Model-based Deep Reinforcement Learning for Robotic Systems

Anusha Nagabandi

August 13, 2020

Model-based Deep Reinforcement Learning for Robotic Systems

by

Anusha Nagabandi

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Summer 2020

Abstract

Model-based Deep Reinforcement Learning for Robotic Systems

by

Anusha Nagabandi

Doctor of Philosophy in Electrical Engineering and Computer Sciences

University of California, Berkeley

Ronald S. Fearing, Co-chair

Sergey Levine, Co-chair

Deep learning has shown promising results in robotics, but we are still far from having intelligent systems that can operate in the unstructured settings of the real world, where disturbances, variations, and unobserved factors lead to a dynamic environment. The premise of the work in this thesis is that model-based deep RL provides an efficient and effective framework for making sense of the world, thus allowing for reasoning and adaptation capabilities that are necessary for successful operation in the dynamic settings of the world.

We first build up a model-based deep RL framework and demonstrate that it can indeed allow for efficient skill acquisition, as well as the ability to repurpose models to solve a variety of tasks. We then scale up these approaches to enable locomotion with a 6-DoF legged robot on varying terrains in the real world, as well as dexterous manipulation with a 24-DoF anthropomorphic hand in the real world. Next, we focus on the inevitable mismatch between an agent's training conditions and the test conditions in which it may actually be deployed, thus illuminating the need for adaptive systems. Inspired by the ability of humans and animals to adapt quickly in the face of unexpected changes, we present a meta-learning algorithm within this model-based RL framework to enable online adaptation of large, high-capacity models using only small amounts of data from the new task. We demonstrate these fast adaptation capabilities in both simulation and the real-world, with experiments such as a 6-legged robot adapting online to an unexpected payload or suddenly losing a leg. We then further extend the capabilities of our robotic systems by enabling the agents to reason directly from raw image observations. Bridging the benefits of representation learning techniques with the adaptation capabilities of meta-RL, we present a unified framework for effective meta-RL from images. With robotic arms in the real world that learn peg insertion and ethernet cable insertion to varying targets, we show the fast acquisition of new skills, directly from raw image observations in the real world. Finally, we conclude by discussing the key limitations of our existing approaches and present promising directions for future work in the area of model-based deep RL for robotic systems.

Acknowledgments

I would like to thank my advisors, Ronald Fearing and Sergey Levine, for their invaluable mentorship these past five years. I would like to thank Claire Tomlin for being a part of my qualification exam committee, and Hannah Stuart for being a part of my qualification exam committee and dissertation committee. Additionally, I would like to thank my wonderful collaborators, without whom the work in this thesis would not have been possible: Greg Kahn, Ignasi Clavera, Kate Rakelly, Guangzhao (Philip) Yang, Thomas Asmar, Simin Liu, Zihao (Tony) Zhao, Vikash Kumar, Chelsea Finn, and Pieter Abbeel. I would also like to thank everyone in my labs (RAIL and BML) and BAIR for always being willing to help each other, provide insightful feedback and discussions about research, create a collaborative atmosphere, talk about life, or even just eat meals together and share memes. In particular, I'd like to thank Abhishek Gupta, Alex Lee, Andrea Bajcsy, Ashvin Nair, Austin Buchan, Carlos Casarez, Carolyn Chen, Coline Devin, Conrad Holda, Duncan Haldane, Eric Mazumdar, Esther Rolf, Ethan Schaler, Greg Kahn, Jessica Lee, Justin Yim, Kate Rakelly, Liyu Wang, Marvin Zhang, Michael Janner, Nick Rhinehart, Sareum Kim, Somil Bansal, Vitchyr Pong, and many others! You have all taught me so much, and you have inspired me daily with your kindness, hard work, dedication, enthusiasm, and thoughtfulness. I truly feel so privileged to have gotten to know you all.

I would like to thank Marcel Bergerman and Maria Yamanaka for sparking my interest in robotics in high school, Aaron Steinfeld for supporting me during my first ever research experience after my freshman year of college, and all of my wonderful colleagues and mentors throughout internships at Carnegie Robotics, Texas Instruments, MIT Lincoln Lab, and Google Brain.

I would also like to thank all of the wonderful friends that I've made in Berkeley, without whom these past 5 years seem unimaginable. Whether we met as roommates, lab mates, ultimate frisbee friends, dance team friends, or just friends, you guys have very quickly become like family. In addition to those mentioned above, I'd like to give a special thank you to Keertana Settaluri, Matt McPhail, and Spencer Kent for making Berkeley feel like home. I would also like to thank my high school friends and my college friends for continuing to be a big part of my life. I am so thankful for all of the wonderful experiences that we have shared together, and there are not enough words that I can use to tell you all how much you mean to me. I would like to give a special thanks to Akshay Ravi for always supporting me, for delicious baked goods, for unlimited laughs and good times, and for making me a better person every day. Finally, a huge thank you to my family, especially to my parents and my sister, without whom I would not be the person that I am today. Thank you for your never-ending love and support.

Contents

Chapter 1

Introduction

From carefully planned-out acrobatics and dynamic walking maneuvers (Kuindersma et al. 2016; Tajima, Honda, and Suga 2009), to precise perception-based manipulation for folding towels (Miller et al. 2012) and reorienting cubes (Akkaya et al. 2019), robots these days can demonstrate very impressive capabilities. Although both traditional control approaches as well as deep learning approaches have demonstrated promising robotics results, we are unfortunately still far from having intelligent robotic systems operating in the real world.

1.1 Motivation

To understand why we don't yet have robotic systems deployed in real-world settings such as homes, we must come to terms with the unstructured nature of the real world, where disturbances, variations, and unobserved factors lead to an extremely dynamic environment. Moving away from controlled and instrumented settings such as labs, what we really want is a robot that can rely on its own on-board sensors (such as cameras) to perceive an unstructured world without the strong assumption of perfect state estimation, and that can very quickly adapt to a variety of new tasks as the situation demands, as opposed to being overly specialized for a single task. For example, a robot that is only capable of moving a certain object from one fixed position to another may be useful in a fixed factory setting where no amount of variation ever occurs (Fig. 1.1), but it will rarely be useful anywhere else.

Figure 1.1: Robots in a Hyundai car factory (Narasimhan 2018), increasing factory efficiency through repeated execution of pre-programmed commands.

What we want instead is a robot that can be trained to sort boxes in a warehouse, for instance, but can then adapt its behavior to heavy or slippery packages, to the presence of an unexpected human arm during its operation, to the fact that the current stack of boxes has fallen, or to the fact that the delivery truck has parked farther away than usual.

Centuries of research efforts have brought us to the point of being able to design complex systems in controlled settings by reasoning through the specific mechanisms at hand; and while we are surprisingly effective at enabling super-human types of capabilities such as beating human Chess champions (Campbell, Hoane Jr, and F.-h. Hsu 2002) and winning Jeopardy (Ferrucci 2012), we unfortunately cannot yet enable the robust and generalizable capabilities of even a young child around the house. Although efforts such as RoboCup (Kitano et al. 1997) and the Darpa Robotics Challenge (Krotkov et al. 2017) have taken steps toward encouraging robots to operate in more realistic scenarios, there is still much progress to be made. The real world is full of challenges for robots, including irregular terrains, the lack of perfect state information for all objects in the world, the resulting difficultly of recognizing or correcting for errors, and even small 1cm errors being enough to spill a coffee mug or drop a bowl. Humans, on the other hand, can feel around their pocket to find a key and then proceed to unlock a door entirely in the dark, without any sense of exactly where the keyhole is. We can pour coffee into a mug regardless of its precise location, using our visual perception. We can continue to move by hobbling on one foot after rolling an ankle. We can walk on stairs or slopes or ice. The development of online reasoning is critical for these types of capabilities, and will directly result in more generalizable and adaptable robots that can be deployed into the world. The premise of the work in this thesis is that model-based deep RL can provide an efficient and effective framework for making sense of the world; and in turn, this ability to make sense of the world can allow for these types of reasoning and adaptation capabilities that are necessary for successful operation in the dynamic settings of the real world.

1.2 Online Reasoning and Decision Making for Generalizable Systems

One way to understand why “hard” tasks such as beating world champions at chess are easy, but “easy tasks” such as cleaning rooms are hard, is to think about the concept of specialists versus generalists. For instance, although the chess robot does not need to know anything other than chess, think about the amount of variations that we need to be able to address when cleaning a room. First, it’s impossible for us to measure everything precisely in the real world. Even if we give our robots vision and allow them to operate directly from image inputs, what does a chair even look like? Although many types of chairs (wooden, red, velvet, stools, etc.) are all clearly identifiable to humans as chairs, they may not actually look alike to a robot that is trying to identify a chair. Even if we assume that computer vision approaches can address this identification problem of knowing all the various forms of what a chair may look like, we still need to address the next problem of actually working with it. Consider the task of putting away a chair. Where do I hold onto it? How do I pull it? The answer is different for every specific instance. For instance, do I hold it at the top and pull on it? Do I hold it in the middle and push on it? How will this type of chair slide on this type of floor? Do I need to understand physics here? Will the chair scratch the floor? Should I try to lift the chair up before moving it? What if the chair has things on top? Do I need to put away all the clothes first? How do I pick up and fold flexible materials like a shirt? As you can see, the difficulty of this problem escalated *very* quickly, and yet we have still only considered one small task involved with cleaning a room, which is moving a chair. Now, consider a robot that learns how to move a certain type of chair from exactly 1 fixed position to some other goal position, in a lab. Unfortunately, that skill is entirely useless in the real world if it cannot generalize to moving chairs from arbitrary positions, or moving a slightly different type of chair, or moving the chair even when the floor is slightly different. In order words, being a specialist is not nearly as important as being a generalist when we want something that works in the real world. Furthermore, it has become clear over the years that it is not sufficient to set rules or specific control laws to dictate what each actuator of a robot should do; rather, this ability to generalize must arise from enabling robots to reason on their own and act accordingly.

Much work has tried to enable generalization to new scenarios through approaches such as training on large amounts of data, domain randomization (Tobin et al. 2017), and domain adaptation (Bousmalis et al. 2018). Other work has instead emphasized the idea of learning predictive models (Pathak 2019) as being the key mechanism for enabling adaptation to scenarios past those seen during training. For example, general concepts such as object permanence (Baillargeon, Spelke, and Wasserman 1985) and approximate physics (Hespos and VanMarle 2012) are learned by kids at a very young age, and they display explicit shock or confusion when faced with seemingly impossible situations that violate their internal predictive models. Prior work in robotics has extended this idea, that "play" teaches infants to use experimentation to learn a model of how things work (Agrawal 2018), which can then be used to perform new tasks. This concept of creating generalist (as opposed to specialist) agents by taking the time to develop good predictive models is further supported by the link between long childhoods and the resulting intelligence of a species (Piantadosi and Kidd 2016). The work in this thesis is inspired by this idea of predictive models as a mechanism for learning to generalize – that perhaps the most general level at which to learn is the predictive level, and that these predictive models can be re-purposed downstream in the process of learning other skills or re-planning in the face of uncertainty.

This observation that the format of predictive models allows us to re-use them for future decision making is powerful, and further supported by evidence in both humans and animals. Humans often perform "what if" types of reasoning by querying their internal predictive models before making decisions; when choosing a future job, for example, we often imagine the consequences and possible futures that may result from each choice, before assessing those possible futures and making our decision. Relatedly, the firing patterns of place cells have been shown (Johnson and Redish 2007) in rats to demonstrate "what if" types of reasoning when the rat comes to critical decision points in a maze. When examined at fine time scales, these place cells, which normally encode concepts such as physical location, were shown to demonstrate a looking-ahead type of firing pattern which scanned both of the possible turn directions before making a decision at the T-junction of a maze. Having explicit predictive mechanisms, then, seems to help both animals and humans to make sense of the world and allow for explicit querying and reasoning in unfamiliar situations.

1.3 Overview, Organization, and Contributions

As alluded to above, a major challenge for deploying robots to real-world environments is that they must be able to perform a wide variety of tasks and acquire new skills as the situation demands, as opposed to repeatedly executing a single and specific fixed task. In other words, we need to extend the current single-task specializations of deep RL into more generalizable and adaptable ones that can succeed in dynamic and unstructured real-world environments.

Consider these definitions of generalization and human-level intelligence:

> "A generalization is the formulation of general concepts from specific instances by **abstracting common properties**" (Merriam-Webster dictionary)
>
> "Generalization is the concept that humans and animals **use past learning in present situations** of learning if the conditions in the situations are regarded as similar." (Gluck, Mercado, and Myers 2007)
>
> "The most difficult problems for computers are informally known as AI-complete. solving them is equivalent to the general aptitude of human intelligence... beyond the capabilities of a purpose-specific algorithm. AI-complete problems are hypothesized to include... **dealing with unexpected circumstances** while solving any real world problem" (Shapiro 1992)

With these in mind, we highlight three main concepts that will reappear in the contents of this thesis: abstracting common properties, using past learning in present situations, and going beyond purpose-specific algorithms to deal with unexpected circumstances. The first concept of abstracting common properties is addressed in this thesis with the building of predictive models; we learn models that can explain what they observe. The second concept is addressed with meta-learning algorithms, which learn to use prior knowledge to quickly solve new tasks. And the third concept of going "beyond purpose-specific" is addressed with work in online adaptation, continual learning, and online inference from raw sensory observations in an effort to succeed even in the face of unexpected circumstances.

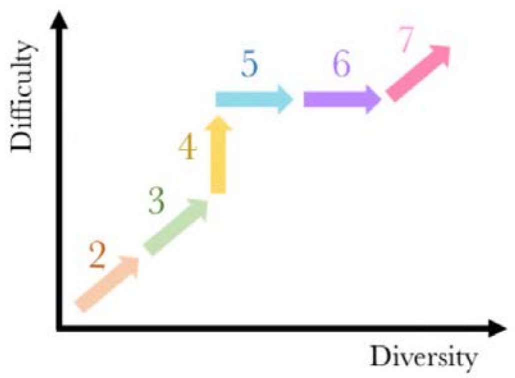


Figure 1.2: Overview figure to visualize the flow and general contributions of each chapter of this thesis, for the overall goal of enabling the deployment of robotic systems into the real world. The **difficulty** axis here encompasses difficulty of the tasks (e.g., contacts), difficulty from state dimensionality (e.g., working with images), difficulty from action dimensionality (e.g., working with high-DoF robots), as well as difficulty induced from operating in the real world. The **diversity** axis, on the other hand, focuses on the generalization aspect of getting robots into the real world. This axis encompasses the re-usability of learned knowledge, the variety of achievable tasks, the capability of adapting online to address unexpected perturbations at run time, and the ability to adjust behaviors as necessary to succeed in new and unseen situations.

We first build up a model-based deep RL framework in chapter 2 and demonstrate that it can indeed allow for efficient and effective autonomous skill acquisition, as well as the ability to repurpose models to execute a variety of paths at test time. Chapter 3 extends this framework to enable locomotion with a 6-DoF legged robot in the real world by learning image-conditioned models that are able to address various types of terrains. Chapter 4 scales up these approaches to address challenging dexterous manipulation tasks with a 24-DoF anthropomorphic hand, both in simulation and in the real world.

At this point, we turn our focus to the inevitable mismatch between an agent's training conditions and the test conditions in which it may actually be deployed. Inspired by the ability of humans and animals to adapt quickly in the face of unexpected changes, chapter 5 presents a meta-learning algorithm within this model-based RL framework to enable online adaptation of large, high-capacity models using only small amounts of data from the new task. These fast adaptation capabilities are shown both in simulation and the real-world, with experiments such as a 6-legged robot adapting online to an unexpected payload or suddenly losing a leg. **Chapter 6** formulates an online learning approach as an expectation maximization problem to construct and build a task distribution with a mixture of models; this work allows generalization to tasks that are further away from the training distribution while also allowing for task recall, generalization, and specialization.

Next, we further extend the capabilities of our robotic systems by enabling the agents to reason directly from raw image observations. Taking the benefits of unsupervised learning techniques for extracting meaningful representations from high-dimensional and partial observations, and bridging them with the adaptation capabilities of meta-RL, chapter

7 presents a unified framework for effective meta-RL from images. Robotic arms in the real world that learn peg insertion and ethernet cable insertion to varying targets show the fast acquisition of new skills, directly from raw image observations in the real world. Finally, **chapter 8** concludes by discussing the key limitations of our existing approaches and presenting promising directions for future work in this area of model-based deep RL for robotic systems.

Chapter 2

MBRL: Learning Neural Network Models for Control

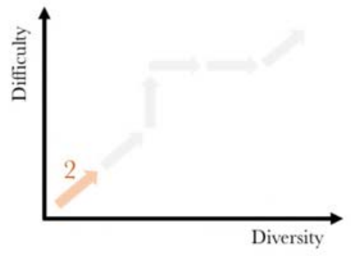


Figure 2.1: Thesis outline, with the current chapter indicated by the last colored arrow.

Model-free deep RL algorithms have been shown to be capable of learning a wide range of robotic skills, but typically require a very large number of samples to achieve good performance. Model-based algorithms, in principle, can provide for much more efficient learning, but have proven difficult to extend to expressive, high-capacity models such as deep neural networks. In this chapter, we demonstrate that neural network dynamics models can in fact be learned and combined with model predictive control (MPC) to achieve excellent sample complexity in a model-based reinforcement learning algorithm. After building up this model-based RL framework and demonstrating its ability to quickly learn stable and plausible gaits that accomplish various complex locomotion tasks, we then propose using those deep neural network dynamics models to initialize a model-free learner as a way to bridge the sample efficiency of model-based approaches with the high task-specific performance of model-free methods. Videos of the experiments as well as the code are available online[1].

2.1 Introduction

Model-free deep reinforcement learning algorithms have been shown to be capable of learning a wide range of tasks, ranging from playing video games from images (V. Mnih, K. Kavukcuoglu, et al. 2013; Oh et al. 2016) to learning complex locomotion skills (J. Schulman et al. 2015). However, such methods suffer from very high sample complexity, often requiring millions of samples to achieve good performance (J. Schulman et al. 2015). Model-based reinforcement learning algorithms are generally regarded as being more efficient (M. P. Deisenroth, Neumann, Peters, et al. 2013). However, to achieve good sample efficiency, these model-based algorithms

[1]`https://sites.google.com/view/mbmf`

have conventionally used either simple function approximators (Lioutikov et al. 2014) or Bayesian models that resist overfitting (M. Deisenroth and C. Rasmussen 2011) in order to effectively learn the dynamics using few samples. This makes them difficult to apply to a wide range of complex, high-dimensional tasks. Although a number of prior works have attempted to mitigate these shortcomings by using large, expressive neural networks to model the complex dynamical systems typically used in deep reinforcement learning benchmarks (Brockman et al. 2016; Emanuel Todorov, Erez, and Tassa 2012a), such models often do not perform well (S. Gu et al. 2016) and have been limited to relatively simple, low-dimensional tasks (Mishra, Abbeel, and Mordatch 2017).

In this work, we demonstrate that multi-layer neural network models can in fact achieve excellent sample complexity in a model-based reinforcement learning algorithm. The resulting models can then be used for model-based control, which we perform using model predictive control (MPC) with a simple random-sampling shooting method (Richards 2004). We demonstrate that this method can acquire effective locomotion gaits for a variety of MuJoCo benchmark systems (Emanuel Todorov, Erez, and Tassa 2012a), including the swimmer, half-cheetah, hopper, and ant. Fig. 7.3 shows that these models can be used at run-time to execute a variety of locomotion tasks such as trajectory following, where the agent executes a path through a given set of sparse waypoints that represent desired center-of-mass positions. Additionally, each systems uses less than four hours worth of data, indicating that the sample complexity of our model-based approach is low enough to be applied in the real world, and is dramatically lower than pure model-free learners. In particular, when comparing this model-based approach's ability to follow arbitrary desired trajectories with a model-free approach's ability to learn just a competent moving forward gait, our results show that the model-based method uses only 3%, 10%, and 14% of the data that is used by a model-free approach (for half-cheetah, swimmer, and ant, respectively). Relatedly, our model-based method can achieve qualitatively good forward gaits for the swimmer, cheetah, hopper, and ant using $20-80\times$ fewer data points than is required by a model-free approach.

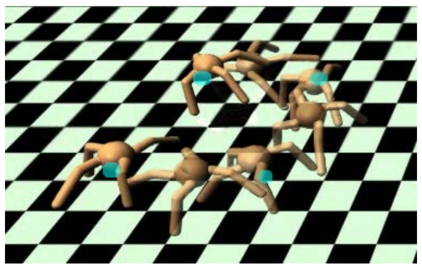


Figure 2.2: Our method learns a dynamics model that enables a simulated quadrupedal robot to autonomously follow user-defined waypoints. Training for this task uses 7×10^5 time steps (collected without knowledge of test-time navigation tasks), and the learned model can be reused at test time to follow arbitrary desired trajectories, such as the U-turn shown above.

Although such model-based methods are drastically more sample efficient and more flexible than task-specific policies learned with model-free reinforcement learning, their asymptotic performance is usually worse than model-free learners due to model bias. Model-free algorithms are not limited by the accuracy of the model, and therefore can achieve better final performance, though at the expense of much higher sample complexity (M. P. Deisenroth, Neumann, Peters, et al. 2013; Kober, J. A. Bagnell, and Peters 2013). To address this issue, we use our model-based algorithm, which can quickly achieve moderately proficient behavior, to initialize a model-free learner, which can slowly achieve near-optimal behavior. The learned model-based controller provides good rollouts, which enable supervised initialization of a policy that can then be fine-tuned with model-free algorithms, such as policy gradients. We empirically demonstrate that the resulting hybrid model-based and model-free (Mb-Mf) algorithm can accelerate model-free learning, achieving sample efficiency gains of $3-5\times$ on the swimmer, half-cheetah, hopper, and ant.

The highlights of this chapter are as follows:

1. We demonstrate effective model-based RL with neural network models for several contact-rich simulated locomotion tasks from standard deep reinforcement learning benchmarks.

2. We empirically evaluate a number of design decisions for neural network dynamics model learning.

3. We show how a model-based learner can be used to initialize a model-free learner to achieve high rewards while significantly reducing sample complexity.

2.2 Related Work

Deep reinforcement learning algorithms based on Q-learning (Volodymyr Mnih, Koray Kavukcuoglu, Silver, Rusu, et al. 2015; Oh et al. 2016; S. Gu et al. 2016), actor-critic methods (T. Lillicrap et al. 2016; V. Mnih, Badia, et al. 2016; John Schulman, Moritz, et al. 2016), and policy gradients (J. Schulman et al. 2015; Shixiang Gu, Timothy Lillicrap, Ghahramani, et al. 2017) have been shown to learn very complex skills in high-dimensional state spaces, including simulated robotic locomotion, driving, video game playing, and navigation. However, the high sample complexity of purely model-free algorithms has made them difficult to use for learning in the real world, where sample collection is limited by the constraints of real-time operation. Model-based algorithms are known in general to outperform model-free learners in terms of sample complexity (M. P. Deisenroth, Neumann, Peters, et al. 2013), and in practice have been applied successfully to control both simulated and real-world robotic systems, such as pendulums (M. Deisenroth and C. Rasmussen 2011), legged robots (Morimoto and Atkeson 2003), swimmers (Meger et al. 2015), and manipulators (M. P. Deisenroth, Carl Edward Rasmussen, and Fox 2011). However, the most efficient model-based algorithms have used relatively simple function approximators, such as Gaussian processes (M. Deisenroth and C. Rasmussen 2011; Boedecker et al. 2014; Ko and Fox 2008), time-varying linear models (Lioutikov et al. 2014; Levine and Abbeel 2014; Yip and Camarillo 2014), and mixtures of Gaussians (Khansari-Zadeh and Billard 2011). PILCO (M. Deisenroth and C. Rasmussen 2011), in particular, is a model-based policy search method which reports excellent sample efficiency by learning probabilistic dynamics models and incorporating model uncertainty into long-term planning. These methods have difficulties, however, in high-dimensional spaces and with nonlinear dynamics. The most high-dimensional task demonstrated with PILCO that we could find has 11 dimensions (Meger et al. 2015), while the most complex task in our work has 49 dimensions and features challenging properties such as frictional contacts.

Although neural networks have been widely used in earlier work to model plant dynamics (K. J. Hunt et al. 1992; Bekey and Goldberg 1992), more recent model-based algorithms have achieved only limited success in applying such models to the more complex benchmark tasks that are commonly used in deep reinforcement learning. Several works have proposed to use deep neural network models for building predictive models of images (Watter et al. 2015), but these methods have either required extremely large datasets for training (Watter et al. 2015) or were applied to short-horizon control tasks (Wahlström, Schön, and M. P. Deisenroth 2015). In contrast, we consider long-horizon simulated locomotion tasks, where the high-dimensional systems and contact-rich environment dynamics provide a considerable modeling challenge. (Mishra, Abbeel, and Mordatch 2017) proposed a relatively complex time-convolutional model for dynamics prediction, but only demonstrated results on low-dimensional (2D) manipulation tasks. (Gal, McAllister, and Carl Edward Rasmussen 2016) extended PILCO (M. Deisenroth and C. Rasmussen 2011) using Bayesian neural networks, but only presented results on a low-dimensional cart-pole swingup task, which does not include contacts.

Aside from training neural network dynamics models for model-based reinforcement learning, we also explore how such models can be used to accelerate a model-free learner. Prior work on model-based acceleration has explored a variety of avenues. The classic Dyna (R. Sutton 1991) algorithm proposed to use a model to generate simulated experience that could be included in a model-free algorithm. This method was extended (Silver, R. S. Sutton, and Müller 2008; Asadi 2015) to work with deep neural network policies, but performed best with models that were not neural networks (S. Gu et al. 2016). Model learning has also been used to accelerate model-free Bellman backups (Heess, Wayne, et al. 2015), but the gains in performance from including the model were relatively modest. Prior work has also used model-based learners to guide policy optimization through supervised learning (Levine, Finn, et al. 2017), but the models that were used were typically local linear models. In a similar way, we also use supervised learning to initialize the policy, but we then fine-tune this policy with model-free learning to achieve the highest returns. Our model-based method is more expressive and flexible than local linear models, and it does not require multiple samples from the same initial state for local linearization.

2.3 The Neural Network Dynamics Model

In the following few sections, we present our model-based deep reinforcement learning framework, which we further build upon in future chapters. We first detail the learned dynamics function $f_\theta(\mathbf{s}_t, \mathbf{a}_t)$ and how to train it, and we then detail how to extract a policy using this learned dynamics function.

We parameterize the learned dynamics function $f_\theta(\mathbf{s}_t, \mathbf{a}_t)$ as a deep neural network, where the parameter vector θ represents the weights of the network. A straightforward parameterization for $f_\theta(\mathbf{s}_t, \mathbf{a}_t)$ would take as input the current state $\mathbf{s}_t$ and action $\mathbf{a}_t$, and output the predicted next state $\hat{\mathbf{s}}_{t+1}$. However, this function can be difficult to learn when the states $\mathbf{s}_t$ and $\mathbf{s}_{t+1}$ are too similar and the action has seemingly little effect on the output; this difficulty becomes more pronounced as the time between states Δt becomes smaller.

We overcome this issue by instead learning a dynamics function that predicts the change in state $\mathbf{s}_t$ over the time step duration of Δt. Thus, the predicted next state is as follows: $\hat{\mathbf{s}}_{t+1} = \mathbf{s}_t + f_\theta(\mathbf{s}_t, \mathbf{a}_t)$. Note that increasing this Δt increases the information available from each data point, and can help with not only dynamics learning but also with planning using the learned dynamics model (Sec. 2.4). However, increasing Δt also increases the discretization and variation of the underlying continuous-time dynamics, which can make the learning process more difficult.

2.3.1 Collecting training data

We collect training data by sampling starting configurations $\mathbf{s}_0 \sim p(\mathbf{s}_0)$, executing random actions at each timestep, and recording the resulting trajectories $\tau = (\mathbf{s}_0, \mathbf{a}_0, \cdots, \mathbf{a}_{T-2}, \mathbf{s}_{T-1})$ of length T. We note that these trajectories are very different from the trajectories the agents will end up executing when planning with this learned dynamics model and a given reward function $r(\mathbf{s}_t, \mathbf{a}_t)$ (Sec. 2.4), showing that model-based methods learn from off-policy data.

2.3.2 Data preprocessing

We slice the trajectories $\{\tau\}$ into training data inputs $(\mathbf{s}_t, \mathbf{a}_t)$ and corresponding output labels $\mathbf{s}_{t+1} - \mathbf{s}_t$. We then subtract the mean of the data and divide by the standard deviation of the data to ensure the loss function weights the different parts of the state (e.g., positions and velocities) equally, regardless of their original scale. We also add zero mean Gaussian noise to the training data (inputs and outputs) to increase model robustness. The training data is then stored in the dataset $\mathcal{D}$.

2.3.3 Training the model

We train the dynamics model $f_\theta(\mathbf{s}_t, \mathbf{a}_t)$ by minimizing the error

$$\mathcal{E}(\theta) = \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \in \mathcal{D}} \frac{1}{2} \|(\mathbf{s}_{t+1} - \mathbf{s}_t) - f_\theta(\mathbf{s}_t, \mathbf{a}_t)\|^2 \tag{2.1}$$

using stochastic gradient descent. While training on the training dataset $\mathcal{D}$, we also calculate the mean squared error in Eqn. 2.1 on a validation set $\mathcal{D}_{val}$, composed of trajectories not stored in the training dataset.

Although this error provides an estimate of how well the learned dynamics function predicts next state, we would in fact like to know how well the model can predict further into the future because we will ultimately use this model for longer-horizon control (Sec. 2.4). We therefore calculate H-step validation errors by propagating the learned dynamics function forward H times to make multi-step open-loop predictions. For each given sequence of true actions $(\mathbf{a}_t, \dots \mathbf{a}_{t+H-1})$ from $\mathcal{D}_{val}$, we compare the corresponding ground-truth states $(\mathbf{s}_{t+1}, \dots \mathbf{s}_{t+H})$ to the dynamics model's multi-step state predictions $(\hat{\mathbf{s}}_{t+1}, \dots \hat{\mathbf{s}}_{t+H})$, calculated as

$$\begin{aligned} \mathcal{E}_{val}^{(H)} &= \frac{1}{\mathcal{D}_{val}} \sum_{\mathcal{D}_{val}} \frac{1}{H} \sum_{h=1}^{H} \frac{1}{2} \|\mathbf{s}_{t+h} - \hat{\mathbf{s}}_{t+h}\|^2 \quad : \\ \hat{\mathbf{s}}_{t+h} &= \begin{cases} \mathbf{s}_t & h = 0 \\ \hat{\mathbf{s}}_{t+h-1} + f_\theta(\hat{\mathbf{s}}_{t+h-1}, \mathbf{a}_{t+h-1}) & h > 0 \end{cases} \end{aligned} \tag{2.2}$$

In this work, we do not use this H-step validation during training; however, this is a meaningful metric that we keep track of, since the controller (introduced in the next section) uses such H-step predictions when performing action selection.

2.4 Using the Learned Model for Control

In order to use the learned model $f_\theta(\mathbf{s}_t, \mathbf{a}_t)$, together with a reward function $r(\mathbf{s}_t, \mathbf{a}_t)$ that encodes some task, we formulate a model-based controller that is both computationally tractable and robust to inaccuracies in the learned dynamics model. We first optimize the sequence of actions $\mathbf{A}_t^{(H)} = (\mathbf{a}_t, \cdots, \mathbf{a}_{t+H-1})$ over a finite horizon H, using the learned dynamics model to predict future states:

$$\begin{aligned} \mathbf{A}_t^{(H)} = \arg\max_{\mathbf{A}_t^{(H)}} \sum_{t'=t}^{t+H-1} r(\hat{\mathbf{s}}_{t'}, \mathbf{a}_{t'}) \quad : \\ \hat{\mathbf{s}}_t = \mathbf{s}_t, \quad \hat{\mathbf{s}}_{t'+1} = \hat{\mathbf{s}}_{t'} + f_\theta(\hat{\mathbf{s}}_{t'}, \mathbf{a}_{t'}). \end{aligned} \tag{2.3}$$

Calculating the exact optimum of Eqn. 2.3 is difficult due to the dynamics and reward functions being nonlinear, but many techniques exist for obtaining approximate solutions to finite-horizon control problems that are sufficient for succeeding at the desired task. In this work, we use a simple random-sampling shooting method (Rao 2009) in which K candidate action sequences are randomly generated, the corresponding state sequences are predicted using the learned dynamics model, the rewards for all sequences are calculated, and the candidate action sequence with the highest expected cumulative reward is chosen. Rather than have the policy execute this action sequence in open-loop, we use model predictive control (MPC): the policy executes only the first action $\mathbf{a}_t$, receives updated state information $\mathbf{s}_{t+1}$, and recalculates the optimal action sequence at the next time step. Note that for higher-dimensional action spaces and longer horizons, random sampling with MPC may be insufficient, and investigating other methods (W. Li and E. Todorov 2004) in future work could improve performance.

Algorithm 1 Model-based RL with MPC Controller

```
1: gather dataset $\mathcal{D}_{\text{RAND}}$ of random trajectories
2: initialize empty dataset $\mathcal{D}_{\text{RL}}$
3: randomly initialize $f_\theta$
4: for iter=1 to max_iter do
5:     train $f_\theta(\mathbf{s}, \mathbf{a})$ via supervised learning (Eqn. 2.1) on mixture of $\mathcal{D}_{\text{RAND}}$ and $\mathcal{D}_{\text{RL}}$
6:     for $t = 1$ to $T$ do
7:         get agent's current state $\mathbf{s}_t$
8:         use $f_\theta$ to select action sequence $\mathbf{A}_t^{(H)}$ by optimizing Eqn. 2.3
9:         execute first action $\mathbf{a}_t$ from selected action sequence $\mathbf{A}_t^{(H)}$
10:        add $(\mathbf{s}_t, \mathbf{a}_t)$ to $\mathcal{D}_{\text{RL}}$
11:    end for
12: end for
```

Note that this combination of predictive dynamics model plus controller is beneficial in that the model is trained only once, but by simply changing the reward function, we can accomplish a variety of goals at run-time, without a need for live task-specific retraining.

To improve the performance of our model-based learning algorithm, we gather additional on-policy data by alternating between gathering data with the current model and retraining the model using the aggregated data. This on-policy data aggregation (i.e., reinforcement learning) improves performance by mitigating the mismatch between the data's state-action distribution and the model-based controller's distribution (Ross, G. J. Gordon, and D. Bagnell 2011). Alg. 1 and Fig. 2.3 provide an overview of our model-based reinforcement learning algorithm.

First, random trajectories are collected and added to dataset $\mathcal{D}_{\text{RAND}}$, which is used to train f_θ by performing gradient descent on the objective in Eqn. 2.1. Then, the model-based MPC controller (Sec. 2.4) gathers T new on-policy datapoints and adds these datapoints to a separate dataset $\mathcal{D}_{\text{RL}}$. The dynamics function f_θ is then retrained using data from both $\mathcal{D}_{\text{RAND}}$ and $\mathcal{D}_{\text{RL}}$. Note that during retraining, the neural network dynamics function's weights are warm-started with the weights from the previous iteration. The algorithm continues alternating between training the model and gathering additional data until a predefined maximum number of

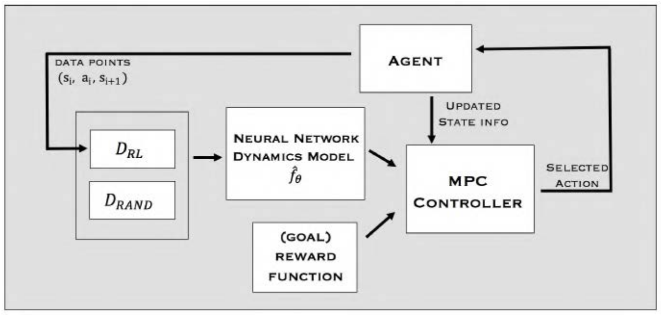


Figure 2.3: Illustration of Algorithm 1. On the first iteration, random actions are performed and used to initialize $\mathcal{D}_{\text{RAND}}$. On all following iterations, this iterative procedure is used to train the dynamics model using the data in the datasets $\mathcal{D}_{\text{RAND}}$ and $\mathcal{D}_{\text{RL}}$, run the MPC controller using the learned model for action selection, aggregate collected data into $\mathcal{D}_{\text{RL}}$, and retrain the model on updated data.

iterations is reached. We evaluate design decisions related to data aggregation in experiments below.

2.5 Model-based RL Results in Simulation

For this set of experiments, we evaluated our model-based RL approach (Alg. 1) on simulated agents (Fig. 2.4) in the MuJoCo (Emanuel Todorov, Erez, and Tassa 2012a) physics engine. The agents we used were swimmer ($\mathcal{S} \in \mathbb{R}^{16}, \mathcal{A} \in \mathbb{R}^{2}$), hopper ($\mathcal{S} \in \mathbb{R}^{17}, \mathcal{A} \in \mathbb{R}^{3}$), half-cheetah ($\mathcal{S} \in \mathbb{R}^{23}, \mathcal{A} \in \mathbb{R}^{6}$), and ant ($\mathcal{S} \in \mathbb{R}^{41}, \mathcal{A} \in \mathbb{R}^{8}$). The code to run these experiments and reproduce these results, as well as videos of all the experiments, are released online[2].

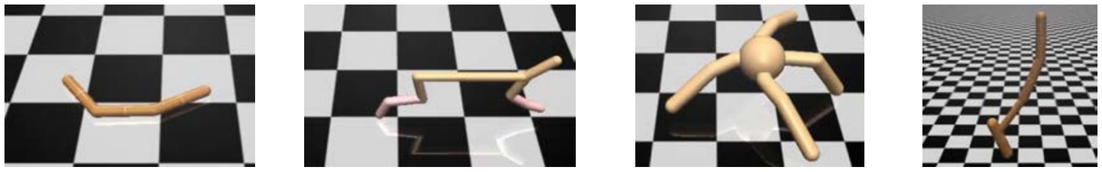

(a) Swimmer (b) Cheetah (c) Ant (d) Hopper

Figure 2.4: Simulated MuJoCo benchmark systems used in these experiments.

2.5.1 Experimental Details and Hyperparameters

In this section, we detail the relevant parameter values and implementation details of our model-based approach. In the tables of parameters listed below, F represents the task of moving forward and TF represents the task of trajectory following.

2.5.1.1 Collecting the initial dataset $\mathcal{D}_{\text{RAND}}$

We populated this initial dataset with rollouts that resulted from the execution of random actions $\mathbf{a}_t \sim \text{Uniform}[-\mathbf{1}, \mathbf{1}]$. Each rollout started at some initial state $\mathbf{s}_0$, but to help with further exploration of the state space, we added noise to this starting state. For all agents, we added noise $\sim \mathcal{N}(0, 0.001)$ to the $\mathbf{s}_0^{\text{POS}}$ and $\mathbf{s}_0^{\text{VEL}}$ elements of the state. The only exception to this starting state noise was for the swimmer on the task of trajectory following. To allow enough exploration of the state space to be able to execute arbitrary trajectories in the future, we found that we had to add more noise to the "heading" element of the swimmer state: We swept this value across the full range of possible headings by adding noise $\sim \text{Uniform}(-\pi, \pi)$. Tables 2.1 and 2.2 below list the number of rollouts and length of each rollout that we collected during this initial random rollout collection, for each domain and task.

[2]`https://sites.google.com/view/mbmf`

Table 2.1: Rollouts in initial random dataset: Trajectory following

	Swimmer TF	Half-Cheetah TF	Ant TF
Number of rollouts	200	200	700
Length of each rollout	500	1000	1000

Table 2.2: Rollouts in initial random dataset: Moving forward

	Swimmer F	Half-Cheetah F	Hopper F	Ant F
Number of rollouts	25	10	20	700
Length of each rollout	333	1000	200	1000

2.5.1.2 Model Training and Control

For all agents, the neural network architecture for the dynamics function consists of two hidden layers, each of dimension 500, with ReLU activations. We trained this dynamics function using the Adam optimizer (Kingma and J. Ba 2014) with learning rate 0.001 and batch size 512. Tables 2.3 and 2.4 below list the amount of training and aggregation performed during the iterative training procedure.

Table 2.3: Training parameters: Trajectory following

	Swimmer TF	Half-Cheetah TF	Ant TF
Training epochs per aggregation iter	70	40	60
Aggregation iters	0	0	0

Table 2.4: Training parameters: Moving forward

	Swimmer F	Half-Cheetah F	Hopper F	Ant F
Training epochs per aggregation iter	30	60	40	20
Aggregation iters	6	7	5	*N/A
Rollouts added per aggregation iter	9	9	10	*N/A
Length of aggregated rollouts	333	1000	**N/A	*N/A
$\mathcal{D}_{\text{RAND}}$-$\mathcal{D}_{\text{RL}}$ split for retraining	10-90	10-90	10-90	*N/A

*N/A : no aggregation performed

**N/A : rollout length varied due to agent's termination conditions

Prior to training, both the inputs and outputs in the dataset were pre-processed to have mean 0 and standard deviation 1. Furthermore, we normalize all environments such that all actions fall in the range $[-\mathbf{1}, \mathbf{1}]$. Shown in tables 2.5 and 2.6 below are additional parameters for these tasks and domains, such as the duration of each environment timestep, the planning horizon for the MPC controller, and the number of action sequences sampled by the controller for each action selection step.

Table 2.5: Other parameters: Trajectory following

	Swimmer TF	Half-Cheetah TF	Ant TF
Timestep dt	0.15s	0.01s	0.02s
Controller horizon H	5	10	15
Number actions sampled K	5000	1000	7000

Table 2.6: Other parameters: Moving forward

	Swimmer F	Half-Cheetah F	Hopper F	Ant F
Timestep dt	0.15s	0.01s	0.02s	0.02s
Controller horizon H	20	20	40	5
Num. actions sampled K	5000	1000	1000	15000

2.5.1.3 Reward Function for Trajectory Following

As described above, the MPC controller is tasked with performing action selection at each time step. To assign a notion of value to a given sequence of actions, for the task for following arbitrarily given trajectories, we use the reward functions shown below.

We formulate this reward function to allow agents to follow trajectories, where the desired trajectory is specified as sparse and lower-dimensional guidance in the form of desired (x, y) center of mass positions. To do this, we first convert the set of desired waypoints into a set of line segments for the agent to travel along. The reward function shown in Alg. 2 computes the reward value R of the given action sequence $\mathbf{A}$, and it does so by penalizing perpendicular distance away from the desired trajectory while encouraging forward progress along the trajectory. Note that this is only one example of a valid reward function for trajectory following, so the reader is free to add other factors to the reward function as desired, such as penalizing for jumping too high or for falling down. We note that the standard MuJoCo reward functions have been similarly tuned to include components such as terminal conditions (e.g., agent falling) and survival rewards.

2.5.2 Evaluating Design Decisions

We first evaluate various design decisions for model-based reinforcement learning with neural networks using empirical evaluations with our model-based approach (Alg. 1). We explored these design decisions for the task of running forward as quickly as possible with the swimmer

Algorithm 2 Reward function for Trajectory Following

```
1: input: current true state $\mathbf{s}_t$,
          sequence of actions $\mathbf{A}=\{\mathbf{a}_0, \mathbf{a}_1, \dots, \mathbf{a}_{H-1}\}$,
          set of desired line segments to follow $L=\{L_0, \dots, L_x\}$
2: reward $R \leftarrow 0$
3: for each action $\mathbf{a}_t$ in $\mathbf{A}$ do
4:     get predicted next state $\hat{\mathbf{s}}_{t+1} = \hat{f}_\theta(\hat{\mathbf{s}}_t, \mathbf{a}_t)$
5:     $L_c \leftarrow$ closest line segment in $L$ to the point $(\hat{\mathbf{s}}^{\text{X}}_{t+1}, \hat{\mathbf{s}}^{\text{Y}}_{t+1})$
6:     $\text{proj}^{\parallel}_t, \text{proj}^{\perp}_t \leftarrow$ project point $(\hat{\mathbf{s}}^{\text{X}}_{t+1}, \hat{\mathbf{s}}^{\text{Y}}_{t+1})$ onto $L_c$
7:     $R \leftarrow R - \alpha(\text{proj}^{\perp}_t) + \beta(\text{proj}^{\parallel}_t - \text{proj}^{\parallel}_{t-1})$
8: end for
9: return: reward $R$
```

and half-cheetah agents; the other agents also exhibited similar trends. After each design decision was evaluated, we used the best outcome of that evaluation for the remainder of the evaluations.

(A) Training steps. Fig. 4.5a shows varying numbers of gradient descent steps taken during each iteration of model learning. As expected, training for too few epochs negatively affects learning performance, with 20 epochs causing swimmer to reach only half of the other experiments' performance.

(B) Dataset aggregation. Fig. 4.5b shows varying amounts of (initial) random data versus (aggregated) on-policy data used within each mini-batch of stochastic gradient descent when training the learned dynamics function. We see that training with at least some aggregated on-policy rollouts significantly improves performance, revealing the benefits of improving learned models with reinforcement learning. However, our method still works well with even just 30% of each mini-batch coming from on-policy rollouts, showing the advantage of model-based reinforcement learning being off-policy.

(C) Controller. Fig. 4.5c shows the effect of varying the horizon H and the number of random samples K used at each time step by the model-based controller. We see that too short of a horizon is harmful for performance, perhaps due to greedy behavior and entry into unrecoverable states. Additionally, the model-based controller for half-cheetah shows worse performance for longer horizons. This is further revealed below in Fig. 2.6, which illustrates a single 100-step validation rollout (as explained in Eqn. 2.2). We see here that the open-loop predictions for certain state elements, such as the center of mass x position, diverge from ground truth. Thus, a large H leads to the use of an inaccurate model for making predictions, which is detrimental to task performance. Finally, with regards to the number of randomly sampled trajectories evaluated, we expect this value needing to be higher for systems with higher-dimensional action spaces.

(D) Number of initial random trajectories. Fig. 4.5d shows varying numbers of random trajectories used to initialize our model-based approach. We see that although a

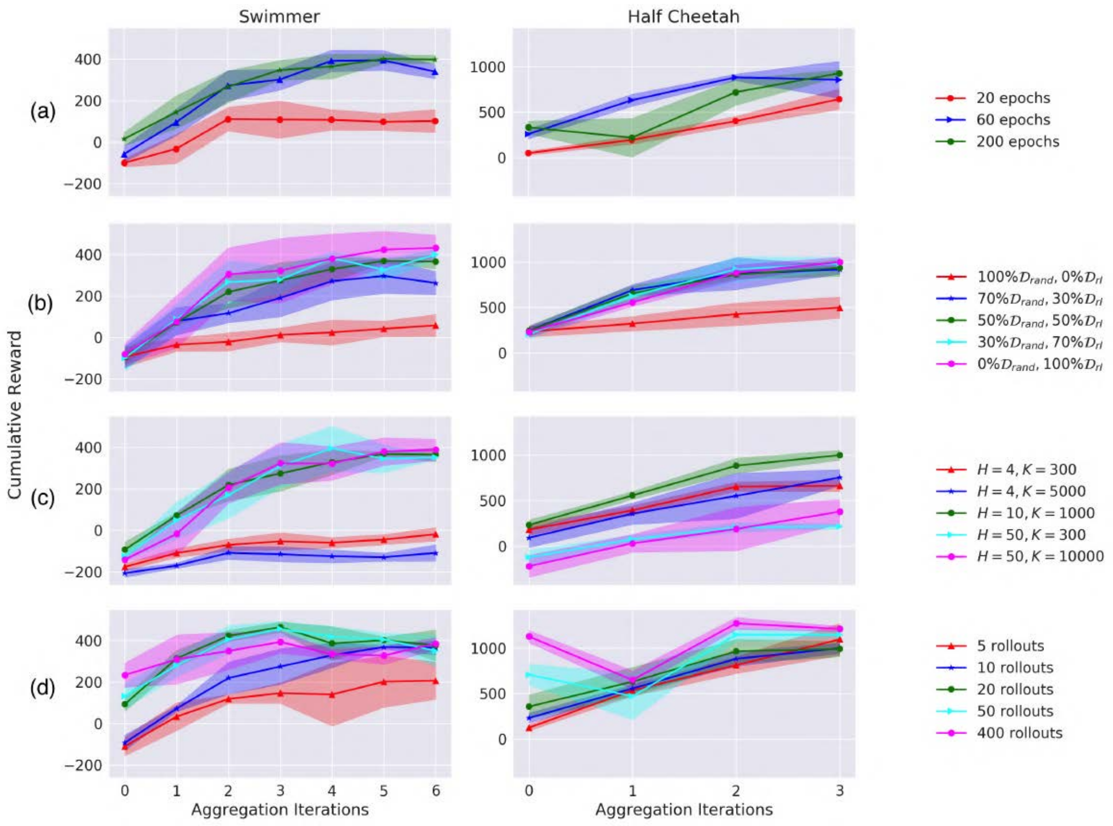


Figure 2.5: Analysis of design decisions for our model-based reinforcement learning approach. (a) Training steps, (b) dataset training split, (c) horizon and number of actions sampled, (d) initial random trajectories. Training for more epochs, leveraging on-policy data, planning with medium-length horizons and many action samples were the best design choices, while data aggregation caused the number of initial trajectories that have little effect.

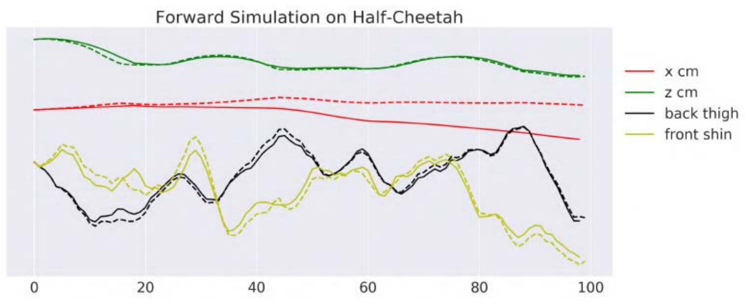


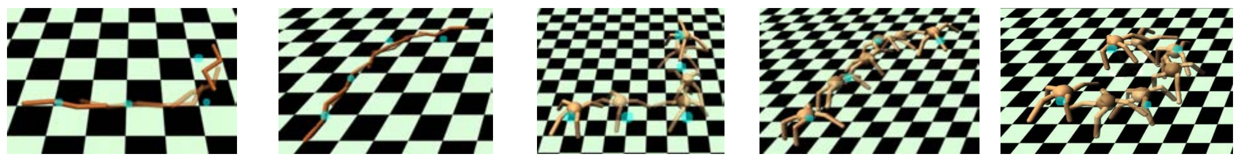
Figure 2.6: We compare a true rollout (solid line) to its corresponding multi-step prediction from the learned model (dotted line) on the half-cheetah. Although we learn to predict certain elements of the state space well, note the eventual divergence of the open-loop predictions over time. Even so, the MPC controller can successfully use the model to control an agent by performing short-horizon planning.

higher amount of initial training data leads to higher initial performance, data aggregation allows low-data initialization runs to reach a high final performance level, highlighting how on-policy data from reinforcement learning improves sample efficiency.

2.5.3 Trajectory Following Results with the MPC Controller

For the task of trajectory following, we evaluated our model-based reinforcement learning approach on the swimmer, ant, and half-cheetah environments (Fig. 2.7). Note that for these tasks, the dynamics model was trained using only random initial trajectories and was trained only once per agent, but the learned model was then used at run-time to accomplish different tasks. These results show that the models learned using our method are general enough to accommodate new tasks at test time, including tasks that are substantially more complex than anything that the robot did during training, such as following a curved path or making a U-turn. Furthermore, we show that even with the use of such a naïve random-sampling controller, the learned dynamics model is powerful enough to perform a variety of tasks.

The reward function we use requires the robot to track the desired x/y center of mass positions. This reward consists of one term to penalize the perpendicular distance away from the desired trajectory, and a second term to encourage forward movement in the direction of the desired trajectory. The reward function does not tell the robot anything about how the limbs should be moved to accomplish the desired center of mass trajectory. The model-based algorithm must discover a suitable gait entirely on its own. Details of this reward are included in Sec. 2.7.1.4.



(a) Swimmer, left (b) Swimmer, right (c) Ant, left (d) Ant, right (e) Ant, u-turn

Figure 2.7: Trajectory following results for the swimmer and ant, with blue dots representing the center-of-mass positions that were specified as the desired trajectory to follow. For each of these agents, we train the dynamics model only once on random trajectories, but can then use it at run-time to execute various desired trajectories.

2.6 Model-free Fine-tuning

The model-based reinforcement learning algorithm described in the previous sections was able to learn complex gaits using very small numbers of samples, when compared to purely model-free learners. However, on benchmark tasks, its final performance still lags behind purely model-free algorithms. To achieve the best final results, we can combine the sample efficiency of model-based learning with the high task-specific performance of model-free learning by using the model-based learner to initialize a model-free learner. We propose a simple but effective method for combining our model-based approach with off-the-shelf, model-free methods by training a policy to mimic the learned model-based controller, and then using the resulting imitation policy as the initialization for a model-free reinforcement learning algorithm.

2.6.1 Initializing the Model-free Learner

We first gather example trajectories with the MPC controller detailed in Sec. 2.4, which uses the learned dynamics function $\hat{f}_\theta$ that was trained using our model-based reinforcement learning algorithm (Alg. 1). We collect the trajectories into a dataset $\mathcal{D}^*$, and we then train a neural network policy $\pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$ to match these "expert" trajectories in $\mathcal{D}^*$. We parameterize π_ϕ as a conditionally Gaussian policy $\pi_\phi(\mathbf{a}_t|\mathbf{s}_t) \sim \mathcal{N}(\mu_\phi(\mathbf{s}_t), \Sigma_{\pi_\phi})$, in which the mean is parameterized by a neural network $\mu_\phi(\mathbf{s})$, and the covariance Σ_{π_ϕ} is a fixed matrix. This policy's parameters are trained using the behavioral cloning objective

$$\min_\phi \frac{1}{2} \sum_{(\mathbf{s}_t, \mathbf{a}_t) \in \mathcal{D}^*} ||\mathbf{a}_t - \mu_\phi(\mathbf{s}_t)||_2^2, \tag{2.4}$$

which we optimize using stochastic gradient descent. To achieve desired performance and address the data distribution problem, we applied DAGGER (Ross, G. J. Gordon, and D. Bagnell 2011): This consisted of iterations of training the policy, performing on-policy rollouts, querying the "expert" MPC controller for "true" action labels for those visited states, and then retraining the policy.

2.6.2 Model-free RL

After initialization, we can use the policy π_ϕ, which was trained on data generated by the learned model-based controller, as an initial policy for a model-free reinforcement learning algorithm. Specifically, we use trust region policy optimization (TRPO) (J. Schulman et al. 2015); such policy gradient algorithms are a good choice for model-free fine-tuning since they do not require any critic or value function for initialization (Grondman et al. 2012), though our method could also be combined with other model-free RL algorithms.

TRPO is also a common choice for the benchmark tasks we consider, and it provides us with a natural way to compare purely model-free learning with our model-based pre-initialization approach. Initializing TRPO with the learned expert policy π_ϕ is as simple as using π_ϕ as the initial policy for TRPO, instead of a standard randomly initialized policy. Although this approach of combining model-based and model-free methods is extremely simple, we demonstrate the efficacy of this approach in our experiments.

2.7 Experimental Mb-Mf Results

We now compare our pure model-based approach with a pure model-free method on standard benchmark locomotion tasks, which require a simulated robot (swimmer, half-cheetah, hopper, ant) to learn the fastest forward-moving gait possible. The model-free approach we compare with is the rllab[3] implementation of trust region policy optimization (TRPO) (J. Schulman et al. 2015), which is know to be successful on these tasks. The code and videos of these experiments are available online[4].

2.7.1 Experimental Details and Hyperparameters

2.7.1.1 The Model-based Component

For the model-based results (which are later used to initialize the model-free policy), we use the parameters listed above in Sec. 2.5.1 for the "forward" tasks of all environments. After we train the model, we collect rollouts from the execution of the MPC controller. At each time step during the collection of these rollouts, noise $\sim \mathcal{N}(0, 0.005)$ is added to the optimal action before execution in order to promote exploration while still achieving good behavior.

2.7.1.2 The Imitation Learning Component

After we collect rollouts from the model-based method, we then train a Gaussian neural network policy to imitate these saved rollouts. We represent the mean of this policy as a neural network composed of tanh nonlinearities and two hidden layers, each of dimension 64. For the imitation learning component, we train this policy using the Adam optimizer (Kingma and J. Ba 2014) with learning rate 0.0001 and batchsize 500. As mentioned in the methods above, supervised learning alone was not enough for this policy, and we had to use the DAGGER algorithm to iteratively improve the behavior cloning results. Table 2.7 below lists relevant parameters for this imitation learning portion of our approach.

Table 2.7: Behavior cloning parameters

	Swimmer	Half-Cheetah	Hopper	Ant
Number of saved MPC rollouts	30	30	60	30
Avg rewards of saved MPC rollouts	30	600	100	110
Number of DAGGER iters	3	3	5	5
Training epochs per DAGGER iter	70	300	200	200
Rollouts aggregated per DAGGER iter	5	2	5	5
Avg rewards of resulting imitation policy	40	500	110	150

2.7.1.3 The Model-free Component

After training the policy to imitate the MPC rollouts, we initialize the model-free TRPO algorithm with that policy and then continue to train it. In addition to training the mean network of that policy initialization using the imitation learning procedure above, the standard

[3]`https://github.com/rll/rllab`
[4]`https://sites.google.com/view/mbmf`

deviation (std) is another parameter of importance. Optimizing this std parameter according to the imitation learning loss function results in worse TRPO performance than arbitrarily using a larger std, perhaps because a higher std on the initial policy leads to more exploration and thus is more beneficial to TRPO. Therefore, we train our policy's mean network using the standard imitation learning loss function, but we manually select the std to be 1.0.

As mentioned above, we use rllab's (Duan, X. Chen, et al. 2016) implementation of the TRPO algorithm for the model-free RL algorithm. We run TRPO with the following parameters for all agents: batch size 50000, base epsilon 10^{-5}, discount factor 0.995, and step size 0.5.

2.7.1.4 The Reward Functions

Table 2.8 below lists the standard reward functions $r(\mathbf{s}_t, \mathbf{a}_t)$ provided online for the benchmark tasks of moving forward with Mujoco agents, which we use in our experiments. As before, the reward R_k corresponding to a given action sequence $\mathbf{A}_k$ of length H is calculated as $R_k = \sum\limits_{t=0}^{H-1} r(\mathbf{s}_t, \mathbf{a}_t^k)$, with r as defined below.

Table 2.8: Standard Mujoco reward functions for moving forward

	Reward r
Swimmer	$\mathbf{s}_{t+1}^{\text{XVEL}} - 0.5\lVert \frac{\mathbf{a}_t}{50} \rVert_2^2$
Half-Cheetah	$\mathbf{s}_{t+1}^{\text{XVEL}} - 0.05\lVert \frac{\mathbf{a}_t}{1} \rVert_2^2$
Hopper	$\mathbf{s}_{t+1}^{\text{XVEL}} + 1 - 0.005\lVert \frac{\mathbf{a}_t}{200} \rVert_2^2$
Ant	$\mathbf{s}_{t+1}^{\text{XVEL}} + 0.5 - 0.005\lVert \frac{\mathbf{a}_t}{150} \rVert_2^2$

2.7.2 Model-free Fine-tuning Results

To compare model-based and model-free results on benchmark tasks, we used the OpenAI gym (Brockman et al. 2016) standard reward functions shown in the previous section. These reward functions primarily reward speed, and are especially difficult for a model-based method due to the myopic nature of the short-horizon MPC that we employ for action selection; therefore, the results of our model-based algorithm on all following plots are lower than would be if we designed our own reward function (for instance, a more dense reward signal such as a straight-line trajectory-following reward function).

Even with these simplistic standard reward functions, our method can very quickly learn a gait that makes forward progress. The swimmer, for example, can quickly achieve qualitatively good moving forward behavior at 20× faster than the model-free method. However, the final achieved reward attained by the pure model-based variant of our approach does not match

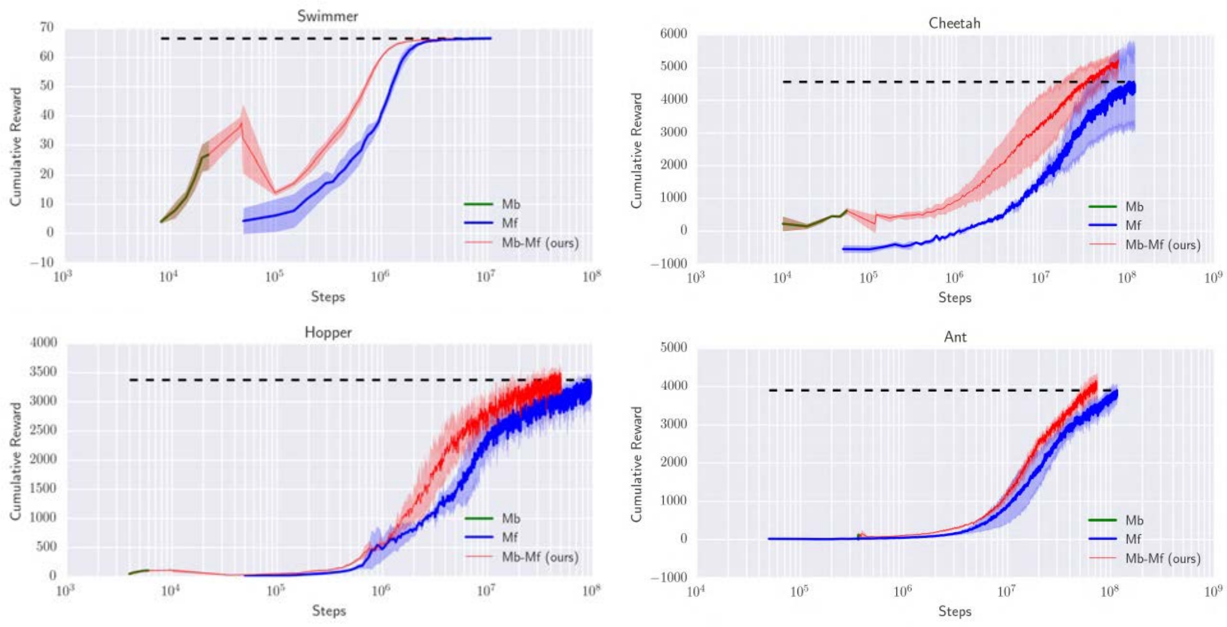


Figure 2.8: Plots show the mean and standard deviation over multiple runs and compare our model-based approach, a model-free approach (TRPO, (J. Schulman et al. 2015)), and our hybrid model-based plus model-free approach. Our combined approach shows a $3-5\times$ improvement in sample efficiency for all shown agents. Note that the x-axis uses a logarithmic scale.

the final performance of state-of-the-art model-free learners, due to an imperfect learned model and the previously discussed sources of suboptimality. When we integrate model-free finetuning (Fig. 2.8), however, the asymptotic performance improves to the level of purely model-free learning. In the case of the hopper, our pure model-based approach learns to perform a double or triple hop very quickly in 1×10^4 steps, but performance plateaus as the reward signal of just forward velocity is not enough for the limited-horizon controller to keep the hopper upright for longer periods of time. Our hybrid Mb-Mf approach takes these quickly-learned gaits and performs model-free fine-tuning in order to achieve high task rewards, achieving $3-5\times$ sample efficiency gains over pure model-free methods for all agents.

2.8 Discussion

In this chapter, we presented a model-based RL algorithm that is able to learn neural network dynamics functions for simulated locomotion tasks using a small number of samples. We described a number of important design decisions for effectively and efficiently training neural network dynamics models, and we presented experiments that evaluated these design parameters. Our method quickly discovered a dynamics model that led to an effective gait; that model could be applied to different trajectory following tasks at run-time, or the initial gait could then be fine-tuned with model-free learning to achieve high task rewards on benchmark Mujoco agents.

In addition to looking at the difference in sample complexity between our hybrid Mb-Mf approach and a pure model-free approach, there are also takeaways from the model-based approach alone. Our model-based algorithm cannot always reach extremely high rewards on its own, but it offers practical use by allowing quick and successful discovery of complex and realistic gaits. In general, our model-based approach can very quickly become competent at a task, whereas model-free approaches can very slowly become experts. For example, when we have a small legged robot with unknown dynamics and we want it to accomplish tasks in the real-world (such as exploration, construction, search and rescue, etc.), achieving reliable walking gaits that can follow any desired trajectory is a superior skill to that of just running straight forward as fast as possible. Additionally, consider the ant: A model-free approach requires 5×10^6 points to achieve a steady walking forward gait, but using just 14% of those data points, our model-based approach can allow for travel in any direction and along arbitrary desired trajectories. Training such a dynamics model only once and applying it to various tasks is compelling; especially when looking toward application to real robots, this sample efficiency can bring these methods out of the simulation world and into the realm of feasibility.

Chapter 3

Scaling Up MBRL for Locomotion

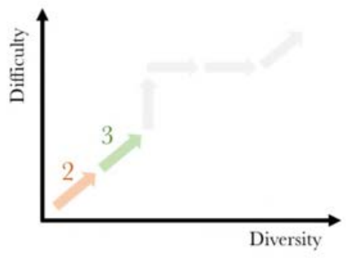


Figure 3.1: Thesis outline, with the current chapter indicated by the last colored arrow.

The work in this chapter extends the model-based learning techniques introduced in the previous chapter along the axis of usefulness and applicability, by training image-conditioned dynamics models and enabling real-world robotic locomotion of legged millirobots on varying terrains.

Millirobots are a promising robotic platform for many applications due to their small size and low manufacturing costs. Legged millirobots, in particular, can provide increased mobility in complex environments and improved scaling of obstacles. However, controlling these small, highly dynamic, and underactuated legged systems is difficult. Hand-engineered controllers can sometimes control these legged millirobots, but they have difficulties with dynamic maneuvers and complex terrains. In this chapter, we present an approach for controlling a real-world legged millirobot that is based on learned neural network models. Using less than 17 minutes of data, our method can learn a predictive model of the robot's dynamics that can enable effective gaits to be synthesized on the fly for following user-specified waypoints on a given terrain. Furthermore, by leveraging expressive, high-capacity neural network models, our approach allows for these predictions to be directly conditioned on camera images, endowing the robot with the ability to predict how different terrains might affect its dynamics. This enables sample-efficient and effective learning for locomotion of a dynamic legged millirobot on various terrains, including gravel, turf, carpet, and styrofoam. Videos of experiments are available online[1].

3.1 Introduction

Legged millirobots are an effective platform for applications, such as exploration, mapping, and search and rescue, because their small size and mobility allows them to navigate through complex, confined, and hard-to-reach environments that are often inaccessible to aerial vehicles

[1]`https://sites.google.com/view/imageconddyn`

and untraversable by wheeled robots. Millirobots also provide additional benefits in the form of low power consumption and low manufacturing costs, which enables scaling them to large teams that can accomplish more complex tasks. This superior mobility, accessibility, and scalability makes legged millirobots some of the most mission-capable small robots available. However, the same properties that enable these systems to traverse complex environments are precisely what make them difficult to control.

Modeling the hybrid dynamics of underactuated legged millirobots from first principles is exceedingly difficult due to complicated ground contact physics that arise while moving dynamically on complex terrains. Furthermore, cheap and rapid manufacturing techniques cause each of these robots to exhibit varying dynamics. Due to these modeling challenges, many locomotion strategies for such systems are hand-engineered and heuristic. These manually designed controllers impose simplifying assumptions, which not only constrain the overall capabilities of these platforms, but also impose a heavy burden on the engineer. Additionally, and perhaps most importantly, they preclude the opportunity for adapting and improving over time.

In this paper, we explore how learning can be used to automatically acquire locomotion strategies in diverse environments for small, low-cost, and highly dynamic legged robots. Choosing an appropriate learning algorithm requires consideration of a number of factors. First, the learned model needs to be expressive enough to cope with the highly dynamic and nonlinear nature of legged millirobots, as well as with high-dimensional sensory observations such as images. Second, the algorithm must allow the robot to learn quickly from modest amounts of data, so as to make it a practical algorithm for real-world application. Third, the learned general-purpose models must be able to be deployed on a wide range of navigational tasks in a diverse set of environments, with minimal human supervision.

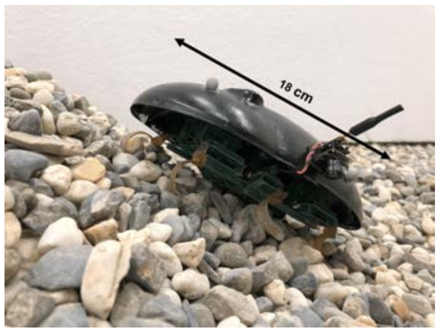


Figure 3.2: VelociRoACH: our small, mobile, highly dynamic, and bio-inspired hexapedal millirobot, shown with a camera mounted for terrain imaging.

The primary contribution of the work work in this chapter is an approach for controlling dynamic legged millirobots that learns an expressive and high-dimensional image-conditioned neural network dynamics model, which is then combined with a model predictive controller (MPC) to follow specified paths. Our sample efficient learning-based approach uses less than 17 minutes of real-world data to learn to follow desired paths in a desired environment, and we empirically show that it outperforms a conventional differential drive control strategy for highly dynamic maneuvers. Our method also enables adaptation to diverse terrains by conditioning its dynamics predictions on its own observed images, allowing it to predict how terrain features such as gravel or turf will alter the system's response. The work in this chapter leverages and builds upon recent advances in learning to achieve a high-performing and sample efficient approach for controlling dynamic legged millirobots on various terrains in the real world.

3.2 Related Work

Controlling Legged Millirobots: Extensive prior work on controlling legged robots includes larger legged robots such as *Anymal* (Hutter et al. 2016), *ASIMO* (Sakagami et al. 2002), and *Big Dog* (Raibert et al. 2008). These systems can achieve successful locomotion, but they have multiple degrees of freedom per leg and a relatively slow stride frequency that allows for more sophisticated control strategies of planned foot placement (Byl 2008; Kolter, Abbeel, and Ng 2008; Kalakrishnan et al. 2010; Zucker et al. 2011). Other prior work includes systems such as RHex (Altendorfer, N. Moore, et al. 2001), where each leg has an independent actuator and can thus execute stable alternating tripod gaits to achieve desired motion. Unlike these systems, however, we are interested in dynamic legged millirobots that are underactuated; these descriptors imply that we cannot move each leg independently, that we have neither the ability nor time to plan specific foot placement, and that we cannot strive for static or quasi-static gaits where stability and well-behaved dynamics can be expected. This realm of steering methods for dynamic running of underactuated legged millirobots includes various methods (McClung III 2006; Zarrouk, Haldane, and Ronald S Fearing 2015), such as actively changing leg kinematics (Clark et al. 2001; Kim, Clark, and Cutkosky 2006), modulating leg impedance (Hoover, Burden, et al. 2010), and executing roll oscillation modulated turning (Haldane and Ronald S Fearing 2014). However, these approaches achieve open-loop turning gaits, while we desire a closed-loop approach to precise path execution. Other traditional methods for both control and modeling of legged systems make simplifying assumptions, such as approximating a system as a spring loaded inverted pendulum (SLIP) model (Komsuoglu, Sohn, et al. 2008; Komsuoglu, Majumdar, et al. 2014) or approximating a system's behavior with a differential drive control strategy. Although these approaches do succeed in certain regimes (Altendorfer, Koditschek, and Holmes 2004), they fail when high speeds or irregular environments lead to more complicated dynamics. In contrast, our neural network learning-based approach can cope with complex dynamics, while also incorporating high-dimensional environmental information in the form of images.

Gait Optimization: Instead of building on simplifying model assumptions to design controllers, prior work has also explored various methods of automatic gait optimization (Da, Hartley, and Grizzle 2017; Gay, Santos-Victor, and Ijspeert 2013). These methods include stochastic gradient descent (Tedrake, T. W. Zhang, and Seung 2005), genetic algorithms (Chernova and Veloso 2004), and Bayesian optimization (Calandra et al. 2014; Lizotte et al. 2007; Tesch, Schneider, and Choset 2011) to reduce the time-consuming design process of manually finding robust parameters. For instance, (Tedrake, T. W. Zhang, and Seung 2005) optimized a control policy for bipedal walking online in less than 20 minutes on a simplified system with 6 joints, and (Gay, Santos-Victor, and Ijspeert 2013) learned model-free sensory feedback controllers to supplement specified open-loop gaits. While these methods are sample efficient and can be applied to real systems, they have not yet been shown to work for high dimensional systems or more complex systems, such as fast robots operating in highly dynamic regimes on irregular surfaces with challenging contact dynamics.

Model-free Policy Learning: Rather than optimizing gaits, prior work in model-free reinforcement learning algorithms has demonstrated the ability to instead learn these behaviors from scratch. Work in this area, including Q-learning (Volodymyr Mnih, Koray Kavukcuoglu, Silver, Rusu, et al. 2015; Oh et al. 2016), actor-critic methods (T. Lillicrap et al. 2016; V. Mnih, Badia, et al. 2016), and policy gradients (J. Schulman et al. 2015), has learned complex skills in high-dimensional state spaces, including skills for simulated robotic locomotion tasks. However, the high sample complexity of such purely model-free algorithms makes them difficult to use for learning in the real world, where sample collection is limited by time and other physical constraints. Unlike these approaches, our model-based learning method uses only minutes of experience to achieve generalizable real-world locomotion skills that were not explicitly seen during training, and it further exemplifies the benefits in sample complexity that arise from incorporating models with learning-based approaches.

Model Learning: Although the sample efficiency of model-based learning is appealing, and although data-driven approaches can eliminate the need to impose restrictive assumptions or approximations, the challenge lies in the difficulty of learning a good model. Relatively simple function approximators such as time-varying linear models have been used to model dynamics of systems (Lioutikov et al. 2014; Yip and Camarillo 2014), including our VeloCiRoACH (Buchan, Haldane, and Ronald S Fearing 2013) platform. However, these models have not yet been shown to posses enough representational power (i.e., accuracy) to generalize to complex locomotion tasks. Prior work has also investigated learning probabilistic dynamics models (M. Deisenroth and C. Rasmussen 2011; Ko and Fox 2008), including Gaussian process models for simulated legged robots (M. P. Deisenroth, Calandra, et al. 2012). While these approaches can be sample efficient, it is intractable to scale them to higher dimensions, as needed especially when incorporating rich sensory inputs such as image observations. In contrast, our method employs expressive neural network dynamics models, which easily scale to high dimensional inputs. Other modeling approaches have leveraged smaller neural networks for dynamics modeling, but they impose strict and potentially restrictive structure to their formulation, such as designing separate modules to represent the various segments of a stride (Crusea et al. 1998), approximating actuators as muscles and tuning these parameters (Xiong, Worgotter, and Manoonpong 2014), or calculating equations of motion and learning error terms on top of these specific models (Grandia, Pardo, and Buchli 2018). Instead, we demonstrate a sample efficient, expressive, and high-dimensional neural network dynamics model that is free to learn without the imposition of an approximated hand-specified structure.

Environment Adaptation: The dynamics of a robot depend not only on its own configuration, but also on its environment. Prior methods generally categorize the problem of adapting to diverse terrains into two stages: first, the terrain is recognized by a classifier trained with human-specified labels (or, less often, using unsupervised learning methods (Leffler 2009)), and second, the gait is adapted to the terrain. This general approach has been used for autonomous vehicles (Thrun, Montemerlo, et al. 2006; Leffler 2009), larger legged robots (Kolter, Abbeel, and Ng 2008; Kalakrishnan et al. 2010; Zucker et al. 2011; Xiong, Worgotter, and Manoonpong 2014; Hoepflinger et al. 2010), and for legged millirobots (Wu et al. 2016; Bermudez et al. 2012). In contrast, our method does not require any human labels at run time, and it adapts to terrains based entirely on autonomous exploration: the dynamics model is simply conditioned on image observations of the terrain, and it automatically learns to recognize the visual cues of terrain features that affect the robot's dynamics.

3.3 Image-Conditioned Models for Capturing the Environment

This work proposes an automated method of acquiring locomotion strategies for small, low-cost, dynamic legged millirobots. In this section, we build on the model-based RL framework from the previous chapter to learn a neural network dynamics model, use the model as part of a model predictive controller, and crucially, enable control on varying real-world terrains by extending the model into an image-conditioned model using features from a pre-trained convolutional neural network.

3.3.1 Overview

We provide an overview of our approach in Fig. 3.3. Note that this model-based RL framework is also the one shown in Fig. 2.3 of the previous chapter, and we will now develop the dynamics model component.

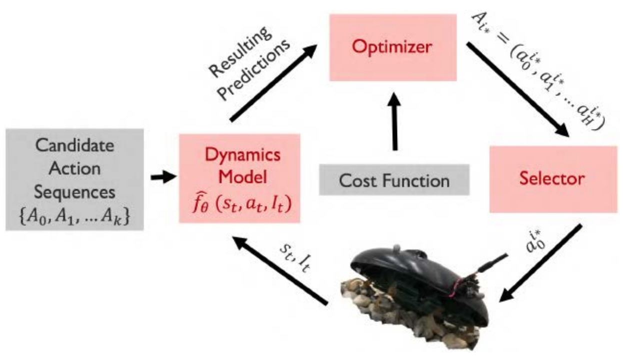

Resulting Predictions
Optimizer
$A_{t*} = (a_0^{t*}, a_1^{t*}, \ldots a_H^{t*})$
Candidate Action Sequences $\{A_0, A_1, \ldots A_k\}$
Dynamics Model $\hat{f}_\theta\,(s_t, a_t, I_t)$
Cost Function
Selector
s_t, I_t
a_0^{t*}

Figure 3.3: Image-conditioned model-based learning for locomotion control: A closed-loop MPC controller performs action selection by using predictions from the learned dynamics model, which is conditioned on the current state $\mathbf{s}_t$, action $\mathbf{a}_t$, and image $\mathbf{I}_t$.

Since we require a parameterization of the dynamics model that can cope with high-dimensional state and action spaces and the complex dynamics of legged millirobots, we represent the dynamics function $f_\theta(\mathbf{s}_t, \mathbf{a}_t)$ as a multilayer neural network, parameterized by θ. As before, this function outputs the predicted change in state that occurs as a result of executing action $\mathbf{a}_t$ from state $\mathbf{s}_t$, over the time step duration of Δt. Thus, the predicted next state is given by: $\hat{\mathbf{s}}_{t+1} = \mathbf{s}_t + f_\theta(\mathbf{s}_t, \mathbf{a}_t)$. While choosing too small of a Δt leads to too small of a state difference to allow meaningful learning, increasing the Δt too much can also make the learning process more difficult because it increases the complexity of the underlying continuous-time dynamics. As described in Algorithm 1 from Section 2.4, initial training data is collected by placing the robot in arbitrary start states and executing random actions at each time step. Each resulting trajectory $\tau = (\mathbf{s}_0, \mathbf{a}_0, \cdots, \mathbf{s}_{T-2}, \mathbf{a}_{T-2}, \mathbf{s}_{T-1})$ of length T is then sliced into training data inputs $(\mathbf{s}_t, \mathbf{a}_t)$ and corresponding output labels $(\mathbf{s}_{t+1} - \mathbf{s}_t)$. We preprocess the training data by normalizing it to be mean 0 and standard deviation 1, which ensures equal weighting of different state elements, regardless of their magnitudes. We then train the dynamics model on data from the training dataset $\mathcal{D}$, using stochastic gradient descent on Equation 2.1.

We formulate a model-based controller which uses the learned model $f_\theta(\mathbf{s}_t, \mathbf{a}_t)$ together with a cost function $c(\mathbf{s}_t, \mathbf{a}_t)$ that encodes some task. Many methods could be used to perform this action selection, and we use a random-sampling shooting method (Rao 2009) which we introduced in the previous chapter, in Section 2.4. At each time step t, we approximate the optimization problem stated in Equation 2.3 by randomly generating K candidate action sequences consisting of H actions each, using the learned dynamics model to predict the resulting states, and then using the cost function to select the action sequence with the lowest cost.

The experiments in this work consist of following arbitrary desired waypoints; for this task of path following, we formulate and use the following cost function:

$$c(\mathbf{s}_t, \mathbf{a}_t) = f_p * p + f_h * h + f_f * f, \tag{3.1}$$

where the parameter f_p penalizes perpendicular distance p away from the desired path, parameter f_f encourages forward progress f along the path, and parameter f_h maintains the heading h of the system toward the desired direction. Rather than executing the entire sequence of selected optimal actions, we use model predictive control (MPC) to execute only the first action $\mathbf{a}_t$, and we then replan at the next time step, given updated state information.

As currently described, our model-based RL approach can successfully follow arbitrary paths when trained and tested on a single terrain. However, in order to traverse complex and varied terrains, it is necessary to adjust the dynamics to the terrain conditions; we will develop such a model in the following section.

3.3.2 Image-Conditioned Dynamics Model

Taking terrain conditions into account is critical for a legged robot to traverse complex and varied terrains in the real world; thus, the model must take environmental information into account, and use that information to correctly be able to predict the future. One approach to succeeding in multiple environments would be to train a separate dynamics model for each terrain. However, in addition to requiring many separate models, this would lead to models that would likely generalize poorly. Furthermore, this approach would require a person to label the training data, as well as each run at test-time, with which terrain the robot is in. All of these aspects are undesirable for an autonomous learning system.

Instead, we propose a simple and highly effective method for incorporating terrain information, using only observations from a monocular color camera mounted on the robotic platform. We formulate an image-conditioned dynamics model $f_\theta(\mathbf{s}_t, \mathbf{a}_t, \mathbf{I}_t)$ that takes as

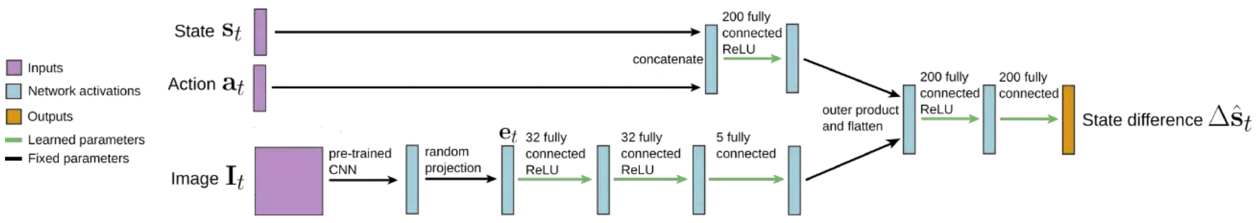


Figure 3.4: Our image-conditioned neural network dynamics model. The model takes as input the current state $\mathbf{s}_t$, action $\mathbf{a}_t$, and image $\mathbf{I}_t$. The image is passed through the convolutional layers of AlexNet (Krizhevsky, Sutskever, and Hinton 2012) pre-trained on ImageNet (Deng et al. 2009), which is then flattened and projected into a lower dimension through multiplication with a random fixed matrix to obtain $\mathbf{e}_t$. The image and concatenated state-action vectors are passed through their own fully connected layers, fused via an outer product, flattened, and passed through more fully connected layers to obtain a predicted state difference $\Delta\hat{\mathbf{s}}_t$.

input not only the current robot state $\mathbf{s}_t$ and action $\mathbf{a}_t$, but also the current image observation $\mathbf{I}_t$. The model (Fig. 3.4) passes image $\mathbf{I}_t$ through the first eight layers of AlexNet (Krizhevsky, Sutskever, and Hinton 2012). The resulting activations are flattened into a vector, and this vector is then multiplied by a fixed random matrix in order to produce a lower dimensional feature vector $\mathbf{e}_t$. The concatenated state-action vector $[\mathbf{s}_t; \mathbf{a}_t]$ is passed through a hidden layer and combined with $\mathbf{e}_t$ through an outer product. As opposed to a straightforward concatenation of $[\mathbf{s}_t; \mathbf{a}_t; \mathbf{e}_t]$, this outer product allows for higher-order integration of terrain information terms with the state and action information terms. This combined layer is then passed through another hidden layer and output layer to produce a prediction of state difference $\Delta\hat{\mathbf{s}}_t$.

Training the entire image-conditioned neural network dynamics model with only minutes of data—corresponding to tens of thousands of datapoints—and in only a few environments would result in catastrophic overfitting. Thus, to perform feature extraction on the images, we use the AlexNet (Krizhevsky, Sutskever, and Hinton 2012) layer weights optimized from training on the task of image classification on the ImageNet (Deng et al. 2009) dataset, which contains 15 million diverse images. Although gathering and labelling this large image dataset was a significant effort, we note that such image datasets are ubiquitous and their learned features have been shown to transfer well to other tasks (Razavian et al. 2014). By using these pre-trained and transferable features, the image-conditioned dynamics model is sample-efficient and can automatically adapt to different terrains without any manual labelling of terrain information.

We show in our experiments that this image-conditioned dynamics model outperforms a naïvely trained dynamics model that is trained simply on an aggregation of all the data. Furthermore, the performance of the image-conditioned dynamics model is comparable, on each terrain, to individual dynamics models that are specifically trained (and tested) on that terrain.

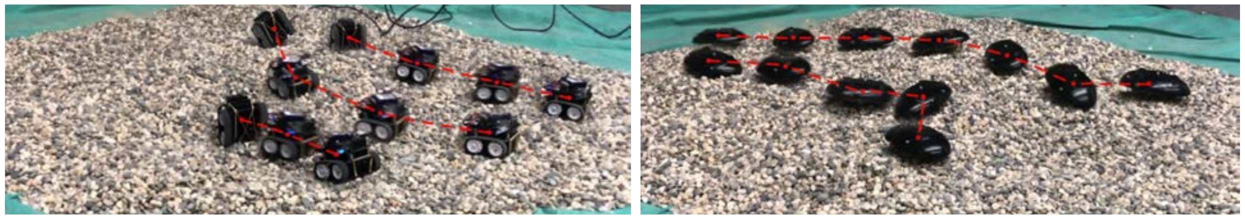


Figure 3.5: Over 15 teleoperated trials performed on rough terrain, a legged robot succeeded in navigating through the terrain 90% of the time, whereas a wheeled robot of comparable size succeeded only 30% of the time.

3.4 Real-World Results of MBRL for Locomotion of Legged Millirobots

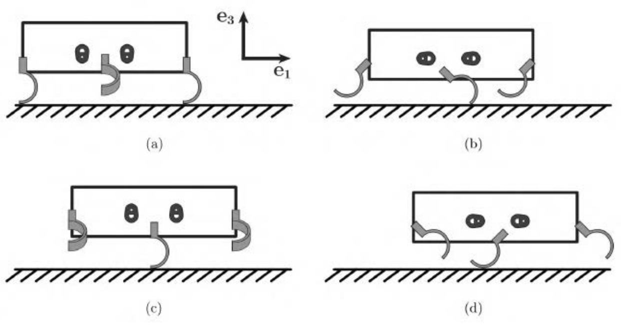
The goal of our experimental evaluation is to study how well our model-based learning algorithm can control a real-world VelociRoACH to follow user-defined paths on various surfaces. Videos of experiments are available online[2].

3.4.1 System Description

The VelociRoACH (Figure 3.2) is a minimally actuated, small, legged, and highly dynamic palm-sized robotic platform. It is 10 cm in length without the shell, can move up to 10-20 body-lengths per second, and uses two motors to control all six legs. Compared to wheeled/treaded robots of similar size (Fig. 3.5), this legged system is able to successfully navigate over more complex terrains. Several design features on the VelociRoACH aim to attain the dynamic performance of its biological inspiration, the American cockroach Periplaneta americana. Details of design techniques and parameter selection can be found in the original VelociRoACH paper (Haldane, Peterson, et al. 2013), and details of the newer modular and more powerful transmission shown in Figure 3.6 can be found in Carlos Casarez's dissertation (Casarez 2018).

The VelociRoACH is constructed through a rapid manufacturing process known as smart composite microstructure (SCM) process (Hoover and Ronald S Fearing 2008). This process allows for the creation of lightweight linkages, enabling the rapid realization of fully functional prototypes of folded flexure-based mobile millirobots. The general SCM process can be thought of as creating flexures by embedding a layer of flexible material in between layers of cut-out rigid material. The VelociRoACH's robot chassis can be constructed for just \$2, and this rigid structural core houses the battery, two motors, transmission, microcontroller, and all sensors. The core also provides mechanical grounding points for the kinematic linkages, which couple each of the two motors to three legs in order to reduce the number of required actuators. The custom transmission independently drives the left and right sets of legs, and

[2]`https://sites.google.com/view/imageconddyn`



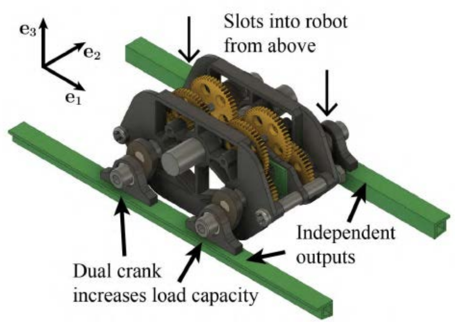

e3
e2
e1
Slots into robot from above
Independent outputs
Dual crank increases load capacity

Figure 3.6: (a-d) Conceptual saggital plane drawing of robot leg positions as a function of crank angle, and (e) isometric solid model view of the leg transmission model. Both of these images are borrowed with permission from (Casarez 2018).

each transmission side is driven by a 3.6 ohm DC motor with a 21.3:1 gear reduction. As shown in Fig. 3.6, the fore and aft legs are constrained to be 180° out of phase from the middle leg. Similar to the design of the X2-VelociRoACH (Haldane and Ronald S Fearing 2015), this VelociRoACH uses two connected output cranks per side to transmit forces at each leg contact.

The VelociRoACH carries an ImageProc embedded circuit board[3], which includes a 40 MHz Microchip dsPIC33F microprocessor, a six axis inertial measurement unit (IMU), an 802.15.4 wireless radio (XBee), and motor control circuitry. We added a 14-bit magnetic rotary encoders to the motors on each side of the robot to monitor absolute position. Additional sensory information includes battery voltage and back-EMF signals from the motors.

The onboard microcontroller runs a low-level 1 kHz control loop and processes communication signals from the XBee. Due to computational limits of the microprocessor, we stream data from the robot to a laptop for calculating controller commands, and then stream these commands back to the microprocessor for execution. To bypass the problem of using only on-board sensors for state estimation, we also use an OptiTrack motion capture system to stream robot pose information during experiments. The motion capture system does not provide any information about the environment terrain, so we also mounted a 3.4 gram monocular color camera onto the VelociRoACH, which communicates directly with the laptop via a radio frequency USB receiver.

3.4.2 Implementation Details

The learned dynamics function $f_\theta(\mathbf{s}_t, \mathbf{a}_t, \mathbf{I}_t)$ is the neural network depicted in Fig. 3.4. For all experiments and results reported below, we use only 17 minutes (10,000 datapoints) worth of data from each terrain to train the dynamics model: This consists of 200 rollouts, each containing 50 data points that are collected at 10 Hz. We train each dynamics model for 50

[3] `https://github.com/biomimetics/imageproc_pcb`

epochs, using the Adam optimizer (Kingma and J. Ba 2014) with learning rate 0.001 and batchsize 1000.

The process of using the neural network dynamics model and the cost function to select the best candidate action sequence at each time step is done in real-time. Relevant parameters for our model-based controller are the number of candidate action sequences sampled at each time step $N = 500$, the horizon $H = 4$, and parameters $f_p = 50$, $f_f = 10$, and $f_h = 5$ for the perpendicular, forward, and heading components of the trajectory following cost function from Eqn. 3.1.

Note that the training data is gathered entirely using random trajectories, and therefore, the paths executed by the controller at run-time differ substantially from the training data. This illustrates the use of off-policy training data, and that the model exhibits considerable generalization. Furthermore, although the model is trained only once, we use it to accomplish a variety of tasks at run-time by simply changing the desired path in the cost function. This decoupling of the task from the dynamics eliminates the need for task-specific training, which further improves overall sample efficiency.

We define the state $\mathbf{s}_t$ of the VelociRoACH to be $[x,\ y,\ z,\ v_x,\ v_y,\ v_z,\ \cos(\phi_r),\ \sin(\phi_r),\ \cos(\phi_p),\ \sin(\phi_p),\ \cos(\phi_y),\ \sin(\phi_y), \omega_x,\ \omega_y,\ \omega_z,\ \cos(a_L),\ \sin(a_L),\ \cos(a_R),\ \sin(a_R),\ v_{a_L},\ v_{a_R},\ \text{bemf}_L,\ \text{bemf}_R,\ V_{bat}]^T$. The center of mass positions (x, y, z) and the Euler angles to describe the center of mass pose (ϕ_r, ϕ_p, ϕ_y) come from the OptiTrack motion capture system. The angular velocities $(\omega_x, \omega_y, \omega_z)$ come from the gyroscope onboard the IMU, and the motor crank positions (a_L, a_R) come from the magnetic rotary encoders, which give a notion of leg position. We include $(\text{bemf}_L, \text{bemf}_R)$ because back-EMF provides a notion of motor torque/velocity, and (V_{bat}) because the voltage of the battery affects the VelociRoACH's performance. Note that the state includes sin and cos of angular values, which is common practice and allows the neural network to avoid wrapping issues.

Finally, we define the action space $\mathbf{a}_t$ of the VelociRoACH to represent the desired velocity setpoints for the rotation of the legs, and we achieve these setpoints using a lower-level PID controller onboard the system. We discuss two of the possible action abstraction choices below in Section 3.4.3.

3.4.3 Action Abstraction

Our presented method allows users the freedom to vary the level of abstraction at which they would like to control their robotic system. Two possible options, which we illustrate in Fig. 3.7 as exhibiting comparable task performance, include directly controlling the motor pulse width modulation (PWM) values or specifying desired velocity setpoints instead.

Directly sending motor commands, instead of velocity setpoints, precludes the need to tune another layer of feedback control (i.e. lower-level PID controller) for calculating motor commands. This method of directly sending commands, however, encounters problems involved with the lack of a feedback loop. In the case of the VelociRoACH, for example, a given PWM value can result in different amounts of leg movement, due to both variations in

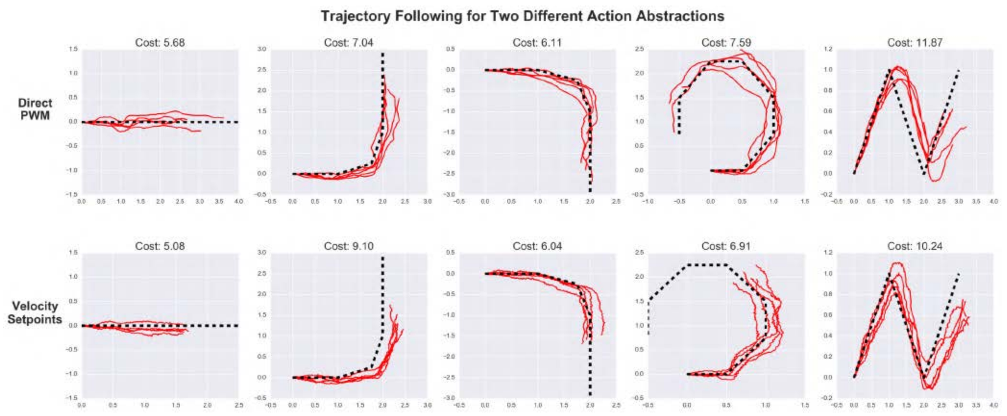


Figure 3.7: Trajectories executed by the model-based controller when the control outputs are (Top:) direct motor PWM values and (Bottom:) leg velocity setpoints, which a lower-level controller is tasked with achieving. Note that for each of these options, the corresponding dynamics model is trained using data where the $\mathbf{a}_t$ represents the indicated choice of action abstraction.

the battery level, as well as due to the leg kinematics leading to different forces at different stages of the leg rotation.

At the same time, outputting desired velocities and then designing a lower-level PID controller to achieve those velocities involves an additional stage of parameter tuning, and one concern includes unpredictable behavior caused by not achieving the desired velocity within the time Δt before the next setpoint is received. Each of these action abstraction options has pros and cons that manifest themselves differently on different systems. Thus, it is an enticing feature to have an algorithm easily adapt to the user's choice of action abstraction, since the best choice may change based on the available system's details.

3.4.4 Improving Performance with More Data

Starting with an experiment on only one type of terrain, we first trained models with different amounts of training data and investigated the effect of the quantity of training data on trajectory following task performance. We trained one model with 50 rollouts (4 minutes), one with 200 rollouts (17 minutes), and one with 400 rollouts (32 minutes). Table 3.1 indicates that more training data can indeed improve task performance. This is an encouraging indication that improvement can occur over time when using a data-driven approach, which is not the case for hand-engineered solutions.

Table 3.1: Trajectory following cost incurred vs. amount of training data

	Straight	Left	Right
50 rollouts (4 minutes)	14.4	16.6	29.4
200 rollouts (17 minutes)	10.3	13.6	17.1
400 rollouts (32 minutes)	10.8	11.3	11.5

3.4.5 Comparing to Differential Drive

To provide a comparison for our model-based learning algorithm's performance, we execute a differential drive controller. This is a common steering method used for robots with wheel or leg-like mechanisms on both sides. In this control scheme, the turn rate ω_{robot} of the robot is proportional to the difference between left ω_l and right ω_r leg velocities. A differential drive controller assumes the behavior that moving the right wheel would turn the robot to the left, and moving the left wheel would turn the robot to the right. Note that this general idea of a difference in leg velocities translating to heading change of the entire system can be implemented in many ways, and we describe our implementation below merely as a guideline.

As described in Algorithm. 3, our implementation of a differential drive controller uses robot heading, as well as perpendicular distance away from the desired path, in order to set velocity setpoints for each side of robot. In addition to standard heading control where the robot turns such that its heading matches the angle of the line, it also incorporates the perpendicular error metric to say that its heading should be more or less than the heading of the line, in order to actually move back toward the line. This controller outputs desired leg velocities at a rate of 10 Hz. To enable the realization of these leg velocities, we also implement a low-level PID controller that runs in the firmware at 1000 Hz. Encoder readings of the leg positions provide feedback, and the PID controller monitors proportional, integral, and derivative errors in order to output the PWM values required for achieving the desired leg velocities.

Algorithm 3 A Differential Drive Algorithm for Trajectory Following

```
 1: Inputs: Current state $(x, y, z, \text{roll}, \text{pitch}, \text{yaw})$,
            Desired waypoints $W = [w_0, w_1, \dots]$,
            Controller parameters $f1$ and $f2$
 2: Line segment $L \leftarrow$ closest $[w_i, w_{i+1}]$ to $(x, y)$
 3: $d_{\text{line}} \leftarrow$ angle of $L$
 4: $p \leftarrow$ perpendicular distance of $(x, y)$ to line segment $L$
 5: if $(x, y)$ to right of $L$
 6: then $d = d_{\text{line}} + f1 * p$
 7: else $d = d_{\text{line}} - f1 * p$
 8: left leg velocity $\omega_l \leftarrow \omega_{\text{nom}} - d * f2$
 9: right leg velocity $\omega_r \leftarrow \omega_{\text{nom}} + d * f2$
10: Outputs: leg PID velocity setpoints $\omega_l$ and $\omega_r$
```

In comparing our method to the differential drive controller, all cost numbers reported in the tables below are calculated on the same cost function (Eqn. 3.1) that indicates how well the executed path aligns with the desired path. Each reported number represents an average over 10 runs. Fig. 3.8 illustrates that the model-based learning method and the differential drive control strategy are comparable at low speeds, across different trajectories on carpet. However, the learned model-based approach outperforms the differential drive strategy at higher speeds; here, differential drive performance deteriorates as leg speeds increase, because traction decreases and causes the legs to have less control over heading. Also, at high speeds, the dynamics of the legged robot can produce significant roll oscillations of the entire system, depending on the leg phasing (Haldane and Ronald S Fearing 2014). Therefore, based on the timing of left and right foot contacts, the system can produce turns inconsistent with a differential drive control strategy. Fig. 3.14 illustrates that for different paths across various surfaces, our model-based learning method outperforms the differential drive control strategy. Furthermore, we note that this difference in performance is most pronounced on surfaces with less traction, such as styrofoam and carpet.

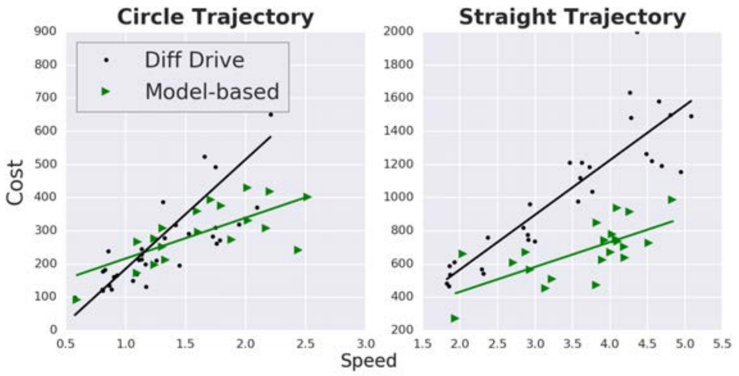


Figure 3.8: An analysis of cost incurred during trajectory following, as a function of the speed of the robot, shows that our model-based learning method is comparable to a differential drive control strategy at low speeds, but outperforms differential drive at high speeds. Each point in these plots represents a separate run, and all runs were conducted on carpet.

3.4.6 A Look at the Telemetry Data

Figure 3.9 shows saved telemetry data (including gyro data, accelerometer data, back EMF, duty cycle, battery voltage, torque, leg positions, commanded actions, robot heading, x, y, roll, pitch, yaw) from the execution of a "left" trajectory by our learned method. These plots illustrate the types of data that are available from the robot and are used for control.

Figure 3.10 shows the distribution of commanded actions for the right vs. left side of the robot during various "straight" and "left" path following runs. The relationship between right and left motor cranks is clear for the differential drive data, and it follows the relationship that we explicitly prescribed in the implementation. The learned approach, on the other hand, exhibits a completely different relationship between the right and left motor commands.

Next, Figure 3.11 shows 4 "left" runs from the learned method (top), and from the differential drive method (bottom). We see a clear correlation between desired robot heading and resulting control commands for the differential drive strategy (as prescribed), but a less clear pattern for the learned method, indicating that the learned approach is doing something different.

Finally, Figure 3.12 shows histograms of the robot's roll angle during the execution of "straight," "right," and "left" trajectories. These histograms show more symmetric distributions of roll angles for the differential drive method, and more skewed roll angles for the learned method, perhaps indicating the enablement of less conservative movements.

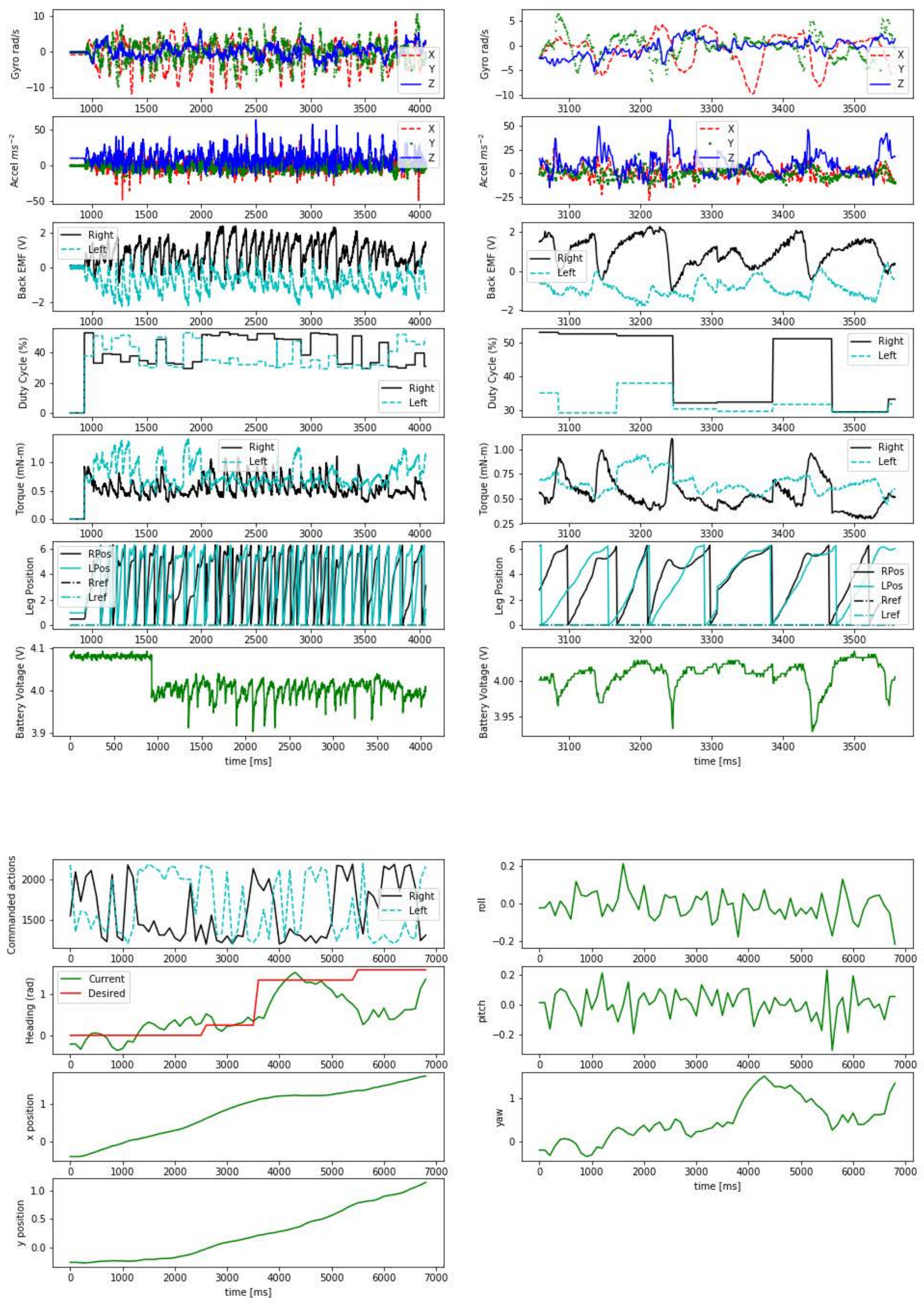


Figure 3.9: **VelociRoACH telemetry data from the learned model performing a left turn.** The top plots show 1kHz data from on-board the robot, with zoomed-in plots on the right. The bottom plots show 10Hz data from the motion capture system, during the same run.

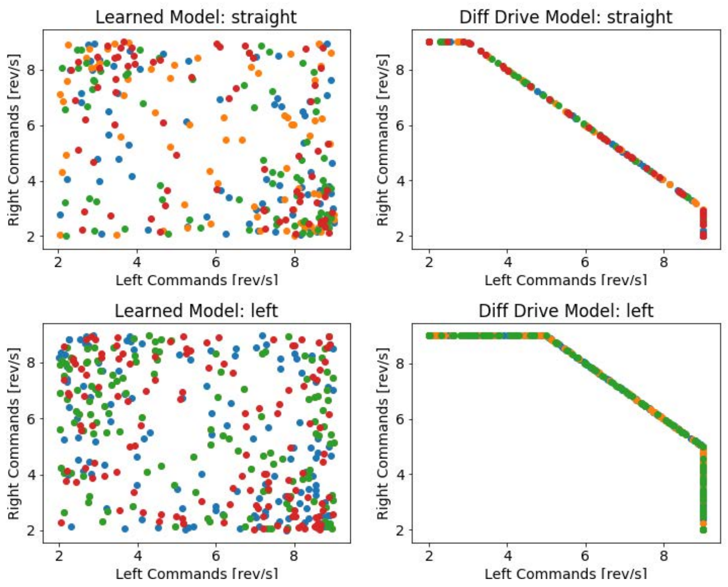


Figure 3.10: Distribution of commanded actions for the right vs. left side of the VelociRoACH, where the commanded actions are velocity setpoints for the legs, in units of leg revolutions per second. The top plots show values from multiple "straight" runs, and the bottom plots show values from multiple "left" runs.

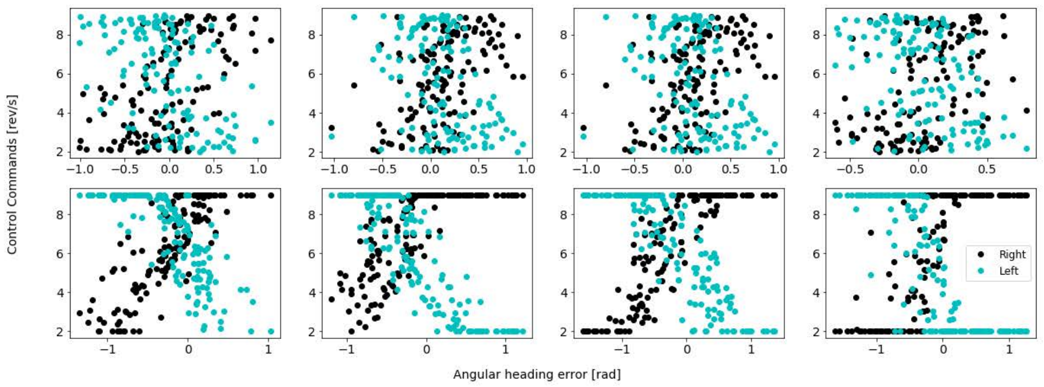


Figure 3.11: Four "left" runs (top) from the learned method, and four "left" runs (bottom) from the differential drive method on the VelociRoACH. The blue dots correspond to the right motor, and the black dots correspond to the left motor. The differential drive runs show a clear correlation between heading error and resulting control commands (as prescribed), whereas the learned model does something different.

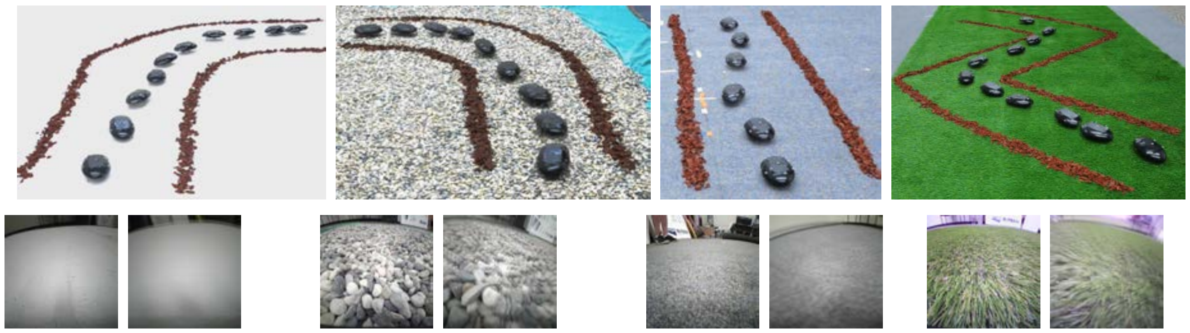


Figure 3.13: Top: Execution of our model-based learning method, using an image-conditioned dynamics model, on various desired paths on four terrains (styrofoam, gravel, carpet, and turf). Note that the path boundaries are outlined for visualization purposes only, and were not present during the experiments. Bottom: Example images from the onboard camera during the runs (shaky images due to body motion).

3.4.7 Qualitative Results of Image-Conditioned Models for Locomotion on Various Terrains

We show in Fig. 3.13 the result of our model-based learning method executing various paths on different surfaces, as well as sample images seen by the VelociRoACH's onboard camera. The demonstrated set of paths captures the key path following primitives of straight lines, arcs, and sharp changes of direction.

3.4.8 Quantitative Results of Image-Conditioned Models for Locomotion on Various Terrains

To analyze the effect of the learned model itself on the controller's performance, we conducted experiments on two materials: a carpet material and a slippery styrofoam material. The costs incurred in Table 3.2 show that the baseline differential drive controller performs relatively poorly on both surfaces. For the model-based approach, the model trained on the carpet works well on the carpet, and the model trained on the styrofoam works well on the styrofoam. The poor performance of either model on the other surface illustrates that the learned dynamics model does in fact encode

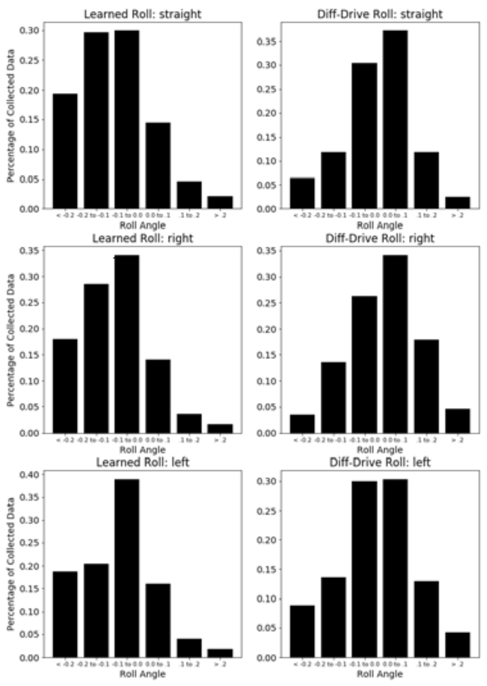


Figure 3.12: Histograms of the VelociRoACH's roll angles during the execution of "straight," "right," and "left" trajectories, showing more symmetric distributions of roll angles for the differential drive method and more skewed roll angles for the learned method.

some knowledge about the surface. Also, performance diminishes when the model is trained on data from both terrains, which indicates that naïvely combining data in order to learn a joint dynamics model is insufficient.

Table 3.2: Costs incurred by the VelociRoACH while executing a straight line path. The model-based controller has the best performance when executed on the surface that it was trained on, indicating that the model incorporates knowledge about the environment. Performance deteriorates when the model is trained on one terrain but tested on another, as well as when the model is jointly trained on all data from all surfaces.

	Carpet	Styrofoam
Differential Drive	13.85	15.45
Model trained on carpet	5.69	18.62
Model trained on styrofoam	22.25	8.15
Model trained on both	7.52	15.76

We have shown so far that when trained on data gathered from a single terrain, our model-based approach is superior to the common differential drive approach, and that our approach improves with more data. Furthermore, the experiment above demonstrated that the robot's dynamics depend on the environment. Thus, we would like our approach to be able to control the VelociRoACH on a variety of terrains.

A standard approach for this would be to train a dynamics model using data from all terrains. However, as shown above in Table 3.2 as well as below in Fig. 3.14, a model that is naïvely trained on all data from multiple terrains and then tested on one of those terrains is significantly worse than a model that is trained solely on that particular terrain. The main reason that this naïve approach does not work well is that the dynamics themselves differ greatly with terrain, and a dynamics model that takes only the robot's current state and action as inputs receives a weak and indirect signal about the robot's environment.

To have a direct signal about the environment, our image-conditioned model takes an additional input: an image taken from an onboard camera, as shown in Fig. 3.13. In Figure 3.14, we compare the performance of this image-conditioned approach to that of alternative approaches on the task of path following for four different paths (straight, left, right, zigzag) on four different surfaces (styrofoam, carpet, gravel, turf). We compare the image-conditioned learned dynamics approach to various alternate approaches:

1. Training a separate dynamics model on each terrain, and testing on that same terrain.

2. Naïvely training one joint dynamics model on all training data, with no images or labels of which terrain the data came from.

3. Training one joint dynamics model using data with explicit terrain labels in the form of a one-hot vector (where the activation of a single vector element directly corresponds to a terrain).

The naïve approach of training one joint dynamics model using an aggregation of all data performs worse than the other learning-based methods. The method of having a separate

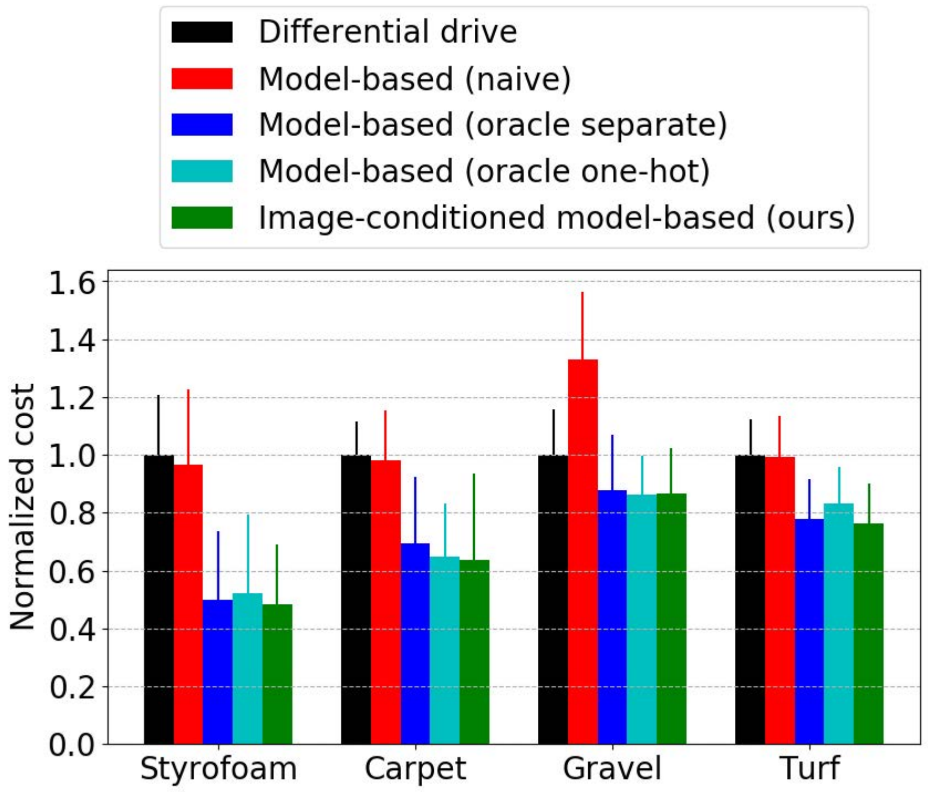


Figure 3.14: Comparison of our image-conditioned model-based approach vs. alternate methods, evaluated on four different terrains, each with four different paths (straight, left, right, and zigzag) on the VelociRoACH. The methods that we compare to include: a hand-engineered differential drive controller, a joint dynamics model that is naïvely trained on all data from all terrains, an "oracle" approach that uses a separate dynamics model on each terrain, and another "oracle" approach where the joint dynamics model is trained using all data with extra one-hot vector labels of the terrain. Our method outperforms the differential drive method and the naïve model-based controller, while performing similarly to the oracle baselines without needing any explicit labels.

dynamics model for each terrain, as well as the method of training one joint dynamics model using one-hot vectors as terrain labels, both perform well on all terrains. However, both of these methods require human supervision to label the training data and to specify which terrain the robot is on at test time. In contrast, our image-conditioned approach performs just as well as the separate and one-hot models, but does not require any additional supervision beyond an onboard monocular camera. Finally, our image-conditioned approach also substantially outperforms the differential drive baseline on all terrains.

3.4.9 Discussion

In this chapter, we presented a sample-efficient model-based learning algorithm using image-conditioned neural network dynamics models that enabled accurate locomotion of a low-cost, under-actuated, legged, and highly dynamic VelociRoACH robot in a variety of environments. Using only 17 minutes of real-world data for each terrain, our method outperformed a commonly used differential drive control strategy, showed improvement with more data, and was able to use features from camera images in order to execute successful locomotion on various terrains with human-provided labels or task-specific training.

Chapter 4

Scaling Up MBRL for Dexterous Manipulation

The work in this chapter extends the capabilities of model-based learning techniques along the axis of task complexity. By scaling up aspects of both the modeling and control techniques of the model-based approach introduced in Chapter 2, the work in this chapter enables efficient and effective learning of various flexible contact-rich dexterous manipulation skills both in simulation and in the real world.

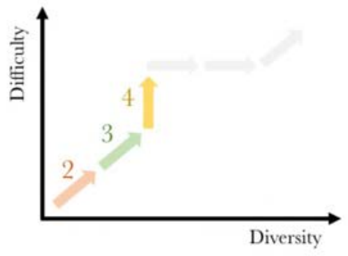


Figure 4.1: Thesis outline, with the current chapter indicated by the last colored arrow.

Dexterous multi-fingered hands can provide robots with the ability to flexibly perform a wide range of manipulation skills. However, many of the more complex behaviors are also notoriously difficult to control: Performing in-hand object manipulation, executing finger gaits to move objects, and exhibiting precise fine motor skills such as writing, all require finely balancing contact forces, breaking and reestablishing contacts repeatedly, and maintaining control of unactuated objects. Learning-based techniques provide the appealing possibility of acquiring these skills directly from data, but current learning approaches either require large amounts of data and produce task-specific policies, or they have not yet been shown

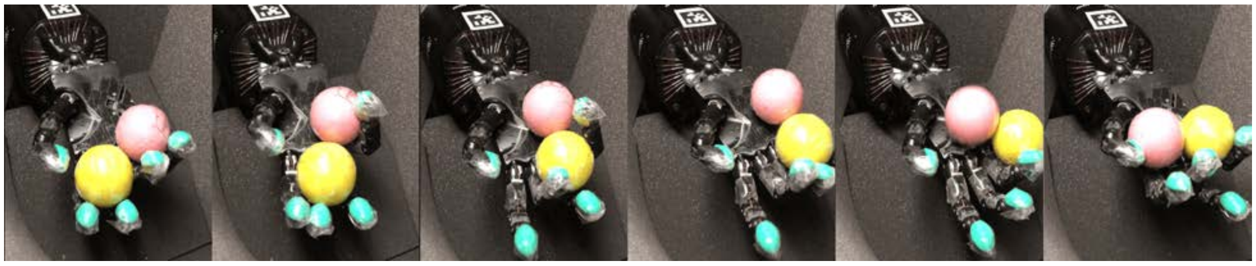


Figure 4.2: PDDM can efficiently and effectively learn complex dexterous manipulation skills in both simulation and the real world. Here, less than 4 hours of experience is needed for the Shadow Hand to learn to rotate two free-floating Baoding balls in the palm, without any prior knowledge of system dynamics.

to scale up to more complex and realistic tasks requiring fine motor skills. In this chapter, we demonstrate that our method of online planning with deep dynamics models (PDDM) addresses both of these limitations; we show that improvements in learned dynamics models, together with improvements in online model-predictive control, can indeed enable efficient and effective learning of flexible contact-rich dexterous manipulation skills – and that too, on a 24-DoF anthropomorphic hand in the real world, using just 4 hours of purely real-world data to learn to simultaneously coordinate multiple free-floating objects. Videos of the experiments as well as the code are available online[1].

4.1 Introduction

Dexterous manipulation with multi-fingered hands represents a grand challenge in robotics: the versatility of the human hand is as yet unrivaled by the capabilities of robotic systems, and bridging this gap will enable more general and capable robots. Although some real-world tasks can be accomplished with simple parallel jaw grippers, there are countless tasks in which dexterity in the form of redundant degrees of freedom is critical. In fact, dexterous manipulation is defined (Okamura, Smaby, and Cutkosky 2000) as being object-centric, with the goal of controlling object movement through precise control of forces and motions – something that is not possible without the ability to simultaneously impact the object from multiple directions. Through added controllability and stability, multi-fingered hands enable useful fine motor skills that are necessary for deliberate interaction with objects. For example, using only two fingers to attempt common tasks such as opening the lid of a jar, hitting a nail with a hammer, or writing on paper with a pencil would quickly encounter the challenges of slippage, complex contact forces, and underactuation. Success in such settings requires a sufficiently dexterous hand, as well as an intelligent policy that can endow such a hand with the appropriate control strategy.

The principle challenges in dexterous manipulation stem from the need to coordinate numerous joints and impart complex forces onto the object of interest. The need to repeatedly establish and break contacts presents an especially difficult problem for analytic approaches, which require accurate models of the physics of the system. Learning offers a promising data-driven alternative. Model-free reinforcement learning (RL) methods can learn policies that achieve good performance on complex tasks (Van Hoof et al. 2015; Levine, Finn, et al. 2016; Rajeswaran et al. 2017); however, we will show that these state-of-the-art algorithms struggle when a high degree of flexibility is required, such as moving a pencil to follow *arbitrary* user-specified strokes. Here, complex contact dynamics and high chances of task failure make the overall skill much more difficult. Model-free methods also require large amounts of data, making them difficult to use in the real world. Model-based RL methods, on the other hand, can be much more efficient, but have not yet been scaled up to such complex tasks. In this work, we aim to push the boundary on this task complexity; consider, for instance, the task of rotating two Baoding balls around the palm of your hand (Figure 4.2).

[1]`https://sites.google.com/view/pddm`

We will discuss how model-based RL methods can solve such tasks, both in simulation and on a real-world robot.

Algorithmically, we present a technique that combines elements of recently-developed uncertainty-aware neural network models with state-of-the-art gradient-free trajectory optimization. While the individual components of our method are based heavily on prior work, we show that their combination is both novel and critical. Our approach, based on deep model-based RL, challenges the general machine learning community's notion that models are difficult to learn and do not yet deliver control results that are as impressive as model-free methods. In this chapter, we push forward the empirical results of model-based RL, in both simulation and the real world, on a suite of dexterous manipulation tasks starting with a 9-DoF three-fingered hand (Zhu et al. 2019) rotating a valve, and scaling up to a 24-DoF anthropomorphic hand executing handwriting and manipulating free-floating objects (Figure 4.3). These realistic tasks require not only learning about interactions between the robot and objects in the world, but also effective planning to find precise and coordinated maneuvers while avoiding task failure (e.g., dropping objects). The work in this chapter demonstrates for the first time that deep neural network models can indeed enable sample-efficient and autonomous discovery of fine motor skills with high-dimensional manipulators, including a real-world dexterous hand trained entirely using just 4 hours of real-world data.

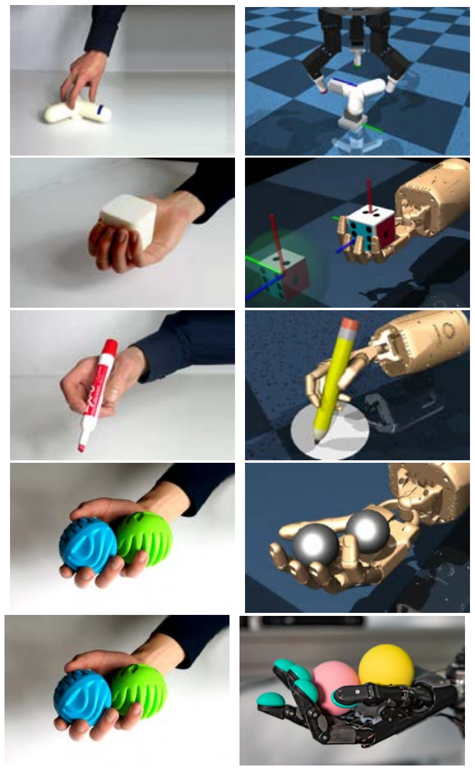


Figure 4.3: Task suite of simulated and real-world dexterous manipulation: valve rotation, in-hand reorientation, handwriting, and manipulating Baoding balls.

4.2 Related Work

In recent years, the mechanical and hardware development of multi-fingered robotic hands has significantly advanced (Butterfaß et al. 2001; Xu and Emanuel Todorov 2016), but our manipulation capabilities haven't scaled similarly. Prior work in this area has explored a wide spectrum of manipulation strategies: (Sundaralingam and Hermans 2018)'s optimizer used geometric meshes of the object to plan finger-gaiting maneuvers, (Andrews and Kry 2013)'s multi-phase planner used the help of offline simulations and a reduced basis of hand poses to accelerate the parameter search for in-hand rotation of a sphere, (Dogar and Srinivasa 2010)'s motion planning algorithm used the mechanics of pushing to funnel an object into a stable grasp state, and (Bai and Liu 2014)'s controller used the conservation of mechanical energy to control the tilt of a palm and roll objects to desired positions on the hand. These types of manipulation techniques have thus far struggled to scale to more complex tasks or sophisticated manipulators in simulation as well as the real world, perhaps due to their need for precise characterization (Okada 1982) of the system and its environment. Reasoning through contact models and motions cones (Chavan-Dafle, Holladay, and Rodriguez 2018; Kolbert, Chavan-Dafle, and Rodriguez 2016), for example, requires computation time that scales exponentially with the number of contacts, and has thus been limited to simpler manipulators and more controlled tasks. With this work, we aim to significantly scale up the complexity of feasible tasks, while also minimizing such task-specific formulations.

More recent work in deep RL has studied this question through the use of data-driven learning to make sense of observed phenomenon (Andrychowicz, B. Baker, et al. 2018; Van Hoof et al. 2015). These methods, while powerful, require large amounts of system interaction to learn successful control policies, making them difficult to apply in the real world. Some work (Rajeswaran et al. 2017; Zhu et al. 2019) has used expert demonstrations to improve this sample efficiency. In contrast, our method is sample efficient without requiring any expert demonstrations, and is still able to leverage data-driven learning techniques to acquire challenging dexterous manipulation skills.

Model-based RL has the potential to provide both efficient and flexible learning. In fact, methods that assume perfect knowledge of system dynamics can achieve very impressive manipulation behaviors (Mordatch, Popović, and Emanuel Todorov 2012; Lowrey et al. 2018) using generally applicable learning and control techniques. Other work has focused on learning these models using high-capacity function approximators (M. P. Deisenroth, Neumann, Peters, et al. 2013; Lenz, Knepper, and Saxena 2015; Levine, Finn, et al. 2016; Nagabandi, Yang, et al. 2017; Nagabandi, Kahn, Ronald S. Fearing, et al. 2018; Williams, Wagener, et al. 2017; Chua et al. 2018a) and probabilistic dynamics models (M. Deisenroth and C. Rasmussen 2011; Ko and Fox 2008; M. P. Deisenroth, Calandra, et al. 2012; Doerr et al. 2017). Our method combines components from multiple prior works, including uncertainty estimation (M. Deisenroth and C. Rasmussen 2011; Chua et al. 2018a; Kurutach et al. 2018a), deep models and model-predictive control (MPC) (Nagabandi, Yang, et al. 2017), and stochastic optimization for planning (Williams, Aldrich, and Theodorou 2015). Model-based RL methods, including recent work in uncertainty estimation (A. Malik et al. 2019) and combining policy networks with online planning (T. Wang and Jimmy Ba 2019), have unfortunately mostly been studied and shown on lower-dimensional (and often, simulated) benchmark tasks, and scaling these methods to higher dimensional tasks such as dexterous manipulation has proven to be a challenge. As illustrated in our evaluation, the particular synthesis of different ideas in this work allows model-based RL to push forward the task complexity of achievable dexterous manipulation skills, and to extend this progress to even a real-world robotic hand.

4.3 PDDM: Online Planning with Deep Dynamics Models

In order to leverage the benefits of autonomous learning from data-driven methods while also enabling efficient and flexible task execution, we extend deep model-based RL approaches to the domain of dexterous manipulation. Our method of online planning with deep dynamics models (PDDM) builds on the model-based RL framework that we presented in the previous chapters by improving both the modeling as well as the planning. As we illustrate in our experiments, the particular combination of components is critical for the success of our method on complex dexterous manipulation tasks and allows it to substantially outperform these prior model-based algorithms as well as strong model-free baselines.

4.3.1 Learning Deep Dynamics Models

Recall the model-based RL problem that we introduced in the previous chapters, where there is a Markov decision process (MDP) with a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, and a state transition distribution $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ describing the result of taking action $\mathbf{a}$ from state $\mathbf{s}$. The task is specified by a bounded reward function $r(\mathbf{s}, \mathbf{a})$, and the goal of RL is to select actions in such a way as to maximize the expected sum of rewards over a trajectory. Model-based RL aims to solve this problem by first learning an approximate model $\hat{p}_\theta(\mathbf{s}'|\mathbf{s}, \mathbf{a})$, parameterized by θ, that approximates the unknown transition distribution $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ of the underlying system dynamics. The parameters θ can be learned to maximize the log-likelihood of observed data $\mathcal{D}$, and the learned model's predictions can then be used to either learn a policy or, as we do, to perform online planning to select optimal actions.

In our work, we use a deep neural network model to represent $\hat{p}_\theta(\mathbf{s}'|\mathbf{s}, \mathbf{a})$, such that the model has enough capacity to capture the complex interactions involved in dexterous manipulation. As in the previous chapters, we use a parameterization of the form $\hat{p}_\theta(\mathbf{s}'|\mathbf{s}, \mathbf{a}) = \mathcal{N}(f_\theta(\mathbf{s}, \mathbf{a}), \mathbf{\Sigma})$, where the mean $f_\theta(\mathbf{s}, \mathbf{a})$ is given by a neural network, and the covariance $\mathbf{\Sigma}$ of the conditional Gaussian distribution can also be learned (although we found this to be unnecessary for good results).

As prior work has indicated, capturing epistemic uncertainty in the network weights is indeed important in model-based RL, especially with high-capacity models that are liable to overfit to the training set and extrapolate erroneously outside of it. A simple and inexpensive way to do this is to employ bootstrap ensembles (Chua et al. 2018a), which approximate the posterior $p(\theta|\mathcal{D})$ with a set of E models, each with parameters θ_i. For deep models, prior work has observed that bootstrap resampling is unnecessary, and it is sufficient to simply initialize each model θ_i with a different random initialization θ_i^0 and use different batches of data $\mathcal{D}_i$ at each train step (Chua et al. 2018a). We note that, as with the work in the previous chapters, this supervised learning setup makes more efficient use of the data than the counterpart model-free methods, since we get dense training signals from each state transition and we are able to use all data (even off-policy data) to make training progress.

4.3.2 Online Planning for Closed-Loop Control

In this work, we use the model predictions in an online planning setup where, at each time step, we perform short-horizon trajectory optimization, as introduced in Equation 2.3. The chosen optimizer will now perform action selection by using the mean predicted reward across all models of the ensemble (instead of the predicted reward from a single model, as before), thus allowing for uncertainty in the form of model disagreement to effect action selection. For example, considering a distribution of reward value predictions as opposed to a single one could discourage the exploitation of an overly-optimistic model. For the choice of optimizer itself, we describe a few particular choices below, each of which builds on the previous ones and ends with the most sophisticated optimizer – the one that is used by PDDM.

Random Shooting: The simplest gradient-free optimizer is the one used in the previous chapters of this work. It simply generates K independent random action sequences $\{\mathbf{A}_0 \dots \mathbf{A}_K\}$, where each sequence $\mathbf{A}_k = \{\mathbf{a}_0^k \dots \mathbf{a}_{H-1}^k\}$ is of length H action. Given a reward function $r(\mathbf{s}_t, \mathbf{a}_t)$ that defines the task, and given future state predictions $\hat{\mathbf{s}}_{t+1} = f_\theta(\hat{\mathbf{s}}_t, \mathbf{a}_t) + \hat{\mathbf{s}}_t$ from the learned dynamics model f_θ, the optimal action sequence $\mathbf{A}_{k^*}$ is selected to be the one corresponding to the sequence with highest predicted reward: $k^* = \arg\max_k R_k = \arg\max_k \sum_{t'=t}^{t+H-1} r(\hat{\mathbf{s}}_{t'}, \mathbf{a}_{t'}^k)$. We showed this approach to achieve success on continuous control tasks with learned models in the previous chapters, but it does have numerous drawbacks: it scales poorly with the dimension of both the planning horizon and the action space, and it often is insufficient for achieving high task performance since a sequence of actions sampled at random often does not directly lead to meaningful behavior.

Iterative Random-Shooting with Refinement: To address these issues, much prior work (Botev et al. 2013) has instead taken a cross-entropy method (CEM) approach, which begins as the random shooting approach, but then does this sampling for multiple iterations $m \in \{0 \dots M\}$ at each time step. The top J highest-scoring action sequences from each iteration are used to update and refine the mean and variance of the sampling distribution for the next iteration, as follows:

$$\begin{aligned}
\mathbf{A}_k &= \{\mathbf{a}_0^k \dots \mathbf{a}_{H-1}^k\}, \text{ where } \mathbf{a}_t^k \sim \mathcal{N}(\boldsymbol{\mu}_t^m, \boldsymbol{\Sigma}_t^m) \;\; \forall k \in K, t \in 0 \dots H-1 \\
\mathbf{A}_{\text{elites}} &= \text{sort}(\mathbf{A}_k)[-J:] \\
\boldsymbol{\mu}_t^{m+1} &= \alpha * \text{mean}(\mathbf{A}_{\text{elites}}) + (1-\alpha)\boldsymbol{\mu}_t^m \;\; \forall t \in 0 \dots H-1 \\
\boldsymbol{\Sigma}_t^{m+1} &= \alpha * \text{var}(\mathbf{A}_{\text{elites}}) + (1-\alpha)\boldsymbol{\Sigma}_t^m \;\; \forall t \in 0 \dots H-1
\end{aligned} \tag{4.1}$$

After M iterations, the optimal actions are selected to be the resulting mean of the action distribution.

Filtering and Reward-Weighted Refinement: While CEM is a stronger optimizer than random shooting, it still scales poorly with dimensionality and is hard to apply when both coordination and precision are required. PDDM instead uses a stronger optimizer that considers covariances between time steps and uses a softer update rule that more effectively integrates a larger number of samples into the distribution update. As derived by recent model-predictive path integral work (Williams, Aldrich, and Theodorou 2015; Lowrey et al. 2018), this general update rule takes the following form for time step t, reward-weighting factor γ, and reward R_k from each of the K predicted trajectories:

$$\boldsymbol{\mu}_t = \frac{\sum_{k=0}^{N}(e^{\gamma \cdot R_k})(\mathbf{a}_t^k)}{\sum_{j=0}^{N} e^{\gamma \cdot (R_j)}} \quad \forall t \in \{0 \ldots H-1\}. \tag{4.2}$$

Rather than sampling the action samples from a random policy or from iteratively refined Gaussians, we instead apply a filtering technique to explicitly produce smoother candidate action sequences. Given the iteratively updating mean distribution $\boldsymbol{\mu}_t$ from above, we generate K action sequences $\mathbf{a}_t^k = \mathbf{n}_t^k + \boldsymbol{\mu}_t$, where each noise sample $\mathbf{n}_t^k$ is generated using filtering coefficient β as follows:

$$\mathbf{u}_t^k \sim \mathcal{N}(0, \boldsymbol{\Sigma}) \quad \forall k \in \{0 \ldots K-1\}, t \in \{0 \ldots H-1\} \tag{4.3}$$

$$\mathbf{n}_t^k = \beta \cdot \mathbf{u}_t^k + (1-\beta) \cdot \mathbf{n}_{t-1}^k \quad \text{where} \quad \mathbf{n}_{t<0} = 0 \tag{4.4}$$

By coupling time steps to each other, this filtering also reduces the effective degrees of freedom or dimensionality of the search space, thus allowing for better scaling with dimensionality.

4.3.3 Overview

After using the models and predicted rewards to perform action selection, we take one step $\mathbf{a}_t$ of the selected action plan, receive updated state information $\mathbf{s}_{t+1}$, and then replan at the following time step. This closed-loop method of replanning using updated information at every time step helps to mitigate some model inaccuraries by preventing accumulating model error. Note that this control procedure also allows us to easily swap out new reward functions or goals at run-time, independent of the trained model. Overall, the full procedure of PDDM involves iteratively performing actions in the real world (through online planning with the use of the learned model) and then using those observations to update that learned model, as stated in Algorithm 1 from Section 2.4. Further implementation details of this procedure are provided in the sections below.

4.4 Results of MBRL for Dexterous Manipulation

Our evaluations in this section aim to address the following questions:

1. Can PDDM autonomously learn to accomplish a variety of complex dexterous manipulation tasks?

2. What is the effect of the various design decisions in PDDM?

3. How does the performance as well as sample efficiency of PDDM compare to that of other state-of-the-art algorithms?

4. How general and versatile is the learned model?

5. Can we apply these lessons learned from simulation to enable a 24-DoF humanoid hand to manipulate free-floating objects in the real world?

All experiment videos as well as the released code can be found online on the project website[2].

4.4.1 Task Suite

Some of the main challenges in dexterous manipulation involve the high dimensionality of the hand, the prevalence of complex contact dynamics that must be utilized and balanced to manipulate free floating objects, and the potential for failure. We identified a set of tasks (Figure 4.3) that specifically highlight these challenges by requiring delicate, precise, and coordinated movement. With the final goal in mind of real-world experiments on a 24-DoF hand, we first set up these selected dexterous manipulation tasks in MuJoCo (Emanuel Todorov, Erez, and Tassa 2012b) and conducted experiments in simulation on two robotic platforms: a 9-DoF three-fingered hand, and a 24-DoF five-fingered hand. We show in Figure 4.4 the successful executions of PDDM on these tasks, which took between 1-2 hours worth of data for simulated tasks and 2-4 hours worth of data for the real-world experiments. We provide brief task overviews below.

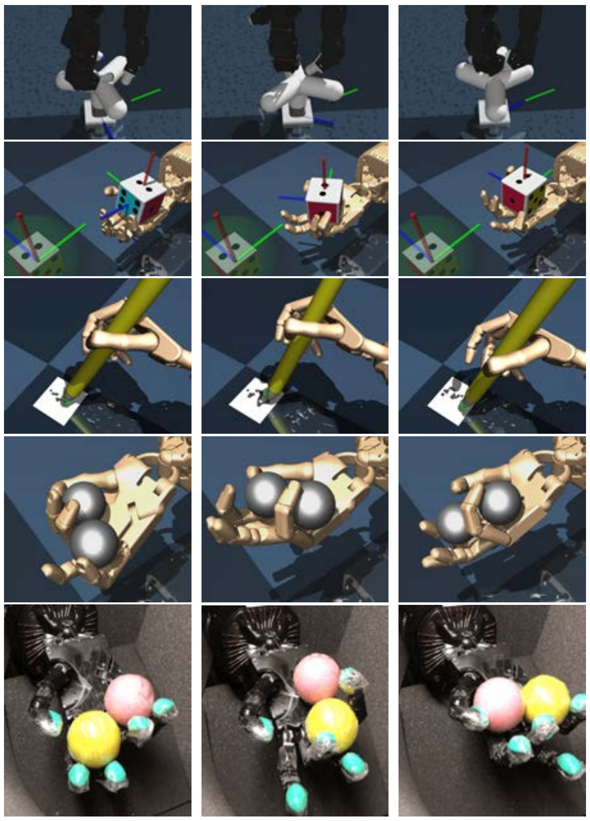


Figure 4.4: Successful executions of PDDM on simulated and real-world dexterous manipulation tasks.

Valve Turning: This starter task for looking into manipulation challenges uses a 9-DoF hand (D'Claw) (Zhu et al. 2019) to turn a 1-DoF valve to arbitrary target locations. Here, the fingers must come into contact with the object and coordinate themselves to make continued progress toward the target.

[2]https://sites.google.com/view/pddm

In-hand Reorientation: Next, we look into the manipulation of free-floating objects, where most maneuvers lead to task failures of dropping objects, and thus sharp discontinuities in the dynamics. Successful manipulation of free-floating objects, such as this task of reorienting a cube into a goal pose, requires careful stabilization strategies as well as re-grasping strategies. Note that these challenges exist even in cases where system dynamics are fully known, let alone with learned models.

Handwriting: In addition to free-floating objects, the requirement of precision is a further challenge when planning with approximate models. Handwriting, in particular, requires precise maneuvering of all joints in order to control the movement of the pencil tip and enable legible writing. Furthermore, the previously mentioned challenges of local minima (i.e., holding the pencil still), abundant terminal states (i.e., dropping the pencil), and the need for simultaneous coordination of numerous joints still apply. This task can also test flexibility of learned behaviors, by requiring the agent to follow arbitrary writing patterns as opposed to only a specific stroke.

Baoding Balls: While manipulation of one free object is already a challenge, sharing the compact workspace with other objects exacerbates the challenge and truly tests the dexterity of the manipulator. We examine this challenge with Baoding balls, where the goal is to rotate two balls around the palm without dropping them. The objects influence the dynamics of not only the hand, but also each other; inconsistent movement of either one knocks the other out of the hand, leading to failure.

4.4.2 Implementation and Hyperparameter Details

We implement the dynamics model as a neural network of 2 fully-connected hidden layers of size 500 with ReLU nonlinearities and a final fully-connected output layer. For all tasks and environments, we train this same model architecture with a standard mean squared error (MSE) supervised learning loss, using the Adam optimizer (Kingma and J. Ba 2014) with learning rate 0.001. This algorithm iteratively alternates between collecting R rollouts of length T to put into the dataset $\mathcal{D}$, and training the model for E epochs, where each epoch consists of a single pass through the shuffled dataset $\mathcal{D}$ while taking a gradient step for every sampled batch of size 500 data points. These hyperparameters, along with controller parameters (H, K, γ, β) and the number of models in the ensemble M are listed below in Table 4.1, referenced by task. For each of these tasks, we normalize the action spaces to $[-1, 1]$ and act at a control frequency of $\frac{1}{dt}$. Reward functions and other relevant task details are listed in Table 4.2.

4.4.3 Ablations and Analysis of Design Decisions

In the first set of experiments (Fig. 4.5), we evaluate the impact of the design decisions for the model and the online planning method. We use the Baoding balls task for these

Table 4.1: Hyperparameters

	R	T	H	K	γ	β	M	E
Valve Turning	20	40	7	200	10	0.6	3	40
In-hand Reorientation	40	70	7	700	50	0.7	3	40
Handwriting	40	40	7	700	0.5	0.5	3	40
Baoding Balls	30	100	7	700	20	0.7	3	40

Table 4.2: Task details

	dt (sec)	Dim of $\mathbf{s}$	Dim of $\mathbf{a}$	Reward $r(\mathbf{s}, \mathbf{a})$
Valve Turning	0.15	21	9	$-10\lvert\text{valve}_\theta - \text{target}_\theta\rvert$ $+\mathbb{1}(\lvert\text{valve}_\theta - \text{target}_\theta\rvert < 0.25)$ $+10 * \mathbb{1}(\lvert\text{valve}_\theta - \text{target}_\theta\rvert < 0.1)$
In-hand Reorientation	0.1	46	24	$-7\lvert\lvert\text{cube}_\text{rpy} - \text{target}_\text{rpy}\rvert\rvert$ $-1000\mathbb{1}(\text{isdrop})$
Handwriting	0.1	48	24	$-100\lvert\lvert\text{tip}_\text{xy} - \text{target}_\text{xy}\rvert\rvert$ $-20\lvert\lvert\text{tip}_\text{z}\rvert\rvert$ $-10\mathbb{1}(\text{forwardtipping} > 0)$
Baoding Balls	0.1	40	24	$-5\lvert\lvert\text{objects}_\text{xyz} - \text{targets}_\text{xyz}\rvert\rvert$ $-500\mathbb{1}(\text{isdrop})$

experiments, though we observed similar trends on other tasks. In the first plot, we see that a sufficiently large architecture is crucial, indicating that the model must have enough capacity to represent the complex dynamical system. In the second plot, we see that the use of ensembles is helpful, especially earlier in training when non-ensembled models can overfit badly and thus exhibit overconfident and harmful behavior. This suggests that ensembles are an enabling factor in using sufficiently high-capacity models. In the third plot, we see that there is not much difference between resetting model weights randomly at each training iteration versus warmstarting them from their previous values.

In the fourth plot, we see that using a planning horizon that is either too long or too short can be detrimental: Short horizons lead to greedy planning, while long horizons suffer from compounding errors in the predictions. In the fifth plot, we study the type of planning algorithm and see that PDDM, with action smoothing and soft updates, greatly

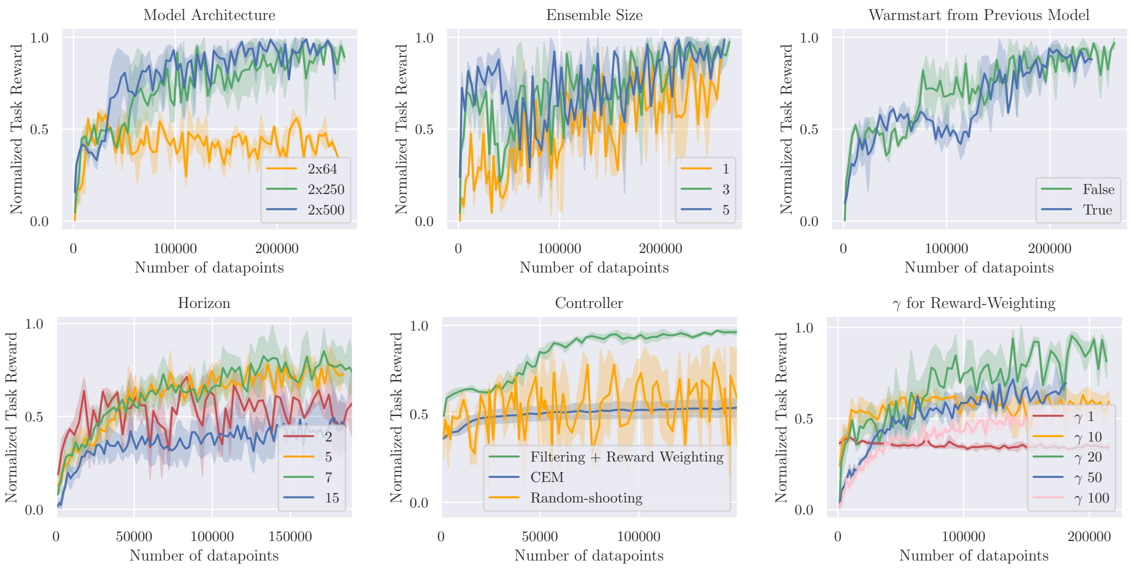


Figure 4.5: Baoding task performance (in simulation) for various design decisions: (Top) model architecture, ensemble size, warmstarting model weights, and (Bottom) planning horizon, controller type, and reward-weighting γ.

outperforms the others. In the final plot, we study the effect of the γ reward-weighting variable, showing that medium values provide the best balance of dimensionality reduction and smooth integration of action samples versus loss of control authority. Here, too soft of a weighting leads to minimal movement of the hand, and too hard of a weighting leads to aggressive behaviors that frequently drop the objects.

4.4.4 Comparisons

In this section, we compare our method to the following state-of-the-art model-based and model-free RL algorithms: **Nagabandi et. al** (Nagabandi, Kahn, Ronald S. Fearing, et al. 2018) is the method introduced in the previous two chapters, which learns a deterministic neural network model combined with a random shooting MPC controller; **PETS** (Kurutach et al. 2018a) combines uncertainty-aware deep network dynamics models with sampling-based uncertainty propagation; **NPG** (Kakade 2002) is a model-free natural policy gradient method, and has been used in prior work on learning manipulation skills (Rajeswaran et al. 2017); **SAC** (Haarnoja, Zhou, Abbeel, et al. 2018) is an off-policy model-free RL algorithm; **MBPO** (Janner et al. 2019) is a recent hybrid approach that uses data from its model to accelerate policy learning. On our suite of dexterous manipulation tasks, PDDM consistently outperforms prior methods both in terms of learning speed and final performance, even solving tasks that prior methods cannot.

Valve turning: We first experiment with a three-fingered hand rotating a valve, with starting and goal positions chosen randomly from the range $[-\pi, \pi]$. On this simpler task, we confirm that most of the prior methods do in fact succeed. We also see that even on this simpler task, policy gradient approaches such as NPG require prohibitively large amounts of data (note the log scale of Figure 4.6).

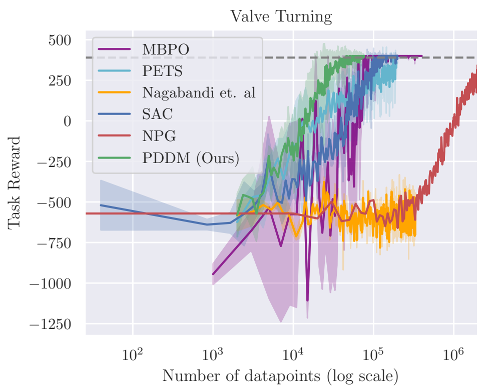


Figure 4.6: Most methods learn this easier task of valve turning, though our method learns fastest.

In-hand reorientation: Next, we scale up our method to a 24-DoF five-fingered hand reorienting a free-floating object to arbitrary goal configurations (Figure 4.7). First, we prescribe two possible goals of either left or right 90° rotations; here, SAC behaves similarly to our method (actually attaining a higher final reward), and NPG is slow as expected, but does achieve the same high reward after 6×10^6 steps. However, when we increase the number of possible goals to 8 different options (90° and 45° rotations in the left, right, up, and down directions), we see that our method still succeeds, but the model-free approaches get stuck in local optima and are unable to fully achieve even the previously attainable goals. This inability to effectively address a "multi-task" or "multi-goal" setup is indeed a known drawback for model-free approaches (and people have been working on this area), and it is particularly pronounced in such goal-conditioned tasks that require flexibility. These additional goals do not make the task harder for PDDM, due to the decoupling of task and dynamics; even in learning 90° rotations, PDDM is building a model of its interactions rather than specifically learning to get to those angles.

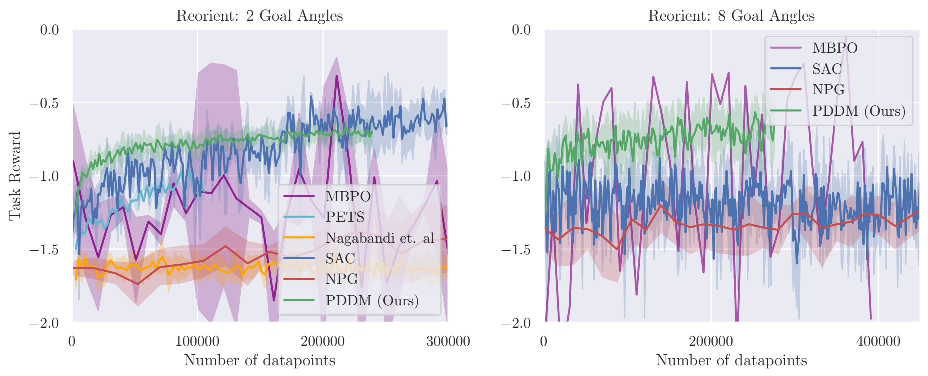


Figure 4.7: In-hand reorientation of a cube. Our method achieves the best results for both (**top**) 2 and (**bottom**) 8 goal angles, and model-free algorithms and methods that directly learn a policy, such as MBPO, struggle with 8 goal angles.

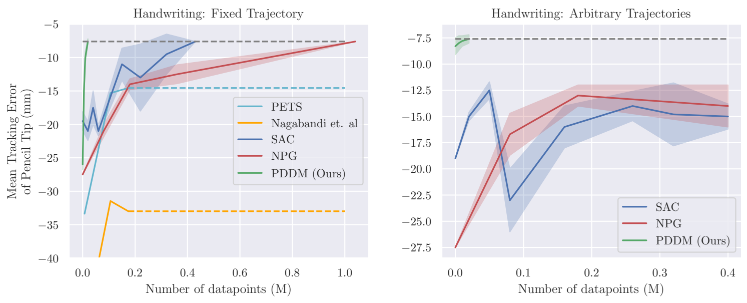


Figure 4.8: **Left**: Manipulating a pencil to make its tip follow a fixed desired path, where PDDM learns substantially faster than SAC and NPG. **Right**: Manipulating a pencil to make its tip follow *arbitrary* paths, where only PDDM succeeds. Note that due to the decoupling of dynamics from task, PDDM requires a similar amount of training data in both of these scenarios.

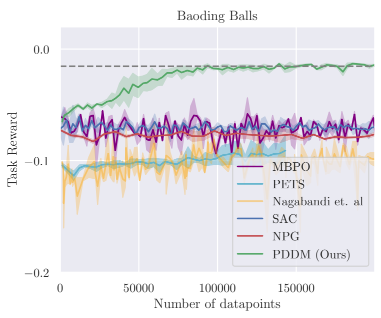


Figure 4.9: PDDM outperforms prior model-based and model-free methods on the simulated Baoding balls task.

Handwriting: To further scale up task complexity, we experiment with handwriting, where the base of the hand is fixed and all writing must be done through coordinated movement of the fingers and the wrist. We perform two variations of this task: (a) the agent is trained and tested on writing a single fixed trajectory, and (b) the agent is trained with the goal of following arbitrary trajectories, but is evaluated on the fixed trajectory from (a). Although even a fixed writing trajectory is challenging, writing arbitrary trajectories requires a degree of flexibility that is exceptionally difficult for prior methods. We see in Figure 4.8 that prior model-based approaches don't actually solve this task (values below the grey line correspond to holding the pencil still near the middle of the paper). Our method, SAC, and NPG solve the task for a single fixed trajectory, but the model-free methods fail when presented with arbitrary trajectories and become stuck in a local optima when trying to write arbitrary trajectories. Even SAC, which has a higher entropy action distribution and therefore achieves better exploration, is unable to extract the finer underlying skill due to the landscape for successful behaviors being quite narrow.

Baoding Balls: This task is particularly challenging due to the inter-object interactions, which can lead to drastically discontinuous dynamics and frequent failures from dropping the objects. We were unable to get the other model-based or model-free methods to succeed at this task (Figure 4.9), but PDDM solves it using just 100,000 data points, or 2.7 hours worth of data. Additionally, we can employ the model that was trained on 100-step rollouts to then run for much longer (1000 steps) at test time. The model learned for this task can also be repurposed, without additional training, to perform a variety of related tasks (see video[3]): moving a single ball to a goal location in the hand, posing the hand, and performing clockwise rotations instead of the learned counter-clockwise ones.

[3]`https://sites.google.com/view/pddm`

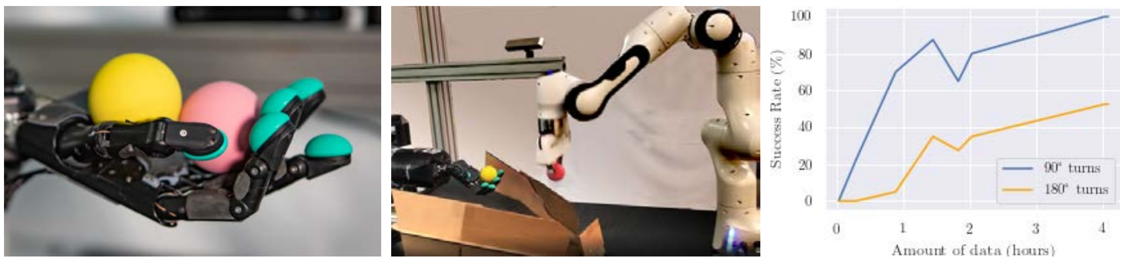


Figure 4.10: Real-world Baoding balls hardware setup with the ShadowHand (left), Franka-Emika arm used for the automated reset mechanism (middle), and the resulting success rates for 90° and 180° turns (right).

4.4.5 Learning Real-World Manipulation of Baoding Balls

Finally, we present an evaluation of PDDM on a real-world anthropomorphic robotic hand. We use the 24-DoF Shadow Hand to manipulate Baoding balls, and we train our method entirely with real-world experience, without any simulation or prior knowledge of the system.

Hardware setup: In order to run this experiment in the real world (Figure 4.10), we use a camera tracker to produce 3D position estimates for the Baoding balls. The camera tracker serves the purpose of providing low latency, robust, and accurate 3D position estimates of the Baoding balls. To enable this tracking, we employ a dilated CNN modeled after the one in KeypointNet (Suwajanakorn et al. 2018). The input to the system is a 280x180 RGB stereo pair (no explicit depth) from a calibrated 12 cm baseline camera rig. The output is a spatial softmax for the 2D location and depth of the center of each sphere in camera frame. Standard pinhole camera equations convert 2D and depth into 3D points in the camera frame, and an additional calibration finally converts it into the ShadowHand's coordinate system. Training of the model is done in sim, with fine-tuning on real-world data. Our semi-automated process of composing static scenes with the spheres, moving the stereo rig, and using VSLAM algorithms to label the images using relative poses of the camera views substantially decreased the amount of hand-labelling that was requiring. We hand-labeled only about 100 images in 25 videos, generating over 10,000 training images. We observe average tracking errors of 5 mm and latency of 20 ms, split evenly between image capture and model inference.

As shown in the supplementary video, we also implement an automated reset mechanism, which consists of a ramp that funnels the dropped Baoding balls to a specific position and then triggers a pre-preprogrammed 7-DoF Franka-Emika arm to use its parallel jaw gripper to pick them up and return them to the Shadow Hand's palm. The planner commands the hand at 10Hz, which is communicated via a 0-order hold to the low-level position controller that operates at 1kHz. The episode terminates if the specific task horizon of 10 seconds has elapsed or if the hand drops either ball, at which point a reset request is issued again. Numerous sources of delays in real robotic systems, in addition to the underactuated nature of the real Shadow Hand, make the task quite challenging in the real world.

Results: After less than 2 hours of real-world training, PDDM is able to learn 90° rotations without dropping the two Baoding balls with a success rate of about 100%, and can achieve a success rate of about 54% on the challenging 180° rotation task, as shown in Figure 4.10. An example trajectory of the robot rotating the Baoding balls using PDDM is shown in Figure 4.2, and videos on the project website [4] illustrate task progress through various stages of training. Qualitatively, we note that performance improves fastest during the first 1.5 hours of training; after this, the system must learn the more complex transition of transferring the control of a Baoding ball from the pinky to the thumb (with a period of time in between, where the hand has only indirect control of the ball through wrist movement). These results illustrate that, although the real-world version of this task is substantially more challenging than its simulated counterpart, our method can learn to perform it with considerable proficiency using a modest amount of real-world training data.

4.5 Discussion

In this chapter, we presented a method for using deep model-based RL to learn dexterous manipulation skills with multi-fingered hands. We demonstrated results on challenging non-prehensile manipulation tasks, including controlling free-floating objects, agile finger gaits for repositioning objects in the hand, and precise control of a pencil to write user-specified strokes. As we showed in our experiments, our method achieves substantially better results than prior deep model-based RL methods, and also demonstrates advantages over model-free RL: it requires substantially less training data and results in a model that can be flexibly reused to perform a wide variety of user-specified tasks. In addition to analyzing the approach on our simulated suite of tasks using 1-2 hours worth of training data, we demonstrated PDDM on a real-world 24 DoF anthropomorphic hand, showing successful in-hand manipulation of objects using just 4 hours worth of entirely real-world interactions.

[4]`https://sites.google.com/view/pddm/`

Chapter 5

Online Model Adaptation via Meta-learning

Although reinforcement learning methods can achieve impressive results in simulation, the real world presents a major challenges in the form of unexpected perturbations or unseen situations that cause proficient but specialized policies to fail at test time. Given that it is impractical to train separate policies to accommodate all situations the agent may see in the real world, the work in this chapter proposes to learn how to quickly and effectively adapt online to new tasks. The work in this chapter extends the capabilities of model-based learning techniques along the axis of usefulness and applicability, by going past the paradigm of autonomous task learning and entering into the paradigm of online adaptation of those learned skills to handle the diversity of variations that an agent is bound to face when deployed in the world.

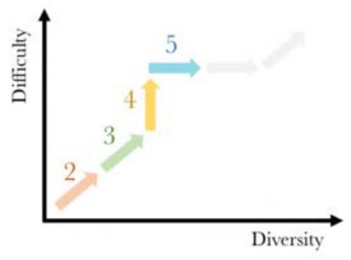


Figure 5.1: Thesis outline, with the current chapter indicated by the last colored arrow.

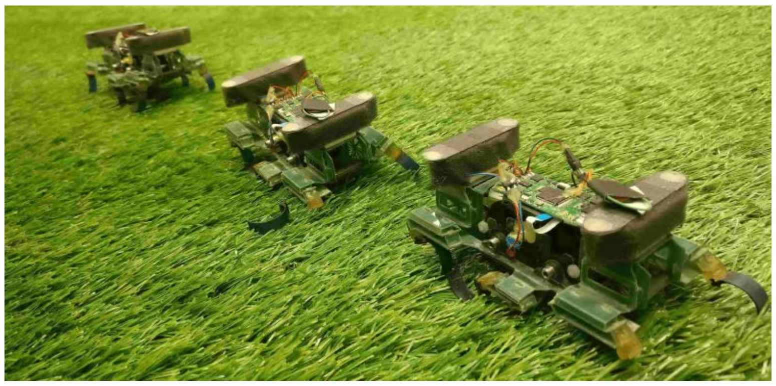


Figure 5.2: We implement our sample-efficient meta-reinforcement learning algorithm on a real legged millirobot, enabling **online** adaptation to new tasks and unexpected occurrences such as losing a leg (shown here), novel terrains and slopes, errors in pose estimation, and pulling payloads.

Our approach uses meta-learning to train a dynamics model prior such that, when combined with a small amount of recent data, this meta-learned prior can be rapidly adapted to the local context. Our experiments demonstrate online adaptation for continuous control tasks on both simulated and real-world agents. We first show simulated agents adapting their behavior online to novel terrains, crippled body parts, and highly-dynamic environments. We also illustrate the importance of incorporating online adaptation into autonomous agents that operate in the real world by applying our method to a real dynamic legged millirobot. We demonstrate the agent's learned ability to quickly adapt online to a missing leg, adjust to novel terrains and slopes, account for miscalibration or errors in pose estimation, and compensate for pulling payloads. Videos of the experiments as well as the code are available online[1].

5.1 Introduction

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-RL algorithm. In this model-based setting, data for updating the model is readily available at every timestep in the form of recent experiences. Crucially, the meta-training process for training such an adaptive model can be much more sample efficient than model-free meta-RL approaches (Duan, John Schulman, X. Chen, Peter L. Bartlett, et al. 2016a; J. X. Wang et al. 2016a; Finn, Abbeel, and Levine 2017c). 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

[1]`https://sites.google.com/berkeley.edu/metaadaptivecontrol`

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 RL: 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; P. Pastor et al. 2011; Meier, Kappler, et al. 2016); but, scaling such methods to complex tasks and nonlinear systems is challenging. Even when working with deep neural networks, which have been used to model complex nonlinear systems (Kurutach et al. 2018b), it is exceptionally difficult to enable fast 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 itself, 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-RL approach that achieves online adaptation in dynamic environments. To the best of knowledge, this is the first meta-RL algorithm to be applied on a real robotic system. Our algorithm efficiently trains a global model that is capable of using its recent experiences to quickly adapt, achieving fast online adaptation in dynamic environments. We evaluate two versions of our approach, a recurrence-based adaptive learner (ReBAL) and a gradient-based adaptive learner (GrBAL), on stochastic and simulated continuous control tasks with complex contact dynamics (Fig. 5.3). 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 of 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 (such as those introduced in the previous chapters), online model-adaptive methods, model-free methods, and prior meta-RL 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 (as introduced in Chapter 3, and also shown in Fig. 5.3). To highlight not only the sample efficiency of our meta model-based RL 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.

5.2 Related Work

Advances in learning control policies have shown success on numerous complex and high dimensional tasks (John Schulman, Levine, et al. 2015; Timothy Lillicrap et al. 2015; Volodymyr Mnih, Koray Kavukcuoglu, Silver, Rusu, et al. 2015; Levine, Finn, et al. 2016; Silver, Schrittwieser, et al. 2017). While RL algorithms provide a framework for learning new tasks, they primarily focus on mastery of individual skills, rather than generalizing and quickly adapting to new scenarios. Furthermore, model-free approaches (Peters and Schaal 2008) require large amounts of system interaction to learn successful control policies, which often makes them impractical for real-world systems. In contrast, model-based methods attain superior sample efficiency by first learning a model of system dynamics, and then using that model to optimize a policy (M. P. Deisenroth, Neumann, Peters, et al. 2013; Lenz, Knepper, and Saxena 2015; Levine, Finn, et al. 2016; Nagabandi, Yang, et al. 2017; Williams, Wagener, et al. 2017). Our approach alleviates the need to learn a single global model by allowing the model to be adapted automatically to different scenarios online based on recent observations. A key challenge with model-based RL approaches is the difficulty of learning a global model that is accurate for the entire state space. Prior model-based approaches tackled this problem by incorporating model uncertainty using Gaussian Processes (GPs) (Ko and Fox 2009; Marc Deisenroth and Carl E Rasmussen 2011; Doerr et al. 2017). However, these methods make additional assumptions on the system (such as smoothness), and do not scale to high dimensional environments. (Chua et al. 2018b) has recently showed that neural networks models can also benefit from incorporating uncertainty, and it can lead to model-based methods that attain model-free performance with a significant reduction on sample complexity. The work in this chapter is orthogonal to theirs, and can benefit from incorporating such uncertainty.

Prior online adaptation approaches (Tanaskovic et al. 2013; Aswani, Bouffard, and Tomlin 2012) have aimed to learn an approximate global model and then adapt it at test time. Dynamic evaluation algorithms (Rei 2015; Krause, Kahembwe, et al. 2017; Krause, Lu, et al. 2016; Fortunato, Blundell, and Vinyals 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 RL and adaptive control (Sastry and Isidori 1989). In contrast from inverse model adaptation (Kelouwani et al. 2012; Underwood and Husain 2010; P. Pastor et al. 2011; Meier, Kappler, 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; Shixiang Gu, Timothy Lillicrap, Sutskever, et al. 2016; Fu, Levine, and Abbeel 2015; 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 work, 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 1998b; Juergen Schmidhuber and Huber 1991; Naik and Mammone 1992b; Lake, Salakhutdinov, and Tenenbaum 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 (Y. Bengio, S. Bengio, and Cloutier 1990; Jürgen Schmidhuber 1992; Younger, Hochreiter, and Conwell 2001; Andrychowicz, Denil, et al. 2016; K. Li and J. 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. 2016b; Munkhdalai and H. Yu 2017b; Munkhdalai, Yuan, et al. 2017; Mishra, Rohaninejad, et al. 2017a) 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, Abbeel, and Levine 2017c). 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.

Meta-learning in the context of RL has largely focused on model-free approaches (Duan, John Schulman, X. Chen, Peter L. Bartlett, et al. 2016a; J. X. Wang et al. 2016a; Sung et al. 2017; Al-Shedivat et al. 2017a). 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, Hofmann, and M. P. Deisenroth 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 and real-world robots.

5.3 Preliminaries

In this section, we present model-based RL, introduce the meta-learning formulation, and describe two main meta-learning approaches.

5.3.1 Model-Based Reinforcement Learning

As introduced in the previous chapters, RL agents aim to perform actions that maximize some notion of cumulative reward. Concretely, consider a Markov decision process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma, \rho_0, H)$. Here, $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$

is the state transition distribution, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a bounded reward function, $\rho_0 : \mathcal{S} \to \mathbb{R}_+$ is the initial state distribution, γ is the discount factor, and H is the horizon. A trajectory segment is denoted by $\tau(i, j) := (\mathbf{s}_i, \mathbf{a}_i, ..., \mathbf{s}_j, \mathbf{a}_j, \mathbf{s}_{j+1})$. Finally, the sum of discounted rewards from a trajectory is the return. In this framework, RL aims to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that prescribes the optimal action to take from each state in order to maximize the expected return.

Model-based RL aims to solve this problem by learning the transition distribution $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$, which is also referred to as the dynamics model. This can be done using a function approximator $\hat{p}_\theta(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ to approximate the dynamics, where the weights θ are optimized to maximize the log-likelihood of the observed data $\mathcal{D}$. In practice, this model can then be used in the process of action selection by either producing data from which to train a policy, or by producing predictions and dynamics constraints to be optimized by a controller.

5.3.2 Meta-Learning

Meta-learning is concerned with automatically learning learning algorithms that are more efficient and effective than learning each task 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 $\rho(\mathcal{T})$ and share a common structure that can be exploited for fast learning. In the supervised learning setting, we aim to learn a function f_θ with parameters θ that minimizes a supervised learning loss $\mathcal{L}$. Then, the goal of meta-learning is to find a learning procedure, denoted as $\theta' = u_\psi(\mathcal{D}^{\text{tr}}_{\mathcal{T}}, \theta)$, that can use just small amounts of data $\mathcal{D}^{\text{tr}}_{\mathcal{T}}$ from each task $\mathcal{T}$ to generate updated or adapated parameters θ' such that the objective $\mathcal{L}$ is well optimized on other data $\mathcal{D}^{\text{test}}_{\mathcal{T}}$ from that same task.

We can formalize this meta-learning problem setting as optimizing for the parameters of the learning procedure θ, ψ as follows:

$$\min_{\theta,\psi} \ \mathbb{E}_{\mathcal{T}\sim\rho(\mathcal{T})}\big[\mathcal{L}(\mathcal{D}^{\text{test}}_{\mathcal{T}}, \theta')\big] \quad \text{s.t.} \quad \theta' = u_\psi(\mathcal{D}^{\text{tr}}_{\mathcal{T}}, \theta) \tag{5.1}$$

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

Once meta-training optimizes for the parameters θ_*, ψ_*, the learning procedure $u_\psi(\cdot, \theta)$ can then be used to learn new held-out tasks from small amounts of data. We will also refer to this learned learning procedure u as the update function.

Gradient-based meta-learning. Model-agnostic meta-learning (MAML) (Finn, Abbeel, and Levine 2017c) 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, MAML prescribed the learning algorithm above to be gradient descent:

$$u_\psi(\mathcal{D}^{\text{tr}}_{\mathcal{T}}, \theta) = \theta - \alpha \nabla_\theta \mathcal{L}(\mathcal{D}^{\text{tr}}_{\mathcal{T}}, \theta) \tag{5.2}$$

The learning rate α may be a learnable parameter (in which case $\psi = \alpha$) or fixed as a hyperparameter, leading to $\psi = \varnothing$. Despite the update rule being fixed, a meta-learned initialization θ 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 θ 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, Abbeel, and Levine 2017c; Duan, John Schulman, X. Chen, Peter L. Bartlett, et al. 2016a). We will build upon these ideas and instantiate them in the model-based setting, to develop a meta model-based RL algorithm that not only is sample-efficient in the meta-training process, but also enables effective online adaptation in dynamic environments.

5.4 Meta-Learning for Online Model Adaptation

In this section, we present our approach for meta-learning for online model adaptation. As explained in Section 5.3.2, standard meta-learning formulations require the learned model θ_*, ψ_* to learn using only $\mathcal{D}^{\text{tr}}_{\mathcal{T}}$ from some new "task". We will define this data to consist of M data points. In prior gradient-based and model-based meta-RL approaches (Finn, Abbeel, and Levine 2017c; Sæmundsson, Hofmann, and M. P. Deisenroth 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 M *timesteps* (rather than the past M 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 M past time steps to predict the next K timesteps. In this setting, M and K are pre-specified hyperparameters.

In this work, we use the notion of environment $\mathcal{E}$ 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 $\rho(\mathcal{E})$ that share some common structure, such as the same observation and action space, but may differ in their dynamics $p_{\mathcal{E}}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$. We denote a trajectory segment by $\tau_{\mathcal{E}}(i, j)$, 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 $\mathcal{E}$. Our algorithm assumes that the environment is locally consistent, in that every segment of length $j - i$ is from 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 (θ, ψ) with respect to a maximum likelihood meta-objective. The meta-objective is the likelihood of the data under a predictive model $\hat{p}_{\theta'}(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ with parameters θ', where $\theta' = u_\psi(\tau_\mathcal{E}(t-M, t-1), \theta)$ corresponds to model parameters that were updated using the past M data points. Concretely, this corresponds to the following optimization:

$$\min_{\theta,\psi} \ \mathbb{E}_{\tau_\mathcal{E}(t-M,t+K)\sim\mathcal{D}}\big[\mathcal{L}(\tau_\mathcal{E}(t, t+K), \theta'_\mathcal{E})\big] \quad \text{s.t.:} \quad \theta'_\mathcal{E} = u_\psi(\tau_\mathcal{E}(t-M, t-1), \theta), \tag{5.3}$$

where the $\tau_\mathcal{E}(t-M, t+K) \sim \mathcal{D}$ corresponds to trajectory segments sampled from the replay buffer, and the loss $\mathcal{L}$ corresponds to the negative log likelihood of the data under the model:

$$\mathcal{L}(\tau_\mathcal{E}(t, t+K), \theta'_\mathcal{E}) \triangleq -\frac{1}{K}\sum_{k=t}^{t+K} \log \hat{p}_{\theta'_\mathcal{E}}(\mathbf{s}_{k+1}|\mathbf{s}_k, \mathbf{a}_k). \tag{5.4}$$

In the meta-objective in Equation 5.3, note that the past M points are used to adapt θ into θ', and the loss of this θ' is evaluated on the future K points. Thus, at every time step t, we use the past M timesteps to provide insight into how to adapt our model such that it performs well for timesteps in the near future. As outlined in Algorithm 4 below, the update rule u_ψ for the inner update and a gradient step on θ for the outer update allow us to optimize this meta-objective of adaptation. By training with this objective during meta-train time, we learn the ability to fine-tune the model using just M data points and thus achieve fast online adaptation. Note that, in our experiments, we compare this approach to using the same M data points to adapt a model that was *not* meta-learned, and we see that this meta-training objective is indeed necessary to enable fast adaptation at test time. We instantiate two versions of our algorithm below, using a recurrence-based meta-learner and a gradient-based meta-learner.

(1) Gradient-Based Adaptive Learner (GrBAL). GrBAL uses a gradient-based meta-learner to perform online adaptation; in particular, we use MAML (Finn, Abbeel, and Levine 2017c). In this case, our update rule is prescribed by gradient descent (5.5).

$$\theta'_\mathcal{E} = u_\psi(\tau_\mathcal{E}(t-M, t-1), \theta) = \theta + \psi \nabla_\theta \frac{1}{M} \sum_{m=t-M}^{t-1} \log \hat{p}_\theta(\mathbf{s}_{m+1}|\mathbf{s}_m, \mathbf{a}_m) \tag{5.5}$$

(2) 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, u_ψ corresponds to the weights of the recurrent model that update its hidden state.

Algorithm 4 Model-Based Meta-RL (train time)

```
Require: Distribution $\rho_{\mathcal{E}}$ 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 $\theta$
 2: for $i = 1, \ldots$ do
 3:     if $i \bmod n_S = 0$ then
 4:         Sample $\mathcal{E} \sim \rho(\mathcal{E})$
 5:         Collect $\tau_{\mathcal{E}}$ using Alg. 5
 6:         $\mathcal{D} \leftarrow \mathcal{D} \cup \{\tau_{\mathcal{E}}\}$
 7:     end if
 8:     for $j = 1 \ldots N$ do
 9:         $\tau_{\mathcal{E}}(t-M, t-1), \tau_{\mathcal{E}}(t, t+K) \sim \mathcal{D}$
10:         $\theta'_{\mathcal{E}} \leftarrow u_{\psi}(\tau_{\mathcal{E}}(t-M, t-1), \theta)$
11:         $\mathcal{L}_j \leftarrow \mathcal{L}(\tau_{\mathcal{E}}(t, t+K), \theta'_{\mathcal{E}})$
12:     end for
13:     $\theta \leftarrow \theta - \beta \nabla_{\theta} \frac{1}{N} \sum_{j=1}^{N} \mathcal{L}_j$
14:     $\psi \leftarrow \psi - \eta \nabla_{\psi} \frac{1}{N} \sum_{j=1}^{N} \mathcal{L}_j$
15: end for
16: Return $(\theta, \psi)$ as $(\theta_*, \psi_*)$
```

Algorithm 5 Online Model Adaptation (test time)

```
Require: Meta-learned params $\theta_*, \psi_*$
Require: controller(), $H$, $r$, $n_A$, $M$
1: $\tau \leftarrow \emptyset$
2: for each timestep $t$ do
3:     $\theta'_* \leftarrow u_{\psi_*}(\tau(t-M, t-1), \theta_*)$
4:     $\mathbf{a_t} \leftarrow$ controller$(\theta'_*, r, H, n_A)$
5:     Execute $\mathbf{a_t}$, add result to $\tau$
6: end for
7: Return rollout $\tau$
```

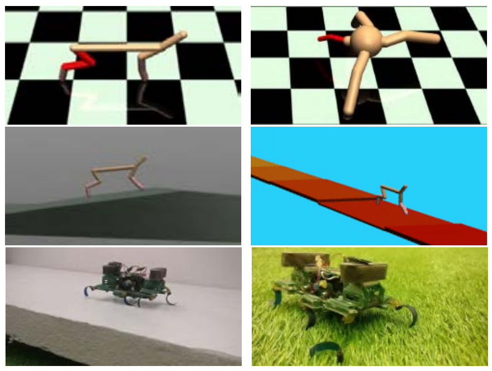


Figure 5.3: Two real-world and four simulated environments on which our method is evaluated and adaptation is crucial for success (e.g., adapting to different slopes and leg failures)

5.5 Model-Based Meta-Reinforcement Learning

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 θ_* and ψ_* from optimizing the meta-learning objective.

Given θ_* and ψ_*, we use the agent's recent experience to adapt the model parameters: $\theta'_* = u_{\psi_*}(\tau(t-M, t-1), \theta_*)$. 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 K, since the adapted model is only valid within the current context. We use model predictive path integral control (MPPI) (Williams, Aldrich, and Theodorou 2015) as the choice of MPC controller for performing action selection, but, in principle, our model adaptation approach is agnostic to the model predictive control

(MPC) method used.

As discussed in the previous chapters, 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 $\hat{p}_{\theta'_*}$ itself will also improve by the next time step. After taking each step, we add the resulting state transition into our dataset, reset the model parameters back to the meta-learned θ_*, and repeat the entire planning process for each timestep. See Algorithm 5 for this adaptation procedure. Finally, in addition to performing this model adaptation at each time step during 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 4.

5.6 Simulated Experimental Results of Online Adaptation via Meta-learning

Before testing our approach on a real robot in the next section, we first perform extensive evaluation in simulation to answer the following questions:

1. Is adaptation actually changing the model?
2. Does our approach enable fast adaptation to varying dynamics, tasks, and environments, both inside and outside of the training distribution?
3. How does our method's performance compare to that of other methods?
4. How do GrBAL and ReBAL compare?
5. How does meta model-based RL compare to meta model-free RL in terms of meta-training sample efficiency and meta-test performance for these experiments?

Videos of these experiments as well as the code are available online[2].

We first conduct a comparative evaluation of our algorithm on a variety of simulated robots using the MuJoCo physics engine (Emanuel Todorov, Erez, and Tassa 2012a). 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. 5.3), where each agent requires different types of adaptation to succeed at run-time, as described below.

[2]`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: (a) disabling a joint unseen during training, to test generalization in the face of a new environment (referred to as "HC Dis. Gen"), and (b) switching between disabled joints during a rollout, to test fast adaptation to changing dynamics (referred to as "HC Dis. F.A.").

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 an unseen setting of a steep hill that first goes up and then down (referred to as "HC Hill").

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 frequent 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 in two different situations: (a) crippling a leg unseen during training, to test generalization in the face of a new environment (referred to as "Ant Crip. Gen."), and (b) switching between normal operation and having a crippled leg within a rollout, to test fast adaptation to changing dynamics (referred to "Ant Crip. F.A.").

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

- **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 (John Schulman, Levine, et al. 2015).

- **Model-free meta-RL (MAML-RL)**: We compare to a state-of-the-art model-free meta-RL method, MAML-RL (Finn, Abbeel, and Levine 2017c).

- **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. This is the model-based approach developed in the previous chapters.

- **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 M observations, akin to dynamic evaluation (Krause, Kahembwe, et al. 2017). This final comparison evaluates the benefit of explicitly training for adaptability (i.e., meta-training).

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

5.6.1 Effect of Adaptation: Pre-update vs. Post-update Model Prediction Errors

First, we analyze the effect of the model adaptation, and show results from test-time runs on various environments. Figures 5.4 and 5.5 display the distribution shift between the pre-update and post-update model prediction errors of three GrBAL runs. Across all tasks and environments, the post-updated model $\hat{p}_{\boldsymbol{\theta}'_*}$ achieves lower prediction error than the pre-updated model $\hat{p}_{\boldsymbol{\theta}_*}$. This shows that using the past M timesteps to update θ_* (pre) into θ'_* (post) does indeed reduce the model prediction error when querying the model for predictions of the future K timesteps.

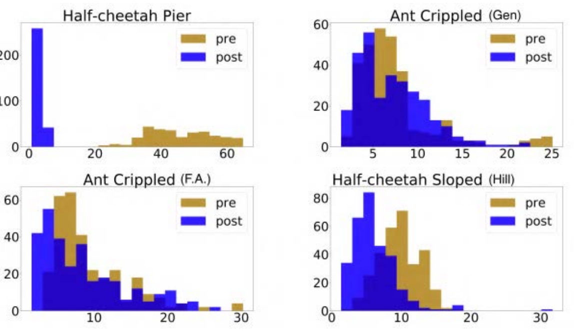


Figure 5.4: Histogram of the K step normalized model prediction error across different tasks. GrBAL achieves lower model error when using the post-update parameters as given by the update rule.

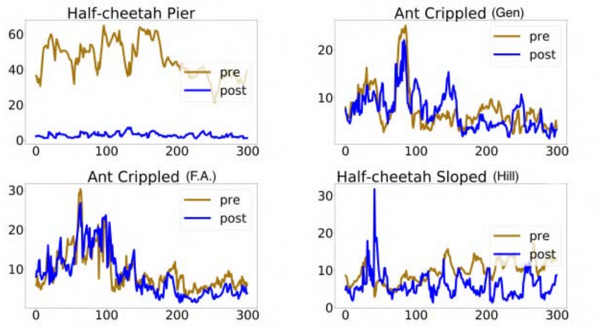


Figure 5.5: An alternate visualization of the information from the previous figure. Here, at each timestep, we show the K step normalized model prediction error across different tasks. GrBAL achieves lower model error when using the post-update parameters as given by the update rule.

5.6.2 Test Performance and Meta-training Sample Efficiency

We first study the sample efficiency of the meta-training process. Figure 5.6 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× 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× 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 (Marc Deisenroth and Carl E Rasmussen 2011; Nagabandi, Kahn, Ronald S Fearing, et al. 2017; Kurutach et al. 2018b); 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.

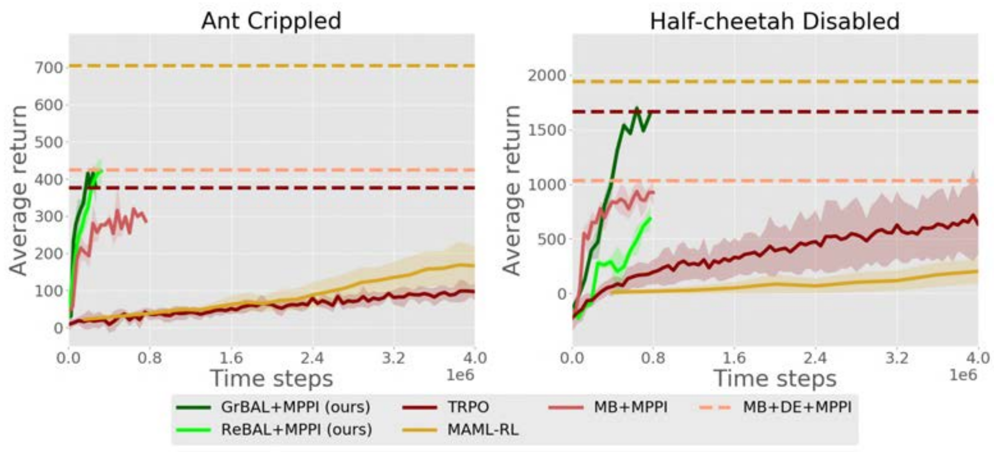


Figure 5.6: Compared to model-free RL, model-free meta-RL, and model-based RL methods, our model-based meta-RL methods achieve good performance with 1000× less data. Dotted lines indicate performance at convergence. For MB+DE+MPPI, we perform dynamic evaluation at test time on the final MB+MPPI model.

5.6.3 Test-time Performance: Online Adaptation & Generalization

In our second comparative evaluation, we evaluate final test time performance for 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 for (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 *new* (i.e., 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.) capability 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 (Gen.) 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. 5.7, show returns that are normalized such that the MB oracle achieves a return of 1.

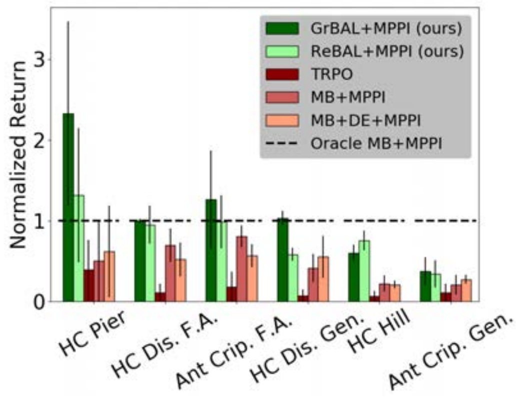


Figure 5.7: Simulated results in a variety of dynamic test environments. GrBAL outperforms other methods, even the MB oracle which is specialized to that particular test environment, in all experiments where fast adaptation is necessary. These results highlight the difficulty of training a global model, 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 all 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.

5.6.4 Effect of Meta-Training Distribution

To see how training distribution affects test performance, we ran an experiment that used GrBAL to train models of a 7-DOF robotic 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 Fig. 5.8 that

1. Seeing a larger range of tasks during training is helpful during testing – a model that saw a large range of force perturbations during training performed the best.

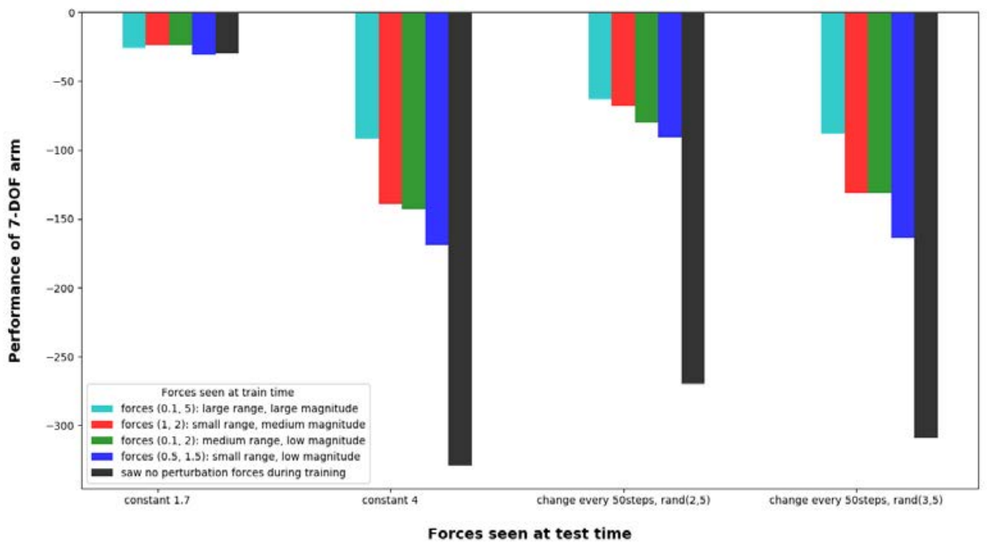


Figure 5.8: Effect of the meta-training distribution on test performance

2. A model that saw no perturbation forces (e.g., trained on a single task) during training did the worst at test time.

3. The middle 3 training ranges 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).

5.6.5 Sensitivity of K and M

In this section we analyze how sensitive is our algorithm w.r.t the hyperparameters K and M. In all experiments, we set K equal to M. Figure 5.9 shows the average return of GrBAL across meta-training iterations of our algorithm for different values of $K = M$. 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 (e.g., 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).

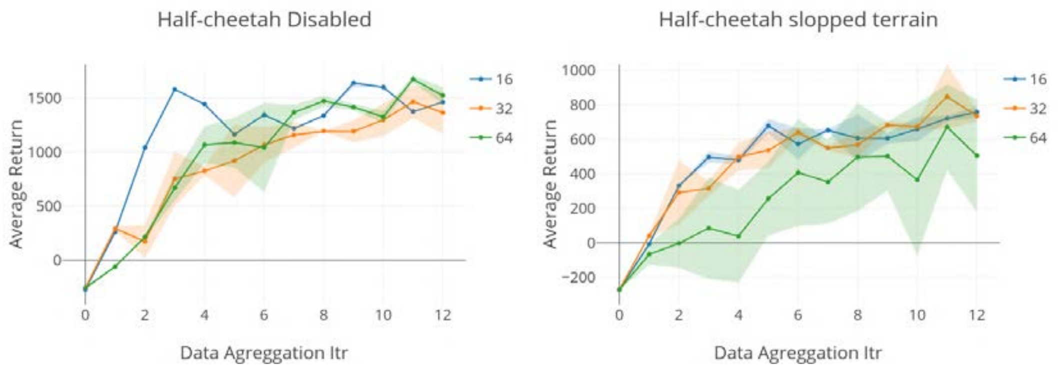


Figure 5.9: Learning curves, for different values of $K = M$, of GrBAL in the half-cheetah disabled and sloped terrain environments. curves suggest that GrBAL performance is fairly robust to the values of these hyperparameters.

5.6.6 Reward functions

For each MuJoCo agent, the same reward function is used in these experiments, across all tasks. Table 5.1 shows the reward functions used for each agent. In this table, x_t refers to 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 $\boldsymbol{g}$ corresponds to the position of the desired goal.

Table 5.1: Reward functions

	Reward function
Half-cheetah	$\frac{x_{t+1}-x_t}{0.01} - 0.05\|\boldsymbol{a}_t\|_2^2$
Ant	$\frac{x_{t+1}-x_t}{0.02} - 0.005\|\boldsymbol{a}_t\|_2^2 + 0.05$
7-DoF Arm	$-\|\boldsymbol{ee}_t - \boldsymbol{g}\|_2^2$

5.6.7 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 in the table below corresponds to the Kullback–Leibler divergence constraint. # Task/itr corresponds to the number of tasks sampled while collecting new data, whereas # TS/itr is the total number of times steps collected (across all tasks). T refers to the number of steps in each rollout, n_A is the number of actions sampled for each controller step of performing action selection, H is the planning horizon for the MPC planner, and $K = M$ is the number of past steps used for adapting the model as well as the number of future steps on which to validate the adapted model's prediction loss.

Table 5.2: Hyperparameters for the half-cheetah 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	0.01	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 5.3: Hyperparameters for the ant 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	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 5.4: 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	-	-	-	-

5.7 Real-world Results of Online Adaptation via Meta-learning

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 used in Chapter 3 and shown in Figures 7.3 and 5.3, 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 for each manufactured robot, its linkages and other body parts deteriorate over time, and it moves very quickly and dynamically with bounding-style gaits; hence, its dynamics are strongly dependent on the terrain or task at hand.

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 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. 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. All experiments were conducted in a motion capture room to get center of mass position and velocity of the robot.

We meta-train a dynamics model for this robot using the meta-objective described in Equation 5.3, 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 set of results in Table 5.5 show that both (a) a model trained with a standard supervised learning objective (MB, as introduced in the previous chapters) and (b) a GrBAL model (which is trained with a meta-learning objective instead of a standard one) achieve comparable performance for executing desired trajectories on terrains that were *seen during training.*

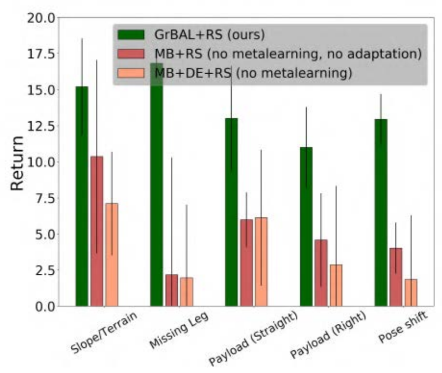


Figure 5.10: GrBAL significantly outperforms both MB and MB+DE, when tested on environments that require online adaptation and/or were never seen during training.

		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 5.5: 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 settings that are indeed seen during training, but including it is not detrimental.

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 in new and changing 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 in Fig. 5.10 demonstrate that GrBAL substantially outperforms MB and MB+DE. Unlike MB and MB+DE, 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 unepxected 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 these

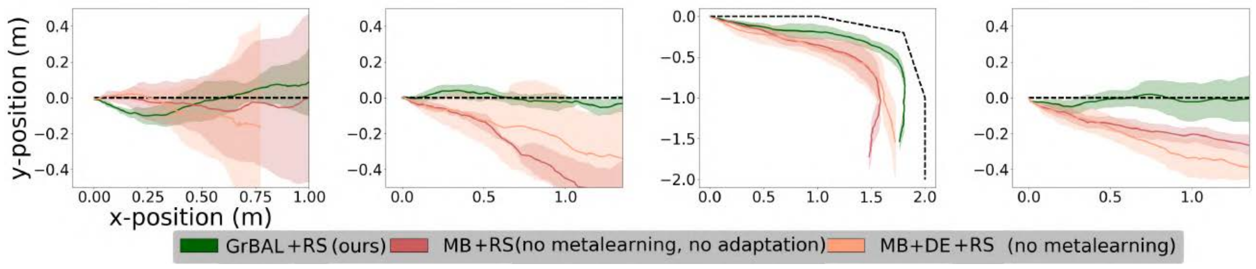


Figure 5.11: The dotted black line indicates the desired trajectory in the xy plane. GrBAL *adapts online* to prevent drift from a missing leg, prevents sliding sideways down a slope, accounts for pose miscalibration errors, and adjusts to pulling payloads (left to right). Note that none of these tasks/environments were seen during training time, and they require fast and effective online adaptation for success.

new environments *online* (i.e., in less than a second). Furthermore, the poor performance of the MB and MB+DE baselines demonstrate not only the need for adaptation, but also the importance of meta-learning to give us good initial parameters to adapt from. The qualitative results of these experiments, shown in Fig. 5.11, show that GrBAL allows the legged robot to adapt online and effectively follow the target trajectories, even in the presence of new environments and unexpected perturbations at test time.

5.8 Discussion

In this chapter, we presented an approach for model-based meta-RL that enabled fast, online adaptation of large and expressive models in dynamic environments. We showed that meta-learning a model for online adaptation resulted 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 provided two instantiations of our approach (ReBAL and GrBAL), and we provided a comparison with other prior methods on a range of continuous control tasks. Finally, we showed 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.

Chapter 6

Continual Model Adaptation via Expectation Maximization

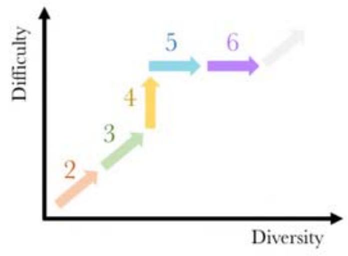


Figure 6.1: Thesis outline, with the current chapter indicated by the last colored arrow.

In the first few chapters, we presented model-based RL methods for performing effective and sample-efficient learning, and we demonstrated results in both simulation and the real world in the areas of locomotion and dexterous manipulation. These models already demonstrated some level of generalization in that, due to the decoupling of the tasks from the dynamics, a single model could be used to perform different tasks such as the handwriting of different trajectories. Although this led to a degree of generalization and flexibility that surpassed being able to only solve a single task, the previous chapter showed that those approaches to model-based RL were not quite sufficient for robots to be viable for deployment in the real world. The previous chapter discussed the inevitable mismatch between an agent's training conditions and the test conditions in which it may actually be deployed, and the need to address that mismatch. From the fact that a learned model will never be perfectly correct, to the fact that the real world has numerous sources of perturbations (e.g., wind, lighting, unexpected payloads, etc.) or unexpected changes (e.g., motor malfunctions, change of terrain, etc.), it becomes clear that online adaptation of our models is critical for enabling systems to operate in more realistic and uncontrolled environments. Inspired by the ability of humans and animals to adapt extremely quickly in the face of unexpected changes, the work in the previous chapter enabled this fast adaptation for deep neural network models, which normally lack this capacity for rapid online adaptation. By designing an effective and efficient meta-learning algorithm, the work in the previous chapter enabled a learned model to adapt *online* to previously unseen situations or changes in the environment.

The work in this chapter further extends those online adaptation capabilities along the axis of usefulness and applicability by developing a more continual adaptation procedure at test time. In this chapter, we present our meta-learning for online learning (MOLe) approach: We formulate an online learning procedure that uses stochastic gradient descent to update model parameters, and an expectation maximization algorithm with a Chinese restaurant process prior to develop and maintain a mixture of models to handle non-stationary task distributions. This improved adaptation procedure allows for all models to be adapted as necessary, with new models instantiated for task changes and old models recalled when previously seen tasks are encountered again. By enabling specialization without sacrificing the generalization that was introduced in the previous chapter, and thus allowing for adaptation to tasks that are further outside of the training distribution, MOLe outperforms prior methods in enabling effective continuous adaptation in non-stationary task distributions on various simulated environments. Videos of the experiments are available online[1].

6.1 Introduction

Human and animal learning is characterized not just by a capacity to acquire complex skills, but also the ability to adapt rapidly when those skills must be carried out under new or changing conditions. For example, animals can quickly adapt to walking and running on different surfaces (Herman 2017) and humans can easily modulate force during reaching movements in the presence of unexpected perturbations (Flanagan and Wing 1993). Furthermore, these experiences are remembered, and can be recalled to adapt more quickly when similar disturbances occur in the future (Doyon and Benali 2005). Since learning entirely new models on such short time-scales is impractical, we can devise algorithms that explicitly train models to adapt quickly from small amounts of data. Such online adaptation is crucial for intelligent systems operating in the real world, where changing factors and unexpected perturbations are the norm. In this chapter, we propose an algorithm for fast and continuous online learning that utilizes deep neural network models to build and maintain a task distribution, allowing for the natural development of both generalization as well as task specialization.

Our working example is continuous adaptation in the model-based reinforcement learning (RL) setting, though our approach generally addresses any online learning scenario with streaming data. We assume that each "trial" consists of multiple tasks, and that the delineation between the tasks is not provided explicitly to the learner – instead, the method must adaptively decide what "tasks" even represent, when to instantiate new tasks, and when to continue updating old ones. For example, a robot running over changing terrain might need to handle uphill and downhill slopes, and might choose to maintain separate models that become specialized to each slope, adapting to each one in turn based on the currently inferred surface.

We perform adaptation simply by using online stochastic gradient descent (SGD) on the model parameters, while maintaining a mixture model over model parameters for different tasks. The mixture is updated via the Chinese restaurant process (Stimberg, Ruttor, and Opper 2012), which enables new tasks to be instantiated as needed over the course of a trial. Although online learning is perhaps one of the oldest applications of SGD (Bottou 1998), modern parametric models such as deep neural networks are exceedingly difficult to train

[1]`https://sites.google.com/berkeley.edu/onlineviameta`

online with this method. They typically require medium-sized minibatches and multiple epochs to arrive at sensible solutions, which is not suitable when receiving data in an online streaming setting. One of our key observations is that meta-learning can be used to learn a prior initialization for the parameters that makes such direct online adaptation feasible, with only a handful of gradient steps. The meta-training procedure we use is based on model-agnostic meta-learning (MAML) (Finn, Abbeel, and Levine 2017a), where a prior weight initialization is learned for a model so as to optimize improvement on any task from a meta-training task distribution after a small number of gradient steps.

Meta-learning with MAML has previously been extended to model-based RL (Nagabandi, Clavera, et al. 2018), but only for the k-shot adaptation setting: The meta-learned prior model is adapted to the k most recent time steps, but the adaptation is not carried forward in time (i.e., adaptation is always performed from the prior itself). Note that this is the method presented in the previous chapter. This rigid batch-mode setting is restrictive in an online learning setup and is insufficient for tasks that are further outside of the training distribution. A more natural formulation is one where the model receives a continuous stream of data and must adapt online to a potentially non-stationary task distribution. This requires both fast adaptation and the ability to recall prior tasks, as well as an effective adaptation strategy to interpolate as needed between the two.

The primary contribution of this work is a meta-learning for online learning (MOLe) algorithm that uses expectation maximization, in conjunction with a Chinese restaurant process prior on the task distribution, to learn mixtures of neural network models that are each updated with online SGD. In contrast to prior multi-task and meta-learning methods, our method's online assignment of soft task probabilities allows for task specialization to emerge naturally, without requiring task delineations to be specified in advance. We evaluate MOLe in the context of model-based RL on a suite of challenging simulated robotic tasks including disturbances, environmental changes, and simulated motor failures. The simulated experiments show a half-cheetah agent and a hexapedal crawler robot performing continuous model adaptation in an online setting. Our results show online instantiation of new tasks, the ability to adapt to out-of-distribution tasks, and the ability to recognize and revert back to prior tasks. Additionally, we demonstrate that MOLe outperforms the state-of-the-art prior method that was introduced in the previous chapter (which does k-shot model-based meta-RL), as well as natural baselines such as continuous gradient updates for adaptation and online learning without meta-training.

6.2 Related Work

Online learning is one of the oldest subfields of machine learning (Bottou 1998; Jafari et al. 2001). Prior algorithms have used online gradient updates (Duchi, Hazan, and Singer 2011) and probabilistic filtering formulations (Murphy 2002; Hoffman, Bach, and Blei 2010; Broderick et al. 2013). In principle, commonly used gradient-based learning methods, such as SGD, can easily be used as online learning algorithms (Bottou 1998). In practice, their performance with deep neural network function approximators is limited (Sahoo et al. 2017): such high-dimensional models must be trained with batch-mode methods, minibatches, and multiple passes over the data. We aim to lift this restriction by using model-agnostic meta-learning (MAML) to explicitly pretrain a model that enables fast adaptation, which we then use for continuous online adaptation via an expectation maximization algorithm with a Chinese restaurant process (Blei, Ng, and Michael I Jordan 2003) prior for dynamic allocation of new tasks in a non-stationary task distribution.

Online learning is related to that of continual or lifelong learning (Thrun 1998), where the agent faces a non-stationary distribution of tasks over time. However, unlike works that focus on avoiding negative transfer, i.e. catastrophic forgetting (Kirkpatrick et al. 2017; Rebuffi, Kolesnikov, and Lampert 2017; Zenke, Poole, and Ganguli 2017; Lopez-Paz et al. 2017; Nguyen et al. 2017), online learning focuses on the ability to rapidly learn and adapt in the presence of non-stationarity. While some continual learning works consider the problem of forward transfer, e.g. (Rusu et al. 2016; Aljundi, Chakravarty, and Tuytelaars 2017; Y.-X. Wang, Ramanan, and Hebert 2017), these works and others in continual learning generally focus on small sets of tasks where fast, online learning is not realistically possible, since there are simply not enough tasks to recover structure that enables fast, few-shot learning in new tasks or environments.

Our approach builds on techniques for meta-learning or learning-to-learn (Thrun and Pratt 1998a; Jurgen Schmidhuber 1987; S. Bengio et al. 1992; Naik and Mammone 1992a). However, most recent meta-learning work considers a setting where one task is learned at a time, often from a single batch of data (Santoro et al. 2016a; Ravi and Larochelle 2017; Munkhdalai and H. Yu 2017a; J. X. Wang et al. 2016b; Duan, John Schulman, X. Chen, Peter L Bartlett, et al. 2016c). In our work, we specifically address non-stationary task distributions and do not assume that task boundaries are known. Prior work (Jerfel et al. 2018) has also considered non-stationary task distributions; whereas (Jerfel et al. 2018) use the meta-gradient to estimate the parameters of a mixture over the task-specific parameters, we focus on fast adaptation and accumulation of task-specific mixture components during run-time optimization. Other meta-learning works have considered non-stationarity within a task (Al-Shedivat et al. 2017b) and episodes involving multiple tasks at meta-test time (Ritter et al. 2018), but they do not consider continual online adaptation with unknown task separation. Prior work has also studied meta-learning for model-based RL (Nagabandi, Clavera, et al. 2018), as presented in the previous chapter. This prior method updates the model every time step, but each update is a batch-mode k-shot update, using exactly k prior transitions and resetting the model at each step. This allows for adaptive control, but does not enable continual online adaptation, since updates from previous steps are always discarded. In our comparisons, we find that our approach substantially outperforms this prior method. To our knowledge, this work is the first to apply meta-learning to learn streaming online updates.

6.3 MOLe: Meta-learning for Online Learning

We formalize the online learning problem setting as follows: at each time step, the model receives an input $\mathbf{x}_t$ and produces a prediction $\hat{\mathbf{y}}_t$. It then receives a ground truth label $\mathbf{y}_t$, which must be used to adapt the model to increase its prediction accuracy on the next input $\mathbf{x}_{t+1}$. The true labels are assumed to come from some task distribution $P(Y_t|X_t, T_t)$, where T_t is the task at time t. The tasks themselves change over time, resulting in a non-stationary task distribution, and the identity of the task T_t is unknown to the learner. In real-world settings, tasks might correspond to unknown parameters of the system (e.g., whether there's a motor malfunction on a robot), unknown underlying user preferences, or other unexpected events. This problem statement covers a range of online learning problems that all require continual adaptation to streaming data and trading off between generalization and specialization.

In our experiments, we use model-based RL as our working example, where the input $\mathbf{x}_t$ is a state-action pair, and the output $\mathbf{y}_t$ is the next state. We discuss this application to model-based RL in Section 6.4, but we keep the following derivation of our method general for the case of arbitrary online prediction problems.

6.3.1 Online Learning with a Mixture of Meta-Trained Networks

We discuss our meta-learning for online learning (MOLe) algorithm in two parts: online learning in this section, and meta-learning in the next. In this section, we explain our online learning method that enables effective online learning using a continuous stream of incoming data from a non-stationary task distribution. We discuss the process of obtaining a meta-learned prior in Sec. 6.3.2, but we first formulate in this section an online adaptation algorithm using SGD with expectation maximization to maintain and adapt a mixture model over task model parameters (i.e., a probabilistic task distribution).

6.3.1.1 Overview

Let $p_{\theta(T_t)}(\mathbf{y}_t|\mathbf{x}_t)$ represent the predictive distribution of the model on input $\mathbf{x}_t$, for an unknown task T_t at time step t. In our mixture model, each option T_i for the task T_t corresponds to its own set of model parameters $\theta(T_i)$. Our goal is to estimate model parameters $\theta_t(T_i)$ for each task T_i in the non-stationary task distribution: This requires inferring the distribution over tasks at each step $P(T_t = T_i|\mathbf{x}_t, \mathbf{y}_t)\ \ \forall T_i$ given some data observations, using that inferred task distribution to make predictions $\hat{\mathbf{y}}_t = p_{\theta(T_t)}(\mathbf{y}_t|\mathbf{x}_t)$, and also using it to update model parameters from $\theta_t(T_i)$ to $\theta_{t+1}(T_i)\ \ \forall T_i$. In practice, the parameters $\theta(T_i)$ of each model will correspond to the weights of a neural network $f_{\theta(T_i)}$.

Each model begins with some prior parameter vector θ^*, which we will discuss in more detail in Section 6.3.2. Since the number of tasks is also unknown, we begin with one task at time step 0, where $|T| = 1$ and thus $\theta_0(T) = \{\theta_0(T_0)\} = \{\theta^*\}$. From here, we continuously update all parameters in $\theta_t(T)$ at each time step (as explained below) and add new tasks as needed, in the attempt to model the true underlying process $P(Y_t|X_t, T_t)$, which we only observe in the form of incoming $\mathbf{x}_t$ and $\mathbf{y}_t$ observations. Since underlying task identities T_t are unknown, we must also estimate this $P(T_t)$ at each time step.

Thus, the online learning problem consists of iteratively inferring task probabilities $P(T_t = T_i|\mathbf{x}_t, \mathbf{y}_t)$, and then using that inferred task distribution to adapt $\theta_t(T_i)\ \ \forall T_i$ at each time step t. The process of inferring the task probabilities is described in further details below, but the model update step under the current inferred task distribution is done by optimizing the expected log-likelihood of the data, given by

$$\mathcal{L} = -E_{T_t \sim P(T_t|\mathbf{x}_t, \mathbf{y}_t)}[\log p_{\theta_t(T_t)}(\mathbf{y}_t|\mathbf{x}_t)], . \tag{6.1}$$

Intuitively, this objective for updating the model seeks model parameters that best explain the observed data, under the current task distribution. Overall, the algorithm iterates between (a) using observed data to infer posterior task probabilities and then (b) using the inferred task probabilities to perform a soft update of model parameters for all tasks. This iterative procedure is detailed below, along with a mechanism for automatically instantiating new tasks as needed.

6.3.1.2 Approximate Online Inference

We use expectation maximization (EM) to update the model parameters. In our case, the E step in EM involves estimating the task distribution $P(T_t = T_i|\mathbf{x}_t, \mathbf{y}_t)$ at the current time step, while the M step involves updating model parameters for all tasks T from $\theta_t(T)$ to obtain the new model parameters $\theta_{t+1}(T)$. The parameters are always updated by one gradient step per time step, according to the inferred task responsibilities.

We first estimate the expectations over all tasks T_i in the task distribution, where the posterior of each task probability can be written as follows:

$$P(T_t = T_i|\mathbf{x}_t, \mathbf{y}_t) \propto p(\mathbf{y}_t|\mathbf{x}_t, T_t = T_i)P(T_t = T_i). \tag{6.2}$$

Here, the likelihood of the data $p(\mathbf{y}_t|\mathbf{x}_t, T_t = T_i)$ is directly given by the model as $p_{\theta_t(T_i)}(\mathbf{y}_t|\mathbf{x}_t)$, and the task prior can be chosen as desired. In this work, we choose to formulate the task prior $P(T_t = T_i)$ using a Chinese restaurant process (CRP). The CRP is an instantiation of a Dirichlet process. In the CRP, at time t, the probability of each task T_i should be given by

$$P(T_t = T_i) = \frac{n_{T_i}}{t - 1 + \alpha} \tag{6.3}$$

where n_{T_i} is the expected number of datapoints in task T_i for all steps $1, \dots, t-1$, and α is a hyperparameter that controls the instantiation of new tasks. The prior therefore becomes

$$P(T_t = T_i) = \frac{\sum_{t'=1}^{t-1} P(T_{t'} = T_i|\mathbf{x}_{t'}, \mathbf{y}_{t'})}{t - 1 + \alpha} \quad \text{and} \quad P(T_t = T_{\text{new}}) = \frac{\alpha}{t - 1 + \alpha}. \tag{6.4}$$

Intuitively, this prior induces a bias that says that tasks seen more often are more likely, and α controls the possibility of a new task. Combining this choice of prior with the likelihood given by the predictive model, we derive the following posterior task probability distribution:

$$P(T_t = T_i|\mathbf{x}_t, \mathbf{y}_t) \propto p_{\theta_t(T_i)}(\mathbf{y}_t|\mathbf{x}_t) \left[\sum_{t'=1}^{t-1} P(T_{t'} = T_i|\mathbf{x}_{t'}, \mathbf{y}_{t'}) + \mathbb{1}(T_{t'} = T_{\text{new}})\alpha\right]. \tag{6.5}$$

Having estimated the latent task probabilities, we next perform the M step, which improves the expected log-likelihood in Equation 6.1 based on the inferred task distribution. Since each task starts at $t = 0$ from the prior θ^*, the values of all parameters after $t + 1$ update steps becomes $\theta_{t+1}(T)$ as follows:

$$\theta_{t+1}(T_i) = \theta^* - \beta \sum_{t'=0}^{t} P_t(T_{t'} = T_i|\mathbf{x}_{t'}, \mathbf{y}_{t'}) \nabla_{\theta_{t'}(T_i)} \log p_{\theta_{t'}(T_i)}(\mathbf{y}_{t'}|\mathbf{x}_{t'}) \quad \forall\ T_i. \tag{6.6}$$

If we assume that all parameters of $\theta_t(T)$ have already been updated for the previous time steps $0, \ldots, t$, we can approximate the last iteration of this update by simply updating all parameters from $\theta_t(T)$ to $\theta_{t+1}(T)$ on the newest data:

$$\theta_{t+1}(T_i) = \theta_t(T_i) - \beta P_t(T_t = T_i|\mathbf{x}_t, \mathbf{y}_t) \nabla_{\theta_t(T_i)} \log p_{\theta_t(T_i)}(\mathbf{y}_t|\mathbf{x}_t) \quad \forall\ T_i. \tag{6.7}$$

This procedure is an approximation, since updates to task parameters $\theta_t(T)$ will in reality also change the corresponding task probabilities at previous time steps. However, this approximation removes the need to store previously seen data points and yields a fully online, streaming algorithm. Finally, to fully implement the EM algorithm, we should alternate the E and M steps to convergence at each time step, rolling back the previous gradient update to $\theta_t(T)$ at each iteration. In practice, we found it sufficient to perform the E and M steps only once per time step. While this is a crude simplification, successive time steps in the online learning scenario are likely to be correlated, making this procedure reasonable. However, it is also straightforward to perform multiple steps of EM while still remaining fully online.

Algorithm 6 Online Learning with Mixture of Meta-Trained Networks

```
Require: $\theta^*$ from meta-training
  Initialize $t = 0$, $\theta_0(T) = \{\theta_0(T_0)\} = \{\theta^*\}$
  for each time step $t$ do
    Calculate $p_{\theta_t(T_i)}(\mathbf{y}_t|\mathbf{x}_t)$  $\forall T_i$
    Calculate $P_t(T_i) = P_t(T_t = T_i|\mathbf{x}_t, \mathbf{y}_t)$  $\forall T_i$
    Calculate $\theta_{t+1}(T_i)$ by adapting from $\theta_t(T_i)$  $\forall T_i$
    Calculate $\theta_{t+1}(T_{\text{new}})$ by adapting from $\theta^*$
    if $P_t(T_{\text{new}}) > P_t(T_i)$  $\forall T_i$ then
      Add $\theta_{t+1}(T_{\text{new}})$ to $\theta_{t+1}(T)$
      Recalculate $P_t(T_i)$ using $\theta_{t+1}(T_i)$  $\forall T_i$
      Recalculate $\theta_{t+1}(T_i)$ using updated $P_t(T_i)$  $\forall T_i$
    end if
    $T^* = \text{argmax}_{T_i}\ \ p_{\theta_{t+1}(T_i)}(\mathbf{y}_t|\mathbf{x}_t)$
    Receive next data point $\mathbf{x}_{t+1}$
  end for
```

We now summarize this full online learning portion of MOLe, outline it in Alg. 6, and illustrate it in Figure 6.2. At the first time step $t = 0$, the task distribution is initialized to contain one entry: $\theta_0(T) = \{\theta_0(T_0)\} = \{\theta^*\}$. At every time step after that, an E step is performed to estimate the posterior task distribution and an M step is performed to update the model parameters corresponding to all tasks, weighted by the likelihood of that task. The CRP prior also assigns, at each time step, the probability of adding a new task T_{new} at the given time step. The parameters $\theta_{t+1}(T_{\text{new}})$ of this new task are adapted from θ^* on the latest data, and this new task is added only if the posterior probability of the new task is greater than that of all currently existing tasks. Recall that this posterior task probability depends on both the likelihood of the observed data under that task, as well as the probability of that task under the CRP prior. After all of these steps, the prediction on the next datapoint is then made using the model parameters $\theta_{t+1}(T^*)$ corresponding to the most likely task T^*, and the entire process repeats.

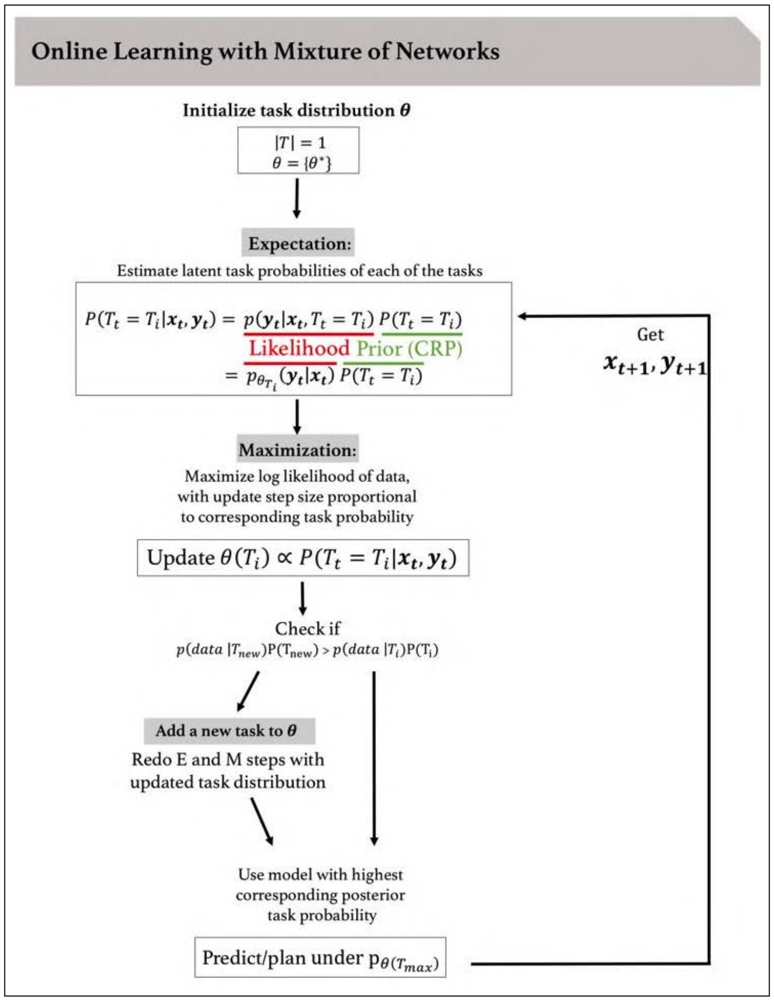

Online Learning with Mixture of Networks
Initialize task distribution θ
$|T| = 1$
$\theta = \{\theta^*\}$
Expectation:
Estimate latent task probabilities of each of the tasks
$P(T_t = T_i|x_t, y_t) = p(y_t|x_t, T_t = T_i)\, P(T_t = T_i)$
Likelihood
Prior (CRP)
$= p_{\theta_{T_i}}(y_t|x_t)\, P(T_t = T_i)$
Get x_{t+1}, y_{t+1}
Maximization:
Maximize log likelihood of data, with update step size proportional to corresponding task probability
Update $\theta(T_i) \propto P(T_t = T_i|x_t, y_t)$
Check if $p(data|T_{new})P(T_{new}) > p(data|T_i)P(T_i)$
Add a new task to θ
Redo E and M steps with updated task distribution
Use model with highest corresponding posterior task probability
Predict/plan under $p_{\theta(T_{max})}$

Figure 6.2: Overview of our algorithm for online learning with mixture of networks. The algorithm decides "task" delineations on its own, starting with only the meta-learned prior θ^* and adding new tasks as it deems necessary for the overall objective of log-likelihood of the observed data. Each instantiated task has to its own set of model parameters, and the algorithm alternates between an expectation (E) step of estimating the posterior task probabilities, and a maximization (M) step of optimizing the log likelihood of data under that inferred task distribution with respect to the model parameters.

6.3.2 Meta-Learning the Prior

We formulated an algorithm above for performing online adaptation using continually incoming data. For this method, we choose to meta-train the prior using the model-agnostic meta-learning (MAML) algorithm. This meta-training algorithm is an appropriate choice, because it results in a prior that is specifically intended for gradient-based fine-tuning. Before we further discuss our choice in meta-training procedure, we first give an overview of MAML and meta-learning in general.

Recall from the previous chapter that, given a distribution of tasks, a meta-learning algorithm produces a learning procedure which can quickly adapt to a new task. MAML optimizes for an initialization of a deep network θ^* that achieves good k-shot task generalization when fine-tuned using just a few (k) datapoints from that task. At train time, MAML sees small amounts of data from a distribution tasks, where data $\mathcal{D}_T$ from each task T can be split into training and validation subsets ($\mathcal{D}_T^{\text{tr}}$ and $\mathcal{D}_T^{\text{val}}$). We will define $\mathcal{D}_T^{\text{tr}}$ as having k datapoints. MAML optimizes for model parameters θ such that one or more gradients steps on $\mathcal{D}_T^{\text{tr}}$ results in a minimal loss L on $\mathcal{D}_T^{\text{val}}$. In our case, we will set $\mathcal{D}_{T_t}^{\text{tr}} = (\mathbf{x}_t, \mathbf{y}_t)$ and $\mathcal{D}_{T_t}^{\text{val}} = (\mathbf{x}_{t+1}, \mathbf{y}_{t+1})$, and the loss L will correspond to the negative log likelihood objective introduced in the previous section.

The MAML meta-RL objective is defined as follows:

$$\min_\theta \sum_T L(\theta - \eta \nabla_\theta L(\theta, \mathcal{D}_T^{\text{tr}}), \mathcal{D}_T^{\text{val}}) = \min_\theta \sum_T L(\phi_T, \mathcal{D}_T^{\text{val}}), \tag{6.8}$$

where a good θ is one that uses a small amount of data $\mathcal{D}_T^{\text{tr}}$ to perform an inner update $\phi_T = \theta - \eta \nabla_\theta L(\theta, \mathcal{D}_T^{\text{tr}})$ with learning rate η, such that this updated information ϕ_T is then able to optimize the objective well on the unseen data $\mathcal{D}_T^{\text{val}}$ from that same task. After meta-training with this objective, the resulting θ^* acts as a prior from which effective fine-tuning can occur on a new task T_{test} at test-time. Here, only a small amount of recent experience from $\mathcal{D}_{T_{\text{test}}}^{\text{tr}}$ is needed in order to update the meta-learned prior θ^* into a $\phi_{T_{\text{test}}}$ that is more representative of the current task at hand:

$$\phi_{T_{\text{test}}} = \theta^* - \eta \nabla_{\theta^*} L(\theta^*, \mathcal{D}_{T_{\text{test}}}^{\text{tr}}). \tag{6.9}$$

Although MAML (Finn, Abbeel, and Levine 2017a) demonstrated this fast adaptation of deep neural networks and the work in the previous chapter (Nagabandi, Clavera, et al. 2018) extended this framework to model-based meta RL, these methods address adaptation in the k-shot setting, always adapting directly from the meta-learned prior and not allowing further adaptation or specialization. In this work, we have extended these capabilities by enabling more evolution of knowledge through a temporally-extended online adaptation procedure, which was presented in the previous section. Note that our procedure for continual online learning is initialized with this prior that was meta-trained for k-shot adaptation, with the intuitive rational that MAML trains this model to be able to change significantly using only a small number of datapoints and gradient steps – which is not true in general for a deep neural network. We show in Sec. 6.5 that our method for continual online learning with a mixture of models (initialized from this meta-learned prior) outperforms both standard k-shot adaptation with the same prior (where model parameters are updated at each step from the prior itself), and also outperforms naively taking many adaptation steps away from that prior (where model parameters are updated at each step directly from the parameters values of the previous time step).

We note that it is quite possible to modify the MAML meta-training algorithm to optimize the model directly with respect to the weighted updates discussed in Section 6.3.1.2. This simply requires computing the task weights (the E step) on each batch during meta-training, and then constructing a computation graph where all gradient updates are multiplied by their respective weights. Standard automatic differentiation software can then compute the corresponding meta-gradient. For short trial lengths, this is not substantially more complex than standard MAML; for longer trial lengths, truncated backpropagation is an option. Although such a meta-training procedure better matches the way that the model is used during online adaptation, we found that it did not substantially improve our results. While it's possible that the difference might be more significant if meta-training for longer-term adaptation, this observation does suggest that simply meta-training with MAML is sufficient for enabling effective continuous online adaptation in non-stationary multi-task settings.

6.4 Application of MOLe to Model-Based RL

In our experiments, we apply MOLe to model-based reinforcement learning. RL in general aims to act in a way that maximizes the sum of future rewards. At each time step t, the agent executes action $\mathbf{a}_t \in \mathcal{A}$ from state $\mathbf{s}_t \in \mathcal{S}$, transitions to the next state $\mathbf{s}_{t+1}$ according to the transition probabilities (i.e., dynamics) $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ and receives rewards $r_t = r(\mathbf{s}_t, \mathbf{a}_t)$. The goal at each step is to execute the action $\mathbf{a}_t$ that maximizes the discounted sum of future rewards $\sum_{t'=t}^{\infty} \gamma^{t'-t} r(\mathbf{s}_{t'}, \mathbf{a}_{t'})$, where discount factor $\gamma \in [0, 1]$ prioritizes near-term rewards. In model-based RL, in particular, the predictions from a known or learned dynamics model are used to either learn a policy that selects actions, or are used directly inside a planning algorithm to select actions that maximize reward.

In this work, the underlying distribution that we aim to model is the dynamics distribution $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, T_t)$, where the unknown T_t represents the underlying settings (e.g., state of the system, external details, environmental perturbations, etc.). The goal for MOLe is to estimate this distribution with a predictive model p_θ. To instantiate MOLe in this context of model-based RL, we follow Algorithm 6 with the following specifications:

(1) We set the input $\mathbf{x}_t$ to be the concatenation of K previous states and actions, given by $\mathbf{x}_t = [\mathbf{s}_{t-K}, \mathbf{a}_{t-K}, \ldots, \mathbf{s}_{t-2}, \mathbf{a}_{t-2}, \mathbf{s}_{t-1}, \mathbf{a}_{t-1}]$, and the output to be the corresponding next states $\mathbf{y}_t = [\mathbf{s}_{t-K+1}, \ldots, \mathbf{s}_{t-1}, \mathbf{s}_t]$. This provides us with a slightly larger batch of data for each online update, as compared to using only the data from the given time step. Since individual time steps at high frequency can be very noisy, using the past K transitions helps to damp

out the updates.

(2) The predictive model p_θ represents each of these underlying transitions as an independent Gaussian such that $p_\theta(\mathbf{y}_t|\mathbf{x}_t) = \prod_{t'=t-K}^{t-1} p(\mathbf{s}_{t'+1}|\mathbf{s}_{t'}, \mathbf{a}_{t'})$, where each $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ is parameterized with a Gaussian given by mean $f_\theta(\mathbf{s}_t, \mathbf{a}_t)$ and constant variance σ^2. We implement this mean dynamics function $f_\theta(s_t, a_t)$ as a neural network model with three hidden layers each of dimension 500, and ReLU nonlinearities.

(3) To calculate the new task parameter $\theta_{t+1}(T_{\text{new}})$, which may or may not be added to the task distribution $\theta_{t+1}(T)$, we must use *some* data to adapt from θ^* into this $\theta(T_{\text{new}})$. For this, we use a set of K nearby datapoints that are *not* the set $\mathbf{x}_t$ itself. This is done to avoid calculating the parameter using the same dataset on which it is evaluated; since $P_t(T_{\text{new}}|\mathbf{x}_t, \mathbf{y}_t)$ is calculated by evaluating the likelihood of that data under this new task, the new task itself cannot be generated using that same data.

(4) Unlike standard online streaming tasks where the next data point $\mathbf{x}_{t+1}$ is just given (independent of the current modeling), the incoming data point in this model-based RL case is the next state of the system and is thus influenced by the predictive model itself. This is because, after the most likely task T^* is selected from the possible tasks, the predictions from the model $p_{\theta_{t+1}(T^*)}$ are used by the controller to plan over a sequence of future actions $\mathbf{a}_0, \dots, \mathbf{a}_H$ and select the actions that maximize future reward. Note that the planning procedure is based on stochastic optimization, following prior work (Nagabandi, Clavera, et al. 2018) as shown in the previous chapters. Since the estimated model parameters affect the controller's action choice, and since that controller's action choice determines the next state of the system (i.e., the next incoming data point), it is even more crucial in this setting to appropriately adapt the model.

5) Finally, note that we attain θ^* from meta-training using model-agnostic meta-learning (MAML), as mentioned in the method above. However, in this case, MAML is performed in the loop of model-based RL. In other words, the model parameters at a given iteration of meta-training are used by the controller to generate on-policy rollouts, the data from these rollouts is then added to the dataset for MAML, and this process repeats until the end of meta-training. This meta-training process is exactly the same as the one proposed in the previous chapter.

6.5 Results of Continual Online Adaptation with MOLe

The questions that we aim to study from our experiments include the following:

1. Can MOLe autonomously discover some task structure amid a stream of non-stationary data?

2. Can MOLe adapt to tasks that are further outside of the task distribution than can be handled by a k-shot learning approach?

3. Can MOLe recognize and revert to tasks it has seen before?

4. Can MOLe avoid overfitting to a recent task, to prevent deterioration of performance upon the next task switch?

5. Can MOLe outperform other methods?

To study these questions, we conduct experiments on agents in the MuJoCo physics engine (Emanuel Todorov, Erez, and Tassa 2012a). The agents we use are a half-cheetah ($\mathcal{S} \in R^{21}, \mathcal{A} \in R^{6}$) and a hexapedal crawler ($\mathcal{S} \in R^{50}, \mathcal{A} \in R^{12}$). Using these agents, we design a number of challenging online learning problems that involve multiple sudden and gradual changes in the underlying task distribution. These include tasks that are extrapolated from those seen previously during training, where online learning is critical for success. Through these experiments, we aim to build problem settings that are representative of the types of disturbances and shifts that a real RL agent might encounter when deployed into the world.

We present results and analysis of our findings in the following three sections, and videos can be found online[2]. In our experiments, we compare to several alternative methods, including two approaches that leverage meta-training and two approaches that do not, as described below:

(a) **k-shot adaptation with meta-learning**: Always adapt from the meta-trained prior θ^*, as typically done with meta-learning methods (Nagabandi, Clavera, et al. 2018), including the one introduced in the previous chapted. This method is often insufficient for adapting to tasks that are further outside of the training distribution, and the adaptation is also not carried forward in time for future use.

(b) **continued adaptation with meta-learning**: Always take gradient steps from the previous time step's parameters (without revert back to the meta-learning prior). This method oftens overfits to recently observed tasks, so it should indicate the importance of our method effectively identifying task structure to avoid overfitting and enable recall.

(c) **model-based RL**: Train a model on the same data as the methods above, using standard supervised learning, and keep this model fixed throughout the trials (i.e., no meta-learning and no adaptation). For context, this is the model-based RL framework that was introduced in the first few chapters, containing no explicit meta-learning or adaptation mechanisms.

(d) **model-based RL with online gradient updates**: Use the same model from model-based RL (i.e., no meta-learning), but adapt it online at test using gradient-descent. This is representative of commonly used dynamic evaluation methods (Rei 2015; Krause, Kahembwe, et al. 2017; Krause, Lu, et al. 2016; Fortunato, Blundell, and Vinyals 2017).

[2]`https://sites.google.com/berkeley.edu/onlineviameta`

Table 6.1: Hyperparameters for train-time

	Iters	Epochs	# Tasks/itr	# TS/itr	K	outer LR	inner LR (η)
Meta-learned approaches (3)	12	50	16	2000-3000	16	0.001	0.01
Non-meta-learned approaches (2)	12	50	16	2000-3000	16	0.001	N/A

Table 6.2: Hyperparameters for run-time

	α (CRP)	LR (model update)	K (previous data)
MOLe (ours)	1	0.01	16
continued adaptation with meta-learning	N/A	0.01	16
k-shot adaptation with meta-learning	N/A	0.01	16
model-based RL	N/A	N/A	N/A
model-based RL with online gradient updates	N/A	0.01	16

6.5.1 Hyperparameters

In all experiments, we use a dynamics model consisting of three hidden layers, each of dimension 500, with ReLU nonlinearities. The control method that we use is random-shooting model predictive control (MPC) where 1000 candidate action sequences each of horizon length H=10 are sampled at each time step, fed through the predictive model, and ranked by their expected reward. The first action step from the highest-scoring candidate action sequence is then executed before the entire planning process repeats again at the next time step.

In Tables 6.1 and 6.2, we list relevant training and testing parameters for the various methods used in our experiments. # Task/itr corresponds to the number of tasks sampled during each iteration of collecting data to train the model, and # TS/itr is the total number of times steps collected during that iteration (sum over all tasks). The inner and outer learning rates control the sizes of the gradient steps during meta-training, where each iteration out of the "Iters" iterations consists of "Epochs" epochs of model training, with each epoch consisting of a full pass through the dataset.

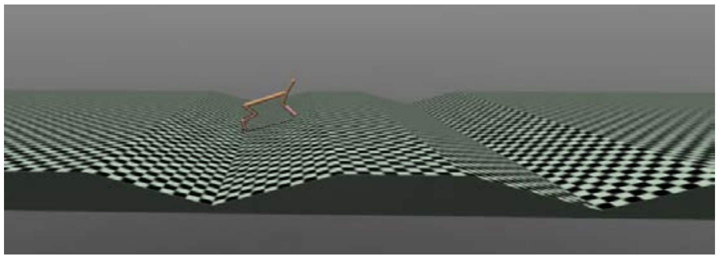


Figure 6.3: Half-cheetah robot, shown traversing a landscape with 'basins' that was not encountered during training.

6.5.2 Terrain Slopes on Half-Cheetah

We start with the task of a half-cheetah (Fig. 6.3) robot, traversing terrains of varying slopes. The prior model is meta-trained on data from terrains with random slopes of low magnitudes, and the test trials are executed on difficult out-of-distribution tasks such as basins, steep hills, etc. As shown in Fig. 6.4, neither model-based RL nor model-based RL with online gradient updates perform well on these out-of-distribution tasks, even though those models were trained using the same data that the meta-trained model received. The bad performance of the model-based RL approach indicates the need for model adaptation (as opposed to assuming that a single model can encompass everything), while the bad performance of adaptive model-based RL (with online gradient updates) indicates the need for a meta-learned initialization to enable effective online adaptation with neural networks.

For the three meta-learning and adaptation methods, we expect continued adaptation with meta-learning to perform poorly due to continuous gradient steps causing it to overfit to recent data; that is, we expect that experience on the upward slopes to lead to deterioration of performance on downward slopes, or something similar. However, based on both our qualitative and quantitative results, we see that the meta-learning procedure seems to have initialized the agent with a parameter space in which these various "tasks" are not seen as substantially different, where online learning by SGD performs well. This suggests that the meta-learning process finds a task space where there is an easy skill transfer between slopes; thus, even when MOLe is faced with the option of switching tasks or adding new tasks to its dynamic latent task distribution, we see in Fig. 6.5 that it chooses not to do so. Unlike findings that we will see later, it is interesting that the discovered task space here does not correspond to human-distinguishable categorical labels of uphill and downhill. Finally, note that these tasks of changing slopes are not naturally similar to each other, because the two non-meta-learning baselines do indeed fail on these test tasks, despite having similar training performance on the shallow training slopes.

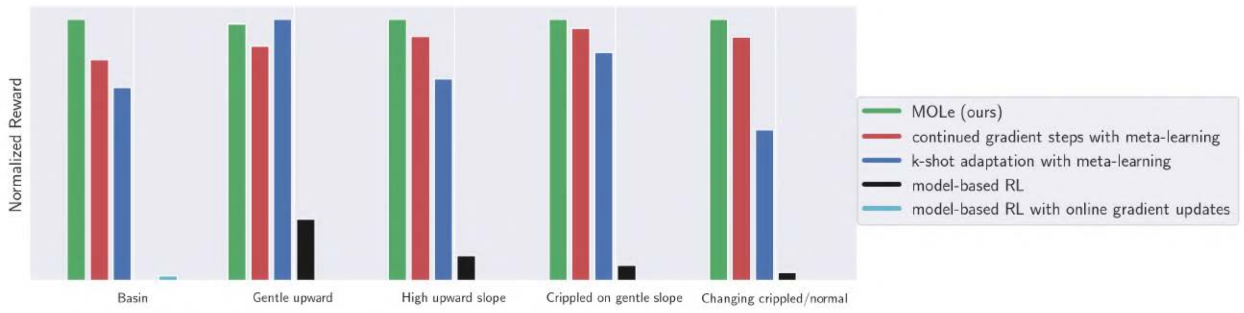


Figure 6.4: Results on half-cheetah terrain traversal. The poorly performing model-based RL shows that a single model is not sufficient, and the poorly performing model-based RL with online gradient updates shows that a meta-learned initialization is critical. The three meta-learning approaches perform similarly on these tasks of different slopes. Note, however, that the performance of k-shot adaptation does deteriorate when the tasks are further away from the training task distribution, such as the last column above where the test tasks introduce crippling of joints. Since this unexpected perturbation is far from what was seen during training, it calls for taking multiple gradient steps away from the prior in order to actually succeed. We see that MOLe succeeds in all of these task settings.

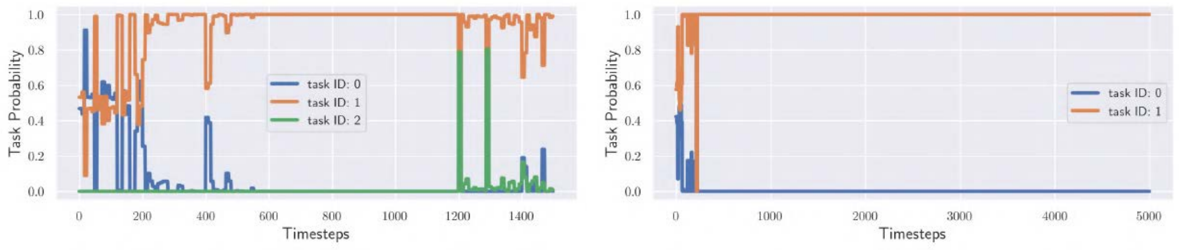


Figure 6.5: Latent task distribution over time for two half-cheetah landscape traversal tasks, where encountered terrain slopes vary within each run. Interestingly, we find that MOLe chooses to only use a single latent task variable to describe varying terrain.

6.5.3 Half-Cheetah Motor Malfunctions

While the findings from the half-cheetah on sloped terrains illustrate that separate task parameters aren't always necessary for what might externally seem like separate tasks, we also want to study agents that experience more drastically-changing non-stationary task distributions during their experience in the world. For this set of experiments, we train all models on data where a single actuator is selected at random to experience a malfunction during the rollout. In this case, malfunction means that the polarity or magnitude of actions applied to that actuator are altered. Fig. 6.6 shows the results of various methods on drastically out-of-distribution test tasks, such as altering all actuators at once. The left of Fig. 6.6 shows that when the task distribution during the test trials contains only a single task, such as 'sign negative' where all actuators are prescribed to be the opposite polarity, then continued adaptation performs well by repeatedly performing gradient updates on incoming data. However, as shown in the other tasks of Fig. 6.6, the performance of this continued adaptation substantially deteriorates when the agent experiences a non-stationary task distribution. Due to overspecialization on recent incoming data, such methods that *continuously* adapt tend to forget and lose previously existing skills. This overfitting and forgetting of past skills is also illustrated in the consistent performance deterioration shown in Fig. 6.6(right). MOLe, on the other hand, dynamically builds a probabilistic task distribution and allows adaptation to these difficult tasks, without forgetting past skills. We visualize a dynamically built task distribution in Fig. 6.7, where the agent experiences alternating periods of normal and crippled-leg operation. This plot shows the successful instantiation of new tasks as well as recall of old tasks; note that both the recognition and adaptation are all done online, without using a bank of past data to perform the adaptation, and without a human-specified set of task categories.

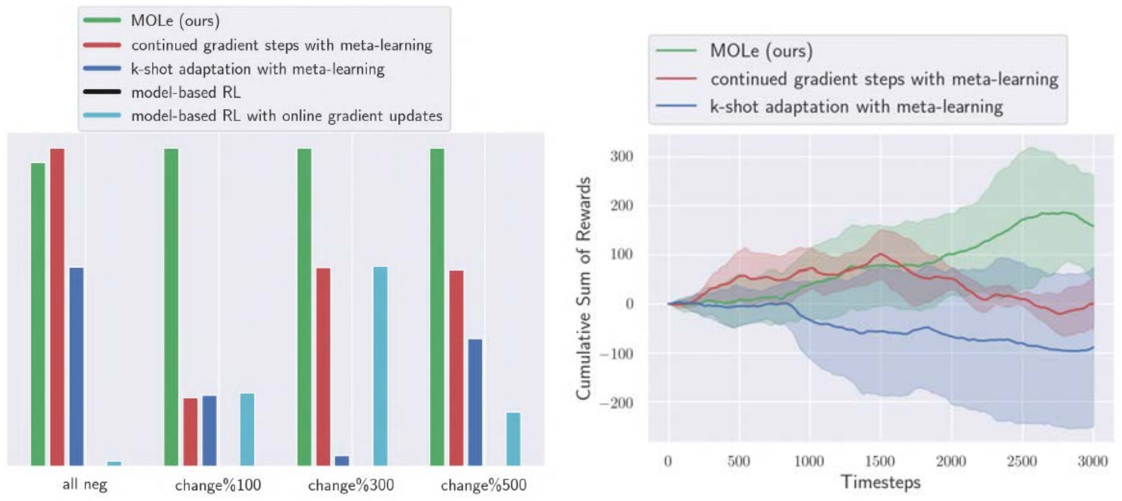


Figure 6.6: Results on the motor malfunction trials, where different trials are shown task distributions that modulate at different frequencies (or stay constant, in the first column). Here, online learning is critical for good performance, k-shot adaptation is insufficient for these tasks that are very different from the tasks seen during training, and continued gradient steps leads to overfitting to recently seen data. MOLe, however, demonstrates high performance in all of these types of task distributions.

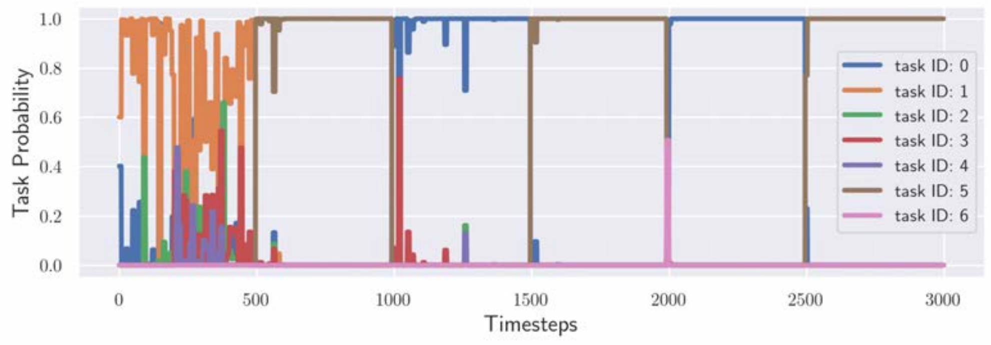


Figure 6.7: Latent task variable distribution over the course of an online learning trial where the underlying motor malfunction changes every 500 timesteps. We find that MOLe is able to successfully recover the underlying task structure by recognizing when the underlying task has changed, and even recalling previously seen tasks. As such, MOLe allows for both specialization as well as generalization.

6.5.4 Crippling of End Effectors on Six-Legged Crawler

To further examine the effects of our continual online adaptation algorithm, we study another, more complex agent: a 6-legged crawler (Fig. 6.8). In these experiments, models are trained on random joints being crippled (i.e., unable to apply actuator commands). In Fig. 6.9, we present two illustrative test tasks: (1) the agent sees a set configuration of crippling for the duration of its test-time experience, and (2) the agent receives alternating periods of experience, between regions of normal operation and regions of having crippled legs.

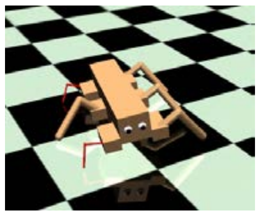


Figure 6.8: Six-legged crawler robot, shown with crippled legs at run time.

The first setting is similar to data seen during training, and thus, we see that even the model-based RL and model based-RL with online gradient updates baselines do not fail. The methods that include both meta-learning and adaptation, however, do have higher performance. Furthermore, we see again that continued gradient steps in this case of a single-task setting is not detrimental.

The second setting's non-stationary task distribution (when the leg crippling is dynamic) illustrates the need for online adaptation (model-based RL fails), the need for a good meta-learning prior to adapt from (failure of model-based RL with online gradient updates), the harm of overfitting to recent experience and thus forgetting older skills (low performance of continued gradient steps), and the need for further adaptation away from the prior (limited performance of k-shot adaptation).

With MOLe, this agent is able to build its own representation of "task" switches, and we see that this switch does indeed correspond to recognizing regions of leg crippling (left of Fig. 6.10). The plot of the cumulative sum of rewards (right of Fig. 6.10) of each of the three meta-learning plus adaptation methods includes this same task switch pattern every 500 steps: Here, we can clearly see that steps 500-1000 and 1500-2000 were the crippled regions. Continued gradient steps actually performs worse on the second and third times it sees normal operation, whereas MOLe is noticeably better as it sees the task more often. Note this improvement of both skills with MOLe, where development of one skill actually does not hinder the other.

Finally, we examine experiments where the crawler experiences (during each trial) walking straight, making turns, and sometimes having a crippled leg. The performance during the first 500 time steps of "walking forward in a normal configuration" for continued gradient steps was comparable to MOLe (+/-10% difference), but its performance during the last 500 time steps of "walking forward in a normal configuration" was **200%** lower (for continued gradient steps). Note this detrimental effect of performing continual updates at each step without allowing for separate task specialization/adaptation.

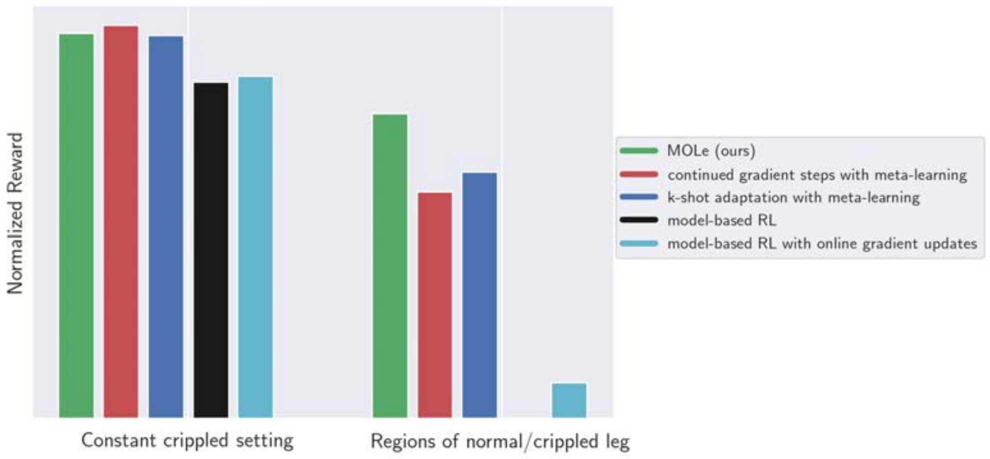


Figure 6.9: Quantitative results on crawler. For a fixed task (left column), adaptation is not necessary and all methods perform well. In contrast, when tasks change dynamically within the trial, only MOLe effectively learns online.

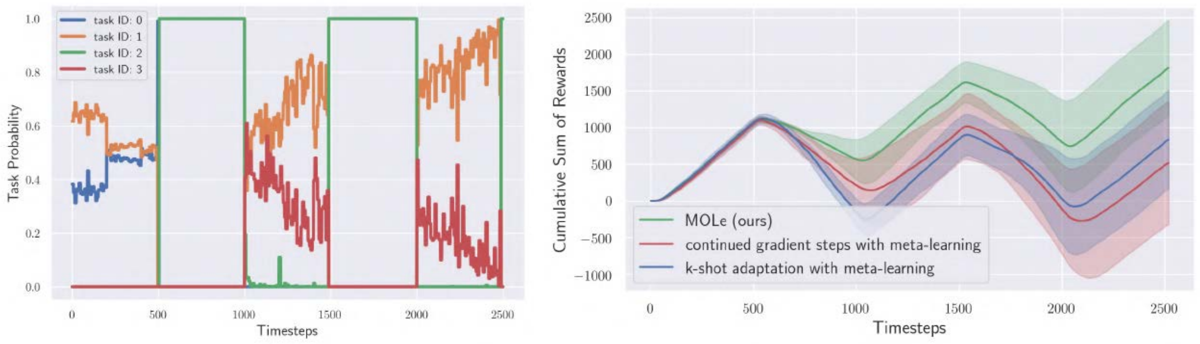


Figure 6.10: Results on crawler experiments. Left: Online recognition of latent task probabilities for alternating periods of normal/crippled experience. Right: MOLe improves from seeing the same tasks multiple times; MOLe allows for improvement of both skills, without letting the development of one skill hinder the other.

6.6 Discussion

In this chapter, we presented an online learning method for neural network models that can handle non-stationary, multi-task settings within each trial. Our method adapts the model directly with SGD, where an EM algorithm uses a Chinese restaurant process prior to maintain a distribution over tasks and handle non-stationarity. Although SGD generally makes for a poor online learning algorithm in the streaming setting for large parametric models such as deep neural networks, we observed that, by (1) meta-training the model for fast adaptation with MAML and (2) employing our online algorithm for probabilistic updates at test time, we can enable effective online learning with neural networks.

In our experiments, we applied this approach to model-based RL, and we demonstrated that it could be used to adapt the behavior of simulated robots faced with various new and unexpected tasks. The results showed that our method can develop its own notion of task, continuously adapt away from the meta-learned prior as necessary (to succeed at tasks further outside of the training distribution, which require more adaptation), and recall tasks it has seen before. This ability to effectively adapt online and develop specializations as well as maintain generalization is a critical step toward enabling the deployment of robots into the real world, where the uncontrolled settings are never exactly what was seen during train time.

Chapter 7

Adaptive Online Reasoning from Images via Latent Dynamics Models

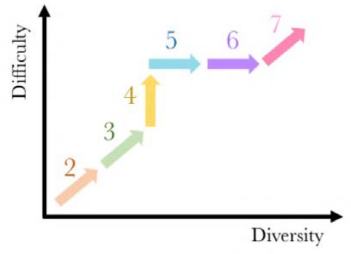


Figure 7.1: Thesis outline, with the current chapter indicated by the last colored arrow.

The work in the previous chapters has built up multiple concepts. First, learning models introduced a significant improvement in the sample efficiency of deep learning methods. Second, the decoupling of tasks from dynamics gave us the ability to train these models with even off-policy data and then use them to solve multiple tasks at test time. Third, these efficient learning approaches were extended further along the axis of generalization and diversity by incorporating various meta-learning techniques to enable online adaptation of these learned models. We saw that these adaptation capabilities were critical for addressing the inevitable mismatch between an agent's training conditions and the test conditions in which it may actually be deployed, and that this adaptation to new conditions needed to happen *quickly* (i.e., it is not feasible to learn each new task from scratch). In this chapter, we extend these capabilities further along the axis of difficulty; in particular, we

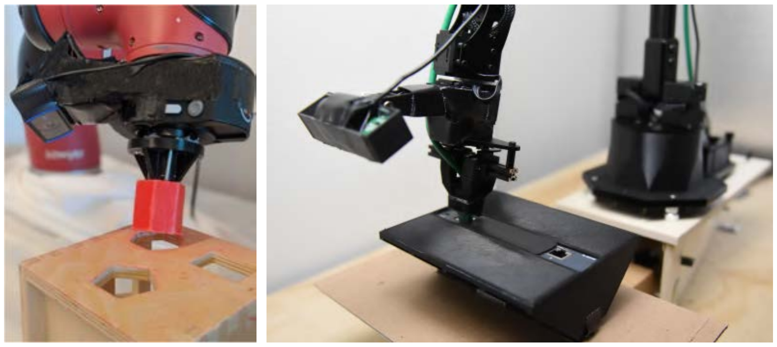


Figure 7.2: Our approach (MELD) enables meta-RL directly from raw image observations in the real world, as shown here by the learned tasks of peg insertion and ethernet cable insertion. MELD allows for sample-efficiency during meta-training time (enabling us to perform this training procedure in the real world), as well as the learned ability to perform *fast adaptation to new tasks* at test time. This ability to perform broad ranges of skills as necessitated by the changing conditions of test time settings is critical towards the goal of deploying robots into the real world.

address the difficulty of working from raw sensory observations such as a robot's onboard cameras. Unlike the previous chapters, whose methods operated directly from information such as joint positions/velocities and object positions/velocities, we now scale up our learning and adaptation capabilities to the more general setting of working directly from raw image observations.

Operating from raw image observations introduces challenges in the form of high-dimensional and partially observable inputs. Making sense of these pixel inputs adds a representation learning burden on top of the already challenging task learning problem. Prior work has demonstrated success in single-task RL from these types of inputs by explicitly addressing the representation learning problem via the learning of latent dynamics models to make sense of the inputs. Unlike the work in previous chapters which learned models and then used the model predictions for action selection, the work in this chapter instead uses models to address the representation learning and meta-learning aspects of the problem, while allowing for model-free RL techniques to address the task learning problem within this learned representation space.

We refer to the problem of making sense of incoming observations in order to understand the underlying state of the system as performing "latent state estimation." By learning a latent dynamics model to address this representation learning problem, we still make use of all of the benefits of models that we've seen thus far – their efficient training using even off-policy data, their adaptability, and their generalization capabilities. Inspired by the use of these latent state dynamics models to improve single-task RL from high dimensional inputs, this work aims to extend these capabilities into the meta-RL regime of being able to perform a broad range of skills. In this work, we posit that the task inference problem in meta-RL can actually be cast into this same framework of latent state estimation, with the unknown task variable viewed as part of the hidden variables that must be inferred. Leveraging this idea of fusing both task and state inference into a unified framework, we present our algorithm MELD, Meta-RL with Latent Dynamics: a practical algorithm for meta-RL from image observations that quickly acquires new skills at test time via posterior inference in a learned latent state model over joint state and task variables. We show that MELD outperforms prior meta-RL methods on a range of simulated robotic locomotion and manipulation problems including peg insertion and object placing. Furthermore, we demonstrate MELD on two real robots, learning to perform peg insertion into varying target boxes with a Sawyer robot, and learning to insert an ethernet cable into new locations after only 4 hours of meta-training on a WidowX robot. Videos of the experiments are available online[1].

7.1 Introduction

Robots that operate in unstructured environments, such as homes and offices, must be able to perform a broad range of skills using only their on-board sensors. The problem of learning from raw sensory observations is often tackled in the context of state estimation: building

[1]`https://sites.google.com/view/meld-lsm/home`

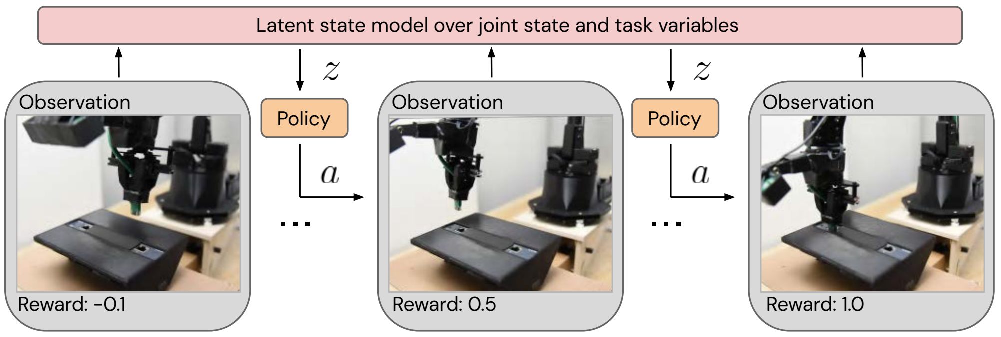


Figure 7.3: **Real-world cable insertion**: At test time our algorithm (MELD) enables a 7-DoF WidowX robot to insert the ethernet cable into novel locations and orientations within a single trial, operating from image observations. MELD achieves this result by meta-training a latent dynamics model to capture task and state information in the hidden variables, as well a policy that conditions on these variables, across 20 meta-training tasks with varying locations and orientations.

models that infer the unknown state variables from sensory readings (Thrun, Burgard, Fox, et al. 2005; Barfoot 2017; Tremblay et al. 2018). In principle, deep reinforcement learning (RL) algorithms can automate this process by directly learning to map sensory inputs to actions, obviating the need for explicit state estimation. This automation comes at a steep cost in sample efficiency since the agent must learn to interpret observations from reward supervision alone. Fortunately, unsupervised learning in the form of general-purpose latent state (or dynamics) models can serve as an additional training signal to help solve the representation learning problem (Finn, X. Y. Tan, et al. 2016; Ghadirzadeh et al. 2017; A. X. Lee et al. 2019; M. Zhang et al. 2019), substantially improving the performance of end-to-end RL. However, even the best-performing algorithms in this class require hours of training to learn a single task (Hafner et al. 2018; M. Zhang et al. 2019; A. X. Lee et al. 2019) and lack a mechanism to transfer knowledge to subsequent tasks.

General purpose autonomous robots must be able to perform a wide variety of tasks and quickly acquire new skills. For example, consider a robot tasked with assembling electronics in a data center. This robot must be able to insert cables of varying shapes, sizes, colors, and weights into the correct ports with the appropriate amounts of force. While standard RL algorithms require hundreds or thousands of trials to learn a policy for each new setting, meta-RL methods hold the promise of drastically reducing the number of trials required. Given a task distribution, such as the variety of ways to insert electronic cables described above, meta-RL algorithms leverage a set of training tasks to meta-learn a mechanism that can quickly learn unseen tasks from the same distribution. Despite impressive results in simulation demonstrating that agents can learn new tasks in a handful of trials (J. X. Wang et al. 2016a; Duan, John Schulman, X. Chen, Peter L Bartlett, et al. 2016b; Finn, Abbeel, and Levine 2017b; Rakelly et al. 2019), these model-free meta-RL algorithms remain largely unproven on real-world robotic systems, largely due to the inefficiency of the meta-training process. In this paper, we show that the same latent dynamics models that greatly improve efficiency in end-to-end single-task RL can *also*, with minimal modification, be used for meta-RL by treating the unknown task information as a hidden variable to be estimated from experience. We formalize the connection between latent state inference and meta-RL, and leverage this insight in our proposed algorithm MELD, Meta-RL with Latent Dynamics.

To derive our algorithm, we cast meta-RL and latent state inference into a single partially observed Markov decision process (POMDP) in which task and state variables are aspects of a more general per-time step hidden variable. Concretely, we represent the agent's belief over the hidden variable as the variational posterior in a sequential VAE latent state model that takes observations and rewards as input, and we condition the agent's policy on this belief. During meta-training, the latent state model and policy are trained across a fixed set of training tasks sampled from the task distribution. The trained system can then quickly learn to succeed on a new unseen task by using a small amount of data from the new task to infer the posterior belief over the hidden variable, and then executing the meta-learned policy conditioned on this belief.

We show in simulation that MELD substantially outperforms prior work on several challenging locomotion and manipulation problems such as running at varying velocities, inserting a peg into varying targets, and putting away mugs of unknown weights to varying locations on a shelf. Next, we run MELD on a real Sawyer robot to solve the problem of peg insertion into varying targets, verifying that MELD can reason jointly about state and task information. Finally, we evaluate our method with a real WidowX robotic arm, where we show that MELD is able to successfully perform ethernet cable insertion into ports at novel locations and orientations after four hours of meta-training.

7.2 Related Work

Our approach can be viewed methodologically as a bridge between meta-RL methods for fast skill acquisition and latent state models for state estimation. In this section, we discuss these areas of work, as well as RL on real-world robotic platforms.

RL for Robotics. While prior work has obtained good results with geometric and force control approaches for a wide range of manipulation tasks (Bicchi and Vijay Kumar 2000; Pereira, Campos, and Vijay Kumar 2004; Henrich and Wörn 2012), including insertion tasks (Kronander, Burdet, and Billard 2014; Newman, Zhao, and Pao 2001) such as those in our evaluation, such approaches typically require considerable manual design effort for each task. RL algorithms offer an automated alternative that has been demonstrated on a variety of robotic tasks (Kober, J. A. Bagnell, and Peters 2013) including balancing a unicycle robot (Marc Deisenroth and Carl E Rasmussen 2011), pushing boxes (Mahadevan and Connell 1992; Finn and Levine 2017a), playing hockey (Chebotar et al. 2017), opening doors (Shixiang Gu, Holly, et al. 2017), valve turning and block stacking (Haarnoja, Zhou, Hartikainen, et al. 2018; M. Zhang et al. 2019), grasping objects (Pinto and Abhinav Gupta 2016; Kalashnikov et al. 2018), throwing objects (Ghadirzadeh et al. 2017; Zeng, S. Song,

J. Lee, et al. 2019), hand manipulation (Zhu et al. 2019), and including insertion (Gullapalli, Franklin, and Benbrahim 1994; Levine, Finn, et al. 2016; Zeng, S. Song, Welker, et al. 2018; M. A. Lee et al. 2018; Schoettler et al. 2019). Although these policies learn impressive skills, they typically do not transfer to other tasks and must be re-trained from scratch for each task.

Meta-RL. Meta-reinforcement learning algorithms learn how to acquire new skills quickly by using experience from a large number of meta-training tasks. Current meta-RL methods differ in how this acquisition procedure is represented, ranging from directly representing the learned learning process via a black-box function approximator such as a recurrent (J. X. Wang et al. 2016a; Duan, John Schulman, X. Chen, Peter L Bartlett, et al. 2016b) or recursive (Mishra, Rohaninejad, et al. 2017b) deep network, to learning initial parameters for gradient-based optimization (Finn, Abbeel, and Levine 2017b; Rothfuss et al. 2018; Houthooft et al. 2018; Mendonca et al. 2019), and learning tasks via variational inference (Rakelly et al. 2019; Zintgraf et al. 2019; Perez, Such, and Karaletsos 2020). Some of these works have formalized meta-RL as a special kind of POMDP in which the hidden state is held constant throughout a task (Rakelly et al. 2019; Zintgraf et al. 2019; Humplik et al. 2019; Perez, Such, and Karaletsos 2020). Taking a broader view, we show that meta-RL can be tackled with a general POMDP algorithm that estimates a time-varying hidden state, allowing the same algorithm to be used for problems with both stationary and non-stationary sources of uncertainty.

Within this area, meta-learning approaches that enable few-shot adaptation have been studied with real systems for imitation learning (Finn, T. Yu, et al. 2017; T. Yu et al. 2018; James, Bloesch, and Davison 2018; Bonardi, James, and Davison 2019) and goal inference (Xie et al. 2018), but direct meta-RL in the real world has received comparatively little attention. Adapting to different environment parameters has been explored in the context of sim2real for table-top hockey (Arndt et al. 2019) and legged locomotion (X. Song et al. 2020), and in the model-based RL setting for millirobot locomotion (Nagabandi, Clavera, et al. 2018) (as presented in the previous two chapters).

Latent State Inference in RL. A significant challenge in real-world robotic learning is contending with the complex, high-dimensional, and noisy observations from the robot's sensors. To handle general partial observability, recurrent policies can persist information over longer time horizons (Yadaiah and Sowmya 2006; Heess, J. J. Hunt, et al. 2015; Hausknecht and Stone 2015), while explicit state estimation approaches maintain a probabilistic belief over the current state of the agent and update it given experience (Kaelbling, Littman, and Cassandra 1998; Pineau, G. Gordon, Thrun, et al. 2003; Ross, Pineau, et al. 2011; M. P. Deisenroth and Peters n.d.; Karkus, D. Hsu, and W. S. Lee 2017; Igl et al. 2018; Gregor and Besse 2018). In our experiments we focus on learning from image observations, which presents a state representation learning challenge that has been studied in detail. End-to-end deep RL algorithms can learn state representations implicitly, but currently suffer from poor sample efficiency due to the added burden of representation learning (Volodymyr Mnih, Koray Kavukcuoglu, Silver, Graves, et al. 2013; Levine, Finn, et al. 2016; Singh et al. n.d.). Pre-trained state estimation systems can predict potentially useful features such as object locations and pose (Tremblay et al. 2018; Visak Kumar et al. 2019); however, these approaches require ground truth supervision. On the other hand, unsupervised learning techniques can improve sample efficiency without access to additional supervision (Lange, Riedmiller, and Voigtlander 2012; Finn, X. Y. Tan, et al. 2016; Schmidt, Newcombe, and Fox 2016; Ghadirzadeh et al. 2017; Florence, Manuelli, and Tedrake 2019; Yarats et al. 2019; Sax et al. 2019). Latent dynamics models capture the time-dependence of observations and provide a learned latent space in which RL can be tractably performed (Watter et al. 2015; Karl et al. 2016; M. Zhang et al. 2019; Hafner et al. 2019; Gelada et al. 2019; A. X. Lee et al. 2019). In our work, we generalize the learned latent variable to encode not only the state but also the task at hand, enabling efficient meta-RL from images.

7.3 Preliminaries

In this work, we leverage tools from latent state modeling to design an efficient meta-RL method that can operate in the real world from image observations. In this section, we review latent state models and meta-RL, developing a formalism that will allow us to derive our algorithm in Section 7.4.

7.3.1 POMDPs and Latent State Models

We first define the RL problem, as also seen in the previous chapters. Recall a Markov decision process (MDP), which consists of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, an initial state distribution $p(\mathbf{s}_1)$, a state transition distribution $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, a discount factor γ, and a reward function $r(\mathbf{s}_t, \mathbf{a}_t)$. We assume the transition and reward functions are unknown, but can be sampled by taking actions in the environment. The goal of RL is to learn a policy $\pi(\mathbf{a}_t|\mathbf{s}_t)$ that selects actions that maximize the sum of discounted rewards. However, robots operating in the real world do not have access to the underlying state $\mathbf{s}_t$, and must instead select actions using high-dimensional and often incomplete observations from cameras and other sensors. Such a system can be described as a partially observed Markov decision process (POMDP), where observations $\mathbf{x}_t$ are a noisy or incomplete function of the unknown underlying state $\mathbf{s}_t$, and the policy is conditioned on a history of these observations as $\pi(\mathbf{a}_t|\mathbf{x}_{1:t})$.

While end-to-end RL methods can implicitly acquire representations of the incoming observations from reward supervision alone (Levine, Finn, et al. 2016; Zeng, S. Song, Welker, et al. 2018), this added burden of representation learning limits sample efficiency and can make optimization more difficult. Methods that explicitly address this representation learning problem have been shown to be more efficient and scale more effectively to harder domains (Igl et al. 2018; Gelada et al. 2019; A. X. Lee et al. 2019). These approaches train latent state models to learn meaningful representations of the incoming observations by explicitly representing the unknown Markovian state as a hidden variable $\mathbf{z}_t$ in a graphical model, as shown in Figure 7.4 (a). The parameters of these graphical models can be trained by approximately maximizing the log-likelihood of the observations: $\log p(\mathbf{x}_{1:T}|\mathbf{a}_{1:T-1}) = \log \int p(\mathbf{x}_T|\mathbf{z}_T)p(\mathbf{z}_T|\mathbf{z}_{T-1}, \mathbf{a}_{T-1})...p(\mathbf{z}_1)dz$. Given a history of observations and actions seen so far, the posterior distribution over the hidden variable captures the agent's belief over the current underlying state, and can be written as $\mathbf{b}_t = p(\mathbf{z}_t|\mathbf{x}_{1:t}, \mathbf{a}_{1:t-1})$. Then, rather than conditioning the policy on raw observations, these methods learn a policy $\pi(\mathbf{a}_t|\mathbf{b}_t)$ as a function of this belief state.

7.3.2 Meta-Reinforcement Learning

In this work, we would like a robot to learn to acquire new skills quickly. We formalize this problem statement as meta-RL, where each task $\mathcal{T}$ from a distribution of tasks $p(\mathcal{T})$ is a POMDP as described above, with initial state distribution $p_\mathcal{T}(\mathbf{s}_1)$, dynamics function $p_\mathcal{T}(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, observation function $p_\mathcal{T}(\mathbf{x}_t|\mathbf{s}_t)$, and reward function $r_\mathcal{T}(r_t|\mathbf{s}_t, \mathbf{a}_t)$. For example, a task distribution that varies both dynamics and rewards across tasks may consist of placing mugs of varying weights (dynamics) in different locations (rewards) on a kitchen shelf. The meta-training process leverages a set of training tasks sampled from $p(\mathcal{T})$ to learn an adaptation procedure that can adapt to a new task from $p(\mathcal{T})$ using a small amount of experience. Similar to the framework in probabilistic inference-based and recurrence-based meta-RL approaches (Zintgraf et al. 2019; Rakelly et al. 2019; Duan, John Schulman, X. Chen, Peter L Bartlett, et al. 2016b; J. X. Wang et al. 2016a), we formalize the adaptation procedure f_ϕ as a function of experience $(\mathbf{x}_{1:t}, r_{1:t}, \mathbf{a}_{1:t-1})$ that summarizes task-relevant information into the variable $\mathbf{c}_t$. The policy is conditioned on this updated variable as $\pi_\theta(\mathbf{a}_t|\mathbf{x}_t, \mathbf{c}_t)$ to adapt to the underlying task. By training the adaptation mechanism f_ϕ and the policy π_θ end-to-end to maximize returns of the adapted policy, meta-RL algorithms can learn policies that effectively modulate and adapt their behavior with small amounts of experience in new tasks. We formalize this meta-RL objective as:

$$\max_{\theta,\phi} \mathop{\mathbb{E}}_{\mathcal{T}\sim p(\mathcal{T})} \mathop{\mathbb{E}}_{\substack{\mathbf{x}_t\sim p_\mathcal{T}(\cdot|\mathbf{s}_t)\\ \mathbf{a}_t\sim\pi_\theta(\cdot|\mathbf{x}_t, b_t)\\ \mathbf{s}_{t+1}\sim p_\mathcal{T}(\cdot|\mathbf{s}_t,\mathbf{a}_t)\\ r_t\sim r_\mathcal{T}(\cdot|\mathbf{s}_t,\mathbf{a}_t)}} \left[\sum_{t=1}^{T} \gamma^t r_t\right] \quad \text{where} \quad \mathbf{c}_t = f_\phi(\mathbf{x}_{1:t}, r_{1:t}, \mathbf{a}_{1:t-1}). \tag{7.1}$$

To put this formulation in context with the meta-learning formulations introduced in the previous two chapters, there are two main aspects to look at. First, the objective here is the direct optimization of rewards, unlike the previous chapters where the objective was model prediction error. As discussed at the beginning of this chapter, this change corresponds with the decision to allow a model-free RL algorithm to learn a policy, rather than taking a model-based planning approach to action selection. Second, the adaptation mechanism (e.g., update rule) f_ϕ was prescribed in the previous chapters to be gradient descent, whereas it is now left to be a more general learned procedure.

In general, meta-RL methods may differ in how the adaptation procedure f_ϕ is represented (e.g., as probabilistic inference (Rakelly et al. 2019; Zintgraf et al. 2019), as a recurrent

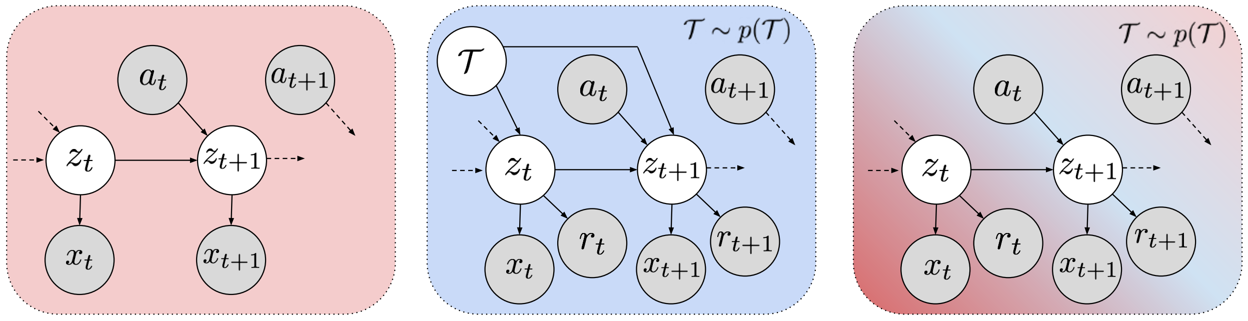


Figure 7.4: **(a)**: When only partial observations of the underlying state are available, latent dynamics models can glean state information $\mathbf{z}_t$ from a history of observations. **(b)**: Meta-RL considers a task distribution where the current task $\mathcal{T}$ is an unobserved variable that controls dynamics and rewards. **(c)**: We interpret $\mathcal{T}$ as part of $\mathbf{z}_t$, allowing us to leverage latent dynamics models for efficient image-based meta-RL.

update (Duan, John Schulman, X. Chen, Peter L Bartlett, et al. 2016b; J. X. Wang et al. 2016a), as a gradient step (Finn, Abbeel, and Levine 2017b)), how often the adaptation procedure occurs (e.g., at every timestep (Duan, John Schulman, X. Chen, Peter L Bartlett, et al. 2016b; Zintgraf et al. 2019) or once per episode (Rakelly et al. 2019; Humplik et al. 2019)), and also in how the optimization is performed (e.g., on-policy (Duan, John Schulman, X. Chen, Peter L Bartlett, et al. 2016b), off-policy (Rakelly et al. 2019)). Differences aside, these methods all typically optimize this objective end-to-end, creating a representation learning bottleneck when learning from image inputs that are ubiquitous in real-world robotics.

In the following section, we show how the latent state models discussed in Section 7.3.1 can be re-purposed for joint representation and task learning, and how this insight leads to a practical algorithm for image-based meta-RL.

7.4 MELD : Meta-RL with Latent Dynamics Models

In this section, we present MELD: an efficient algorithm for meta-RL from images. We first develop the algorithm in Section 7.4.1 and then describe its implementation in Section 7.4.2.

7.4.1 MELDing State and Task Inference for Meta-RL from Images

To see how task inference in meta-RL can be cast as latent state inference, consider the graphical models depicted in Figure 7.4. Panel (a) illustrates a standard POMDP with underlying latent state $\mathbf{z}_t$ and observations $\mathbf{x}_t$, and panel (b) depicts standard meta-RL, where the hidden task variable $\mathcal{T}$ is assumed constant throughout the episode. In the meta-RL setting, the policy must then be conditioned on both the observation and the task variable in order to adapt to a new task (see Equation 7.1). Casting this task variable as part of the latent state, panel (c) illustrates our graphical model, where the states $\mathbf{z}_t$ now contain both state and task information. In effect, we cast the task distribution over POMDPs as a POMDP itself, where the state variables now additionally capture task information. This meld allows us to draw on the rich literature of latent state models discussed in Section 7.3.1, and use them here to tackle the problem of meta-RL from sensory observations. Note that the task is not explicitly handled since it is simply another hidden state variable, providing a seamless integration of meta-RL with learning from sensory observations.

Concretely, we learn a latent state model over hidden variables by optimizing the log-likelihood of the evidence (observations and rewards) in the graphical model in Figure 7.4c:

$$\max_{\phi} \mathop{\mathbb{E}}_{\mathcal{T}\sim p(\mathcal{T})} \mathop{\mathbb{E}}_{\substack{\mathbf{x}_t\sim p_{\mathcal{T}}(\cdot|\mathbf{s}_t)\\ \mathbf{a}_t\sim\pi_\theta(\cdot|\mathbf{b}_t)\\ \mathbf{s}_{t+1}\sim p_{\mathcal{T}}(\cdot|\mathbf{s}_t,\mathbf{a}_t)\\ r_t\sim r_{\mathcal{T}}(\cdot|\mathbf{s}_t,\mathbf{a}_t)}} \left[\log p_\phi(\mathbf{x}_{1:T}, r_{1:T}|\mathbf{a}_{1:T-1})\right]. \tag{7.2}$$

Note that the only change from the latent state model from Section 7.3.1 is the inclusion of rewards as part of the observed evidence, and while this change appears simple, it enables meta-learning by allowing the hidden state to capture task information.

Posterior inference in this model then gives the agent's belief $\mathbf{b}_t = p(\mathbf{z}_t|\mathbf{x}_{1:t}, r_{1:t}, \mathbf{a}_{1:t-1})$ over latent state and task variables $\mathbf{z}_t$. Conditioned on this belief, the policy $\pi_\theta(\mathbf{a}_t|\mathbf{b}_t)$ can learn to modulate its actions and adapt its behavior to the task. Prescribing the adaptation procedure f_ϕ from Equation 7.1 to be posterior inference in our latent state model, the meta-training objective in MELD is:

$$\max_{\theta} \mathop{\mathbb{E}}_{\mathcal{T}\sim p(\mathcal{T})} \mathop{\mathbb{E}}_{\substack{\mathbf{x}_t\sim p_{\mathcal{T}}(\cdot|\mathbf{s}_t)\\ \mathbf{a}_t\sim\pi_\theta(\cdot|\mathbf{b}_t)\\ \mathbf{s}_{t+1}\sim p_{\mathcal{T}}(\cdot|\mathbf{s}_t,\mathbf{a}_t)\\ r_t\sim r_{\mathcal{T}}(\cdot|\mathbf{s}_t,\mathbf{a}_t)}} \left[\sum_{t=1}^{T} \gamma^t r_t\right] \quad \text{where} \quad \mathbf{b}_t = p(\mathbf{z}_t|\mathbf{x}_{1:t}, r_{1:t}, \mathbf{a}_{1:t-1}). \tag{7.3}$$

By melding state and task inference into a unified framework of latent state estimation, MELD inherits the same representation learning mechanism as latent state models discussed in Section 7.3.1 to enable efficient meta-RL with images.

7.4.2 Implementing MELD

Exactly computing the posterior distribution over the latent state variable is intractable, so we take a variational inference approach to maximize a lower bound on the log-likelihood objective (Wainwright and Michael Irwin Jordan 2008) in Equation 7.2. We factorize the variational posterior as $q(\mathbf{z}_{1:T}|\mathbf{x}_{1:T}, r_{1:T}, \mathbf{a}_{1:T-1}) = q(\mathbf{z}_T|\mathbf{x}_T, r_T, \mathbf{z}_{T-1}, \mathbf{a}_{T-1}) \dots q(\mathbf{z}_2|\mathbf{x}_2, r_2, \mathbf{z}_1, \mathbf{a}_1) q(\mathbf{z}_1|\mathbf{x}_1, r_1)$. With this factorization, we implement each component as a deep neural network and optimize the evidence lower bound of the joint objective, where $\mathbb{E}_{\mathbf{z}_{1:t}\sim q_\phi}\left[\log p(\mathbf{x}_{1:T}, r_{1:T}|\mathbf{a}_{1:T-1})\right] \geq$

$\mathcal{L}_{model}$, with $\mathcal{L}_{model}$ defined as:

$$\mathcal{L}_{model}(\mathbf{x}_{1:T}, r_{1:T}, \mathbf{a}_{1:T-1}) = \mathop{\mathbb{E}}_{\mathbf{z}_{1:T}\sim q_\phi} \sum_{t=1}^{T} \log p_\phi(\mathbf{x}_t|\mathbf{z}_t) + \log p_\phi(r_t|\mathbf{z}_t)$$
$$- D_{\mathrm{KL}}(q_\phi(\mathbf{z}_1|\mathbf{x}_1, r_1)\|p(\mathbf{z}_1)) - \sum_{t=2}^{T} D_{\mathrm{KL}}(q_\phi(\mathbf{z}_t|\mathbf{x}_t, r_t, \mathbf{z}_{t-1}, \mathbf{a}_{t-1})\|p_\phi(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{a}_{t-1})). \quad (7.4)$$

The first two terms encourage a rich latent representation $\mathbf{z}_t$ by requiring it to reconstruct observations and rewards, while the last term keeps the inference network consistent with latent dynamics. The first timestep posterior $q_\phi(\mathbf{z}_1|\mathbf{x}_1, r_1)$ is modeled separately from the remaining steps, and $p(\mathbf{z}_1)$ is chosen to be a fixed unit Gaussian $\mathcal{N}(0, I)$. The learned inference networks $q_\phi(\mathbf{z}_1|\mathbf{x}_1, r_1)$ and $q_\phi(\mathbf{z}_t|\mathbf{x}_t, r_t, \mathbf{z}_{t-1}, \mathbf{a}_{t-1})$, decoder networks $p_\phi(\mathbf{x}_t|\mathbf{z}_t)$ and $p_\phi(r_t|\mathbf{z}_t)$, and dynamics $p_\phi(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{a}_{t-1})$ are all fully connected networks that predict the output parameters of Gaussian distributions. We follow the architecture of the latent variable model from SLAC (A. X. Lee et al. 2019), which models two layers of latent variables. Since our observations consist of RGB camera images, we use convolutional layers in the observation encoder and decoder. Both of these networks include the same convolutional architecture (the decoder simply the transpose of the encoder) that consists of five convolutional layers. The layers have 32, 64, 128, 256, and 256 filters and the corresponding filter sizes are 5, 3, 3, 3, 4. For environments in which the robot observes a two images (such as a fixed scene image as well as a first-person image from a wrist camera), we concatenate the images together and apply rectangular filters. All other model networks are fully connected and consist of 2 hidden

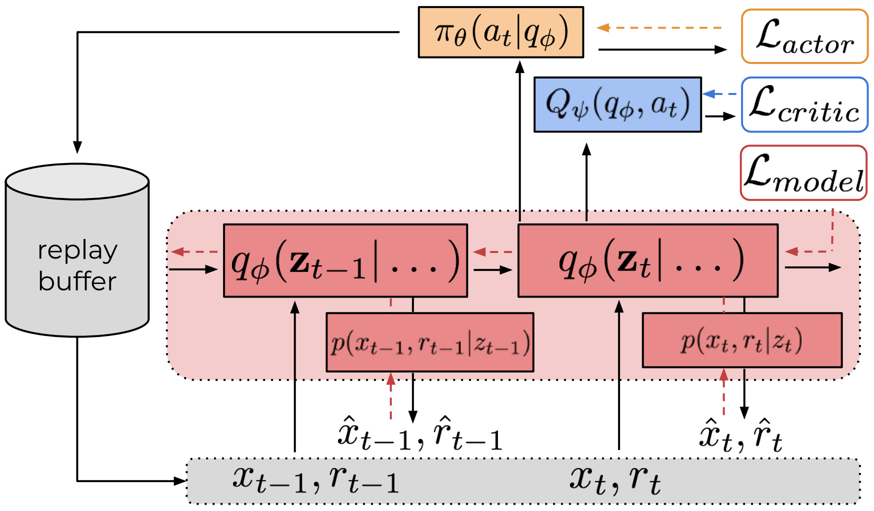


Figure 7.5: MELD meta-training alternates between collecting data with π_θ, training the latent state model using the collected data from the replay buffer, and training the actor π_θ and critic $\mathcal{Q}_\zeta$ conditioned on the belief from the current model.

layers of 32 units each. We use ReLU activations after each layer. We train all networks with the Adam optimizer with learning rate 0.001. We use the soft actor-critic (SAC) (Haarnoja, Zhou, Abbeel, et al. 2018) RL algorithm in this work, due to its high sample efficiency and performance. The SAC algorithm maximizes discounted returns as well as policy entropy via policy iteration. The actor $\pi_\theta(\mathbf{a}_t|\mathbf{b}_t)$ and the critic $Q_\psi(\mathbf{b}_t, \mathbf{a}_t)$ are conditioned on the posterior belief $\mathbf{b}_t$, modeled as fully connected neural networks, and trained as prescribed by the SAC algorithm. The critic is trained to minimize the soft Bellman error, which takes the entropy of the policy into account in the backup. We instantiate the actor and critic as fully connected networks with 2 hidden layers of 256 units each. We follow the implementation of SAC, including the use of 2 Q-networks and the tanh actor output activation.

During meta-training, MELD alternates between collecting data with the current policy, training the model by optimizing $\mathcal{L}_{model}$, and training the policy with the current model. Relevant hyper-parameters for meta-training can be found in Table 7.1. Meta-training and meta-testing are described in Algorithm 7 and Algorithm 8 respectively.

Table 7.1: Meta-training hyper-parameters

Parameter	Value
num. training tasks	30
num. eval tasks	10
actor, critic learning rates	3e-4
model learning rate	1e-4
size of per-task replay buffers $\mathcal{B}_i$	1e5
num. tasks collect data	20
num. rollouts per task	1
num. train steps per epoch	640
num. tasks sample for update	20
model batch size	512
actor, critic batch size	512

Algorithm 7 MELD Meta-training

```
Require: Training tasks $\{\mathcal{T}_i\}_{i=1...J}$ from $p(\mathcal{T})$
Require: learning rates $\eta_1, \eta_2, \eta_3$
 1: Init. model $p_\phi$, $q_\phi$, actor $\pi_\theta$, critic $Q_\zeta$
 2: Init. replay buffers $\mathcal{B}_i$ for each train task
 3: while not done do
 4:   for each task $\mathcal{T}_i$ do                          ▷ collect data
 5:     Infer $\mathbf{b}_1 = q_\phi(\mathbf{z}_1|\mathbf{x}_1, r_1)$
 6:     Step with $\mathbf{a}_1 \sim \pi_\theta(\mathbf{a}|\mathbf{b}_1)$, get $\mathbf{x}_2$, $r_2$
 7:     for $t = 2, \dots T-1$ do
 8:       Infer $\mathbf{b}_t = q_\phi(\mathbf{z}_t|\mathbf{x}_t, r_t, \mathbf{z}_{t-1}, \mathbf{a}_{t-1})$
 9:       Step $\mathbf{a}_t \sim \pi_\theta(\mathbf{a}|\mathbf{b}_t)$, get $\mathbf{x}_{t+1}$, $r_{t+1}$
10:     end for
11:     Add data $\{\mathbf{x}_{1:T}, r_{1:T}, \mathbf{a}_{1:T-1}\}$ to $\mathcal{B}^i$
12:   end for
13:   for step in train steps do
14:     for each task $\mathcal{T}_i$ do
15:       Sample $\{\mathbf{x}_{1:T}, r_{1:T}, \mathbf{a}_{1:T-1}\} \sim \mathcal{B}_i$
16:       Infer beliefs $\mathbf{b}_{1:T} = \{q_\phi(\mathbf{z}_t| \dots)\}_{1:T}$
17:       Predict reconstructions $\{\hat{\mathbf{x}}_t, \hat{r}_t\}_{1:T}$
18:       $\mathcal{L}_m^i = \mathcal{L}_{model}(\{\mathbf{x}_t, \hat{\mathbf{x}}_t, r_t, \hat{r}_t, \mathbf{a}_t\}_{1:T})$
19:     end for
20:     $\phi \leftarrow \phi - \eta_1 \nabla_\phi \sum_i \mathcal{L}_m^i$           ▷ train model
21:     Update $\theta, \zeta$ with SAC($\eta_2, \eta_3$)              ▷ train AC
22:   end for
23: end while
```

Algorithm 8 MELD Meta-testing

```
Require: Test task $\mathcal{T} \sim p(\mathcal{T})$
1: Infer $\mathbf{b}_1 = q_\phi(\mathbf{z}_1|\mathbf{x}_1, r_1)$
2: Step with $\mathbf{a}_1 \sim \pi_\theta(\mathbf{a}|\mathbf{b}_1)$, get $\mathbf{x}_2$, $r_2$
3: for $t = 2, \dots, T-1$ do
4:   Infer $\mathbf{b}_t = q_\phi(\mathbf{z}_t|\mathbf{x}_t, r_t, \mathbf{z}_{t-1}, \mathbf{a}_{t-1})$
5:   Step with $\mathbf{a}_t \sim \pi_\theta(\mathbf{a}|\mathbf{b}_t)$, get $\mathbf{x}_{t+1}$, $r_{t+1}$
6: end for
```

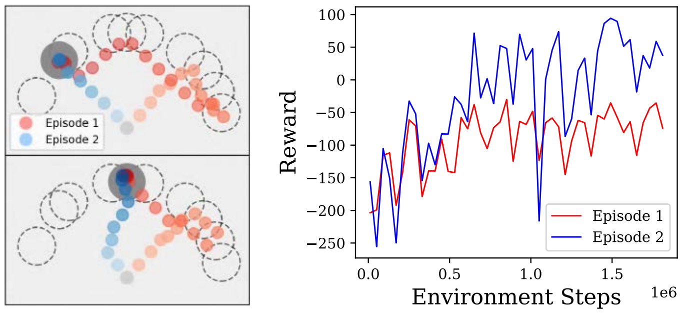


Figure 7.6: Image-based 2D navigation: trajectory traces (left) and rewards (right) of meta-learned exploration to find goal in Episode-1 (red) and persisting information across episodes going directly there in Episode-2 (blue).

7.5 Simulated and Real-world Results of Meta-RL from Images

In our experiments, we aim to answer the following questions:

1. Can MELD capture and propagate state and task information over time to explore effectively in a new task?
2. How does MELD compare to prior meta-RL methods in enabling fast acquisition of new skills at test time in challenging simulated control problems?
3. Can MELD enable real robots to quickly acquire skills via meta-RL from images?

7.5.1 Learning to Explore with MELD

We first present a didactic image-based 2D navigation problem to illustrate how MELD can learn extended exploration strategies and adapt to a new task, within a single episode. The task distribution consists of goal locations located along a semi-circle around the start state. The agent receives inputs in the form of 64x64 image observations and rewards that are non-zero only upon reaching the correct goal. Because meta-training with sparse rewards is challenging, we make use of dense rewards (e.g., distance from goal at each step) during meta-training, as in prior works (Abhishek Gupta et al. 2018; Rakelly et al. 2019; Mendonca et al. 2019). In this hybrid instantiation of MELD, the reward input to the model is the sparse reward, as during test time, but dense reward is used during training for the actor-critic and reconstruction losses.

Intuitively, the latent state model objective encourages the belief to capture and propagate uncertainty over task; for example, after exploring the right half of the semi-circle and receiving no rewards, the belief should capture that the goal lies elsewhere. Conditioned on this belief, the agent can learn an exploration policy that searches regions until the goal is found. We observe this behavior experimentally, see Figure 7.6a. By updating the posterior belief at each step, MELD is able to find the goal within $10 - 20$ steps of the first episode, rather than requiring multiple full episodes, as in the case of posterior sampling methods that hold the task variable constant across each episode (Rakelly et al. 2019; Abhishek Gupta et al. 2018). Once the goal is found, MELD can navigate to it immediately (Figure 7.6b) in the next episode by persisting latent variables across episodes (as shown by the higher "episode-2" rewards).

7.5.2 Fast Skill Acquisition at Test Time in Simulated Environments

In our next set of experiments, we evaluate MELD on the four simulated image-based continuous control problems in Figure 7.7. We describe the simulated experimental setup below and then present the results of fast acquisition.

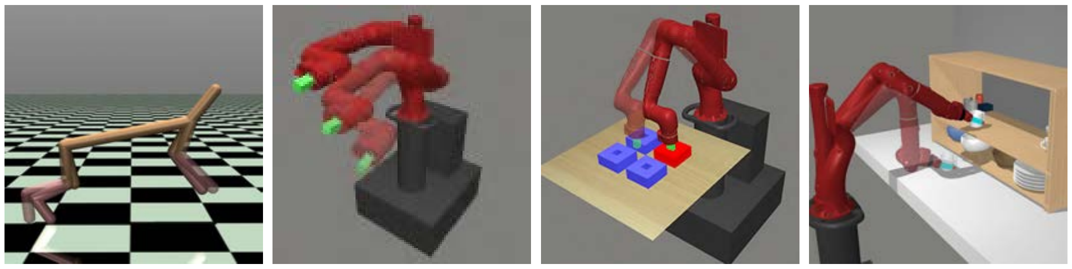


Figure 7.7: Simulated locomotion and manipulation meta-RL environments in the MuJoCo simulator (Emanuel Todorov, Erez, and Tassa 2012a): running at different velocities, reaching to varying goals, inserting a peg into varying boxes in varying locations, and putting away mugs of varying weights to varying locations. The goal for each task is available to the robot only via per-timestep rewards, but is illustrated here for visualization purposes.

7.5.2.1 Simulation Environment Details

In (a) Cheetah-vel, each task is a different target running velocity for the 6-DoF legged robot. The remainder of the problems use a 7-DOF Sawyer robotic arm. In (b) Reacher, each task is a different goal position for the end-effector. In (c) Peg-insertion, the robot must insert the peg into the correct box, where each task varies the goal box as well as locations of all four boxes. In (d) Shelf-placing, each task varies the weight (dynamics change) and target location (reward change) of a mug that the robot must move from the table to the shelf.

In the Cheetah-vel environment, we control the robot by commanding the torques on the robot's 6 joints. The reward function consists of the difference between the target velocity v_{target} and the current velocity v_x of the cheetah's center of mass, as well a small control cost on the torques sent to the joints:

$$r_{\text{cheetal-vel}} = -|v_x - v_{\text{target}}| + 0.01||a_t||_2. \tag{7.5}$$

The episode length is 50 time steps, and the observation consists of a single 64x64 pixel image from a tracking camera (as shown in Fig. 7.8a), which sees a view of the full cheetah.

In all three Sawyer environments, we control the robot by commanding joint delta-positions for all 7 joints. The reward function indicates the difference between the current end-effector pose x_{ee} and a goal pose x_{goal}, as follows:

$$r_{\text{sawyer-envs}} = -(d^2 + \log(d + 1\text{e-}5)), \quad \text{where } d = ||x_{\text{ee}} - x_{\text{goal}}||_2. \tag{7.6}$$

This reward function encourages precision near the goal, which is particularly important for the peg insertion task. We impose a maximum episode length of 40 time steps for these environments. The observations for all three of these environments consist of two images concatenated to form a 64x128 image: one from a fixed scene camera, and one from a wrist-mounted first-person view camera. These image observations for each environment are shown in Fig. 7.8b-d. The simulation time step and control frequency for each of these simulated environments is listed in Table 7.2.

Table 7.2: Simulation Environments

Environment	Sim. time step	Control freq.
Cheetah-vel	0.01	10Hz
Reacher	0.0025	4Hz
Peg-insert	0.0025	4Hz
Shelf-placing	0.0025	4Hz

For all environments, we train with 30 meta-training tasks and evaluate on 10 meta-test tasks from the same distribution that are not seen during training.

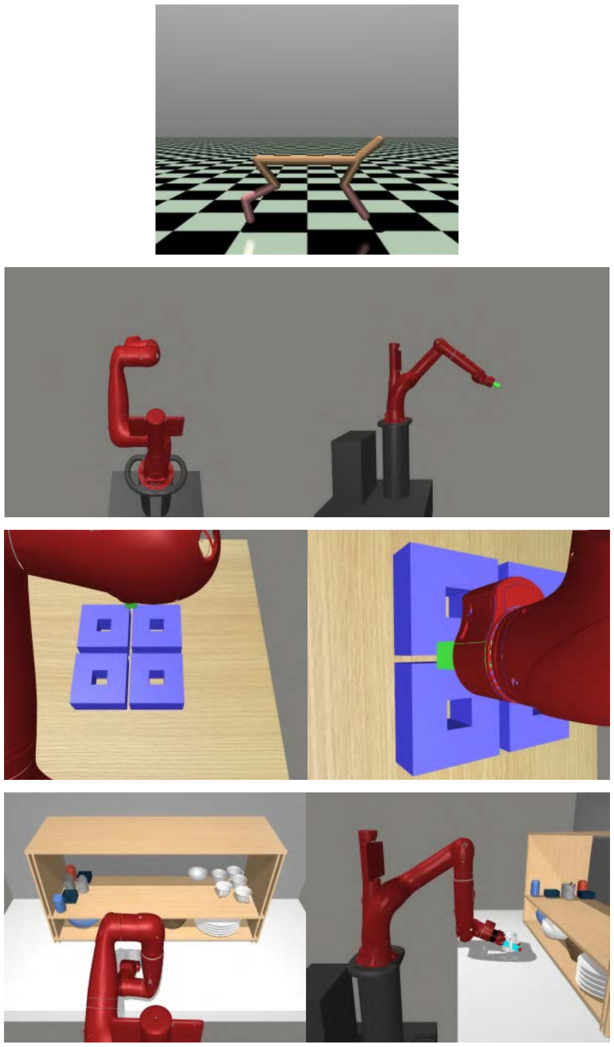


Figure 7.8: 64x64 and 64x128 image observations, seen as input by MELD for (a) Cheetah-vel, (b) Reacher, (c) Peg-insertion, and (d) Shelf-placing.

7.5.2.2 Comparisons

We compare MELD to two representative state-of-the-art meta-RL algorithms, PEARL (Rakelly et al. 2019) and RL^2 (Duan, John Schulman, X. Chen, Peter L Bartlett, et al. 2016b). PEARL models a belief over a probabilistic latent task variable as a function of un-ordered batches of transitions, and conditions the policy on both the current observation and this inferred task belief. Unlike MELD, this algorithm assumes an exploration phase of several trajectories in the new task to gather information before it adapting, so to get its best performance, we evaluate only *after* this exploration phase. RL^2 models the policy as a recurrent network that directly maps observations, actions, and rewards to actions. To apply PEARL and RL^2

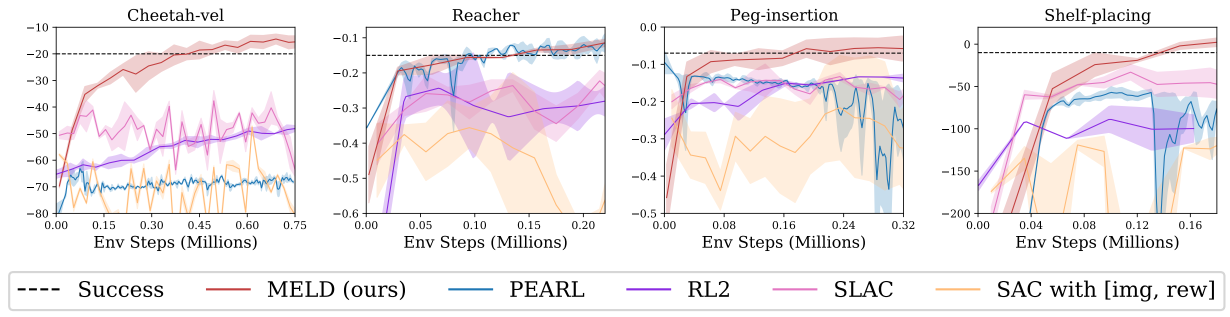


Figure 7.9: Rewards on test tasks versus meta-training environment steps, comparing MELD to prior methods. See text for definitions of these success metrics, and discussion on the scale of these metrics.

to environments with image observations, we augment them with the same convolutional encoder architecture used by MELD. We also compare MELD to the end-to-end RL algorithm SAC (Haarnoja, Zhou, Abbeel, et al. 2018), with a sequence of observations and rewards as input. This input provides SAC with enough information to solve these dense reward tasks, so this comparison reveals the importance of the explicit representation learning and meta-learning mechanisms in MELD. Finally, to verify the need for meta-learning on these held-out tasks, we compare to SLAC (A. X. Lee et al. 2019), which infers a latent state from observations but does *not* perform meta-learning.

7.5.2.3 Results of Fast Skill Acquisition

Figure 7.12 shows average performance on meta-test tasks over the course of meta-training, across 3 random seeds per algorithm. We define a success metric for each environment that correlates with qualitatively solving the task: Cheetah-vel: within 0.2m/s of target velocity, Reacher: within 10cm of goal, Peg: complete insertion with $5cm$ variation possible inside the site, Shelf: mug within 5cm of goal. Although these were selected thresholds, we emphasize that all prior methods except PEARL on the reacher environment objectively failed on these image-based meta-RL tasks; as can be seen in the videos on the project website [2], they were not simply a threshold-choice away from success. We use these success metrics because task reward is often misleading when averaged across a distribution of tasks; in peg-insertion, for example, the numerical difference between always inserting the peg correctly versus never inserting it can be as low as 0.1, since the distance between the center of the goal distribution and each goal is quite small and accuracy is required.

MELD achieves the highest performance in each environment and is the only method to fully solve Cheetah-vel, Peg-insertion, and Shelf-placing. Notably, end-to-end RL with SAC performs poorly across all of these image-based meta-RL tasks. The SLAC baseline also fails in this meta-RL setting, as expected, with qualitative behavior of always executing a single "average" motion, such as reaching toward a mean goal location and running at a

[2] `https://sites.google.com/view/meld-lsm/home`

medium speed. While PEARL infers a latent task variable, it relies on the current observation alone for state information; thus it is able to succeed only on Reacher, which is the only environment where enough information can be inferred from a single image. While RL^2 is capable of propagating both state and task information over time, we observe that it overfits heavily to training tasks and struggles on evaluation tasks.

The experiments above consider the standard meta-RL paradigm, where the agent adapts to one test task at a time. However, many realistic scenarios consist of a sequence of tasks. For example, consider a robot moving a mug filled with liquid; if some of the liquid spills, the robot must adapt to the new dynamics of the lighter mug to finish the job. Unlike previous methods that hold task variables constant across episodes (Abhishek Gupta et al. 2018; Rakelly et al. 2019), the ability to update its belief at each step allows MELD to adapt quickly to task changes. We evaluate MELD in the Cheetah-vel environment on a sequence of 3 different target velocities within a single episode and observe in Figure 7.10 that MELD adapts to track each velocity within a few time steps.

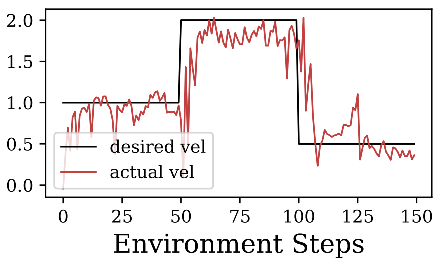


Figure 7.10: MELD tracking changing velocity targets for Cheetah-vel.

7.5.3 MELD in the Real World

We now evaluate MELD on two real-world robotic systems: a 7-DoF Sawyer arm performing peg insertion, and a 6-DoF WidowX arm performing ethernet cable insertion. The policy sends joint velocity controls over a ROS interface to a low-level PID controller to move the joints of each robot. The reward function for all tasks is the sum of the L2-norms of translational and rotational distances between the pose of the object in the end-effector and a goal pose. Note that the goal is not provided to the agent, but must be inferred from its history of observations and rewards. The agent's observations are concatenated images from two webcams (Figure 7.11): one fixed view and one first-person view from a wrist-mounted camera.

We first test if the latent state model learned by MELD can reason about state and task information and perform challenging image-based control in the real world. On the Sawyer robot, we learn precise peg insertion where the task distribution consists of three tasks, each corresponding to a different target box. The robot succeeds on all three tasks after training on 4 hours of data ($60,000$ samples at 4Hz), as shown in Figure 7.13.

Next, to demonstrate the fast acquisition of *new* tasks via meta-learning with MELD in the real world, we set up a more realistic problem of ethernet cable insertion, where the task distribution consists of inserting the cable into different ports in a router that also varies in location and orientation. To instrument these distinct tasks in the real world, we build an automatic task reset mechanism that moves and rotates the router. This mechanism controls the translational and rotational displacement of the network switch. The network

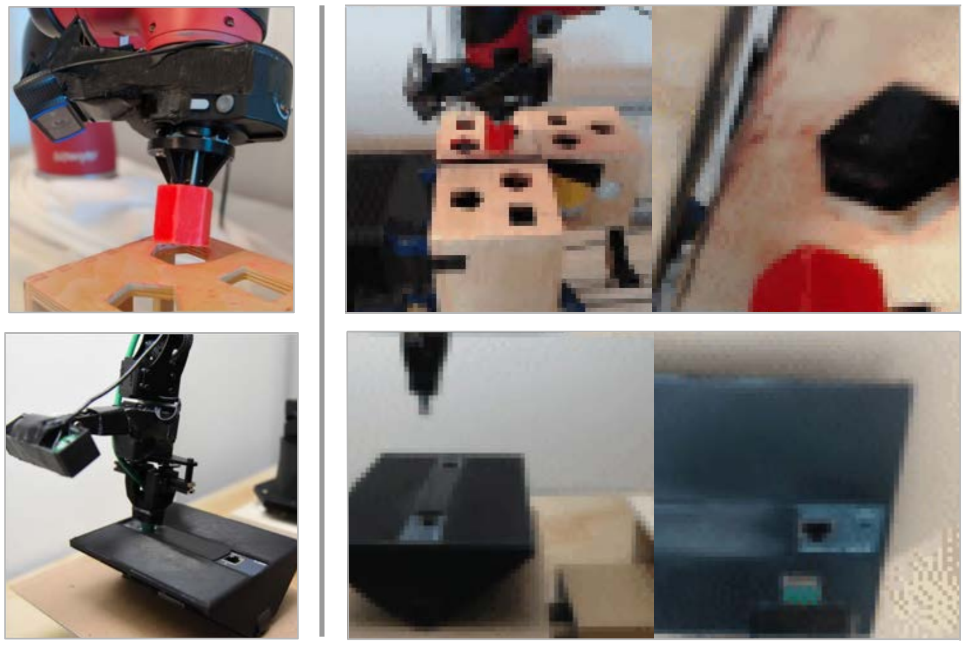


Figure 7.11: (**Left**) The peg and ethernet cable insertion problems. (**Right**) 64x128 image observations seen by the Sawyer for peg insertion and the WidowX for ethernet cable insertion.

switch(A) is mounted to a 3D printed housing(B) with gear attached. We control the rotation of the housing through motor 1. This setup is then mounted on top of a linear rail(C) and motor 2 controls its translational displacement through a timing pulley. In our experiments, the training task distribution consisted of 20 different tasks, where each task was randomly assigned from a rotational range of 16 degrees and a translational range of 2cm.

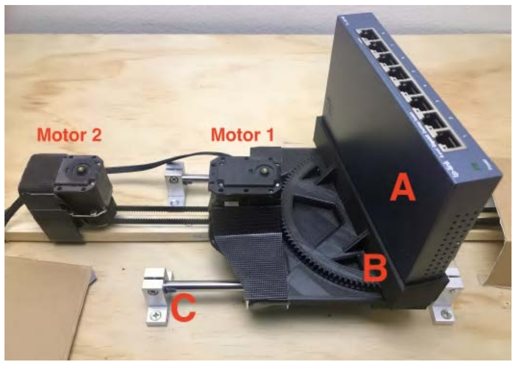


Figure 7.12: Automatic task reset mechanism for ethernet cable insertion: The network switch is rotated and translated by a series of motors in order to generate different tasks for meta-learning. This allows our meta-learning process to be entirely automated, without needing human intervention to reset either the robot or the task at the beginning of each rollout.

After training across these 20 meta-training tasks using a total of 4 hours $(50,000$ samples

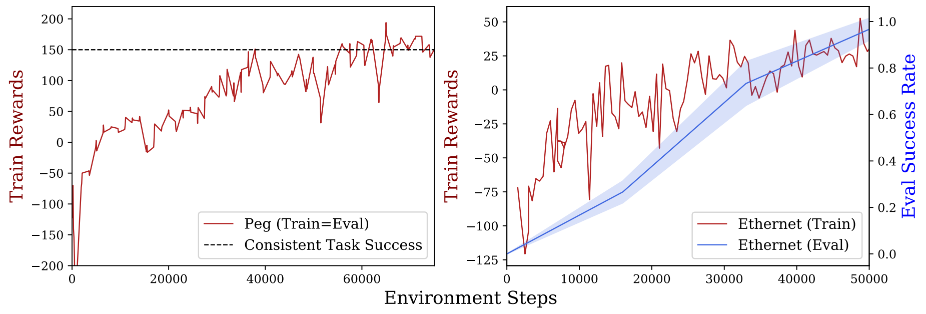


Figure 7.13: In red, rewards on train tasks during meta-training for Sawyer peg insertion (**left**) and WidowX ethernet cable insertion (**right**). In blue, success rate of ethernet cable insertion on unseen eval tasks.

at 3.3Hz) worth of data, MELD achieves a success rate of 96% over three rounds of evaluation in each of the 10 randomly sampled evaluation tasks that were not seen during training. Videos and more experiments can be found online[3].

7.6 Discussion

In this chapter, we drew upon the insight that meta-RL can be cast into the framework of latent state inference. This allowed us to combine the adaptation and fast skill-acquisition capabilities of meta-learning with the efficiency of unsupervised latent state models when learning from raw image observations. Based on this principle, we designed MELD, a practical algorithm for meta-RL with image observations. We showed that MELD outperforms prior methods on simulated locomotion and manipulation tasks with image observations, and is efficient enough to perform meta-RL directly from images in the real world. By operating directly from raw image observations rather than assuming access to the underlying state information, as well as by explicitly enabling the *fast acquisition* of new skills, the work in this chapter takes a critical step toward the deployment of intelligent and adaptive robots into the real world.

[3]`https://sites.google.com/view/meld-lsm/home`

Chapter 8

Conclusion

In this thesis, we presented the idea that model-based deep RL provides an efficient and effective framework for making sense of the world; and in turn, this ability to make sense of the world allows for reasoning and adaptation capabilities that give rise to the generalization that is necessary for successful operation in the real world. We started in chapter 2 by building up a model-based deep RL framework and demonstrating that it allows for efficient and effective autonomous skill acquisition for various agents in simulation. We also demonstrated the ability to repurpose the learned models to execute a variety of paths at test time. Next, in chapter 3, we extended this framework to enable locomotion with a 6-DoF legged robot in the real world by learning image-conditioned models that are able to address various types of terrains. Then, in chapter 4, we scaled up both modeling and control aspects of our model-based deep RL framework and demonstrated efficient and effective learning of challenging dexterous manipulation skills with a 24-DoF anthropomorphic hand, both in simulation and in the real world. We then switched focus to the inevitable mismatch between an agent's training conditions and the test conditions in which it may actually be deployed. In chapter 5, we presented a meta-learning algorithm within our model-based deep RL framework to enable online adaptation of large, high-capacity models using only small amounts of data from new tasks. We demonstrated these fast adaptation capabilities in both simulation and the real-world, with experiments such as a 6-legged robot adapting online to an unexpected payload or suddenly losing a leg. Next, in chapter 6, we further improved these adaptation capabilities by formulating an online learning approach as an expectation maximization problem. By maintaining and building a task distribution with a mixture of models, this algorithm demonstrated both generalization as well as specialization. Finally, in chapter 7, we further extended the capabilities of our robotic systems by enabling the agents to reason directly from raw image observations. By combining efficiency benefits from representation learning with the adaptation capabilities of meta-RL, we presented a unified framework for effective meta-RL from images. This approach demonstrated not only efficient meta-learning due to the learned latent dynamics model, but it also demonstrated fast acquisition of new skills at test time due to the meta-learned procedure for performing online inference. We demonstrated these results on robotic arms in the real world performing peg insertion and ethernet cable insertion to varying targets, thus showing the fast acquisition of new skills directly from raw image observations in the real world.

Looking ahead, we see a few promising directions for future work in this area of model-based deep RL for robotic systems. We discuss some open questions and future directions below.

Incorporating other sensing modalities: Thanks to the large amounts of prior work in computer vision research, great strides have been made in learning directly from image inputs. While there is still work to be done in that direction, there has been a relatively much smaller effort in incorporating other sources of sensing modalities such as touch and sound. These additional senses can be critical for many tasks in the real world, such as buttoning a shirt or braiding our own hair, where most of the work happens without vision to guide our actions. Even tasks that do incorporate vision can benefit from additional sensing modalities, including operating in environments with occlusions, or reasoning further about details that are not obvious from vision alone, such as the weight, composition, or interactions between objects.

What to model and predict: Throughout this thesis, we talked about the importance and effectiveness of learning predictive models. However, in the real world, it's often not obvious what to predict. In most of the chapters in this thesis, we had some "state" representation and thus formulated the problem as predicting the next state. However, knowing what to include in this state ahead of time is a challenge in the real world; it is impossible to include the position of every possible item that may be relevant, and then predict the future value of that entry. Instead, the last chapter presented an algorithm that operated directly from image inputs. As is common for representation learning approaches, however, we used pixel-based reconstruction loss as part of the training objective. These types of pixel-based predictions are unfortunately challenging as well as severely limiting. Consider, for example, a robot arm that might push a bowl of cereal off of the edge of the table. In this case, what we care to predict is that the bowl will break and the milk will splatter. Predicting each pixel value of the resulting image is not at all appropriate in these types of situations, because trying to figure out where every drop of milk or every piece of shattered glass might end up is not only unimportant, but also impossible. Recent work has shown promising alternatives for pixel-based reconstruction losses, such as contrastive learning (Srinivas, Laskin, and Abbeel 2020; T. Chen et al. 2020) and other similarity-based (R. Zhang et al. 2018) learning approaches. Additionally, there may also be effective ways to combine autonomous image segmentation processes with predictive processes on those segments. The development and integration of such approaches – to produce predictive models that don't rely on fixed state representations or on pixel-based predictions – are a promising direction for future work.

Planning over long horizons: RL can be described as decision making to control a system in order to accomplish some given goal. This problem statement encompasses many types of challenges, from flying a quadcopter to unloading a dishwasher. While we can reason over short time horizons for tasks such as maneuvering a quadcopter to follow a given trajectory or figuring out in-hand manipulation of an object, we must reason over a much longer time horizon to do multi-stage reasoning or tasks with sparse rewards. Consider setting a dinner table or cleaning an entire room, where "success" cannot be achieved without reasoning over a long time horizon and doing many intermediate things before actually achieving that success. Although model-based deep RL algorithms can now solve tasks that are challenging in many ways, from locomotion (M. P. Deisenroth, Calandra, et al. 2012; Hester, Quinlan, and Stone 2010; Morimoto and Atkeson 2003) to manipulation (M. P. Deisenroth, Englert, et al. 2014; Depraetere et al. 2014; M. P. Deisenroth, Carl Edward Rasmussen, and Fox 2011; Atkeson 1998), these approaches still struggle with compounding model errors when trying to scale to multi-step reasoning or long-horizon planning tasks. Work in the areas of gradient-based optimization techniques (Mordatch, Emanuel Todorov, and Popović 2012; Ratliff et al. 2009), generating sub-goals (McGovern and Barto 2001), learning options (Stolle and Precup 2002), generating successor representations (Kulkarni et al. 2016), and hierarchical reinforcement learning (Peng et al. 2017; Vezhnevets et al. 2017; Stulp and Schaal 2011; Kaelbling, Littman, and A. W. Moore 1996) have shown promise in this area of addressing long-horizon tasks. Integrating, developing, and applying these ideas to enable reasoning over varying time horizons and autonomous behaviors in the dynamic conditions of real-world environments is a critical direction for future work.

Bibliography

[1] Pulkit Agrawal. *Computational sensorimotor learning.* eScholarship, University of California, 2018.

[2] Ilge Akkaya et al. “Solving rubik’s cube with a robot hand”. In: *arXiv preprint arXiv:1910.07113* (2019).

[3] Maruan Al-Shedivat et al. “Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments”. In: *CoRR* abs/1710.03641 (2017). arXiv: `1710.03641`.

[4] Rahaf Aljundi, Punarjay Chakravarty, and Tinne Tuytelaars. “Expert Gate: Lifelong Learning With a Network of Experts”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 2017.

[5] Richard Altendorfer, Daniel E Koditschek, and Philip Holmes. “Stability analysis of a clock-driven rigid-body SLIP model for RHex”. In: *The International Journal of Robotics Research* 23.10-11 (2004), pp. 1001–1012.

[6] Richard Altendorfer, Ned Moore, et al. “RHex: A biologically inspired hexapod runner”. In: *Autonomous Robots* 11.3 (2001), pp. 207–213.

[7] Sheldon Andrews and Paul G Kry. “Goal directed multi-finger manipulation: Control policies and analysis”. In: *Computers & Graphics* 37.7 (2013), pp. 830–839.

[8] Marcin Andrychowicz, Bowen Baker, et al. “Learning dexterous in-hand manipulation”. In: *arXiv preprint arXiv:1808.00177* (2018).

[9] Marcin Andrychowicz, Misha Denil, et al. “Learning to learn by gradient descent by gradient descent”. In: *CoRR* abs/1606.04474 (2016). arXiv: `1606.04474`.

[10] Karol Arndt et al. “Meta Reinforcement Learning for Sim-to-real Domain Adaptation”. In: *arXiv preprint arXiv:1909.12906* (2019).

[11] Kavosh Asadi. “Strengths, weaknesses, and combinations of model-based and model-free reinforcement learning”. PhD thesis. University of Alberta, 2015.

[12] Karl J Åström and Björn Wittenmark. *Adaptive control.* Courier Corporation, 2013.

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

[14] Christopher G Atkeson. “Nonparametric model-based reinforcement learning”. In: *Advances in neural information processing systems*. 1998, pp. 1008–1014.

[15] Yunfei Bai and C Karen Liu. “Dexterous manipulation using both palm and fingers”. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 1560–1565.

[16] Renee Baillargeon, Elizabeth S Spelke, and Stanley Wasserman. “Object permanence in five-month-old infants”. In: *Cognition* 20.3 (1985), pp. 191–208.

[17] Bowen Baker et al. “Designing neural network architectures using reinforcement learning”. In: *arXiv preprint arXiv:1611.02167* (2016).

[18] Timothy D Barfoot. *State estimation for robotics*. Cambridge University Press, 2017.

[19] George Bekey and Kenneth Y Goldberg. *Neural networks in robotics*. Springer US, 1992.

[20] Samy Bengio et al. “On the optimization of a synaptic learning rule”. In: *Optimality in Artificial and Biological Neural Networks*. 1992.

[21] Yoshua Bengio, Samy Bengio, and Jocelyn Cloutier. *Learning a synaptic learning rule*. Université de Montréal, Département d’informatique et de recherche opérationnelle, 1990.

[22] Fernando L Garcia Bermudez et al. “Performance analysis and terrain classification for a legged robot over rough terrain”. In: *IROS*. 2012.

[23] Antonio Bicchi and Vijay Kumar. “Robotic grasping and contact: A review”. In: *ICRA*. Vol. 1. IEEE. 2000, pp. 348–353.

[24] David M Blei, Andrew Y Ng, and Michael I Jordan. “Latent dirichlet allocation”. In: *Journal of machine Learning research* 3.Jan (2003), pp. 993–1022.

[25] Joschka Boedecker et al. “Approximate real-time optimal control based on sparse gaussian process models”. In: *ADPRL*. 2014.

[26] Alessandro Bonardi, Stephen James, and Andrew J Davison. “Learning One-Shot Imitation from Humans without Humans”. In: *arXiv preprint arXiv:1911.01103* (2019).

[27] Zdravko I Botev et al. “The cross-entropy method for optimization”. In: *Handbook of statistics*. Vol. 31. Elsevier, 2013, pp. 35–59.

[28] Léon Bottou. “Online learning and stochastic approximations”. In: *On-line learning in neural networks* 17.9 (1998), p. 142.

[29] Konstantinos Bousmalis et al. “Using simulation and domain adaptation to improve efficiency of deep robotic grasping”. In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 4243–4250.

[30] Daniel A Braun et al. “Learning optimal adaptation strategies in unpredictable motor tasks”. In: *Journal of Neuroscience* (2009).

[31] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: `arXiv:1606.01540`.

[32] Tamara Broderick et al. "Streaming variational bayes". In: *Advances in Neural Information Processing Systems*. 2013, pp. 1727–1735.

[33] Austin D Buchan, Duncan W Haldane, and Ronald S Fearing. "Automatic identification of dynamic piecewise affine models for a running robot". In: *IEEE/RSJ Int. Conf. on Intell. Robots and Systems*. 2013, pp. 5600–5607.

[34] Jörg Butterfaß et al. "DLR-Hand II: Next generation of a dextrous robot hand". In: *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*. Vol. 1. IEEE. 2001, pp. 109–114.

[35] Katie Byl. "Metastable legged-robot locomotion". PhD thesis. Massachusetts Institute of Technology, 2008.

[36] Roberto Calandra et al. "An experimental comparison of Bayesian optimization for bipedal locomotion". In: *IEEE Int. Conf. on Robotics and Automation*. 2014, pp. 1951–1958.

[37] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. "Deep blue". In: *Artificial intelligence* 134.1-2 (2002), pp. 57–83.

[38] Carlos Sebastian Casarez. *Tail-Augmented Self-Righting and Turning of a Dynamic Legged Millirobot*. University of California, Berkeley, 2018.

[39] Nikhil Chavan-Dafle, Rachel Holladay, and Alberto Rodriguez. "In-hand manipulation via motion cones". In: *arXiv preprint arXiv:1810.00219* (2018).

[40] Yevgen Chebotar et al. "Combining Model-Based and Model-Free Updates for Trajectory-Centric Reinforcement Learning". In: *arXiv preprint arXiv:1703.03078* (2017).

[41] Ting Chen et al. "A simple framework for contrastive learning of visual representations". In: *arXiv preprint arXiv:2002.05709* (2020).

[42] Sonia Chernova and Manuela Veloso. "An evolutionary approach to gait learning for four-legged robots". In: *IEEE/RSJ Int. Conf. on Intell. Robots and Systems*. Vol. 3. 2004, pp. 2562–2567.

[43] Kurtland Chua et al. "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*. 2018, pp. 4759–4770.

[44] Kurtland Chua et al. "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models". In: *arXiv preprint arXiv:1805.12114* (2018).

[45] Jonathan E Clark et al. "Biomimetic design and fabrication of a hexapedal running robot". In: *IEEE Int. Conf. on Robotics and Automation*. Vol. 4. 2001, pp. 3643–3649.

[46] Holk Crusea et al. "Walknet—a biologically inspired network to control six-legged walking". In: *Neural Networks*. 1998.

[47] Xingye Da, Ross Hartley, and Jessy Grizzle. “Supervised Learning for Stabilizing Underactuated Bipedal Robot Locomotion, with Outdoor Experiments on the Wave Field”. In: *ICRA*. 2017.

[48] M. Deisenroth and C. Rasmussen. “A model-based and data-efficient approach to policy search”. In: *ICML*. 2011.

[49] Marc Peter Deisenroth, Roberto Calandra, et al. “Toward fast policy search for learning legged locomotion”. In: *IROS*. 2012.

[50] Marc Peter Deisenroth, Peter Englert, et al. “Multi-task policy search for robotics”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2014, pp. 3876–3881.

[51] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. “A survey on policy search for robotics”. In: *Foundations and Trends in Robotics*. 2013.

[52] Marc Peter Deisenroth and Jan Peters. “Solving nonlinear continuous state-action-observation POMDPs for mechanical systems with Gaussian noise”. In: ().

[53] Marc Peter Deisenroth, Carl Edward Rasmussen, and Dieter Fox. “Learning to control a low-cost manipulator using data-efficient reinforcement learning”. In: (2011).

[54] Marc Deisenroth and Carl E Rasmussen. “PILCO: A model-based and data-efficient approach to policy search”. In: *International Conference on machine learning (ICML)*. 2011, pp. 465–472.

[55] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *CVPR*. 2009.

[56] B Depraetere et al. “Comparison of model-free and model-based methods for time optimal hit control of a badminton robot”. In: *Mechatronics* 24.8 (2014), pp. 1021–1030.

[57] Andreas Doerr et al. “Model-Based Policy Search for Automatic Tuning of Multivariate PID Controllers”. In: *CoRR* abs/1703.02899 (2017). arXiv: `1703.02899`. URL: `http://arxiv.org/abs/1703.02899`.

[58] Mehmet R Dogar and Siddhartha S Srinivasa. “Push-grasping with dexterous hands: Mechanics and a method”. In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE. 2010, pp. 2123–2130.

[59] Julien Doyon and Habib Benali. “Reorganization and plasticity in the adult brain during learning of motor skills”. In: *Current opinion in neurobiology* 15.2 (2005), pp. 161–167.

[60] Yan Duan, Xi Chen, et al. “Benchmarking deep reinforcement learning for continuous control”. In: *ICML*. 2016.

[61] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, et al. “RL2: Fast Reinforcement Learning via Slow Reinforcement Learning”. In: *CoRR* abs/1611.02779 (2016). arXiv: `1611.02779`.

[62] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, et al. "RL2: Fast Reinforcement Learning via Slow Reinforcement Learning". In: *arXiv preprint arXiv:1611.02779* (2016).

[63] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, et al. "RL2: Fast Reinforcement Learning via Slow Reinforcement Learning". In: *arXiv:1611.02779* (2016).

[64] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* (2011).

[65] David A Ferrucci. "Introduction to "this is watson"". In: *IBM Journal of Research and Development* 56.3.4 (2012), pp. 1–1.

[66] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *International Conference on Machine Learning (ICML)* (2017).

[67] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *arXiv preprint arXiv:1703.03400* (2017).

[68] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *CoRR* abs/1703.03400 (2017). arXiv: `1703.03400`.

[69] Chelsea Finn and Sergey Levine. "Deep visual foresight for planning robot motion". In: *ICRA*. 2017.

[70] Chelsea Finn and Sergey Levine. "Meta-Learning and Universality: Deep Representations and Gradient Descent can Approximate any Learning Algorithm". In: *CoRR* abs/1710.11622 (2017). arXiv: `1710.11622`.

[71] Chelsea Finn, Xin Yu Tan, et al. "Deep spatial autoencoders for visuomotor learning". In: *ICRA*. IEEE. 2016, pp. 512–519.

[72] Chelsea Finn, Tianhe Yu, et al. "One-Shot Visual Imitation Learning via Meta-Learning". In: *CoRL*. 2017, pp. 357–368.

[73] J Randall Flanagan and Alan M Wing. "Modulation of grip force with load force during point-to-point arm movements". In: *Experimental Brain Research* 95.1 (1993), pp. 131–143.

[74] Peter Florence, Lucas Manuelli, and Russ Tedrake. "Self-supervised correspondence in visuomotor policy learning". In: *IEEE Robotics and Automation Letters* 5.2 (2019), pp. 492–499.

[75] Meire Fortunato, Charles Blundell, and Oriol Vinyals. "Bayesian recurrent neural networks". In: *arXiv preprint arXiv:1704.02798* (2017).

[76] Justin Fu, Sergey Levine, and Pieter Abbeel. "One-Shot Learning of Manipulation Skills with Online Dynamics Adaptation and Neural Network Priors". In: *CoRR* abs/1509.06841 (2015). arXiv: `1509.06841`.

[77] Yarin Gal, Rowan Thomas McAllister, and Carl Edward Rasmussen. “Improving PILCO with bayesian neural network dynamics models”. In: *Data-Efficient Machine Learning workshop*. 2016.

[78] Sebastien Gay, Jose Santos-Victor, and Auke Ijspeert. “Learning Robot Gait Stability using Neural Networks as Sensory Feedback Function for Central Pattern Generators”. In: 2013.

[79] Carles Gelada et al. “DeepMDP: Learning Continuous Latent Space Models for Representation Learning”. In: *ICML*. 2019, pp. 2170–2179.

[80] Ali Ghadirzadeh et al. “Deep predictive policy training using reinforcement learning”. In: *IROS*. IEEE. 2017, pp. 2351–2358.

[81] Mark A Gluck, Eduardo Mercado, and Catherine E Myers. *Learning and memory: From brain to behavior*. Macmillan Higher Education, 2007.

[82] Ruben Grandia, Diego Pardo, and Jonas Buchli. “Contact Invariant Model Learning for Legged Robot Locomotion”. In: *RAL*. 2018.

[83] Karol Gregor and Frederic Besse. “Temporal difference variational auto-encoder”. In: *arXiv preprint arXiv:1806.03107* (2018).

[84] Ivo Grondman et al. “A survey of actor-critic reinforcement learning: standard and natural policy gradients”. In: *IEEE Transactions on Systems, Man, and Cybernetics*. 2012.

[85] S. Gu et al. “Continuous deep Q-learning with model-based acceleration”. In: *ICML*. 2016.

[86] Shixiang Gu, Ethan Holly, et al. “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates”. In: *ICRA*. IEEE. 2017, pp. 3389–3396.

[87] Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, et al. “Q-Prop: sample-efficient policy gradient with an off-policy critic”. In: *ICLR*. 2017.

[88] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, et al. “Continuous Deep Q-Learning with Model-based Acceleration”. In: *International Conference on Machine Learning*. 2016, pp. 2829–2838.

[89] Vijaykumar Gullapalli, Judy A Franklin, and Hamid Benbrahim. “Acquiring robot skills via reinforcement learning”. In: *IEEE CSM* 14.1 (1994), pp. 13–24.

[90] Abhishek Gupta et al. “Meta-reinforcement learning of structured exploration strategies”. In: *NeurIPS*. 2018, pp. 5302–5311.

[91] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, et al. “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor”. In: *arXiv preprint arXiv:1801.01290* (2018).

[92] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, et al. “Soft Actor-Critic Algorithms and Applications”. In: *arXiv preprint arXiv:1812.05905* (2018).

[93] Danijar Hafner et al. “Learning latent dynamics for planning from pixels”. In: *arXiv preprint arXiv:1811.04551* (2018).

[94] Danijar Hafner et al. “Learning latent dynamics for planning from pixels”. In: *ICML*. 2019, pp. 2555–2565.

[95] Duncan W Haldane and Ronald S Fearing. “Roll oscillation modulated turning in dynamic millirobots”. In: *IEEE Int. Conf. on Robotics and Automation*. 2014, pp. 4569–4575.

[96] Duncan W Haldane and Ronald S Fearing. “Running beyond the bio-inspired regime”. In: *IEEE Int. Conf. on Robotics and Automation*. 2015, pp. 4539–4546.

[97] Duncan W Haldane, Kevin C Peterson, et al. “Animal-inspired design and aerodynamic stabilization of a hexapedal millirobot”. In: *IEEE Int. Conf. on Robotics and Automation*. 2013, pp. 3279–3286.

[98] Matthew Hausknecht and Peter Stone. “Deep Recurrent Q-Learning for Partially Observable MDPs”. In: *2015 AAAI Fall Symposium Series*. 2015.

[99] Nicolas Heess, Jonathan J Hunt, et al. “Memory-based control with recurrent neural networks”. In: *arXiv preprint arXiv:1512.04455* (2015).

[100] Nicolas Heess, Gregory Wayne, et al. “Learning continuous control policies by stochastic value gradients”. In: *NIPS*. 2015.

[101] Dominik Henrich and Heinz Wörn. *Robot manipulation of deformable objects*. Springer Science & Business Media, 2012.

[102] Robert Herman. *Neural control of locomotion*. Vol. 18. Springer, 2017.

[103] Susan J Hespos and Kristy VanMarle. “Physics for infants: Characterizing the origins of knowledge about objects, substances, and number”. In: *Wiley Interdisciplinary Reviews: Cognitive Science* 3.1 (2012), pp. 19–27.

[104] Todd Hester, Michael Quinlan, and Peter Stone. “Generalized model learning for reinforcement learning on a humanoid robot”. In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 2369–2374.

[105] Mark A Hoepflinger et al. “Haptic terrain classification for legged robots”. In: *ICRA*. 2010.

[106] Matthew Hoffman, Francis R Bach, and David M Blei. “Online learning for latent dirichlet allocation”. In: *advances in neural information processing systems*. 2010, pp. 856–864.

[107] Aaron M Hoover, Samuel Burden, et al. “Bio-inspired design and dynamic maneuverability of a minimally actuated six-legged robot”. In: *IEEE RAS and EMBS Int. Conf. on Biomedical Robotics and Biomechatronics*. 2010, pp. 869–876.

[108] Aaron M Hoover and Ronald S Fearing. “Fast scale prototyping for folded millirobots”. In: *IEEE Int. Conf. on Robotics and Automation*. 2008, pp. 886–892.

[109] Rein Houthooft et al. "Evolved policy gradients". In: *NeurIPS*. 2018.

[110] Jan Humplik et al. "Meta reinforcement learning as task inference". In: *arXiv preprint arXiv:1905.06424* (2019).

[111] K Jetal Hunt et al. "Neural networks for control systems—a survey". In: *Automatica*. 1992.

[112] Marco Hutter et al. "Anymal-a highly mobile and dynamic quadrupedal robot". In: *IEEE/RSJ Int. Conf. on Intell. Robots and Systems*. 2016, pp. 38–44.

[113] Maximilian Igl et al. "Deep Variational Reinforcement Learning for POMDPs". In: *ICML*. 2018, pp. 2122–2131.

[114] Amir Jafari et al. "On no-regret learning, fictitious play, and nash equilibrium". In: *ICML*. Vol. 1. 2001, pp. 226–233.

[115] Stephen James, Michael Bloesch, and Andrew J Davison. "Task-Embedded Control Networks for Few-Shot Imitation Learning". In: *CoRL*. 2018, pp. 783–795.

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

[117] Ghassen Jerfel et al. "Online gradient-based mixtures for transfer modulation in meta-learning". In: *arXiv preprint arXiv:1812.06080* (2018).

[118] Adam Johnson and A David Redish. "Neural ensembles in CA3 transiently encode paths forward of the animal at a decision point". In: *Journal of Neuroscience* 27.45 (2007), pp. 12176–12189.

[119] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. "Planning and acting in partially observable stochastic domains". In: *AI* 101.1-2 (1998), pp. 99–134.

[120] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. "Reinforcement learning: A survey". In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[121] Sham M Kakade. "A natural policy gradient". In: *Advances in neural information processing systems*. 2002, pp. 1531–1538.

[122] Mrinal Kalakrishnan et al. "Fast, robust quadruped locomotion over challenging terrain". In: *IEEE Int. Conf. on Robotics and Automation*. 2010, pp. 2665–2670.

[123] Dmitry Kalashnikov et al. "Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". In: *CoRL*. 2018, pp. 651–673.

[124] Peter Karkus, David Hsu, and Wee Sun Lee. "Qmdp-net: Deep learning for planning under partial observability". In: *NeurIPS*. 2017, pp. 4694–4704.

[125] Maximilian Karl et al. "Deep Variational Bayes Filters: Unsupervised Learning of State Space Models from Raw Data". In: *ICLR*. 2016.

[126] Sousso Kelouwani et al. "Online system identification and adaptive control for PEM fuel cell maximum efficiency tracking". In: *IEEE Transactions on Energy Conversion* 27.3 (2012), pp. 580–592.

[127] S Mohammad Khansari-Zadeh and Aude Billard. "Learning stable nonlinear dynamical systems with gaussian mixture models". In: *IEEE Transactions on Robotics*. 2011.

[128] Sangbae Kim, Jonathan E Clark, and Mark R Cutkosky. "iSprawl: Design and tuning for high-speed autonomous open-loop running". In: *The International Journal of Robotics Research* 25.9 (2006), pp. 903–912.

[129] D. Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *ICLR*. 2014.

[130] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the National Academy of Sciences* (2017).

[131] Hiroaki Kitano et al. "RoboCup: A challenge problem for AI". In: *AI magazine* 18.1 (1997), pp. 73–73.

[132] Jonathan Ko and Dieter Fox. "GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models". In: *Autonomous Robots* 27.1 (2009), pp. 75–90.

[133] Jonathan Ko and Dieter Fox. "GP-BayesFilters: Bayesian filtering using gaussian process prediction and observation models". In: *IROS*. 2008.

[134] Jens Kober, J Andrew Bagnell, and Jan Peters. "Reinforcement learning in robotics: A survey". In: *IJRR* (2013).

[135] Roman Kolbert, Nikhil Chavan-Dafle, and Alberto Rodriguez. "Experimental validation of contact dynamics for in-hand manipulation". In: *International Symposium on Experimental Robotics*. Springer. 2016, pp. 633–645.

[136] J Zico Kolter, Pieter Abbeel, and Andrew Y Ng. "Hierarchical apprenticeship learning with application to quadruped locomotion". In: *Advances in Neural Information Processing Systems*. 2008, pp. 769–776.

[137] Haldun Komsuoglu, Anirudha Majumdar, et al. "Characterization of dynamic behaviors in a hexapod robot". In: *Experimental Robotics*. Springer. 2014, pp. 667–684.

[138] Haldun Komsuoglu, Kiwon Sohn, et al. "A physical model for dynamical arthropod running on level ground". In: *Departmental Papers (ESE)* (2008), p. 466.

[139] Ben Krause, Emmanuel Kahembwe, et al. "Dynamic Evaluation of Neural Sequence Models". In: *CoRR* abs/1709.07432 (2017). arXiv: `1709.07432`.

[140] Ben Krause, Liang Lu, et al. "Multiplicative LSTM for sequence modelling". In: *arXiv preprint arXiv:1609.07959* (2016).

[141] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *NIPS*. 2012.

[142] Klas Kronander, Etienne Burdet, and Aude Billard. "Task transfer via collaborative manipulation for insertion assembly". In: *WHRI*. Citeseer. 2014.

[143] Eric Krotkov et al. "The darpa robotics challenge finals: Results and perspectives". In: *Journal of Field Robotics* 34.2 (2017), pp. 229–240.

[144] Scott Kuindersma et al. “Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot”. In: *Autonomous robots* 40.3 (2016), pp. 429–455.

[145] Tejas D Kulkarni et al. “Deep successor reinforcement learning”. In: *arXiv preprint arXiv:1606.02396* (2016).

[146] Visak Kumar et al. “Contextual Reinforcement Learning of Visuo-tactile Multi-fingered Grasping Policies”. In: *arXiv preprint arXiv:1911.09233* (2019).

[147] Thanard Kurutach et al. “Model-Ensemble Trust-Region Policy Optimization”. In: *arXiv preprint arXiv:1802.10592* (2018).

[148] Thanard Kurutach et al. “Model-Ensemble Trust-Region Policy Optimization”. In: *arXiv preprint arXiv:1802.10592* (2018).

[149] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. “Human-level concept learning through probabilistic program induction”. In: *Science* (2015).

[150] Sascha Lange, Martin Riedmiller, and Arne Voigtlander. “Autonomous reinforcement learning on raw visual input data in a real world application”. In: *IJCNN*. 2012.

[151] Alex X Lee et al. “Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model”. In: *arXiv preprint arXiv:1907.00953* (2019).

[152] Michelle A Lee et al. “Making sense of vision and touch: Self-supervised learning of multimodal representations for contact-rich tasks”. In: *arXiv preprint arXiv:1810.10191* (2018).

[153] Bethany Leffler. “Perception-Based Generalization in Model-Based Reinforcement Learning”. PhD thesis. Rutgers, 2009.

[154] Ian Lenz, Ross A Knepper, and Ashutosh Saxena. “DeepMPC: Learning Deep Latent Features for Model Predictive Control.” In: *Robotics: Science and Systems*. 2015.

[155] Sergey Levine and Pieter Abbeel. “Learning neural network policies with guided policy search under unknown dynamics”. In: *NIPS*. 2014.

[156] Sergey Levine, Chelsea Finn, et al. “End-to-end training of deep visuomotor policies”. In: *Journal of Machine Learning Research (JMLR)* (2016).

[157] Sergey Levine, Chelsea Finn, et al. “End-to-end training of deep visuomotor policies”. In: *JMLR*. 2017.

[158] Sergey Levine and Vladlen Koltun. “Guided policy search”. In: *International Conference on Machine Learning*. 2013, pp. 1–9.

[159] Ke Li and Jitendra Malik. “Learning to optimize”. In: *arXiv preprint arXiv:1606.01885* (2016).

[160] W. Li and E. Todorov. “Iterative linear quadratic regulator design for nonlinear biological movement systems”. In: *ICINCO*. 2004.

[161] T. Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *ICRL*. 2016.

[162] Timothy Lillicrap et al. “Continuous control with deep reinforcement learning”. In: *CoRR* abs/1509.02971 (2015). arXiv: `1509.02971`.

[163] Rudolf Lioutikov et al. “Sample-based information-theoretic stochastic optimal control”. In: *ICRA*. 2014.

[164] Daniel J Lizotte et al. “Automatic Gait Optimization with Gaussian Process Regression.” In: *IJCAI*. Vol. 7. 2007, pp. 944–949.

[165] David Lopez-Paz et al. “Gradient Episodic Memory for Continual Learning”. In: *Advances in Neural Information Processing Systems*. 2017.

[166] Kendall Lowrey et al. “Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control”. In: *arXiv preprint arXiv:1811.01848* (2018).

[167] Sridhar Mahadevan and Jonathan Connell. “Automatic programming of behavior-based robots using reinforcement learning”. In: *AI* 55.2-3 (1992), pp. 311–365.

[168] Ali Malik et al. “Calibrated Model-Based Deep Reinforcement Learning”. In: *arXiv preprint arXiv:1906.08312* (2019).

[169] Patrizio Manganiello et al. “Optimization of Perturbative PV MPPT Methods Through Online System Identification.” In: *IEEE Trans. Industrial Electronics* 61.12 (2014), pp. 6812–6821.

[170] Arthur Joseph McClung III. *Techniques for dynamic maneuvering of hexapedal legged robots*. Vol. 67. 11. 2006.

[171] Amy McGovern and Andrew G Barto. “Automatic discovery of subgoals in reinforcement learning using diverse density”. In: (2001).

[172] David Meger et al. “Learning legged swimming gaits from experience”. In: *ICRA*. 2015.

[173] Franziska Meier, Daniel Kappler, et al. “Towards Robust Online Inverse Dynamics Learning”. In: *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems*. IEEE, 2016.

[174] Franziska Meier and Stefan Schaal. “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, May 2016.

[175] Russell Mendonca et al. “Guided Meta-Policy Search”. In: *arXiv preprint arXiv:1904.00956* (2019).

[176] Stephen Miller et al. “A geometric approach to robotic laundry folding”. In: *The International Journal of Robotics Research* 31.2 (2012), pp. 249–267.

[177] Nikhil Mishra, Pieter Abbeel, and Igor Mordatch. “Prediction and control with temporal segment models”. In: *ICML*. 2017.

[178] Nikhil Mishra, Mostafa Rohaninejad, et al. "A simple neural attentive meta-learner". In: *NIPS 2017 Workshop on Meta-Learning*. 2017.

[179] Nikhil Mishra, Mostafa Rohaninejad, et al. "Meta-learning with temporal convolutions". In: *arXiv preprint arXiv:1707.03141* (2017).

[180] V. Mnih, A. P. Badia, et al. "Asynchronous methods for deep reinforcement learning". In: *ICML*. 2016.

[181] V. Mnih, K. Kavukcuoglu, et al. "Playing Atari with deep reinforcement learning". In: *Workshop on Deep Learning, NIPS*. 2013.

[182] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[183] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, et al. "Human-level control through deep reinforcement learning". In: *Nature*. 2015.

[184] Igor Mordatch, Zoran Popović, and Emanuel Todorov. "Contact-invariant optimization for hand manipulation". In: *Proceedings of the ACM SIGGRAPH/Eurographics symposium on computer animation*. Eurographics Association. 2012, pp. 137–144.

[185] Igor Mordatch, Emanuel Todorov, and Zoran Popović. "Discovery of complex behaviors through contact-invariant optimization". In: *ACM Transactions on Graphics (TOG)* 31.4 (2012), pp. 1–8.

[186] Jun Morimoto and Christopher G Atkeson. "Minimax differential dynamic programming: An application to robust biped walking". In: *NIPS*. 2003.

[187] Tsendsuren Munkhdalai and Hong Yu. "Meta Networks". In: *International Conference on Machine Learning (ICML)* (2017).

[188] Tsendsuren Munkhdalai and Hong Yu. "Meta networks". In: *arXiv preprint arXiv:1703.00837* (2017).

[189] Tsendsuren Munkhdalai, Xingdi Yuan, et al. "Learning Rapid-Temporal Adaptations". In: *arXiv preprint arXiv:1712.09926* (2017).

[190] K. Murphy. "Dynamic Bayesian Networks: Representation, Inference and Learning". PhD thesis. Dept. Computer Science, UC Berkeley, 2002. URL: `https://www.cs.ubc.ca/~murphyk/Thesis/thesis.html`.

[191] Anusha Nagabandi, Ignasi Clavera, et al. "Learning to Adapt: Meta-Learning for Model-Based Control". In: *arXiv preprint arXiv:1803.11347* (2018).

[192] Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, et al. "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning". In: *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*. 2018, pp. 7559–7566.

[193] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, et al. "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning". In: *arXiv preprint arXiv:1708.02596* (2017).

[194] Anusha Nagabandi, Guangzhao Yang, et al. “Learning Image-Conditioned Dynamics Models for Control of Under-actuated Legged Millirobots”. In: *arXiv preprint arXiv:1711.05253* (2017).

[195] Devang K Naik and RJ Mammone. “Meta-neural networks that learn by learning”. In: *International Joint Conference on Neural Netowrks (IJCNN)*. 1992.

[196] Devang K Naik and RJ Mammone. “Meta-neural networks that learn by learning”. In: *Neural Networks, 1992. IJCNN., International Joint Conference on*. Vol. 1. IEEE. 1992, pp. 437–442.

[197] TE Narasimhan. *Robots now rule Indian car manufacture*. 2018. URL: `https://www.rediff.com/business/report/hyundai-uses-580-robots-in-chennai-plant-maruti-5000/20180530.htm` (visited on 05/30/2018).

[198] Wyatt S Newman, Yonghong Zhao, and Y-H Pao. “Interpretation of force and moment signals for compliant peg-in-hole assembly”. In: *ICRA*. Vol. 1. IEEE. 2001, pp. 571–576.

[199] Cuong V Nguyen et al. “Variational Continual Learning”. In: *arXiv:1710.10628* (2017).

[200] J. Oh et al. “Control of memory, active perception, and action in minecraft”. In: *ICML*. 2016.

[201] Tokuji Okada. “Computer control of multijointed finger system for precise object-handling”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 12.3 (1982), pp. 289–299.

[202] Allison M Okamura, Niels Smaby, and Mark R Cutkosky. “An overview of dexterous manipulation”. In: *Robotics and Automation, 2000. Proceedings. ICRA’00. IEEE International Conference on*. Vol. 1. IEEE. 2000, pp. 255–262.

[203] P. Pastor et al. “Online movement adaptation based on previous sensor experiences.” English (US). In: *IEEE International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2011, pp. 365–371.

[204] Deepak Pathak. *Learning to Generalize via Self-Supervised Prediction*. eScholarship, University of California, 2019.

[205] Xue Bin Peng et al. “Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning”. In: *ACM Transactions on Graphics (TOG)* 36.4 (2017), pp. 1–13.

[206] Guilherme AS Pereira, Mario FM Campos, and Vijay Kumar. “Decentralized algorithms for multi-robot manipulation via caging”. In: *IJRR* 23.7-8 (2004), pp. 783–795.

[207] Christian F Perez, Felipe Petroski Such, and Theofanis Karaletsos. “Generalized Hidden Parameter MDPs Transferable Model-based RL in a Handful of Trials”. In: *arXiv preprint arXiv:2002.03072* (2020).

[208] Jan Peters and Stefan Schaal. “Reinforcement learning of motor skills with policy gradients”. In: *Neural networks* (2008).

[209] Steven T Piantadosi and Celeste Kidd. "Extraordinary intelligence and the care of infants". In: *Proceedings of the National Academy of Sciences* 113.25 (2016), pp. 6874–6879.

[210] Joelle Pineau, Geoff Gordon, Sebastian Thrun, et al. "Point-based value iteration: An anytime algorithm for POMDPs". In: *IJCAI*. Vol. 3. 2003, pp. 1025–1032.

[211] Lerrel Pinto and Abhinav Gupta. "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours". In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2016.

[212] Akshara Rai et al. "Learning Feedback Terms for Reactive Planning and Control". In: *Proceedings 2017 IEEE International Conference on Robotics and Automation (ICRA)*. Piscataway, NJ, USA: IEEE, May 2017.

[213] Marc Raibert et al. "Bigdog, the rough-terrain quadruped robot". In: *IFAC Proceedings Volumes* 41.2 (2008), pp. 10822–10825.

[214] Aravind Rajeswaran et al. "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations". In: *arXiv preprint arXiv:1709.10087* (2017).

[215] Kate Rakelly et al. "Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables". In: *ICML*. 2019.

[216] A. Rao. "A survey of numerical methods for optimal control". In: *Advances in the Astronautical Sciences*. 2009.

[217] Nathan Ratliff et al. "CHOMP: Gradient optimization techniques for efficient motion planning". In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 489–494.

[218] Sachin Ravi and Hugo Larochelle. "Optimization as a Model for Few-Shot Learning". In: *International Conference on Learning Representations (ICLR)*. 2017.

[219] Sachin Ravi and Hugo Larochelle. "Optimization as a model for few-shot learning". In: *International Conference on Learning Representations (ICLR)* (2018).

[220] Ali Sharif Razavian et al. "CNN features off-the-shelf: an astounding baseline for recognition". In: *CVPR Workshops (CVPRW)*. 2014.

[221] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, and Christoph H Lampert. "iCaRL: Incremental classifier and representation learning". In: *Proc. CVPR*. 2017.

[222] Marek Rei. "Online Representation Learning in Recurrent Neural Language Models". In: *CoRR* abs/1508.03854 (2015). arXiv: `1508.03854`.

[223] A. Richards. "Robust constrained model predictive control". PhD thesis. MIT, 2004.

[224] Samuel Ritter et al. "Been There, Done That: Meta-Learning with Episodic Recall". In: *arXiv preprint arXiv:1805.09692* (2018).

[225] Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. “A reduction of imitation learning and structured prediction to no-regret online learning”. In: *AISTATS*. 2011.

[226] Stéphane Ross, Joelle Pineau, et al. “A Bayesian approach for learning and planning in partially observable Markov decision processes”. In: *JMLR* 12.May (2011), pp. 1729–1770.

[227] Jonas Rothfuss et al. “ProMP: Proximal Meta-Policy Search”. In: *arXiv preprint arXiv:1810.06784* (2018).

[228] Andrei A Rusu et al. “Progressive neural networks”. In: *arXiv:1606.04671* (2016).

[229] Steindór Sæmundsson, Katja Hofmann, and Marc Peter Deisenroth. “Meta Reinforcement Learning with Latent Variable Gaussian Processes”. In: *arXiv preprint arXiv:1803.07551* (2018).

[230] Doyen Sahoo et al. “Online deep learning: Learning deep neural networks on the fly”. In: *arXiv preprint arXiv:1711.03705* (2017).

[231] Yoshiaki Sakagami et al. “The intelligent ASIMO: System overview and integration”. In: *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*. Vol. 3. IEEE. 2002, pp. 2478–2483.

[232] Adam Santoro et al. “Meta-learning with memory-augmented neural networks”. In: *International Conference on Machine Learning (ICML)*. 2016.

[233] Adam Santoro et al. “One-shot learning with memory-augmented neural networks”. In: *arXiv preprint arXiv:1605.06065* (2016).

[234] Sosale Shankara Sastry and Alberto Isidori. “Adaptive control of linearizable systems”. In: *IEEE Transactions on Automatic Control* (1989).

[235] Alexander Sax et al. “Mid-Level Visual Representations Improve Generalization and Sample Efficiency for Learning Visuomotor Policies”. In: *CoRL*. 2019.

[236] Juergen Schmidhuber and Rudolf Huber. “Learning to generate artificial fovea trajectories for target detection”. In: *International Journal of Neural Systems* (1991).

[237] Jurgen Schmidhuber. “Evolutionary principles in self-referential learning”. In: *Diploma thesis, Institut f. Informatik, Tech. Univ. Munich* (1987).

[238] Jürgen Schmidhuber. “Learning to control fast-weight memories: An alternative to dynamic recurrent networks”. In: *Neural Computation* (1992).

[239] Tanner Schmidt, Richard Newcombe, and Dieter Fox. “Self-supervised visual descriptor learning for dense correspondence”. In: *IEEE Robotics and Automation Letters* 2.2 (2016), pp. 420–427.

[240] Gerrit Schoettler et al. “Deep Reinforcement Learning for Industrial Insertion Tasks with Visual Inputs and Natural Reward Signals”. In: *ICLR*. 2019.

[241] J. Schulman et al. “Trust region policy optimization”. In: *ICML*. 2015.

[242] John Schulman, Sergey Levine, et al. “Trust region policy optimization”. In: *CoRR, abs/1502.05477* (2015).

[243] John Schulman, Philipp Moritz, et al. “High-dimensional continuous control using generalized advantage estimation”. In: *ICLR*. 2016.

[244] Stuart C Shapiro. *Encyclopedia of artificial intelligence second edition*. John, 1992.

[245] Maruan Al-Shedivat et al. “Continuous adaptation via meta-learning in nonstationary and competitive environments”. In: *arXiv preprint arXiv:1710.03641* (2017).

[246] David Silver, Julian Schrittwieser, et al. “Mastering the game of Go without human knowledge”. In: *Nature* (2017).

[247] David Silver, Richard S Sutton, and Martin Müller. “Sample-based learning and search with permanent and transient memories”. In: *ICML*. 2008.

[248] Avi Singh et al. “End-to-End Robotic Reinforcement Learning without Reward Engineering”. In: *environment (eg, by placing additional sensors)* 34 (), p. 44.

[249] Xingyou Song et al. “Rapidly Adaptable Legged Robots via Evolutionary Meta-Learning”. In: *arXiv preprint arXiv:2003.01239* (2020).

[250] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. “Curl: Contrastive unsupervised representations for reinforcement learning”. In: *arXiv preprint arXiv:2004.04136* (2020).

[251] Florian Stimberg, Andreas Ruttor, and Manfred Opper. “Bayesian inference for change points in dynamical systems with reusable states-a chinese restaurant process approach”. In: *Artificial Intelligence and Statistics*. 2012, pp. 1117–1124.

[252] Martin Stolle and Doina Precup. “Learning options in reinforcement learning”. In: *International Symposium on abstraction, reformulation, and approximation*. Springer. 2002, pp. 212–223.

[253] Freek Stulp and Stefan Schaal. “Hierarchical reinforcement learning with movement primitives”. In: *2011 11th IEEE-RAS International Conference on Humanoid Robots*. IEEE. 2011, pp. 231–238.

[254] Balakumar Sundaralingam and Tucker Hermans. “Geometric in-hand regrasp planning: Alternating optimization of finger gaits and in-grasp manipulation”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 231–238.

[255] Flood Sung et al. “Learning to learn: Meta-critic networks for sample efficient learning”. In: *arXiv preprint arXiv:1706.09529* (2017).

[256] R. Sutton. “Dyna, an integrated architecture for learning, planning, and reacting”. In: *AAAI*. 1991.

[257] Supasorn Suwajanakorn et al. “Discovery of Latent 3D Keypoints via End-to-end Geometric Reasoning”. In: *Neural Information Processing Systems (NIPS ’18*. 2018.

[258] Ryosuke Tajima, Daisaku Honda, and Keisuke Suga. "Fast running experiments involving a humanoid robot". In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 1571–1576.

[259] Marko Tanaskovic et al. "Adaptive model predictive control for constrained linear systems". In: *Control Conference (ECC), 2013 European*. IEEE. 2013.

[260] Russ Tedrake, Teresa Weirui Zhang, and H Sebastian Seung. "Learning to walk in 20 minutes". In: *Proceedings of the Fourteenth Yale Workshop on Adaptive and Learning Systems*. Vol. 95585. Yale University New Haven (CT). 2005, pp. 1939–1412.

[261] Matthew Tesch, Jeff Schneider, and Howie Choset. "Using response surfaces and expected improvement to optimize snake robot gait parameters". In: *IEEE/RSJ Int. Conf. on Intell. Robots and Systems*. 2011, pp. 1069–1074.

[262] Sebastian Thrun. "Lifelong learning algorithms". In: *Learning to learn*. Springer, 1998.

[263] Sebastian Thrun, Wolfram Burgard, Dieter Fox, et al. *Probabilistic robotics, vol. 1*. 2005.

[264] Sebastian Thrun, Mike Montemerlo, et al. "Stanley: The robot that won the DARPA Grand Challenge". In: *Journal of field Robotics* (2006).

[265] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 1998.

[266] Sebastian Thrun and Lorien Pratt. "Learning to learn: Introduction and overview". In: *Learning to learn*. Springer, 1998.

[267] Josh Tobin et al. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, pp. 23–30.

[268] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control". In: *IROS*. 2012.

[269] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control". In: *International Conference on Intelligent Robots and Systems (IROS)*. 2012.

[270] Jonathan Tremblay et al. "Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects". In: *CoRL*. 2018, pp. 306–316.

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

[272] Herke Van Hoof et al. "Learning robot in-hand manipulation with tactile features". In: *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*. IEEE. 2015, pp. 121–127.

[273] Alexander Sasha Vezhnevets et al. “Feudal networks for hierarchical reinforcement learning”. In: *arXiv preprint arXiv:1703.01161* (2017).

[274] Niklas Wahlström, Thomas B Schön, and Marc Peter Deisenroth. “From pixels to torques: policy learning with deep dynamical models”. In: *Deep learning workshop at ICML* (2015).

[275] Martin J Wainwright and Michael Irwin Jordan. *Graphical models, exponential families, and variational inference*. Now Publishers Inc, 2008.

[276] Jane X Wang et al. “Learning to reinforcement learn”. In: *arXiv preprint arXiv:1611.05763* (2016).

[277] Jane X Wang et al. “Learning to reinforcement learn”. In: *arXiv:1611.05763* (2016).

[278] Tingwu Wang and Jimmy Ba. “Exploring Model-based Planning with Policy Networks”. In: *arXiv preprint arXiv:1906.08649* (2019).

[279] Yu-Xiong Wang, Deva Ramanan, and Martial Hebert. “Growing a brain: Fine-tuning by increasing model capacity”. In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[280] Manuel Watter et al. “Embed to control: a locally linear latent dynamics model for control from raw images”. In: *NIPS*. 2015.

[281] Ari Weinstein and Matthew Botvinick. “Structure Learning in Motor Control: A Deep Reinforcement Learning Model”. In: *CoRR* abs/1706.06827 (2017). arXiv: `1706.06827`.

[282] Grady Williams, Andrew Aldrich, and Evangelos Theodorou. “Model Predictive Path Integral Control using Covariance Variable Importance Sampling”. In: *CoRR* abs/1509.01149 (2015). arXiv: `1509.01149`.

[283] Grady Williams, Nolan Wagener, et al. “Information theoretic mpc for model-based reinforcement learning”. In: *International Conference on Robotics and Automation (ICRA)*. 2017.

[284] X Alice Wu et al. “Integrated ground reaction force sensing and terrain classification for small legged robots”. In: *RAL* (2016).

[285] Annie Xie et al. “Few-Shot Goal Inference for Visuomotor Learning and Planning”. In: *CoRL*. 2018, pp. 40–52.

[286] Xiaofeng Xiong, Florentin Worgotter, and Poramate Manoonpong. “Neuromechanical Control for Hexapedal Robot Walking on Challenging Surfaces and Surface Classification”. In: *RAS*. 2014.

[287] Zhe Xu and Emanuel Todorov. “Design of a highly biomimetic anthropomorphic robotic hand towards artificial limb regeneration”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 3485–3492.

[288] Narri Yadaiah and G Sowmya. “Neural network based state estimation of dynamical systems”. In: *IJCNN*. IEEE. 2006, pp. 1042–1049.

[289] Denis Yarats et al. “Improving Sample Efficiency in Model-Free Reinforcement Learning from Images”. In: *arXiv preprint arXiv:1910.01741* (2019).

[290] Michael C Yip and David B Camarillo. “Model-less feedback control of continuum manipulators in constrained environments”. In: *IEEE Transactions on Robotics*. 2014.

[291] A Steven Younger, Sepp Hochreiter, and Peter R Conwell. “Meta-learning with backpropagation”. In: *International Joint Conference on Neural Networks*. IEEE. 2001.

[292] Tianhe Yu et al. “One-Shot Imitation from Observing Humans via Domain-Adaptive Meta-Learning”. In: *ICLR*. 2018.

[293] David Zarrouk, Duncan W Haldane, and Ronald S Fearing. “Dynamic legged locomotion for palm-size robots”. In: *SPIE Defense+ Security*. International Society for Optics and Photonics. 2015, 94671S–94671S.

[294] Andy Zeng, Shuran Song, Johnny Lee, et al. “Tossingbot: Learning to throw arbitrary objects with residual physics”. In: *arXiv preprint arXiv:1903.11239* (2019).

[295] Andy Zeng, Shuran Song, Stefan Welker, et al. “Learning Synergies between Pushing and Grasping with Self-supervised Deep RL”. In: *arXiv preprint arXiv:1803.09956* (2018).

[296] Friedemann Zenke, Ben Poole, and Surya Ganguli. “Continual learning through synaptic intelligence”. In: *International Conference on Machine Learning*. 2017.

[297] Marvin Zhang et al. “SOLAR: Deep Structured Representations for Model-Based Reinforcement Learning”. In: *ICML*. 2019, pp. 7444–7453.

[298] Richard Zhang et al. “The unreasonable effectiveness of deep features as a perceptual metric”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 586–595.

[299] Henry Zhu et al. “Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost”. In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 3651–3657.

[300] Luisa Zintgraf et al. “VariBAD: A Very Good Method for Bayes-Adaptive Deep RL via Meta-Learning”. In: *arXiv preprint arXiv:1910.08348* (2019).

[301] Matt Zucker et al. “Optimization and learning for rough terrain legged locomotion”. In: *The International Journal of Robotics Research* 30.2 (2011), pp. 175–191.