectory Length : 200
- Inner LR : 0.05
- Num Adapt Steps: 1

The agent must move forward as quickly as it can. Each task is a different randomization of the simulation parameters, including friction, joint mass, and inertia. The agent receives a reward equal to its velocity.

C.4.3 *Further Experiments Results*

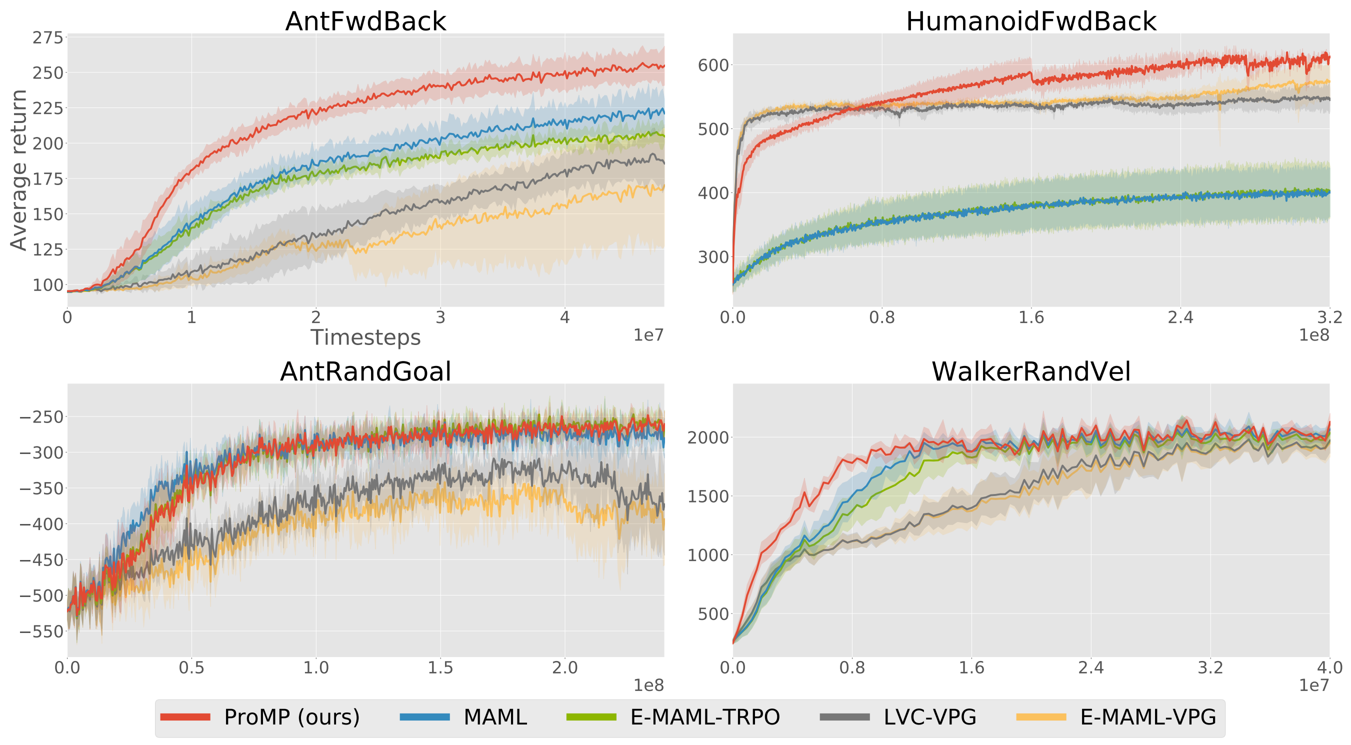


Figure 42: Meta-learning curves of ProMP and four other gradient-based meta-learning algorithms in four new Mujoco environments

In addition to the six environments displayed in Figure 13, we ran experiments on the other four continuous control environments described above. The results are displayed in 42.

In addition to the improved sample complexity and better asymptotic performance, another advantage of ProMP is its computation time. Figure 43 shows the average time spent per iteration throughout the learning process in the humanoid environment differences of ProMP, LVC-VPG, and MAML-TRPO. Due to the expensive conjugate gradient steps used in TRPO, MAML takes far longer than either first order method. Since ProMP takes multiple stochastic gradient descent steps per iteration, it leads to longer outer update times compared to VPG, but in both cases the update time is a fraction of the time spent sampling from the environment.

The difference in sampling time is due to the reset process: resetting the environment when the agent “dies" is an expensive operation. ProMP acquires better performance quicker, and as a result the agent experiences longer trajectories and the environment is reset less often. In our setup, instances of the environment are run in parallel and performing a reset blocks all environments.

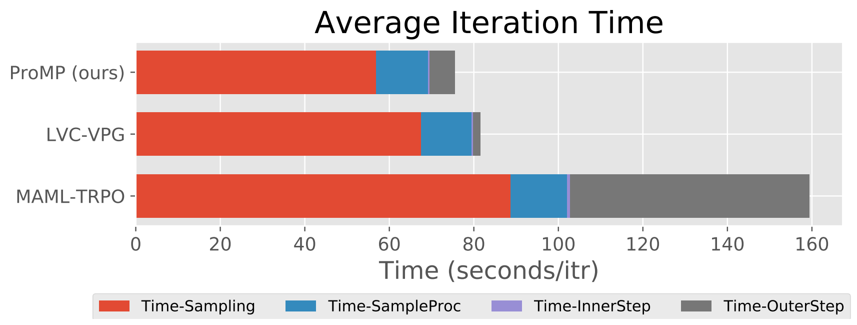


Figure 43: Comparison of wall clock time with different algorithms on HumanoidRandDirec, averaged over all iterations

D

APPENDIX: IMPROVING MODEL-BASED REINFORCEMENT LEARNING VIA MODEL-AUGMENTED PATHWISE DERIVATIVE

D.1 APPENDIX

Here we prove the lemmas and theorems stated in the manuscript.

D.1.1 *Proof of Lemma 1*

Let $J_\pi(\theta)$ and $\hat{J}_\pi(\hat{\theta})$ be the expected return of the policy π_θ under our objective and under the RL objective, respectively. Then, we can write the MSE of the gradient as

$$\begin{aligned}\mathbb{E}[\|\nabla_\theta J_\pi(\theta) - \nabla_\theta \hat{J}_\pi(\theta)\|_2] &= \mathbb{E}[\|\nabla_\theta(M - \hat{M}) + |\nabla_\theta \gamma^H (Q - \hat{Q})\|_2] \\ &\leqslant \mathbb{E}[\|\nabla_\theta(M - \hat{M})\|_2] + \mathbb{E}[\|\nabla_\theta \gamma^H (Q - \hat{Q})\|_2]\end{aligned}$$

whereby, $M = \sum_{t=0}^{H} \gamma^t r(\mathbf{s}_t)$ and $\hat{M} = \sum_{t=0}^{H} \gamma^t r(\hat{s}_t)$.

We will denote as ∇ the gradient w.r.t the inputs of network, $x_t = (\mathbf{s}_t, \mathbf{a}_t)$ and $\hat{x}_t = (\hat{s}_t, \hat{a}_t)$; where $\hat{a}_t \sim \pi_\theta(\hat{s}_t)$. Notice that since $f \hat{f}$ and π are Lipschitz and their gradient is Lipschitz as well, we have that $\nabla_\theta \hat{x}_t \leqslant K^t$, where K depends on the Lipschitz constants of the model and the policy. Without loss of generality, we assume that K is larger than 1. Now, we can bound the error on the Q as

$$\begin{aligned}\|\nabla_\theta(Q - \hat{Q})\|_2 =& \|\nabla Q \nabla_\theta x_H - \nabla \hat{Q} \nabla_\theta \hat{x}_H\|_2 \\ &= \|(\nabla Q - \nabla \hat{Q}) \nabla_\theta x_H - \nabla \hat{Q}(\nabla_\theta \hat{x}_H - \nabla_\theta x_H)\|_2 \\ &\leqslant \|\nabla Q - \nabla \hat{Q}\|_2 \|\nabla_\theta x_H\|_2 + \|\nabla \hat{Q}\|_2 \|\nabla_\theta \hat{x}_H - \nabla_\theta x_H\|_2 \\ &\leqslant \epsilon_Q \|\nabla_\theta x_H\|_2 + L_Q \|\nabla_\theta \hat{x}_H - \nabla_\theta x_H\|_2 \\ &\leqslant \epsilon_Q K^H + L_Q \|\nabla_\theta \hat{x}_H - \nabla_\theta x_H\|_2\end{aligned}$$

Now, we will bound the term $\|\nabla_\theta \hat{s}_{t+1} - \nabla_\theta \mathbf{s}_{t+1}\|_2$:

$$\begin{aligned}
\|\nabla_\theta \hat{s}_{t+1} - \nabla_\theta \mathbf{s}_{t+1}\|_2 &= \|\nabla_s \hat{f} \nabla_\theta \hat{s}_t + \nabla_a \hat{f} \nabla_\theta \hat{a}_t - \nabla_s f \nabla_\theta \mathbf{s}_t - \nabla_a f \nabla_\theta \mathbf{a}_t\|_2 \\
&\leqslant \|\nabla_s \hat{f} \nabla_\theta \hat{s}_t - \nabla_s f \nabla_\theta \mathbf{s}_t\|_2 + \|\nabla_a \hat{f} \nabla_\theta \hat{a}_t - \nabla_a f \nabla_\theta \mathbf{a}_t\|_2 \\
&\leqslant \epsilon_f \|\nabla_\theta \hat{s}_t\|_2 + L_f \|\nabla_\theta \hat{s}_t - \nabla_\theta \mathbf{s}_t\|_2 + L_f \|\nabla_\theta \hat{a}_t - \nabla_\theta \mathbf{a}_t\| + \epsilon_f \|\nabla_\theta \hat{a}_t\|_2 \\
&\leqslant \epsilon_f \|\nabla_\theta \hat{s}_t\|_2 + (L_f + L_f L_\pi) \|\nabla_\theta \hat{s}_t - \nabla_\theta \mathbf{s}_t\|_2 + \epsilon_f \|\nabla_\theta \hat{a}_t \\
&= \epsilon_f \|\nabla_\theta \hat{x}_t\|_2 + (L_f + L_f L_\pi) \|\nabla_\theta \hat{s}_t - \nabla_\theta \mathbf{s}_t\|_2
\end{aligned}$$

Hence, applying this recursion we obtain that

$$\|\nabla_\theta \hat{x}_{t+1} - \nabla_\theta x_{t+1}\|_2 \leqslant \epsilon_f \sum_{k=0}^{t} (L_f + L_f L_\pi)^{t-k} \|\nabla_\theta \hat{x}_k\|_2 \leqslant \epsilon_f \frac{L^{t+1} - 1}{L - 1} K^t$$

where $L = L_f + L_f L_\pi$. Then, the error in the gradient in the previous term is bounded by

$$\|\nabla_\theta (Q - \hat{Q})\|_2 \leqslant \epsilon_Q K^H + L_Q \epsilon_f \frac{L^H - 1}{L - 1} K^H$$

In order to bound the model term we need first to bound the rewards since

$$\|\nabla_\theta (M - \hat{M})\|_2 \leqslant \sum_{t=0}^{H} \gamma^t \|\nabla_\theta (r(\mathbf{s}_t) - r(\hat{s}_t))\|_2$$

Similar to the previous bounds, we can bound now each reward term by

$$\|\nabla_\theta (r(\mathbf{s}_t) - r(\hat{s}_t))\|_2 \leqslant \epsilon_f L_r \frac{L^{t+1} - 1}{L - 1} K^t$$

With this result we can bound the total error in models

$$\|\nabla_\theta (M - \hat{M})\|_2 \leqslant \sum_{t=0}^{H-1} \gamma^t \epsilon_f L_r \frac{L^{t+1} - 1}{L - 1} K^t = \frac{L\epsilon_f}{(L-1)} \left(\frac{(\gamma K L)^H - 1}{\gamma K L - 1} - \frac{(\gamma K)^H - 1}{\gamma K - 1} \right)$$

Then, the gradient error has the form

$$\begin{aligned}
&\mathbb{E}[\|\nabla_\theta J_\pi(\theta) - \nabla_\theta \hat{J}_\pi(\theta)\|_2] \leqslant \\
\frac{L\epsilon_f}{(L-1)} \left(\frac{(\gamma K L)^H - 1}{\gamma K L - 1} - \frac{(\gamma K)^H - 1}{\gamma K - 1} \right) &+ \epsilon_Q (\gamma K)^H + L_Q \epsilon_f \frac{L^H - 1}{L - 1} (\gamma K)^H \\
&= \epsilon_f c_1(H) + \epsilon_Q c_2(H)
\end{aligned}$$

D.1.2 *Proof of Lemma 2*

The total variation distance can be bounded by the KL-divergence using the Pinsker's inequality

$$D_{TV}(\pi_\theta \| \pi_{\hat{\theta}}) \leqslant \sqrt{\frac{D_{KL}(\pi_\theta \| \pi_{\hat{\theta}})}{2}}$$

Then if we assume third order smoothness on our policy, by the Fisher information metric theorem then

$$D_{KL}(\pi_\theta \| \pi_{\hat{\theta}}) = \tilde{c}\|\theta - \hat{\theta}\|_2^2 + \mathcal{O}(\|\theta - \hat{\theta}\|_2^3)$$

Given that $\|\theta - \hat{\theta}\|_2 = \alpha\|\nabla_\theta J_\pi - \nabla_\theta \hat{J}_\pi\|_2$, for a small enough step the following inequality holds

$$D_{KL}(\pi_\theta \| \pi_{\hat{\theta}}) \leqslant \alpha^2 \tilde{c}(\epsilon_f c_1(H) + \epsilon_Q c_2(H))^2 =$$

Combining this bound with the Pinsker inequality

$$D_{TV}(\pi_\theta \| \pi_{\hat{\theta}}) \leqslant \alpha\sqrt{\frac{\tilde{c}}{2}}(\epsilon_f c_1(H) + \epsilon_Q c_2(H)) = \alpha c_3(\epsilon_f c_1(H) + \epsilon_Q c_2(H))$$

D.1.3 *Proof of Theorem 1*

Given the bound on the total variation distance, we can now make use of the monotonic improvement theorem to establish an improvement bound in terms of the gradient error. Let $J_\pi(\theta)$ and $J_\pi(\hat{\theta})$ be the expected return of the policy π_θ and $\pi_{\hat{\theta}}$ under the true dynamics. Let ρ and $\hat{\rho}$ be the discounted state marginal for the policy π_θ and $\pi_{\hat{\theta}}$, respectively

$$\begin{aligned}
|J_\pi(\theta) - J_\pi(\hat{\theta})| &= |\sum_{s,a} \rho(s)\pi_\theta r(s,a) - \hat{\rho}(s)\pi_{\hat{\theta}} r(s,a)| \\
&\leqslant |\sum_{s,a} \rho(s)\pi_\theta(a|s) r(s,a) - \hat{\rho}(s)\pi_{\hat{\theta}}(a|s) r(s,a)| \\
&\leqslant r_{\max}|\sum_{s,a} \rho(s)\pi_\theta(a|s) - \hat{\rho}(s)\pi_{\hat{\theta}}(a|s)| \\
&\leqslant \frac{2r_{\max}}{1-\gamma}\max_s \sum_a |\pi_\theta(a|s) - \pi_{\hat{\theta}}(a|s)| \\
&= \frac{2r_{\max}}{1-\gamma}\max_s D_{TV}(\pi_\theta \| \pi_{\hat{\theta}})
\end{aligned}$$

Then, combining the results from Lemma 2 we obtain the desired bound.

D.1.4 *Ablations*

In order to show the significance of each component of MAAC, we conducted more ablation studies. The results are shown in Figure 44. Here, we analyze the effect of training the Q-function with data coming from just the real environment, not learning a maximum entropy policy, and increasing the batch size instead of increasing the amount of samples to estimate the value function.

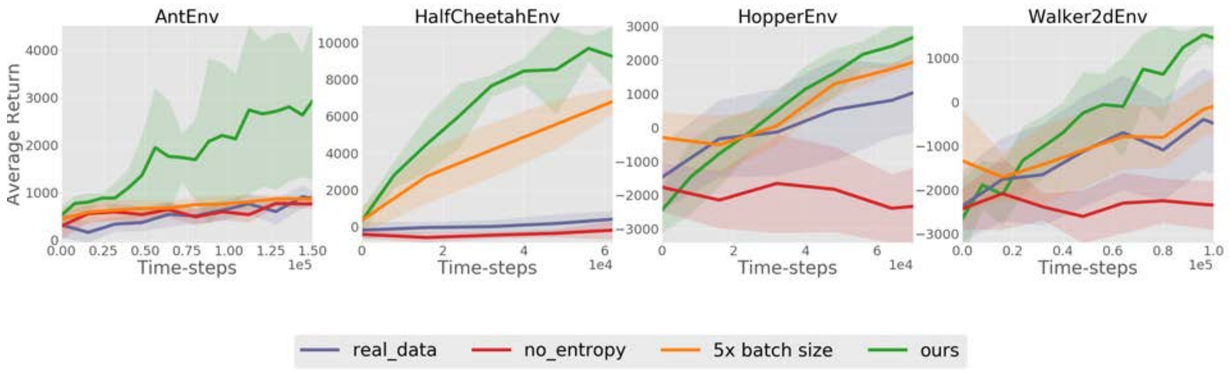


Figure 44: We further test the significance of some components of our method: not use the dynamics to generate data, and only use real data sampled from environments to train policy and Q-functions (real_data), remove entropy from optimization objects (no_entropy), and using a single sample estimate of the pathwise derivative but increase the batch size accordingly (5x batch size). Considering entropy and using dynamic models to augment data set are both very important.

D.1.5 *Execution Time Comparison*

	Iteration (s)	Training Model (s)	Optimization (s)	MBPO Iteration (s)
HalfCheetahEnv	1312	486	738	708
HopperEnv	845	209	517	723

Table 4: This table shows the time that different parts of MAAC need to train for one iteration after 6000 time steps, averaged across 4 seeds. We also add the time needed for MBPO for one iteration here for comparison.

E

APPENDIX: LEARNING TO ADAPT IN DYNAMIC, REAL-WORLD ENVIRONMENTS THROUGH META-REINFORCEMENT LEARNING

E.1 MODEL PREDICTION ERRORS: PRE-UPDATE VS. POST-UPDATE

In this section, we show the effect of adaptation in the case of GrBAL. In particular, we show the histogram of the K step normalized error, as well as the per-timestep visualization of this error during a trajectory. Across all tasks and environments, the post-updated model $\hat{p}_{\theta'_*}$ achieves lower prediction error than the pre-updted model $\hat{p}_{\theta_*}$.

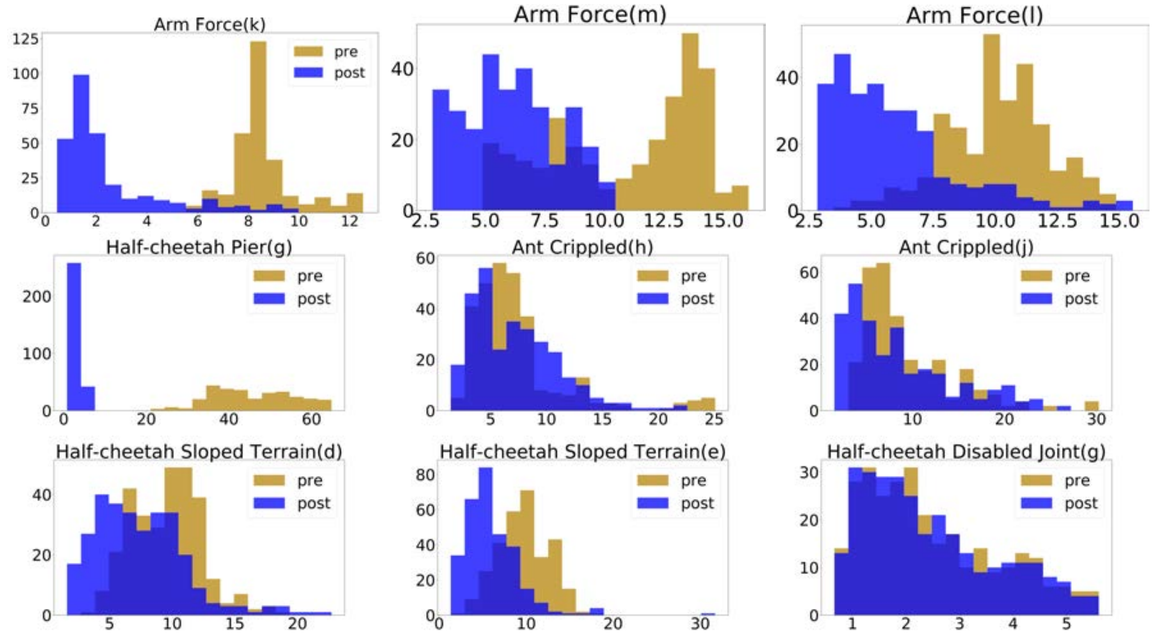


Figure 45: Histogram of the K step normalized error across different tasks. GrBAL accomplishes lower model error when using the parameters given by the update rule.

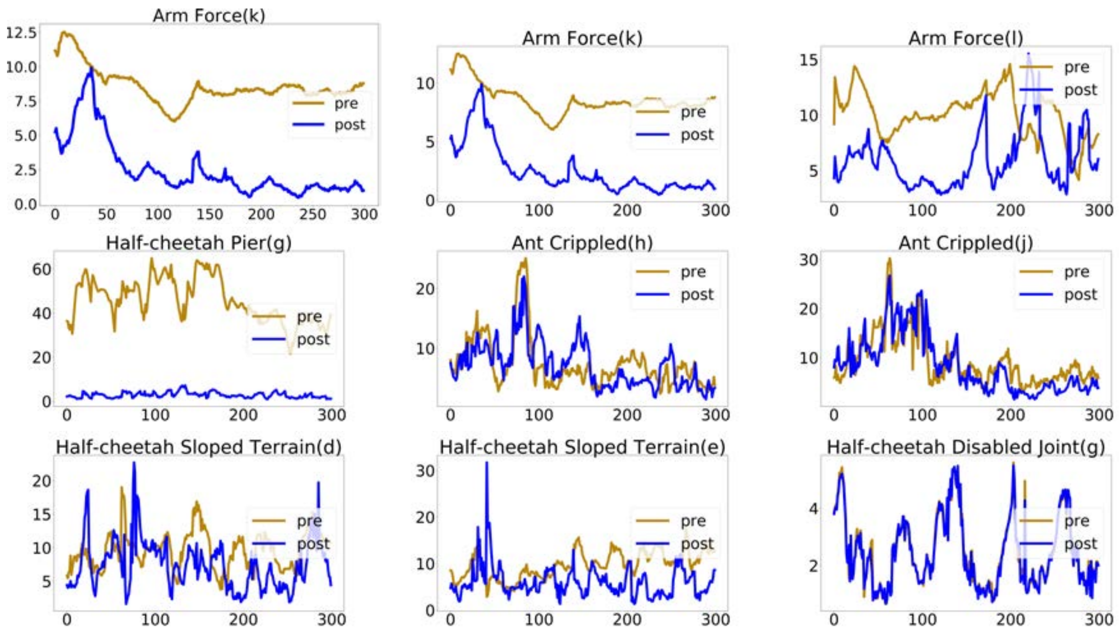


Figure 46: At each time-step we show the K step normalized error across different tasks. GrBAL accomplishes lower model error using the parameters given by the update rule.

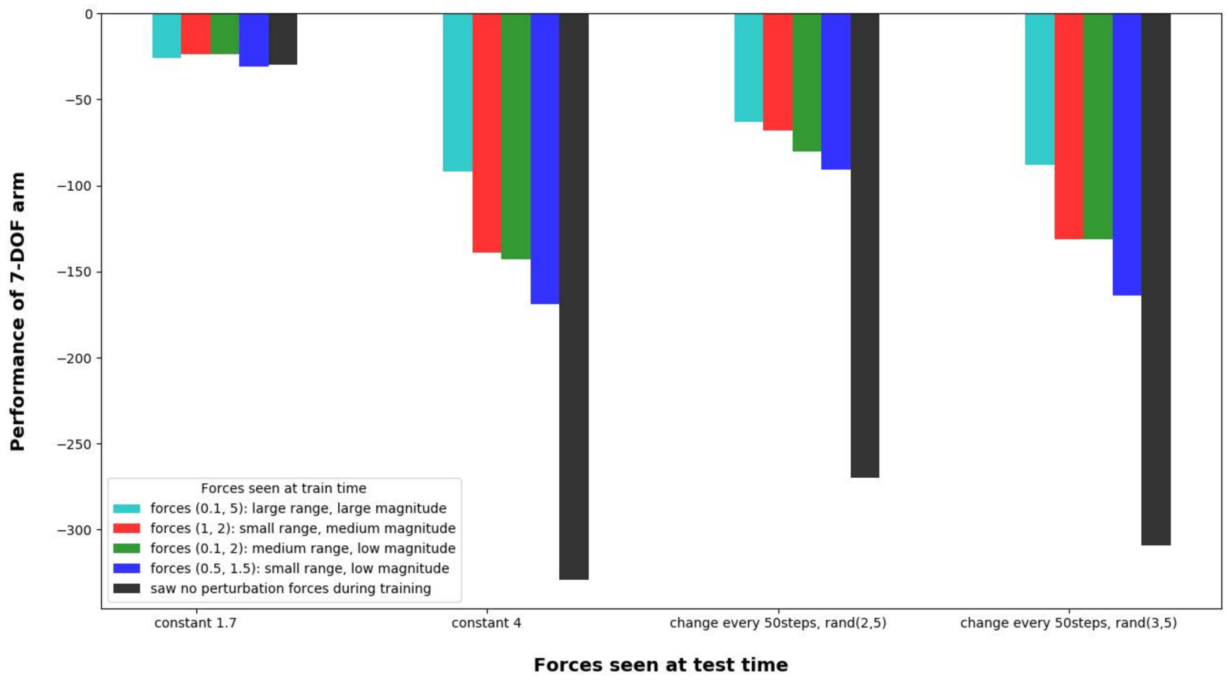


Figure 47: Effect of the meta-training distribution on test performance

E.2 EFFECT OF META-TRAINING DISTRIBUTION

To see how training distribution affects test performance, we ran an experiment that used GrBAL to train models of the 7-DOF arm, where each model was trained on the same number of datapoints during meta-training, but those datapoints came from different ranges of force perturbations. We observe (in the plot below) that

1. Seeing more during training is helpful during testing — a model that saw a large range of force perturbations during training performed the best

2. A model that saw no perturbation forces during training did the worst

3. The middle 3 models show comparable performance in the "constant force = 4" case, which is an out-of-distribution task for those models. Thus, there is not actually a strong restriction on what needs to be seen during training in order for adaptation to occur at train time (though there is a general trend that more is better)

E.3 Sensitivity of T_1 and T_2

In this section we analyze how sensitive is our algorithm w.r.t the hyperparameters T_1 and T_2. In all experiments of the paper, we set T_1 equal to T_2. Figure 48 shows the average return of GrBAL across meta-training iterations of our algorithm for different values of $T_1 = T_2$. The performance of the agent is largely unaffected for different values of these hyperparameters, suggesting that our algorithm is not particularly sensitive to these values. For different agents, the optimal value for these hyperparameters depends on various task details, such as the amount of information present in the state (a fully-informed state variable precludes the need for additional past timesteps) and the duration of a single timestep (a longer timestep duration makes it harder to predict more steps into the future).

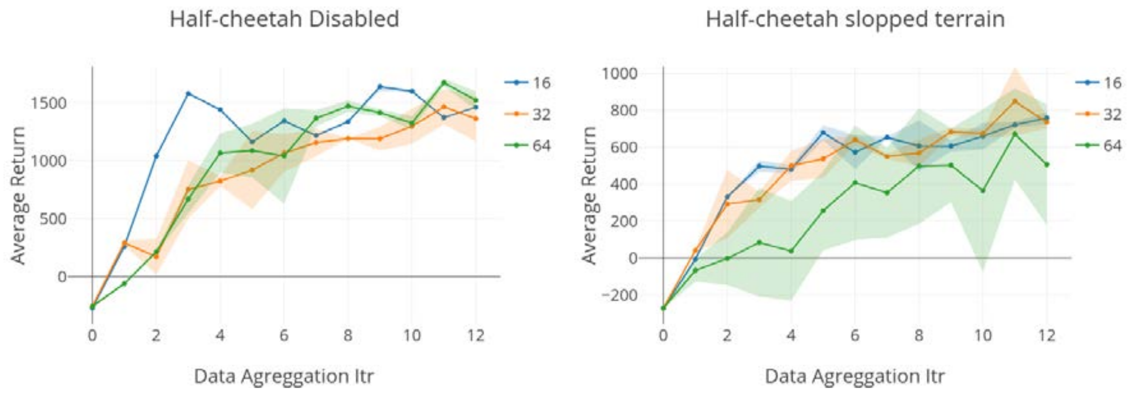


Figure 48: Learning curves, for different values of $T_1 = T_2$, of GrBAL in the half-cheetah disabled and sloped terrain environments. The x-axis shows data aggreation iterations during meta-training, whereas the y-axis shows the average return achieved when running online adaptation with the meta-learned model from the particular iteration. The curves suggest that GrBAL performance is fairly robust to the values of these hyperparameters.

E.4 Reward functions

For each MuJoCo agent, the same reward function is used across its various tasks. Table 5 shows the reward functions used for each agent. We denote by x_t the x-coordinate of the agent at time t, $\boldsymbol{ee}_t$ refers to the position of the end-effector of the 7-DoF arm, and $\mathbf{g}$ corresponds to the position of the desired goal.

Table 5: Reward functions

	Reward function
Half-cheetah	$\frac{x_{t+1}-x_t}{0.01} - 0.05\|\mathbf{a}_t\|_2^2$
Ant	$\frac{x_{t+1}-x_t}{0.0e} - 0.005\|\mathbf{a}_t\|_2^2 + 0.05$
7-DoF Arm	$-\|\boldsymbol{ee}_t - \mathbf{g}\|_2^2$

E.5 HYPERPARAMETERS

Below, we list the hyperparameters of our experiments. In all experiments we used a single gradient step for the update rule of GrBAL. The learning rate (LR) of TRPO corresponds to the Kullback–Leibler divergence constraint. # Task/itr corresponds to the number of tasks sampled for collecting data to train the model or model, whereas # TS/itr is the total number of times steps collected (for all tasks). Finally, H refers to the horizon of the task.

Table 6: Hyperparameters for the half-cheetah tasks

	LR	Inner LR	Epochs	T_2	T_1	Batch Size	# Tasks/itr	# TS/itr	H	n_A Train	H Train	n_A Test	H Test
GrBAL	0.001	0.01	50	32	32	500	32	64000	1000	1000	10	2500	15
ReBAL	0.001	-	50	32	32	500	32	64000	1000	1000	10	2500	15
MB	0.001	-	50	-	-	500	64	64000	1000	1000	10	2500	15
TRPO	0.05	-	-	-	-	50000	50	50000	1000	-	-	-	-

Table 7: Hyperparameters for the ant tasks

	LR	Inner LR	Epochs	T_2	T_1	Batch Size	# Tasks/itr	# TS/itr	H	n_A Train	H Train	n_A Test	H Test
GrBAL	0.001	0.001	50	10	16	500	32	24000	500	1000	15	1000	15
ReBAL	0.001	-	50	32	16	500	32	32000	500	1000	15	1000	15
MB	0.001	-	70	-	-	500	10	10000	500	1000	15	1000	15
TRPO	0.05	-	-	-	-	50000	50	50000	500	-	-	-	-

Table 8: Hyperparameters for the 7-DoF arm tasks

	LR	Inner LR	Epochs	K	M	Batch Size	# Tasks/itr	# TS/itr	T	n_a Train	H Train	n_a Test	H Test
GrBAL	0.001	0.001	50	32	16	1500	32	24000	500	1000	15	1000	15
ReBAL	0.001	-	50	32	16	1500	32	24000	500	1000	15	1000	15
MB	0.001	-	70	-	-	10000	10	10000	500	1000	15	1000	15
TRPO	0.05	-	-	-	-	50000	50	50000	500	-	-	-	-