Broad Generalization through Domain Transfer: Abstractions and Algorithms

Aravind Rajeswaran

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2021

Reading Committee:

Sham Kakade, Chair

Emanuel Todorov, Chair

Sergey Levine

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

Abstract

Broad Generalization through Domain Transfer:
Abstractions and Algorithms

Aravind Rajeswaran

Co-Chairs of the Supervisory Committee:

Professor Sham Kakade
Computer Science & Engineering

Affiliate Professor Emanuel Todorov
Computer Science & Engineering

Deep learning and reinforcement learning have recently had a transformative impact on the fields of computer vision, NLP, and robotics. However, most of the recent progress have been in narrowly defined tasks where training and deployment happen under the same (or similar) conditions. This has resulted in brittle models that catastrophically fail when presented with conditions even moderately different from what they were trained on. Furthermore, current approaches are known to be data hungry, and thus may require prohibitively large datasets for many applications. How can we create intelligent agents that are data-efficient, robust, capable of broad generalization, and fast adaptation?

In this thesis, we outline the importance of *domain transfer* as a key component to achieve the aforementioned capabilities. Domain transfer refers to the ability of an AI agent to draw upon experiences from related tasks and transfer inductive biases, enabling more efficient and proficient learning in downstream tasks. This thesis presents abstractions and algorithms to enable such domain transfer. In particular, we focus on domain transfer that arises in the context of simulation to reality transfer in robotics, learning from static offline datasets in reinforcement learning, and efficient adaptation to new tasks in the case of meta-learning. The algorithms we present enjoy rigorous theoretical guarantees and also demonstrate strong empirical results in a variety of benchmark tasks and real-world case studies spanning perception, control, and robotics.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

This thesis is a culmination of an exciting and eventful academic journey over the past five years. It would not have been possible without guidance and motivation from my advisers, collaborators, and friends. I would like to express my sincere gratitude and appreciation for everyone who has played a part in my PhD journey.

First and foremost, I would like to thank my advisors Sham Kakade and Emo Todorov. The academic guidance, research freedom, and honest feedback I received from them have played invaluable roles in my research trajectory. I greatly admire Sham's ability to ask the correct research questions ahead of their time, and I hope some of this has rubbed off on me. I thank Emo for being a father figure away from home – someone who was approachable and genuinely cared for my success. Thank you so much Sham and Emo – I have learned so much from our interactions, both about research and the world more broadly.

I wish to thank Sergey Levine for being a wonderful mentor and teaching me the cutting edge of deep RL research. Thank you so much Sergey for being approachable, and making me feel welcomed and part of your research group. The interactions and discussions this enabled have had a profound and lasting impact on my research vision. I also thank Byron Boots, Eric Rombokas, and Sergey Levine for being on my PhD committee and sharing valuable time and feedback, which have been very helpful in improving my research presentation skills.

I feel blessed to have had a large number of wonderful collaborators and mentors. I thank Vikash Kumar for consistent mentorship throughout my PhD. I also thank Kendall Lowrey, Igor Mordatch, Chelsea Finn, and Rahul Kidambi, for the recurring collaborations. Our discussions have enriched my research vision and allowed me to give my best to research. I also express my sincere thanks to all my collaborators involved in various projects: Sarvjeet, Ravi, Abhishek, John Schulman, Dibya, Avi, Cathy, Rocky, Pieter, Svet, Jeremy, Divye, Andrew, Shivam, Henry, Colin, Sidd, Praneeth, Thorsten, Rafael, Kevin Yu, Aviral, Misha, Catherine, Kevin Lu, Lili, Aravind Srinivas, Kimin, Aditya.

During my PhD, I had the opportunity to intern at a number of great research labs. My stints at OpenAI and Google Brain were instrumental in establishing multiple long term collaborations. I thank John Schulman and OpenAI for inviting me to intern, even though it was my first year in PhD. I thank Vincent Vanhoucke and Google Brain for inviting me to continue as student researcher after my internship. These environments gave me the opportunity to connect with an amazing intern cohort and some of the best senior deep learning researchers.

The academic and administrative ecosystem built by UW CSE contributed a lot towards my gratifying PhD experience. I thank Elise Dorough and the CSE management for the constant encouragement and help with various administrative tasks. I thank various professors in CSE for stimulating research discussions: Dieter Fox, Sidd Srinivasa, Byron Boots, Kevin Jamieson, Maryam Fazel, Zaid Harchaoui. I also thank all my office mates as well as members of the Robotics and WAIL labs. Special thanks to Kendall, Aaron, Svet, Ben, Colin, Motoya, Zoey, Mohit, Kay, Willie, Rahul, Ramya, John, Krishna, Jennifer, Sam, Aditya, and others for the many fun coffee chats, explorations around the city, and brainstorm sessions.

Outside of CSE research, I thank my friends from Seattle who have made the past five years memorable and enjoyable. In particular, it would have been extremely difficult to survive the Covid-19 lockdowns without them. Thanks so much Guna, Srini, Rahul, Sachin, Mandar, Gagan, Shiva, Bala, Arjun, Ankur. Will greatly miss the shopping runs, movie nights, cocktail mixes, and Biriyani dashes.

I also thank my advisers from my undergraduate days: Balaraman Ravindran, Shankar Narasimhan, Sridharakumar Narasimhan, and Sitabhra Sinha. My enthusiasm for research and science was nurtured under their guidance. I also thank them for helping me get into a PhD program from where I could broaden my horizons and pursue my research vision.

Last but certainly not least, I thank my parents, to whom this thesis is dedicated. Their unbounded love and tireless support have played an instrumental part over the entirety of my life. During the final phases of my PhD involving Covid regulations, their daily calls gave me something to look forward to everyday and helped keep track of time. This thesis would not have been possible without their support.

DEDICATION

To my parents, K. Rajeswaran and S. Banumathy.

Thank you for the eternal love and support.

Chapter 1

INTRODUCTION

A core characteristic of an intelligent system is the ability to quickly learn new skills, as well as adapting to unexpected situations. This requires the ability to organize and draw upon prior experience from related tasks for broad generalization and adaptation. We can all relate through our personal experiences, that humans are remarkably good at such generalization and adaptation capabilities. As an example, we all have experiences of travelling to different countries and staying in hotel rooms that have different appliances and fixtures than what we may be used to. Nevertheless, we can very quickly adapt and manipulate objects in the new scene. Similarly, we can generalize experiences from driving one type of vehicle to another, say from bicycle to motorcycle, by drawing upon general and transferable notions such as balance. We can also quickly adapt to driving in a different country even though visual characteristics and traffic rules may be different, such as left-hand drive vs right-hand drive. Can we create AI systems that are capable of similar characteristics of broad generalization and rapid adaptation in new scenarios? In the future, such intelligent systems would be essential to assist humans in a variety of tasks ranging from everyday chores, to personal assistants, to elder care. To understand and make progress on this question, we must first understand the capabilities and limitations of currently existing AI systems.

Deep learning [1] has emerged as the dominant paradigm in machine learning and AI. It has demonstrated considerable success in various computer vision [2], speech recognition [3], and natural language tasks [4, 5, 6]. In conjunction with reinforcement learning (deep RL), it has demonstrated success in playing games [7, 8] and controlling robots [9, 10]. In these settings, a large randomly initialized deep learning model is trained using examples or interactions to become proficient at the desired task. Despite these impressive feats, deep learning and deep RL approaches suffer from two major challenges. First, they have demonstrated success primarily in narrowly defined tasks, where training and testing happens under the same distribution. Failure on instances even mildly different from training conditions and the existence of adversarial examples [11] point to a lack of broad generalization and distributional robustness in current methods. Secondly, while deep learning models can be trained from scratch with minimal human engineering and priors, the *tabula rasa* learning approach makes them extremely data hungry. Thus, there is a significant gap between the quick learning, robustness, and adaptation capabilities of humans and that of current AI systems.

The premise of this thesis is that to achieve broad generalization and fast adaptation, it is essential for an AI agent to draw upon prior experience and transfer inductive biases from related domains. In this thesis, we refer to this as *Domain Transfer.* By drawing upon prior experience from related domains, we avoid the tabula rasa training, and thus can be more data efficient. Furthermore, the breadth of prior experience may enable broader generalizations as well. In this thesis, we develop new abstractions and algorithms to enable such domain transfer, and ultimately lead to robust and broadly competent AI systems.

A straightforward attempt at such domain transfer is pretraining – where the model is trained on the source domain and either directly tested in the target domain, or is tested after finetuning using limited experience in the target domain. While this is simple and appealing, it can often fail due to the systematic discripancies (or domain gap) that exist between the source and target domains. Since the agent is not aware of the transfer it must eventually do, it may not learn the correct inductive biases that are amenable to transfer or adaptation. For example, an agent trained for the task of image classification might imbibe translation invariance from the data, since the network has to output the same object category regardless of the location of the object in the image. Such a model might perform very poorly for downstream robotic grasping, where the models must be very sensitive to location of objects for an accurate grasp. Similarly, a robot that is trained to walk only from upright configurations may learn an extremely fast walking gait, but such a gait may not be resistant to perturbations, thereby performing very poorly in real-world conditions [12]. This all points to the observation that training AI systems on narrowly defined tasks may lead to solutions that do not generalize broadly and might be detrimental for transfer to new situations. In this thesis, we outline different algorithmic paradigms for learning models with inductive biases that either directly transfer to the target domain or can be adapted efficiently with minimal data.

1.1 Thesis Contributions and Outline

In **Chapter 2**, we study simulation to reality transfer for safe and sample-efficient deep RL. Current state of the art deep RL algorithms require a prohibitively large number of potentially dangerous samples (interactions) for physical robotic systems. To reduce the training time and promote safety, we study transferring controllers from a physics simulator. Data collection from a physics simulator is cheap and easy since it can be run much faster than real-time and avoids safety concerns. However, such simulators only represent a crude approximation of the real world, and thus solutions learned in simulation may not directly transfer. We develop the EPOpt algorithm [13] to address this issue and promote the learning of robust and transferable control policies. This work was published at the International Conference on Learning Representations (ICLR) 2017 [13]. In Chapter 2.6, we present a case study that extends this approach to show positive results on a real physical robot. This work was published at IEEE SIMPAR 2018 [14].

In **Chapter 3**, we study offline RL where the goal is to learn a competent control policy based solely on a dataset of historical interactions with the environment. Due to use of real-world data, this avoids the systematic discrepancies that plague simulators. Since the data has already been collected, or can be collected using some safe policies, this paradigm is suitable for safety critical applications like healthcare, robotics, and industrial automation. Large historical datasets are also readily available in domains like autonomous driving and recommendation systems, where offline RL may be used to improve upon currently deployed policies. However, offline RL is particularly challenging due to a domain shift between the training data distribution and the induced distribution of the trained policy. Since the goal in offline RL is to consume offline data and produce a policy that is better than the data collection policy, the domain shift between training and deployment will necessarily occur. To mitigate the effect of this domain shift, we introduce MOReL [15], a new algorithmic framework that effectively leverages model-based approaches for offline RL. This work was published in Neural Information Processing Systems (NeurIPS) 2020 [15]. MOReL achieves state of the art results in offline RL benchmark tasks while also enjoying strong theoretical guarantees, including a minimax optimality guarantee.

In **Chapter 4**, we extend model-based RL ideas from above setting to the more well studied case of interactive RL, where the agent can interact with the environment for additional feedback. Surprisingly, even in this well-studied setting, we observe the effects of domain shift due to interleaving policy learning with dynamics model learning. We draw upon insights from game theory to design stable algorithms that mitigate against domain shift [16]. Through this endeavor, our work established that model-based RL algorithms can: (a) be highly sample efficient; (b) match the asymptotic performance of model-free policy gradient; (c) scale gracefully to high dimensional tasks like dexterous manipulation; and (d) handle extended rollout horizons of several hundred timesteps. This work was published at the International Conference on Machine Learning (ICML) 2020 [16].

In **Chapter 5**, we study meta-learning, where an agent learns a learning algorithm that is capable of quickly learning in a new domain. While the previous chapters primarily focus on transfering from a single source domain to a target domain, in meta-learning we consider a collection of source domains and aim to learn adaptable representations for accelerating the learning of new tasks. In this chapter, we show how to formalize meta-learning as a *bi-level optimization* problem. In the outer level, we learn shared meta-parameters, while in the inner-level, we learn task-specific deep models. Meta parameters modulate the behavior of a task learning algorithm, and correspond to variables like weight initialization and learning rate. After meta-learning, the meta parameters capture inductive properties and priors for efficiently learning new tasks. A key challenge in scaling optimization based meta learning is back-propagation through the inner level learning process. The computation and memory costs for computing the meta gradient scale linearly with the length of the optimization path. For example, if the learning algorithm uses 100 steps of gradient descent, the computation graph for meta parameters is 100 times the size of the deep model. Back-propagating through such large graphs is infeasible due to memory constraints and numerical instabilities.

To address these limitations, we developed the implicit MAML algorithm [17]. Using the concept of implicit differentiation, we prove that the meta gradient depends only on the solution to the inner-level learning problem, and not the specific path taken by the algorithm. Leveraging this, our implicit MAML algorithm uses only constant memory, independent of length of optimization path. As a result, our algorithm is provably efficient in computation and memory, provably convergent, and demonstrates strong empirical gains in benchmark tasks. This work was published at Neural Information Processing Systems (NeurIPS) 2019 [17].

Finally, in **Chapter 6**, we extend the meta-learning framework from Chapter 5 to continual non-stationary task distributions. A limitation of the standard meta-learning paradigm is that the agent's ability to refine the learning algorithm stops after the initial meta-training phase, and does not continue to improve over time as it encounters more tasks. Furthermore, the quality of meta-learned algorithm is intimately tied to the task distribution, which may drift over time. In our work on online meta-learning [18], we develop a new formulation that presents a more realistic embodiment of the continuous learning process. In this setting, the agent faces tasks sequentially one after the other, potentially from a non-stationary distribution. Learning in the initial tasks may be slow, similar to learning from scratch. As the agent experiences more tasks, the learning progressively becomes more efficient and proficient. To achieve this, we develop the *follow the meta leader* (FTML) algorithm, and prove that it has a no-regret property when competing against a very powerful comparator class that can perform task-specific adaptation. We validated this algorithm on few-shot vision domains, and showed that FTML can enable CNNs to recognize and predict pose of new objects more efficiently than conventional online learning approaches. In essence, this enables meta-learning to continuously accelerate the learning of new tasks. This work was published in the International Conference on Machine Learning (ICML) 2019 [18].

Overall, through this thesis, we hope to establish domain transfer as an important component of broad generalization in learning and control. We also provide abstractions and templates to design scalable and practically efficient algorithms for domain transfer that also enjoy strong theoretical guarantees.

Chapter 2

SIMULATION TO REALITY TRANSFER

2.1 Introduction

In the past half decade, deep reinforcement learning has demonstrated remarkable success in a wide range of tasks including games [19, 20, 21], simulated control problems [22, 23, 24], and animation [25]. However, high sample complexity remains a major challenge for direct use of such algorithms on complex physical robots. Model-free algorithms like Q-learning, actor-critic, and policy gradients are known to suffer from long learning times [26], which is compounded when used in conjunction with expressive function approximators like deep neural networks (DNNs). The challenge of gathering samples from the real world is further exacerbated by issues of safety for the agent and environment, since sampling with partially learned policies could be unstable [27]. Thus, model-free deep RL methods often require a prohibitively large numbers of potentially dangerous samples for physical control tasks.

Simulation to reality (Sim2Real) transfer, where the real-world target domain is approximated with a simulated source domain, has the potential to circumvent above challenges by learning policies using cheap simulated data. Sim2Real can in principle be viewed as a model-based RL approach, where the model class is dictated by the physical effects supported by the simulator. The principal challenge with simulated training is the systematic discrepancy between source and target domains. We show in our experiments that such deep RL algorithms learn policies that are highly optimized for the specific models used in the simulator, but are brittle under model mismatch. This is unsurprising, since deep networks are remarkably proficient at exploiting any systematic regularities in the simulator.

Thus, methods that compensate for systematic discrepancies (modeling errors) are needed to transfer results from simulations to real world using RL. We show that the impact of such discrepancies can be mitigated through two key ideas: (1) training on an ensemble of models in an adversarial fashion to learn policies that are robust to parametric model errors, as well as to unmodeled effects; and (2) adaptation of the source domain ensemble using data from the target domain to progressively make it a better approximation. This can be viewed either as an instance of model-based Bayesian RL [28]; or as transfer learning from a collection of simulated source domains to a real-world target domain [29].

This chapter outlines the Ensemble Policy Optimization (EPOpt$-\epsilon$) algorithm for finding policies that are robust to model mismatch. In line with model-based Bayesian RL, we learn a policy for the target domain by alternating between two phases:

1. Given a source distribution (i.e. ensemble of models), find a robust policy that is competent for the whole distribution.

2. Gather target domain data using the robust policy, and adapt the source distribution.

EPOpt uses an ensemble of models sampled from the source distribution, and a form of adversarial training to learn robust policies that generalize to a broad range of models. By robust, we mean insensitivity to parametric model errors and broadly competent performance for *direct-transfer* (also referred to as *jumpstart* like in Taylor and Stone [29]). Direct-transfer performance refers to the average initial performance (return) in the target domain, without any fine-tuning on the target domain. By adversarial training, we mean that model instances on which the policy performs poorly in the source distribution are sampled more often in order to encourage learning of policies that perform well for a wide range of model instances. This is in contrast to methods which learn highly optimized policies for specific model instances, but brittle under model perturbations.

In our experiments, we did not observe significant loss in performance by requiring the policy to work on multiple models, for example through adopting a more conservative strategy. Further, we show that policies learned using EPOpt are robust even to effects not modeled in the source domain. Such unmodeled effects are a major issue when transferring from simulation to the real world. For the model adaptation step (ii), we present a simple method using approximate Bayesian updates, which progressively makes the source distribution a better approximation of the target domain. We evaluate the proposed methods on OpenAI gym tasks simulated with MuJoCo [30]. Our experimental results suggest that adversarial training on model ensembles produces robust policies which generalize better than policies trained on a single, maximum-likelihood model alone.

2.2 Problem Formulation

We consider parametrized Markov Decision Processes (MDPs), which are tuples of the form: $\mathcal{M}(p) \equiv <\mathcal{S}, \mathcal{A}, \mathcal{T}_p, \mathcal{R}_p, \gamma, S_{0,p}>$ where $\mathcal{S}$, $\mathcal{A}$ are (continuous) states and actions respectively; $\mathcal{T}_p$ $\mathcal{R}_p$, and $S_{0,p}$ are the state transition, reward function, and initial state distribution respectively, all parametrized by p; and γ is the discount factor. Thus, we consider a set of MDPs with the same state and action spaces, but each MDP in this set can have different dynamics and reward functions. We use transition functions of the form $S_{t+1} \equiv \mathcal{T}_p(s_t, a_t)$ where $\mathcal{T}_p$ is a random process and S_{t+1} is a random variable.

We distinguish between source and target MDPs using $\mathcal{M}$ and $\mathcal{W}$ respectively. We also refer to $\mathcal{M}$ and $\mathcal{W}$ as source and target domains respectively, as is common in the transfer learning literature. Our objective is to learn the optimal policy for $\mathcal{W}$; and to do so, we have access to $\mathcal{M}(p)$. We assume that we have a distribution ($\mathcal{D}$) over the source domains (MDPs) generated by a distribution over the parameters $P \equiv \mathcal{P}(p)$ that capture our subjective belief about the parameters of $\mathcal{W}$. Let this distribution $\mathcal{P}$ be parametrized by ψ (e.g. mean, standard deviation). For example, $\mathcal{M}$ could be a hopping task with reward proportional to hopping velocity and falling down corresponds to a terminal state. For this task, p could correspond to parameters like torso mass, ground friction, and damping in joints, all of which affect the dynamics. Ideally, we would like the target domain to be in the model class, i.e. $\{\exists p \mid \mathcal{M}(p) = \mathcal{W}\}$. However, in practice, there are likely to be unmodeled effects, and we analyze this setting in our experiments. We wish to learn a policy $\pi^*_\theta(s)$ that performs well for all $\mathcal{M} \sim \mathcal{D}$. Note that this robust policy does not have an explicit dependence on p, and we require it to perform well without knowledge of p. Our intuition is that if we can find a highly rewarding policy while being oblivious to p, then such a policy is likely to also achieve high rewards in the target domain.

2.3 Learning protocol and EPOpt algorithm

We follow the round-based learning protocol of Bayesian model-based RL. We use the term *rounds* when interacting with the target domain, and *episode* when performing rollouts with the simulator. In each round, we interact with the target domain after computing the robust policy on the current (i.e. posterior) simulated source distribution. Following this, we update the source distribution using data from the target domain collected by executing the robust policy. Thus, in round i, we update two sets of parameters: θ_i, the parameters of the robust policy (neural network); and ψ_i, the parameters of the source distribution. The two key steps in this procedure are finding a robust policy given a source distribution; and updating the source distribution using data from the target domain. In this section, we present our approach for both of these steps.

2.3.1 Robust policy search

We introduce the EPOpt algorithm for finding a robust policy using the source distribution. EPOpt is a policy gradient based meta-algorithm which can use any policy optimization algorithm as a subroutine [31, 32, 33, 12, 34]. In particular, we consider on-policy optimization algorithms which collect a batch of trajectories by rolling out the current policy, and use these trajectories to make a policy update. The basic structure of EPOpt is to sample a collection of models from the source distribution, sample trajectories from each of these models, and make a gradient update based on a subset of sampled trajectories. To describe this formally, we first define the performance of a policy in an MDP:

$$\eta_{\mathcal{M}}(\theta, p) = \mathbb{E}_{\tilde{\tau}} \left[\sum_{t=0}^{T-1} \gamma^t r_t(s_t, a_t) \,\middle|\, p \right]. \tag{2.1}$$

Here, $\eta_{\mathcal{M}}(\theta, p)$ is the evaluation of π_θ on the model $\mathcal{M}(p)$, with $\tilde{\tau}$ being trajectories generated by $\mathcal{M}(p)$ and π_θ: $\tilde{\tau} = \{s_t, a_t, r_t\}_{t=0}^{T}$ where $s_{t+1} \sim \mathcal{T}_p(s_t, a_t)$, $s_0 \sim S_{0,p}$, $r_t \sim \mathcal{R}_p(s_t, a_t)$, and

$a_t \sim \pi_\theta(s_t)$. We can analogously also define the average performance over the source domain distribution as:

$$\eta_{\mathcal{D}}(\theta) = \mathbb{E}_{p\sim\mathcal{P}}\left[\eta_{\mathcal{M}}(\theta, p)\right] = \mathbb{E}_{p\sim\mathcal{P}}\left[\mathbb{E}_{\hat{\tau}}\left[\sum_{t=0}^{T-1}\gamma^t r_t(s_t, a_t) \,\middle|\, p\right]\right] = \mathbb{E}_{\tau}\left[\sum_{t=0}^{T-1}\gamma^t r_t(s_t, a_t)\right]. \tag{2.2}$$

Here, $\eta_{\mathcal{D}}(\theta)$ is the evaluation of π_θ over the source domain distribution. The corresponding expectation is over trajectories τ generated by $\mathcal{D}$ and π_θ: $\tau = \{s_t, a_t, r_t\}_{t=0}^{T}$, where $s_{t+1} \sim \mathcal{T}_{p_t}(s_t, a_t)$, $p_{t+1} = p_t$, $s_0 \sim S_{0,p_0}$, $r_t \sim \mathcal{R}_{p_t}(s_t, a_t)$, $a_t \sim \pi_\theta(s_t)$, and $p_0 \sim \mathcal{P}$. With this modified notation of trajectories, policy optimization can be invoked for policy search.

Optimizing $\eta_{\mathcal{D}}$ allows us to learn a policy that performs best in expectation over models in the source domain distribution. However, this does not necessarily lead to a robust policy, since there could be high variability in performance for different models in the distribution. To explicitly seek a robust policy, we use a softer version of max-min objective suggested in robust control, and optimize for the conditional value at risk (CVaR) [35]:

$$\max_{\theta, y} \int_{\mathcal{F}(\theta)} \eta_{\mathcal{M}}(\theta, p)\mathcal{P}(p)dp \qquad s.t. \quad \mathbb{P}\left(\eta_{\mathcal{M}}(\theta, P) \le y\right) = \epsilon, \tag{2.3}$$

where $\mathcal{F}(\theta) = \{p \mid \eta_{\mathcal{M}}(\theta, p) \le y\}$ is the set of parameters corresponding to models that produce the worst ϵ percentile of returns, and provides the limit for the integral; $\eta_{\mathcal{M}}(\theta, P)$ is the random variable of returns, which is induced by the distribution over model parameters; and ϵ is a hyperparameter which governs the level of relaxation from max-min objective. The interpretation is that (2) maximizes the expected return for the worst ϵ-percentile of MDPs in the source domain distribution. We adapt the previous policy gradient formulation to approximately optimize the objective in (2). The resulting algorithm, which we call EPOpt-ϵ, generalizes learning a policy using an ensemble of source MDPs which are sampled from a source domain distribution.

Algorithm 1 EPOpt–ϵ for Robust Policy Search

```
1:  **Input:** $\psi$, $\theta_0$, $niter$, $N$, $\epsilon$
2:  **for** iteration $i = 0, 1, 2, \dots niter$ **do**
3:      **for** $k = 1, 2, \dots N$ **do**
4:          sample model parameters $p_k \sim \mathcal{P}_\psi$
5:          sample a trajectory $\tau_k = \{s_t, a_t, r_t, s_{t+1}\}_{t=0}^{T-1}$ from $\mathcal{M}(p_k)$ using policy $\pi(\theta_i)$
6:      **end for**
7:      compute $Q_\epsilon = \epsilon$ percentile of $\{R(\tau_k)\}_{k=1}^{N}$
8:      select sub-set $\mathbb{T} = \{\tau_k : R(\tau_k) \le Q_\epsilon\}$
9:      Update policy: $\theta_{i+1} = \text{PolicyUpdate}(\theta_i, \mathbb{T})$
10: **end for**
```

In Algorithm 1, $R(\tau_k) \equiv \sum_{t=0}^{T-1} \gamma^t r_{t,k}$ denotes the discounted return obtained in trajectory sample τ_k. In line 7, we compute the $\epsilon-$percentile value of returns from the N trajectories. In line 8, we find the subset of sampled trajectories which have returns lower than Q_ϵ. Line 9 calls one step of an underlying policy optimization subroutine on the subset of trajectories from line 8. For the CVaR objective, it is important to use a good baseline for the value function. [35] show that without a baseline, the resulting policy gradient is biased and not consistent. We use a linear function as the baseline with a time varying feature vector to approximate the value function, similar to [36]. The parameters of the baseline are estimated using only the subset of trajectories with return less than Q_ϵ. We found that this approach led to empirically good results.

For small values of ϵ, we observed that using the sub-sampling step from the beginning led to unstable learning. Policy gradient methods adjust parameters of policy to increase probability of trajectories with high returns and reduce probability of poor trajectories. EPOpt$-\epsilon$ due to the sub-sampling step emphasizes penalizing poor trajectories more. This might constrain the initial exploration needed to find good trajectories. Thus, we initially use a setting of $\epsilon = 1$ for few iterations before setting epsilon to the desired value. This corresponds to exploring initially to find promising trajectories and rapidly reducing probability of trajectories that do not generalize.

2.3.2 Adapting the source domain distribution

In line with model-based Bayesian RL, we can adapt the ensemble distribution after observing trajectory data from the target domain. The Bayesian update can be written as:

$$\mathbb{P}(P|\tau_k) = \frac{1}{Z} \times \mathbb{P}(\tau_k|P) \times \mathbb{P}(P) \quad = \frac{1}{Z} \times \prod_{t=0}^{T-1} \mathbb{P}(S_{t+1} = s_{t+1}^{(k)}|s_t^{(k)}, a_t^{(k)}, p) \times \mathbb{P}(P = p), \quad (2.4)$$

where $\frac{1}{Z}$ is the partition function (normalization) required to make the probabilities sum to 1, S_{t+1} is the random variable representing the next state, and $\left(s_t^{(k)}, a_t^{(k)}, s_{t+1}^{(k)}\right)_{t=0}^{T}$ are data observed along trajectory τ_k. We try to explain the target trajectory using the stochasticity in the state-transition function, which also models sensor errors. This provides the following expression for the likelihood:

$$\mathbb{P}(S_{t+1}|s_t, a_t, p) \equiv \mathcal{T}_p(s_t, a_t). \quad (2.5)$$

We follow a sampling based approach to calculate the posterior, by sampling a set of model parameters: $p_i = [p_1, p_2, \ldots, p_M]$ from a sampling distribution, $\mathbb{P}_S(p_i)$. Consequently, using Bayes rule and importance sampling, we have:

$$\mathbb{P}(p_i|\tau_k) \propto \mathcal{L}(\tau_k|p_i) \times \frac{\mathbb{P}_P(p_i)}{\mathbb{P}_S(p_i)}, \quad (2.6)$$

where $\mathbb{P}_P(p_i)$ is the probability of drawing p_i from the prior distribution; and $\mathcal{L}(\tau_k|p_i)$ is the likelihood of generating the observed trajectory with model parameters p_i. The weighted samples from the posterior can be used to estimate a parametric model, as we do in this paper. Alternatively, one could approximate the continuous probability distribution using discrete weighted samples like in case of particle filters. In cases where the prior has very low probability density in certain parts of the parameter space, it might be advantageous to choose a sampling distribution different from the prior. The likelihood can be factored using the Markov property as: $\mathcal{L}(\tau_k|p_i) = \prod_t \mathbb{P}(S_{t+1} = s_{t+1}^{(k)}|s_t^{(k)}, a_t^{(k)}, p_i)$. This simple model adaptation rule allows us to illustrate the utility of EPOpt for robust policy search, as well as its integration with model adaptation to learn policies in cases where the target model could be very different from the initially assumed distribution.

2.4 Experiments

We evaluated the proposed EPOpt-ϵ algorithm on the 2D hopper [37] and half-cheetah [38] benchmarks using the MuJoCo physics simulator [30].[1] Both tasks involve complex second order dynamics and direct torque control. Underactuation, high dimensionality, and contact discontinuities make these tasks challenging reinforcement learning benchmarks. These challenges when coupled with systematic parameter discrepancies can quickly degrade the performance of policies and make them unstable, as we show in the experiments. The policy optimization sub-routine is implemented using TRPO. We parametrize the stochastic policy using the scheme presented in [33]. The policy is represented with a Gaussian distribution, the mean of which is parametrized using a neural network with two hidden layers. Each hidden layer has 64 units, with a *tanh* non-linearity, and the final output layer is made of linear units. Normally distributed independent random variables are added to the output of this neural network, and we also learn the standard deviation of their distributions. Our experiments are aimed at answering the following questions:

1. How does the performance of standard policy search methods (like TRPO) degrade in the presence of systematic physical differences between the training and test domains, as might be the case when training in simulation and testing in the real world?

2. Does training on a distribution of models with EPOpt improve the performance of the policy when tested under various model discrepancies, and how much does ensemble training degrade overall performance (e.g. due to acquiring a more conservative strategy)?

3. How does the robustness of the policy to physical parameter discrepancies change when using the robust EPOpt-ϵ variant of our method?

[1]Supplementary video: `https://youtu.be/w1YJ9vwaoto`

4. Can EPOpt learn policies that are robust to unmodeled effects – that is, discrepancies in physical parameters between source and target domains that *do not* vary in the source domain ensemble?
5. When the initial model ensemble differs substantially from the target domain, can the ensemble be adapted efficiently, and how much data from the target domain is required for this?

In all the comparisons, *performance* refers to the average undiscounted return per trajectory or episode (we consider finite horizon episodic problems). In addition to the previously defined performance, we also use the 10^{th} percentile of the return distribution as a proxy for the worst-case return.

2.4.1 Comparison to Standard Policy Search

In Figure 2.1, we evaluate the performance of standard TRPO and EPOpt($\epsilon = 0.1$) on the hopper task, in the presence of a simple parametric discrepancy in the physics of the system between the training (source) and test (target) domains. The plots show the performance of various policies on test domains with different torso mass. The first three plots show policies that are each trained on a single torso mass in the source domain, while the last plot

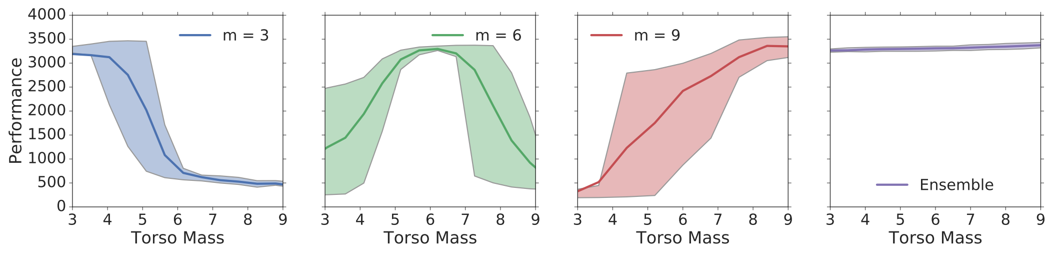


Figure 2.1: Performance of hopper policies when testing on target domains with different torso masses. The first three plots (blue, green, and red) show the performance of policies trained with TRPO on source domains with torso mass 3, 6, and 9, respectively (denoted by $m =$ in the legend). The rightmost plot shows the performance of EPOpt($\epsilon = 0.1$) trained on a Gaussian source distribution with mean mass $\mu = 6$ and standard deviation $\sigma = 1.5$. The shaded regions show the 10^{th} and 90^{th} percentile of the return distribution. Policies trained using traditional approaches on a single mass value are unstable for even slightly different masses, making the hopper fall over when trying to move forward. In contrast, the EPOpt policy is stable and achieves a high level of performance on the entire range of masses considered. Further, the EPOpt policy does not suffer from degradation in performance as a consequence of adopting a more robust policy.

illustrates the performance of EPOpt, which is trained on a Gaussian mass distribution. The results show that no single torso mass value produces a policy that is successful in all target domains. However, the EPOpt policy succeeds almost uniformly for all tested mass values. Furthermore, the results show that there is almost no degradation in the performance of EPOpt for any mass setting, suggesting that the EPOpt policy does not suffer substantially from adopting a more robust strategy.

2.4.2 Analysis of Robustness

Next, we analyze the robustness of policies trained using EPOpt on the hopper domain. For this analysis, we construct a source distribution which varies four different physical parameters: torso mass, ground friction, foot joint damping, and joint inertia (armature). This distribution is presented in Table 2.1. Using this source distribution, we compare between three different methods: (1) standard policy search (TRPO) trained on a single model corresponding to the mean parameters in Table 2.1; (2) EPOpt($\epsilon = 1$) trained on the source distribution; (3) EPOpt($\epsilon = 0.1$) – i.e. the adversarially trained policy, again trained on the previously described source distribution. The aim of the comparison is to study direct-transfer performance, similar to the robustness evaluations common in robust controller design [39]. Hence, we learn a policy using each of the methods, and then test policies on different model instances (i.e. different combinations of physical parameters) without any adaptation. The results of this comparison are summarized in Figure 2.2, where we present the performance of the policy for testing conditions corresponding to different torso mass and friction values, which we found to have the most pronounced impact on performance. The

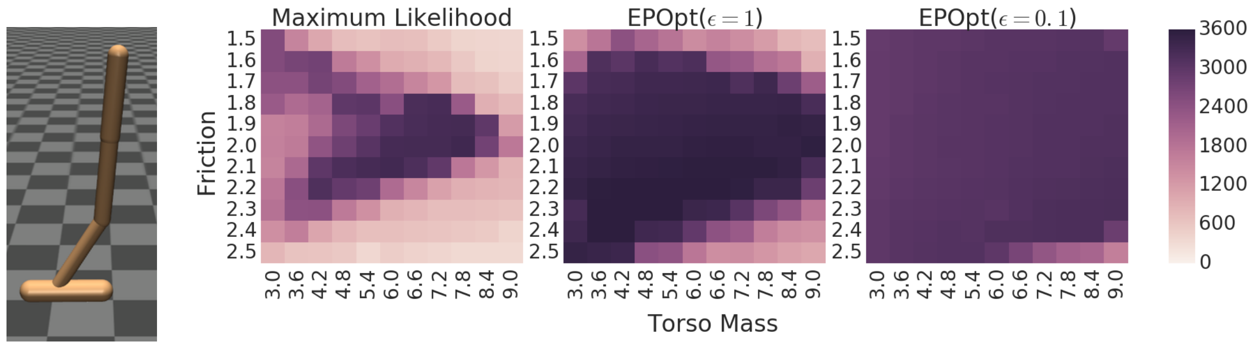


Figure 2.2: On the left, is an illustration of the simulated 2D hopper task studied in this paper. On right, we depict the performance of policies for various model instances of the hopper task. The performance is depicted as a heat map for various model configurations, parameters of which are given in the x and y axis. The adversarially trained policy, EPOpt($\epsilon = 0.1$), is observed to generalize to a wider range of models and is more robust. Table 2.1.

results indicate that EPOpt($\epsilon = 0.1$) produces highly robust policies. A similar analysis for the 10th percentile of the return distribution (softer version of worst-case performance), the half-cheetah task, and different ϵ settings are presented in the appendix.

2.4.3 Robustness to Unmodeled Effects

To analyze the robustness to unmodeled effects, our next experiment considers the setting where the source domain distribution is obtained by varying friction, damping, and armature as in Table 2.1, but does not consider a distribution over torso mass. Specifically, all models in the source domain distribution have the same torso mass (value of 6), but we will evaluate the policy trained on this distribution on target domains where the torso mass is different. Figure 2.3 indicates that the EPOpt($\epsilon = 0.1$) policy is robust to a broad range of torso masses even when its variation is not considered. However, as expected, this policy is not as robust as the case when mass is also modeled as part of the source domain distribution.

Table 2.1: Initial source domain distribution

Hopper	μ	σ	low	high
mass	6.0	1.5	3.0	9.0
ground friction	2.0	0.25	1.5	2.5
joint damping	2.5	1.0	1.0	4.0
armature	1.0	0.25	0.5	1.5
Half-Cheetah	μ	σ	low	high
mass	6.0	1.5	3.0	9.0
ground friction	0.5	0.1	0.3	0.7
joint damping	1.5	0.5	0.5	2.5
armature	0.125	0.04	0.05	0.2

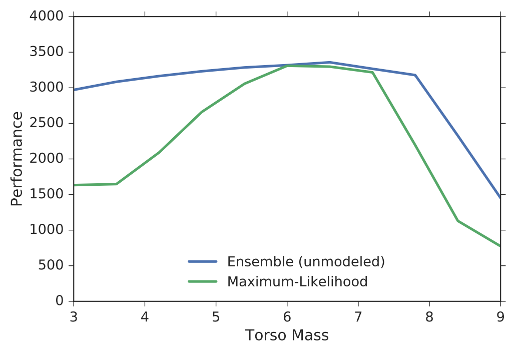
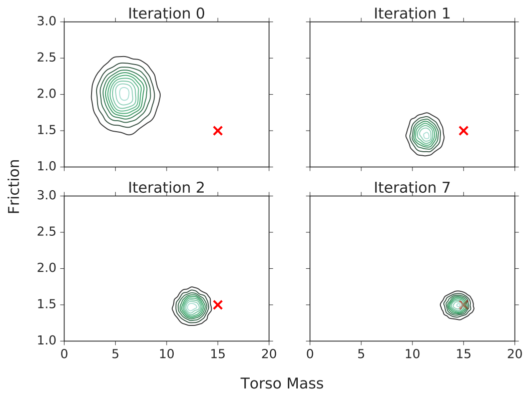


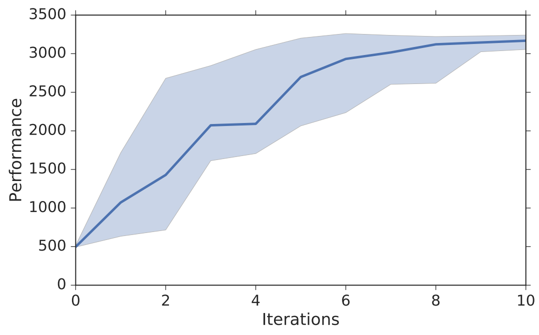
Figure 2.3: Comparison between policies trained on a fixed *maximum-likelihood* model with mass (6), and an ensemble where all models have the same mass (6) and other parameters varying as described in Table 2.1.

2.4.4 Model Adaptation

The preceding experiments show that EPOpt can find robust policies, but the source distribution in these experiments was chosen to be broad enough such that the target domain is not too far from high-density regions of the distribution. However, for real-world problems, we might not have the domain knowledge to identify a good source distribution in advance. In such settings, model (source) adaptation allows us to change the parameters of the source distribution using data gathered from the target domain. Additionally, model adaptation is helpful when the parameters of the target domain could change over time, for example due to wear and tear in a physical system. To illustrate model adaptation, we performed an experiment where the target domain was very far from the high density regions of the initial source distribution, as depicted in Figure 2.4(a). In this experiment, the source distribution varies the torso mass and ground friction. We observe that progressively, the source distribution becomes a better approximation of the target domain and consequently the performance improves. In this case, since we followed a sampling based approach, we used a uniform sampling distribution, and weighted each sample with the importance weight as described in Section 3.2. Eventually, after 10 iterations, the source domain distribution is able to accurately match the target domain. Figure 2.4(b) depicts the learning curve, and we see that a robust policy with return more than 2500, which roughly corresponds to a situation where the hopper is able to move forward without falling down for the duration of the episode, can be discovered with just 5 trajectories from the target domain. Subsequently, the policy improves near monotonically, and EPOpt finds a good policy with just 11 episodes worth of data from the target domain. In contrast, to achieve the same level of performance on the target domain, completely model-free methods like TRPO would require more than 2×10^4 trajectories when the neural network parameters are initialized randomly.



(a)



(b)

Figure 2.4: (a) Visualizes the source distribution during model adaptation on the hopper task, where mass and friction coefficient are varied in the source domain. The red cross indicates the unknown parameters of the target domain. The contours in the plot indicate the distribution over models (we assume a Gaussian distribution). Lighter colors and more concentrated contour lines indicate regions of higher density. Each iteration corresponds to one round (episode) of interaction with the target domain. The high-density regions gradually move toward the true model, while maintaining probability mass over a range of parameters which can explain the behavior of target domain. Figure 2.4(b) presents the corresponding learning curve, where the shaded region describes the 10th and 90th percentiles of the performance distribution, and the solid line is the average performance.

2.5 Relationship to Prior Work

Robust control is a branch of control theory which formally studies development of robust policies [39, 40, 41]. However, typically no distribution over source or target tasks is assumed, and a worst case analysis is performed. Most results from this field have been concentrated around linear systems or finite MDPs, which often cannot adequately model complexities of real-world tasks. The set-up of model-based Bayesian RL maintains a belief over models for decision making under uncertainty [42, 28]. In Bayesian RL, through interaction with the target domain, the uncertainty is reduced to find the correct or closest model. Application of this idea in its full general form is difficult, and requires either restrictive assumptions like finite MDPs [43], gaussian dynamics [44], or task specific innovations. Previous methods have also suggested treating uncertain model parameters as unobserved state variables in a continuous POMDP framework, and solving the POMDP to get optimal exploration-exploitation trade-off [45, 46]. While this approach is general, and allows automatic learning of epistemic actions, extending such methods to large continuous control tasks like those considered in this paper is difficult.

Risk sensitive RL methods [47, 35] have been proposed to act as a bridge between robust control and Bayesian RL. These approaches allow for using subjective model belief priors, prevent overly conservative policies, and enjoy some strong guarantees typically associated with robust control. However, their application in high dimensional continuous control tasks have not been sufficiently explored. We refer readers to García and Fernández [27] for a survey of related risk sensitive RL methods in the context of robustness and safety.

Standard model-based control methods typically operate by finding a maximum-likelihood estimate of the target model [48, 49, 50], followed by policy optimization. Use of model ensembles to produce robust controllers was explored recently in robotics. Mordatch et al. [51] use a trajectory optimization approach and an ensemble with small finite set of models; whereas we follow a sampling based direct policy search approach over a continuous distribution of uncertain parameters, and also show domain adaptation. Sampling based approaches can be applied to complex models and discrete MDPs which cannot be planned through easily. Similarly, Wang et al. [52] use an ensemble of models, but their goal is to optimize for average case performance as opposed to transferring to a target MDP. Wang et al. [52] use a hand engineered policy class whose parameters are optimized with CMA-ES. EPOpt on the other hand can optimize expressive neural network policies directly. In addition, we show model adaptation, effectiveness of the sub-sampling step ($\epsilon < 1$ case), and robustness to unmodeled effects, all of which are important for transfering to a target MDP.

Learning of parametrized skills [53] is also concerned with finding policies for a distribution of parametrized tasks. However, this is primarily geared towards situations where task parameters are revealed during test time. Our work is motivated by situations where target task parameters (e.g. friction) are unknown. A number of methods have also been suggested to reduce sample complexity when provided with either a baseline policy [54, 55], expert demonstration [56, 57], or approximate simulator [58, 59]. These are complimentary to our work, in the sense that our policy, which has good direct-transfer performance, can be used to sample from the target domain and other off-policy methods could be explored for policy improvement.

2.6 Hardware Case Study: Non-Prehensile Manipulation with EPOpt

In a follow up project [14], we employed simulation to reality transfer with EPOPt on a manipulation task involving a physical robot. In particular, we studied non-prehensile object manipulation, which refers to the setting where the object to be manipulated is not under grasp or force closure. Due to the lack of a stable grasp, non-prehensile manipulation remains a challenging control problem in robotics. In this work, we focus on a particularly challenging system using three Phantom robots as fingers. These are haptic robots that are torque-controlled and have higher bandwidth than the fingers of existing robotic hands. In terms of speed and compliance (but not strength), they are close to the capabilities of the human hand. This makes them harder to control, especially in nonprehensile manipulation tasks where the specifics of each contact event and the balance of contact forces exerted on the object are very important and need to be considered by the controller in some form.

Figure 2.5 presents an illustration of the task we consider. We use the MuJoCo [30] simulator to model the dynamics of the environment and use a physically consistent system

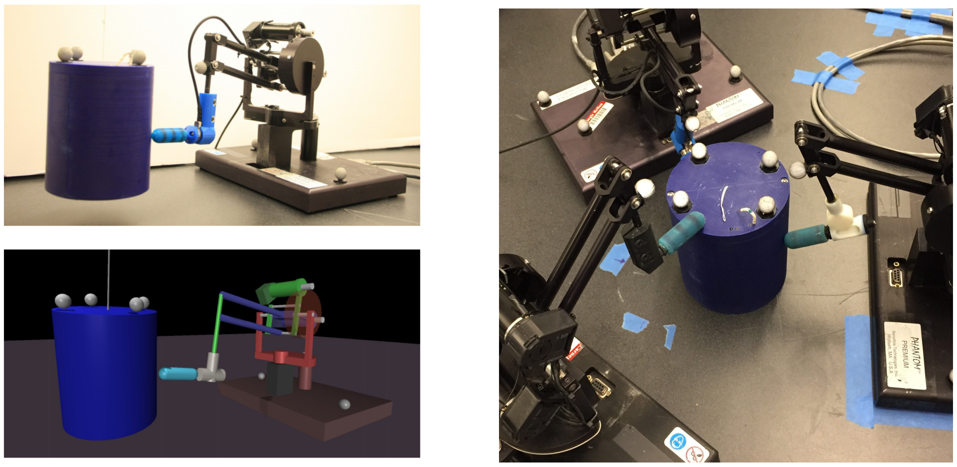


Figure 2.5: (Left) Illustration of the physical phantom robot and the simulated counterpart. We use PhaseSpace markers for tracking the robot and object. The location of the markers along with kinematic information can be used to infer the robot pose. (Right) Illustration of the task setup where three phantom robots must operate together to manipulate the cylindrical object. The task involves moving the object along a desired trajectory.

identification process [60] for parameter estimation to obtain a nominal dynamics model. As the manipulator is non-prehensile, we do not use any demonstrations or guide the policy search. To facilitate transfer to hardware, we also avoid the use of an estimator (i.e. the use of a model to predict state like a Kalman filter) by learning a function that directly converts from sensor values to motor torques. The policy is then transfered to the hardware for evaluation. We show that even for incorrect models used during training, good transfer is achieved by using an ensemble of models. A summary of results in presented in Figure 2.6, and additional details about this project can be found in the publication of Lowrey et al. [14].

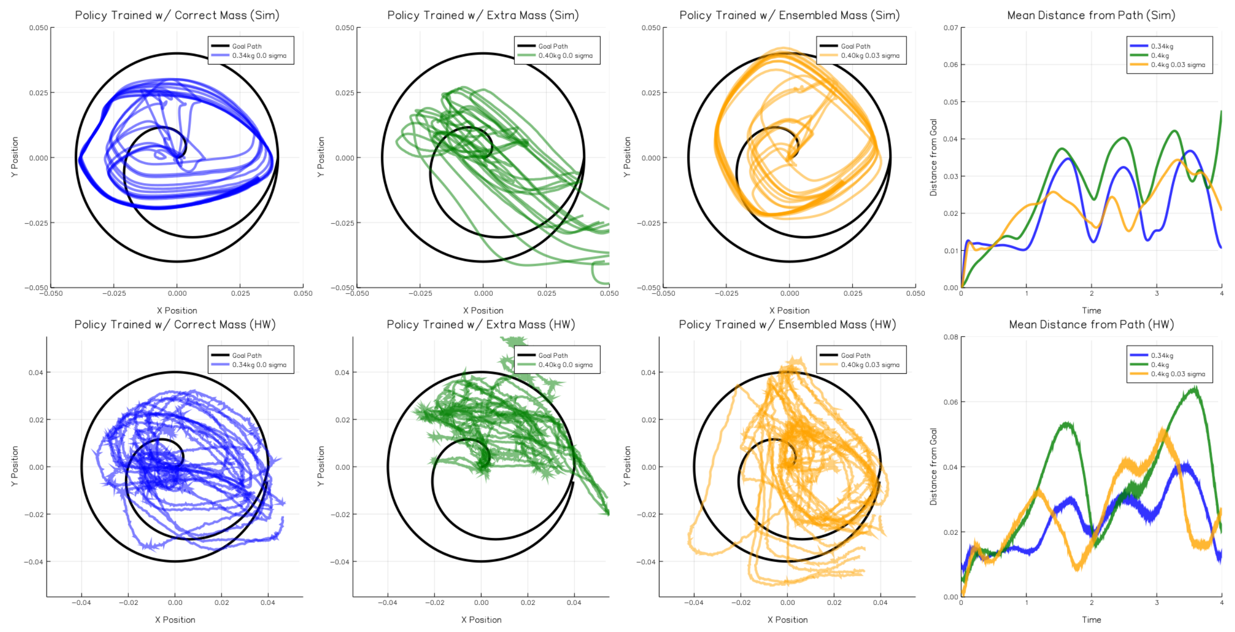


Figure 2.6: 10 rollouts are performed where the target position of the object is the path that spirals from the center outward (in black) and then performs a full circular sweep (the plots represent a top-down view). We compare three differently trained policies: one where the mass of the object cylinder is 0.34kg, one where the mass is increased by 20 percent (to 0.4kg), and finally, we train a policy with the incorrect mass, but add model noise (at standard deviation 0.03) during training to create an ensemble. We evaluate these policies on the correct (0.34kg) mass in both simulation and on hardware. In both, the policy trained with the incorrect mass does not track the goal path, sometimes even losing the object. We also calculate the per time-step error from the goal path, averaged from all 10 rollouts (right-most plots); there is usually a non-zero error between the object and the reference due to the feedback policy having to 'catch up' to the reference.

2.7 Chapter Summary

In this chapter, we discussed our work in simulation to reality transfer with emphasis on our EPOpt algorithm. EPOpt enables training of robust policies, and supports an adversarial training regime designed to provide good direct-transfer performance. We also describe how our approach can be combined with Bayesian model adaptation to adapt the source domain ensemble to a target domain using a small amount of target domain experience. Our experimental results demonstrate that the ensemble approach provides for highly robust and generalizable policies in fairly complex simulated robotic tasks. Our experiments also demonstrate that Bayesian model adaptation can produce distributions over models that lead to better policies on the target domain than more standard maximum likelihood estimation, particularly in presence of unmodeled effects.

Although our method exhibits good generalization performance, the adaptation algorithm we use currently relies on sampling the parameter space, which is computationally intensive as the number of variable physical parameters increase. We observed that (adaptive) sampling from the prior leads to fast and reliable adaptation if the true model does not have very low probability in the prior. However, when this assumption breaks, we require a different sampling distribution which could produce samples from all regions of the parameter space. This is a general drawback of Bayesian adaptation methods. In future work, we plan to explore alternative sampling and parameterization schemes, including non-parametric distributions. An eventual end-goal would be to replace the physics simulator entirely with learned Bayesian neural network models, which could be adapted with limited data from the physical system. These models could be pre-trained using physics based simulators like MuJoCo to get a practical initialization of neural network parameters. Such representations are likely useful when dealing with high dimensional inputs like simulated vision from rendered images or tasks with complex dynamics like deformable bodies, which are needed to train highly generalizable policies that can successfully transfer to physical robots acting in the real world.

Finally, on a historical note, our work was among the first to introduce the idea of using multiple (ensembles) simulated models to gain policy robustness and enable transfer to robotic hardware. Since the publication of our work, this broad research theme has gained major prominence in robot learning under the name or *domain randomization* [61]. In addition to considering multiple models that capture our uncertainty about physics parameters, the rendering characteristics of the simulator can also be randomized encouraging the learning of visuo-motor policies that are robust to various visual distractor effects. Such policies can be more robust to variations in colors, textures, and lighting conditions. Such a combination of model ensembles that encourages robustness to both variations in physical conditions as well as visual characteristics have been successful in learning RL policies for grasping [62], whole-arm manipulation [63, 64], locomotion [65, 66], and dexterous manipulation [67].

Chapter 3

OFFLINE REINFORCEMENT LEARNING

3.1 Introduction

In the previous chapter, we discussed approaches that can benefit from large scale simulated data collection, and enable transfer to a desired target domain, typically the real world. Nevertheless, due to lack of real-world data collection, this approach is fundamentally limited by the capabilities of the simulator. To mitigate against the systematic discrepancies between the simulator and target domain, and to enable the learning of better policies, in this chapter we study the possibility of reusing large scale offline datasets for RL.

The fields of computer vision and NLP have seen tremendous advances by utilizing large-scale offline datasets [68, 69, 70] for training and deploying deep learning models [2, 3, 4, 5]. In contrast, reinforcement learning (RL) [71] is typically viewed as an online learning process. The RL agent iteratively collects data through interactions with the environment while learning the policy. Unfortunately, a direct embodiment of this trial and error learning is often inefficient and feasible only with a simulator [10, 72, 21]. Similar to progress in other fields of AI, the ability to learn from offline datasets may hold the key to unlocking the sample efficiency and widespread use of RL agents.

Offline RL, also known as batch RL [73], involves learning a highly rewarding policy using only a static offline dataset collected by one or more data logging (behavior) policies. Since the data has already been collected, offline RL abstracts away data collection or exploration, and allows prime focus on data-driven learning of policies. This abstraction is suitable for safety sensitive applications like healthcare and industrial automation where careful oversight by a domain expert is necessary for taking exploratory actions or deploying new policies [74, 75]. Additionally, large historical datasets are readily available in domains like autonomous driving and recommendation systems, where offline RL may be used to improve upon currently deployed policies.

Due to use of static dataset, offline RL faces unique challenges. Over the course of learning, the agent has to evaluate and reason about various candidate policy updates. This *offline policy evaluation* is particularly challenging due to deviation between the state visitation distribution of the candidate policy and the logging policy. Furthermore, this difficulty is exacerbated over the course of learning as the candidate policies increasingly deviate from the logging policy. This change in distribution, as a result of policy updates, is typically called *distribution shift* and constitutes a major challenge in offline RL. Recent studies show that directly using off-policy RL algorithms with an offline dataset yields poor results due

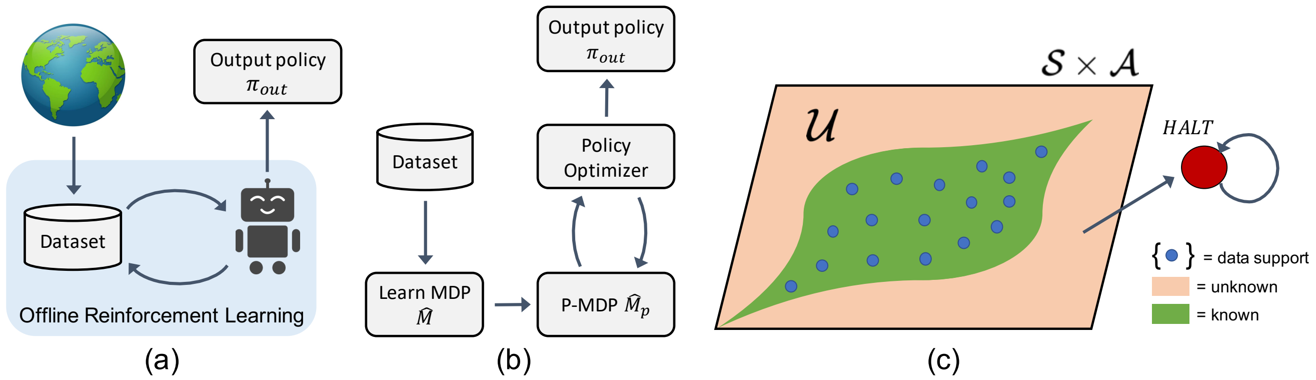


Figure 3.1: (a) Illustration of the offline RL paradigm. (b) Illustration of our framework, `MOReL`, which learns a pessimistic MDP (P-MDP) from the dataset and uses it for policy search. (c) Illustration of the P-MDP, which partitions the state-action space into known (green) and unknown (orange) regions, and also forces a transition to a low reward absorbing state (HALT) from unknown regions. Blue dots denote the support in the dataset. Function approximation and generalization allows us to learn about states not numerically identical to data support. See algorithm 2 for more details.

to distribution shift and function approximation errors [76, 77, 78]. To overcome this, prior works have proposed modifications like Q-network ensembles [76, 79] and regularization towards the data logging policy [80, 77, 79]. Most notably, prior work in offline RL has been confined almost exclusively to model-free methods [81, 76, 77, 80, 78, 79, 82].

Model-based RL (MBRL) presents an alternate set of approaches involving the learning of approximate dynamics models which can subsequently be used for policy search. MBRL enables the use of generic priors like smoothness and physics [83] for model learning, and a wide variety of planning algorithms [84, 85, 86, 87, 34]. As a result, MBRL algorithms have been highly sample efficient for online RL [16, 88]. However, direct use of MBRL algorithms with offline datasets can prove challenging, again due to the distribution shift issue. In particular, since the dataset may not span the entire state-action space, the learned model is unlikely to be globally accurate. As a result, planning using a learned model without any safeguards against model inaccuracy can result in *"model exploitation"* [89, 90, 88, 16], yielding poor results [50]. In this context, we study the pertinent question of how to effectively regularize and adapt model-based methods for offline RL.

Our Contributions: The principal contribution of our work is the development of `MOReL`: Model-based Offline Reinforcement Learning, a novel model-based framework for offline RL (see figure 3.1 for an overview). `MOReL` enjoys rigorous theoretical guarantees, enables transparent algorithm design, and offers state of the art (SOTA) results on widely studied offline RL benchmarks.

- `MOReL` consists of two modular steps: (a) learning a *pessimistic MDP* (P-MDP) using the offline dataset; and (b) learning a near-optimal policy for the P-MDP. For *any* policy, the performance in the true MDP (environment) is approximately lower bounded by the performance in the P-MDP, making it a suitable surrogate for purposes of policy evaluation and learning. This also guards against model exploitation, which often plagues MBRL.
- The P-MDP partitions the state space into "known" and "unknown" regions, and uses a large negative reward for unknown regions. This provides a regularizing effect during policy learning by heavily penalizing policies that visit unknown states. Such a regularization in the space of state visitations, afforded by a model-based approach, is particularly well suited for offline RL. In contrast, model-free algorithms [77, 79] are forced to regularize the policies directly towards the data logging policy, which can be overly conservative.
- Theoretically, we establish upper bounds for the sub-optimality of a policy learned with `MOReL`, and a lower-bound for the sub-optimality of a policy learnable by *any* offline RL algorithm. We find that these bounds match upto log factors, suggesting that `MOReL` is nearly minimax optimal.
- We evaluate `MOReL` on standard benchmark tasks used for offline RL. `MOReL` obtains SOTA results in 12 out of 20 environment-dataset configurations, and performs competitively in the rest. In contrast, the best prior algorithm [79] obtains SOTA results in only 5 (out of 20) configurations.

3.2 Related Work

Offline RL dates to at least the work of Lange et al. [73], and has applications in healthcare [91, 92, 93], recommendation systems [94, 95, 96, 97], dialogue systems [98, 80, 99], and autonomous driving [100]. Algorithms for offline RL typically fall under three categories. The first approach utilizes **importance sampling** and is popular in contextual bandits [101, 94, 95]. For full offline RL, Liu et al. [102] perform planning with learned importance weights [103, 104, 105] while using a notion of pessimism for regularization. However, Liu et al. [102] don't explicitly consider generalization and their guarantees become degenerate if the logging policy does not span the support of the optimal policy. In contrast, our approach accounts for generalization, leads to stronger theoretical guarantees, and obtains SOTA results on challenging offline RL benchmarks. The second, and perhaps most popular approach is based on **approximate dynamic programming (ADP)**. Recent works have proposed modification to standard ADP algorithms [106, 19, 22, 107] towards stabilizing Bellman targets with ensembles [78, 76, 80] and regularizing the learned policy towards the data logging policy [76, 77, 79]. ADP-based offline RL has also be studied theoretically [87, 108]. However, these works again don't study the impact of support mismatch between logging policy and optimal policy. Finally, **model-based RL** has been explored only sparsely for offline RL in literature [50, 109] (see appendix for details). The work of Ross and Bagnell [50] considered a straightforward approach of learning a model from offline data, followed by planning. They showed that this can have arbitrarily large sub-optimality. In contrast, our work develops a new framework utilizing the notion of pessimism, and shows both theoretically and experimentally that MBRL can be highly effective for offline RL. Concurrent to our work, Yu et al. [110] also study a model-based approach to offline RL.

A cornerstone of `MOReL` is the P-MDP which partitions the state space into known and unknown regions. Such a hard partitioning was considered in early works like E^3 [111], R-MAX [112], and metric-E^3 [113], but was not used to encourage pessimism. Similar ideas have been explored in related settings like online RL [114, 115] and imitation learning [116]. Our work differs in its focus on offline RL, where we show the P-MDP construction plays a crucial role. Moreover, direct practical instantiations of E^3 and metric-E^3 with function approximation have remained elusive.

3.3 Problem Formulation

A **Markov Decision Process (MDP)** is represented by $\mathcal{M} = \{S, A, r, P, \rho_0, \gamma\}$, where, S is the state-space, A is the action-space, $r : S \times A \to [-R_{\max}, R_{\max}]$ is the reward function, $P : S \times A \times S \to \mathbb{R}_+$ is the transition kernel, ρ_0 is the initial state distribution, and γ the discount factor. A policy defines a mapping from states to a probability distribution over actions, $\pi : S \times A \to \mathbb{R}_+$. The goal is to obtain a policy that maximizes expected performance with states sampled according to ρ_0, i.e.:

$$\max_{\pi} \quad J_{\rho_0}(\pi, \mathcal{M}) := \mathbb{E}_{s \sim \rho_0}\left[V^{\pi}(s, \mathcal{M})\right], \text{ where, } V^{\pi}(s, \mathcal{M}) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s\right]. \tag{3.1}$$

To avoid notation clutter, we suppress the dependence on ρ_0 when understood from context, i.e. $J(\pi, \mathcal{M}) \equiv J_{\rho_0}(\pi, \mathcal{M})$. We denote the optimal policy using $\pi^* := \arg\max_{\pi} J_{\rho_0}(\pi, \mathcal{M})$. Typically, a class of parameterized policies $\pi_\theta \in \Pi(\Theta)$ are considered, and the parameters θ are optimized.

In **offline RL**, we are provided with a static dataset of interactions with the environment consisting of $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}_{i=1}^{N}$. The data can be collected using one or more logging (or behavioral) policies denoted by π_b. We do not assume logging policies are known in our formulation. Given $\mathcal{D}$, the goal in offline RL is to output a π_{out} with minimal sub-optimality, i.e. $J(\pi^*, \mathcal{M}) - J(\pi_{\text{out}}, \mathcal{M})$. In general, it may not be possible to learn the optimal policy with a static dataset (see section 3.5). Thus, we aim to design algorithms that would result in as low sub-optimality as possible.

Model-Based RL (MBRL) involves learning an MDP $\hat{\mathcal{M}} = \{S, A, r, \hat{P}, \hat{\rho}_0, \gamma\}$ which uses the learned transitions $\hat{P}$ instead of the true transition dynamics P. In this paper, we assume the reward function is known and use it in $\hat{M}$. If $r(\cdot)$ is unknown, it can also be learned from data. The initial state distribution $\hat{\rho}_0$ can either be learned from the data or ρ_0 can be used if known. Analogous to $\mathcal{M}$, we use $J_{\hat{\rho}_0}(\pi, \hat{\mathcal{M}})$ or simply $J(\pi, \hat{\mathcal{M}})$ to denote performance of π in $\hat{M}$.

3.4 Algorithmic Framework

For ease of exposition and clarity, we first begin by presenting an idealized version of `MOReL`, for which we also establish theoretical guarantees. Subsequently, we describe a practical version of `MOReL` that we use in our experiments. Algorithm 2 presents the broad framework of `MOReL`. We now study each component of `MOReL` in greater detail.

Algorithm 2 `MOReL`: Model Based Offline Reinforcement Learning

```
1: Require Dataset $\mathcal{D}$
2: Learn approximate dynamics model $\hat{P} : S \times A \to S$ using $\mathcal{D}$.
3: Construct $\alpha$-USAD, $U^\alpha : S \times A \to \{\text{TRUE}, \text{FALSE}\}$ using $\mathcal{D}$ (see Definition 1).
4: Construct the pessimistic MDP $\hat{\mathcal{M}}_p = \{S \cup \text{HALT}, A, r_p, \hat{P}_p, \hat{\rho}_0, \gamma\}$ (see Definition 2).
5: (OPTIONAL) Use a behavior cloning approach to estimate the behavior policy $\hat{\pi}_b$.
6: $\pi_{\text{out}} \leftarrow \text{PLANNER}(\hat{\mathcal{M}}_p, \pi_{\text{init}} = \hat{\pi}_b)$
7: Return $\pi_{\text{out}}$.
```

Learning the dynamics model: The first step involves using the offline dataset to learn an approximate dynamics model $\hat{P}(\cdot|s,a)$. This can be achived through maximum likelihood estimation or other techniques from generative and dynamics modeling [117, 118, 119]. Since the offline dataset may not span the entire state space, the learned model may not be globally accurate. So, a naïve MBRL approach that directly plans with the learned model may over-estimate rewards in unfamiliar parts of the state space, resulting in a highly sub-optimal policy [50]. We overcome this with the next step.

Unknown state-action detector (USAD): We partition the state-action space into known and unknown regions based on the accuracy of learned model as follows.

Definition 1. *(α-USAD) Given a state-action pair (s, a), define an unknown state action detector as:*

$$U^\alpha(s,a) = \begin{cases} \textit{FALSE} \quad \textit{(i.e. Known)} & \text{if } D_{TV}\left(\hat{P}(\cdot|s,a), P(\cdot|s,a)\right) \le \alpha \quad \textit{can be guaranteed} \\ \textit{TRUE} \quad \textit{(i.e. Unknown)} & \textit{otherwise} \end{cases} \tag{3.2}$$

Here $D_{TV}\left(\hat{P}(\cdot|s,a), P(\cdot|s,a)\right)$ denotes the total variation distance between $\hat{P}(\cdot|s,a)$ and $P(\cdot|s,a)$. Intuitively, USAD provides confidence about where the learned model is accurate. It flags state-actions for which the model is guarenteed to be accurate as "known", while flagging state-actions where such a guarantee cannot be ascertained as "unknown". Note that USAD is based on the ability to guarantee the accuracy, and is not an inherent property of the model. In other words, there could be states where the model is actually accurate, but flagged as unknown due to the agent's inability to guarantee accuracy. Two factors contribute to USAD's effectiveness: (a) data availability: having sufficient data points "close" to the query; (b) quality of representations: certain representations, like those based on physics, can lead to better generalization guarantees. This suggests that larger datasets and research in representation learning can potentially enable stronger offline RL results.

Pessimistic MDP construction: We now construct a pessimistic MDP (P-MDP) using the learned model and USAD, which penalizes policies that venture into unknown parts of state-action space.

Definition 2. *The (α, κ)-pessimistic MDP is described by $\hat{\mathcal{M}}_p := \{S \cup HALT, A, r_p, \hat{P}_p, \hat{\rho}_0, \gamma\}$. Here, S and A are states and actions in the MDP $\mathcal{M}$. HALT is an additional absorbing state we introduce into the state space of $\hat{\mathcal{M}}_p$. $\hat{\rho}_0$ is the initial state distribution learned from the dataset $\mathcal{D}$. γ is the discount factor (same as $\mathcal{M}$). The modified reward and transition dynamics are given by:*

$$\hat{P}_p(s'|s,a) = \begin{cases} \delta(s' = \text{HALT}) & \begin{array}{l}\text{if } U^\alpha(s,a) = \text{TRUE} \\ \text{or} \quad s = \text{HALT}\end{array} \\ \hat{P}(s'|s,a) & \textit{otherwise} \end{cases} \qquad r_p(s,a) = \begin{cases} -\kappa & \text{if } s = \text{HALT} \\ r(s,a) & \textit{otherwise} \end{cases}$$

$\delta(s' = \text{HALT})$ is the Dirac delta function, which forces the MDP to transition to the absorbing state HALT. For unknown state-action pairs, we use a reward of $-\kappa$, while all known state-actions receive the same reward as in the environment. The P-MDP heavily punishes policies that visit unknown states, thereby providing a safeguard against distribution shift and model exploitation.

Planning: The final step in `MOReL` is to perform planning in the P-MDP defined above. For simplicity, we assume a planning oracle that returns an ϵ_π-sub-optimal policy in the P-MDP. A number of algorithms based on MPC [84, 120], search-based planning [121, 86], dynamic programming [19, 87], or policy optimization [34, 107, 12, 122] can be used to approximately realize this.

3.5 Theoretical Results

In order to state our results, we begin by defining the notion of hitting time.

Definition 3. *(Hitting time) Given an MDP $\mathcal{M}$, starting state distribution ρ_0, state-action pair (s,a) and a policy π, the* hitting time *$T^{\pi}_{(s,a)}$ is defined as the random variable denoting the first time action a is taken at state s by π on $\mathcal{M}$, and is equal to ∞ if a is never taken by π from state s. For a set of state-action pairs $\mathcal{S} \subseteq S \times A$, we define $T^{\pi}_{\mathcal{S}} \overset{def}{=} \min_{(s,a)\in\mathcal{S}} T^{\pi}_{(s,a)}$.*

We are now ready to present our main result with the proofs deferred to the appendix.

Theorem 1. *(Policy value with pessimism) The value of any policy π on the original MDP $\mathcal{M}$ and its $(\alpha, R_{\max})$-pessimistic MDP $\hat{\mathcal{M}}_p$ satisfies:*

$$J_{\hat{\rho}_0}(\pi, \hat{\mathcal{M}}_p) \geq J_{\rho_0}(\pi, \mathcal{M}) - \frac{2R_{max}}{1-\gamma} \cdot D_{TV}(\rho_0, \hat{\rho}_0) - \frac{2\gamma R_{max}}{(1-\gamma)^2} \cdot \alpha - \frac{2R_{max}}{1-\gamma} \cdot \mathbb{E}\left[\gamma^{T^{\pi}_{\mathcal{U}}}\right], \text{ and}$$
$$J_{\hat{\rho}_0}(\pi, \hat{\mathcal{M}}_p) \leq J_{\rho_0}(\pi, \mathcal{M}) + \frac{2R_{max}}{1-\gamma} \cdot D_{TV}(\rho_0, \hat{\rho}_0) + \frac{2\gamma R_{max}}{(1-\gamma)^2} \cdot \alpha,$$

where $T^{\pi}_{\mathcal{U}}$ denotes the hitting time of unknown states $\mathcal{U} \overset{def}{=} \{(s,a) : U^{\alpha}(s,a) = TRUE\}$ by policy π in MDP $\mathcal{M}$.

Theorem 1 can be used to bound the suboptimality of output policy π_{out} of Algorithm 2.

Corollary 1. *Suppose PLANNER in Algorithm 2 returns an ϵ_π sub-optimal policy. Then, we have*

$$J_{\rho_0}(\pi^*, \mathcal{M}) - J_{\rho_0}(\pi_{out}, \mathcal{M}) \leq \epsilon_\pi + \frac{4R_{max}}{1-\gamma} \cdot D_{TV}(\rho_0, \hat{\rho}_0) + \frac{4\gamma R_{max}}{(1-\gamma)^2} \cdot \alpha + \frac{2R_{max}}{1-\gamma} \cdot \mathbb{E}\left[\gamma^{T^{\pi^*}_{\mathcal{U}}}\right].$$

Theorem 1 indicates that the difference in any policy π's value in the $(\alpha, R_{\max})$ pessimistic MDP $\hat{\mathcal{M}}_p$ and the original MDP $\mathcal{M}$ depends on: i) the total variation distance between the true and learned starting state distribution $D_{TV}(\rho_0, \hat{\rho}_0)$, ii) the maximum total variation distance α between the learned model $\hat{P}(\cdot|s,a)$ and the true model $P(\cdot|s,a)$ over all *known* states i.e., $\{(s,a)|U^{\alpha}(s,a) = \text{FALSE}\}$ and, iii) the hitting time $T^{\pi^*}_{\mathcal{U}}$ of unknown states $\mathcal{U}$ on the original MDP $\mathcal{M}$ under the optimal policy π^*. As the dataset size increases, $D_{TV}(\rho_0, \hat{\rho}_0)$ and α approach zero, indicating $\mathbb{E}\left[\gamma^{T^{\pi^*}_{\mathcal{U}}}\right]$ determines the sub-optimality in the limit. For comparison to prior work, Lemma 4 in Appendix B.1 bounds this quantity in terms of state-action visitation distribution, which for a policy π on $\mathcal{M}$ is expressed as $d^{\pi,\mathcal{M}}(s,a) \overset{\text{def}}{=} (1-\gamma)\sum_{t=0}^{\infty} \gamma^t P(s_t = s, a_t = a | s_0 \sim \rho_0, \pi, \mathcal{M})$. Furthermore, we can also show that `MOReL` learns a policy that improves over the behavioral policy with high probability. The following lemma presents both of these results:

Lemma 1. *(Upper bound;* `MOReL` *improves over the behavioral policy) Suppose $\rho_{0,min} > 0$, $p_{min} > 0$ and $d^{\pi_b}_{min} > 0$ are the smallest non-zero elements of initial distribution ρ_0, state transition probabilities $P(\cdot|s,a)$, and discounted state probability distribution $d^{\pi_b,\mathcal{M}}(s,a)$ respectively. If the dataset $\mathcal{D}$ consists of $n \geq \frac{C}{\left(d^{\pi_b}_{min}\right)^2} \cdot \log \frac{1}{\delta d^{\pi_b}_{min}}$ independent trajectories sampled according to a behavior policy π_b with initial distribution ρ_0, then the output π_{out} of Algorithm 2 satisfies:*

$$\begin{aligned}
&J_{\rho_0}(\pi_b, \mathcal{M}) - J_{\rho_0}(\pi_{out}, \mathcal{M}) \leq \epsilon_\pi + \epsilon_n, \ \ \textit{and} \\
&J_{\rho_0}(\pi^*, \mathcal{M}) - J_{\rho_0}(\pi_{out}, \mathcal{M}) \leq \epsilon_\pi + \frac{2R_{max}}{1-\gamma} \cdot \mathbb{E}\left[\gamma^{T^{\pi^*}_{\mathcal{U}}}\right] + \epsilon_n \leq \epsilon_\pi + \frac{2R_{\max}}{(1-\gamma)^2} \cdot d^{\pi^*,\mathcal{M}}(\mathcal{U}) + \epsilon_n,
\end{aligned}$$

with probability at least $1 - C\delta$, where C is a large enough constant and

$$\epsilon_n \overset{def}{=} \frac{4CR_{max}}{(1-\gamma)\rho_{0,min}} \cdot \sqrt{\frac{\log \frac{1}{\delta \rho_{0,min}}}{n}} + \frac{4C\gamma R_{max}}{(1-\gamma)^2 p_{min}} \cdot \sqrt{\frac{\log \frac{1}{\delta p_{min} d^{\pi_b}_{min}}}{d^{\pi_b}_{min} \cdot n}}$$

is an error term related to finite samples that goes to zero as $n \to \infty$.

The bound consists of three terms: (i) a sampling error term ϵ_n which decreases with larger dataset sizes that is typical of offline RL; (ii) an optimization error term ϵ_π that can be made small with additional compute to find the optimal policy in the learned model; and (iii) a distribution shift term that depends on the coverage of the offline dataset and overlap with the optimal policy.

Prior results [76, 102] assume that $d^{\pi^*,\mathcal{M}}(\mathcal{U}_D) = 0$, where $\mathcal{U}_D \overset{\text{def}}{=} \{(s,a)|(s,a,r,s') \notin \mathcal{D}\} \supseteq \mathcal{U}$ is the set of state action pairs that don't occur in the offline dataset, and guarantee finding an optimal policy under this assumption. Our result significantly improves upon these in three ways: i) $\mathcal{U}_D$ is replaced by a smaller set $\mathcal{U}$, leveraging the generalization ability of learned dynamics model, ii) the sub-optimality bound is extended to the setting where full support coverage is not satisfied i.e., $d^{\pi^*,\mathcal{M}}(\mathcal{U}) > 0$, and iii) the sub-optimality bound on π_{out} is stated in terms of unknown state hitting time $T^{\pi^*}_{\mathcal{U}}$, which can be significantly better than a bound that depends only on $d^{\pi^*,\mathcal{M}}(\mathcal{U})$. To further strengthen our results, the following proposition shows that Lemma 1 is tight up to log factors.

Proposition 1. *(Lower bound) For any discount factor $\gamma \in [0.95, 1)$, support mismatch $\epsilon \in \left(0, \frac{1-\gamma}{\log \frac{1}{1-\gamma}}\right]$ and reward range $[-R_{max}, R_{max}]$, there is an MDP $\mathcal{M}$, starting state distribution ρ_0, optimal policy π^* and a dataset collection policy π_b such that i) $d^{\pi^*,\mathcal{M}}(\mathcal{U}_D) \leq \epsilon$, and ii) any policy $\hat{\pi}$ that is learned solely using the dataset collected with π_b satisfies:*

$$J_{\rho_0}(\pi^*, \mathcal{M}) - J_{\rho_0}(\hat{\pi}, \mathcal{M}) \geq \frac{R_{max}}{4(1-\gamma)^2} \cdot \frac{\epsilon}{\log \frac{1}{1-\gamma}},$$

where $\mathcal{U}_D \overset{def}{=} \{(s,a) : (s,a,r,s') \notin \mathcal{D} \text{ for any } r, s'\}$ denotes state-actions not in the dataset $\mathcal{D}$.

We see that for $\epsilon < (1-\gamma)/(\log \frac{1}{1-\gamma})$, the lower bound obtained by Proposition 1 on the suboptimality of any offline RL algorithm matches the asymptotic (as $n \to \infty$) upper bound of Lemma 1 up to an additional log factor. For $\epsilon > (1-\gamma)/(\log \frac{1}{1-\gamma})$, Proposition 1 also implies (by choosing $\epsilon' = (1-\gamma)/(\log \frac{1}{1-\gamma}) < \epsilon$) that any offline algorithm must suffer at least constant factor suboptimality in the worst case. Finally, we note that as the size of dataset $\mathcal{D}$ increases to ∞, Theorem 1 and the optimality of PLANNER (i.e., $\epsilon_\pi = 0$) together imply that $J_{\rho_0}(\pi_{\text{out}}, \mathcal{M}) \geq J_{\rho_0}(\pi_b, \mathcal{M})$.

3.6 Practical Implementation Of `MOReL`

We now present a practical instantiation of `MOReL` (algorithm 2) utilizing a recent model-based NPG approach [16]. The principal difference is the specialization to offline RL and construction of the P-MDP using an ensemble of learned dynamics models.

Dynamics model learning: We consider Gaussian dynamics models [16] of the form $\hat{P}(\cdot|s,a) \equiv \mathcal{N}\left(f_\phi(s,a), \Sigma\right)$, with mean $f_\phi(s,a) = s + \sigma_\Delta \ \text{MLP}_\phi\left((s-\mu_s)/\sigma_s, (a-\mu_a)/\sigma_a\right)$, where $\mu_s, \sigma_s, \mu_a, \sigma_a$ are the mean and standard deviations of states/actions in $\mathcal{D}$; σ_Δ is the standard deviation of state differences, i.e. $\Delta = s' - s, (s, s') \in \mathcal{D}$; this parameterization ensures local continuity since the MLP learns only the state differences. The MLP parameters are optimized using maximum likelihood estimation with mini-batch stochastic optimization using Adam [123].

Unknown state-action detector (USAD): In order to partition the state-action space into known and unknown regions, we use uncertainty quantification [124, 125, 126, 127]. In particular, we consider approaches that track uncertainty using the predictions of ensembles of models [124, 127]. We learn multiple models $\{f_{\phi_1}, f_{\phi_2}, \ldots\}$ where each model uses a different weight initialization and are optimized with different mini-batch sequences. Subsequently, we compute the ensemble discrepancy as $\text{disc}(s,a) = \max_{i,j} \ \left\| f_{\phi_i}(s,a) - f_{\phi_j}(s,a) \right\|_2$, where f_{ϕ_i} and f_{ϕ_j} are members of the ensemble. With this, we implement USAD as below, with threshold being a tunable hyperparameter.

$$U_{\text{practical}}(s,a) = \begin{cases} \text{FALSE} \ \ \text{(i.e. Known)} & \text{if} \ \ \text{disc}(s,a) \leq \text{threshold} \\ \text{TRUE} \ \ \text{(i.e. Unknown)} & \text{if} \ \ \text{disc}(s,a) > \text{threshold} \end{cases}. \tag{3.3}$$

3.7 Experiments

Through our experimental evaluation, we aim to answer the following questions:

1. **Comparison to prior work:** How does `MOReL` compare to prior SOTA offline RL algorithms [76, 77, 79] in commonly studied benchmark tasks?
2. **Quality of logging policy:** How does the quality (value) of the data logging (behavior) policy, and by extension the dataset, impact the quality of the policy learned by `MOReL`?
3. **Importance of pessimistic MDP:** How does `MOReL` compare against a naïve model-based RL approach that directly plans in a learned model without any safeguards?
4. **Transfer from pessimistic MDP to environment:** Does learning progress in the P-MDP, which we use for policy learning, effectively translate or transfer to learning progress in the environment?

To answer the above questions, we consider commonly studied benchmark tasks from OpenAI gym [128] simulated with MuJoCo [30]. Our experimental setup closely follows prior work [76, 77, 79]. The tasks considered include `Hopper-v2`, `HalfCheetah-v2`, `Ant-v2`, and `Walker2d-v2`, which are illustrated in Figure 3.2. We consider five different logged datasets for each environment, totalling 20 environment-dataset combinations. Datasets are collected based on the work of Wu et al. [79], with each dataset containing the equivalent of 1 million timesteps of environment interaction. We first partially train a policy (π_p) to obtain values around 1000, 4000, 1000, and 1000 respectively for the four environments. The first exploration strategy, `Pure`, involves collecting the dataset solely using π_p. The four other datasets are collected using a combination of π_p, a *noisy* variant of π_p, and an untrained random policy. The noisy variant of π_p utilizes either epsilon-greedy or Gaussian noise, resulting in configurations `eps-1`, `eps-3`, `gauss-1`, `gauss-3` that signify various types and magnitudes of noise added to π_p. Please see appendix for additional experimental details.

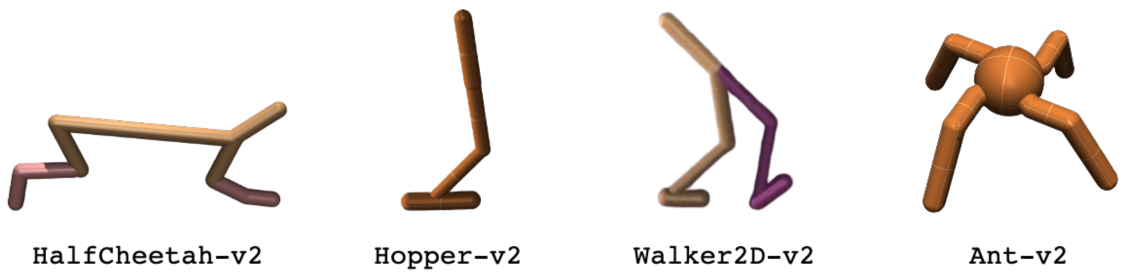


Figure 3.2: Illustration of the suite of tasks considered in this work. These tasks require the RL agent to learn locomotion gaits for the illustrated simulated characters.

Table 3.1: Results in various environment-exploration combinations. Baselines are reproduced from Wu et al. [79]. Prior work does not provide error bars. For MOReL results, error bars indicate the standard deviation across 5 different random seeds. We choose SOTA result based on the average performance.

Environment: Ant-v2

Algorithm	BCQ [76]	BEAR [77]	BRAC [79]	Best Baseline	`MOReL` (Ours)
`Pure`	1921	2100	2839	2839	**3663±247**
`Eps-1`	1864	1897	2672	2672	**3305±413**
`Eps-3`	1504	2008	2602	2602	**3008±231**
`Gauss-1`	1731	2054	2667	2667	**3329±270**
`Gauss-3`	1887	2018	2640	2661	**3693±33**

Environment: Hopper-v2

Algorithm	BCQ [76]	BEAR [77]	BRAC [79]	Best Baseline	`MOReL` (Ours)
`Pure`	1543	0	2291	2774	**3642±54**
`Eps-1`	1652	1620	2282	2360	**3724±46**
`Eps-3`	1632	2213	1892	2892	**3535±91**
`Gauss-1`	1599	1825	2255	2255	**3653±52**
`Gauss-3`	1590	1720	1458	2097	**3648±148**

Environment: HalfCheetah-v2

Algorithm	BCQ [76]	BEAR [77]	BRAC [79]	Best Baseline	`MOReL` (Ours)
`Pure`	5064	5325	6207	**6209**	**6028±192**
`Eps-1`	5693	5435	**6307**	**6307**	5861±192
`Eps-3`	5588	5149	6263	**6359**	5869±139
`Gauss-1`	5614	5394	**6323**	**6323**	6026±74
`Gauss-3`	5837	5329	**6400**	**6400**	5892±128

Environment: Walker-v2

Algorithm	BCQ [76]	BEAR [77]	BRAC [79]	Best Baseline	`MOReL` (Ours)
`Pure`	2095	2646	2694	2907	**3709±159**
`Eps-1`	1921	2695	3241	**3490**	2899±588
`Eps-3`	1953	2608	**3255**	**3255**	**3186±92**
`Gauss-1`	2094	2539	2893	3193	**4027±314**
`Gauss-3`	1734	2194	**3368**	**3368**	**2828±589**

We parameterize the dynamics model using 2-layer ReLU-MLPs and use an ensemble of 4 dynamics models to implement USAD as described in Section 3.6. We parameterize the policy using a 2-layer tanh-MLP, and train it using model-based NPG [16]. We evaluate the learned policies using rollouts in the (real) environment, but these rollouts are not made available to the algorithm in any way for purposes of learning. This is similar to evaluation protocols followed in prior work [79, 76, 77]. We present all our results averaged over 5 different random seeds. Note that we use the same hyperparameters for all random seeds. In contrast, the prior works whose results we compare against tune hyper-parameters separately for each random seed [79].

Comparison of `MOReL`'s performance with prior work We compare results of `MOReL`with prior SOTA algorithms like BCQ, BEAR, and all variants of BRAC. The results are summarized in Table 3.1. For fairness of comparison, we reproduce results from prior work and do not run the algorithms ourselves. We provide a more expansive table with additional baseline algorithms in the appendix. Our algorithm, `MOReL`, achives SOTA results in 12 out of the 20 environment-dataset combinations, overlaps in error bars for 3 other combinations, and is competitive in the remaining cases. In contrast, the next best approach (a variant of BRAC) achieves SOTA results in only 5 out of 20 configurations.

Table 3.2: Results of various algorithms on the D4RL benchmark suite. Each number is the normalized score computed as (score − random policy score) / (expert policy score − random policy score). The raw score for `MOReL` was taken to be the average over the last 100 iterations of policy learning averaged over 3 random seeds. Results of MOPO [110] and CQL [130] are reported from their respective papers. Remaining results are taken from the D4RL benchmark suite white-paper [129].

Dataset	**Environment**	**MOReL (Ours)**	**MOPO**	**CQL**	**SAC-Off**	**BEAR**	**BRAC-p**	**BRAC-v**
random	halfcheetah	25.6	34.4	**35.4**	30.5	25.1	24.1	31.2
random	hopper	**53.6**	11.7	10.8	11.3	11.4	11	12.2
random	walker2d	**37.3**	13.6	7	4.1	7.3	-0.2	1.9
medium	halfcheetah	42.1	42.3	44.4	-4.3	41.7	43.8	**46.3**
medium	hopper	**95.4**	28.0	86.6	0.8	52.1	32.7	31.1
medium	walker2d	77.8	17.8	74.5	0.9	59.1	77.5	**81.1**
medium-replay	halfcheetah	40.2	**53.1**	46.2	-2.4	38.6	45.4	47.7
medium-replay	hopper	**93.6**	67.5	48.6	3.5	33.7	0.6	0.6
medium-replay	walker2d	**49.8**	39.0	32.6	1.9	19.2	-0.3	0.9
medium-expert	halfcheetah	53.3	**63.3**	62.4	1.8	53.4	44.2	41.9
medium-expert	hopper	108.7	23.7	**111**	1.6	96.3	1.9	0.8
medium-expert	walker2d	95.6	44.6	**98.7**	-0.1	40.1	76.9	81.6
Average	Average	**64.42**	36.58	54.85	4.13	39.83	29.80	31.44

Comparison of `MOReL`'s performance in the D4RL benchmark suite The D4RL benchmark suite [129] for offline RL was introduced in concurrent work. We also study the performance of `MOReL` in this benchmark suite. We find that `MOReL` achieves the highest (normalized) score in 5 out of 12 domains studied, while the next best algorithm (CQL) achieves the highest score in only 3 out of 12 domains. Furthermore, we observe that `MOReL` is often very competitive with the best performing algorithm in any given domain even if it doesn't achieve the top score. However, in many domains, `MOReL` significantly improves over the state of the art (e.g. hopper-medium-replay and hopper-random). To aggregate results across multiple domains, we consider the average of the normalized scores as a proxy, and observe that `MOReL` significantly outperforms prior algorithms.

Importance of Pessimistic MDP To highlight the importance of P-MDP, we again consider the `Pure-partial` dataset outlined above. We compare `MOReL` with a naïve MBRL approach that first learns a dynamics model using the offline data, followed by running model-

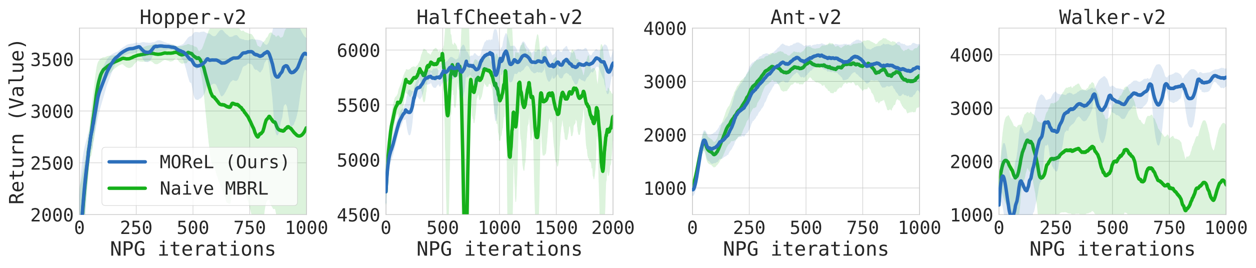


Figure 3.3: `MOReL` and Naive MBRL learning curves. The x-axis plots the number of model-based NPG iterations, while y axis plots the return (value) in the real environment. The naive MBRL algorithm is highly unstable while `MOReL` leads to stable and near-monotonic learning. Notice however that even naive MBRL learns a policy that performs often as well as the best model-free offline RL algorithms.

based NPG without any safeguards against model inaccuracy. The results are summarized in Figure 3.3. We observe that the naïve MBRL approach already works well, achieving results comparable to prior algorithms like BCQ and BEAR. However, `MOReL` clearly exhibits more stable and monotonic learning progress. This is particularly evident in `Hopper-v2`, `HalfCheetah-v2`, and `Walker2d-v2`, where an uncoordinated set of actions can result in the agent falling over. Furthermore, in the case of naïve MBRL, we observe that performance can quickly degrade after a few hundred steps of policy improvement, such as in case of `Hopper-v2`, `HalfCheetah-v2` and `Walker2d-v2`. This suggests that the learned model is being over-exploited. In contrast, with `MOReL`, we observe that the learning curve is stable and nearly monotonic even after many steps of policy improvement.

Quality of logging policy Section 3.5 indicates that it is not possible for any offline RL algorithm to learn a near-optimal policy when faced with support mismatch between the dataset and optimal policy. To verify this experimentally for `MOReL`, we consider two datasets

Table 3.3: Value of the policy learned by `MOReL` (5 random seeds) when working with a dataset collected with a random (untrained) policy (`Pure-random`) and a partially trained policy (`Pure-partial`).

Environment	`Pure-random`	`Pure-partial`
`Hopper-v2`	2354 ± 443	3642 ± 54
`HalfCheetah-v2`	2698 ± 230	6028 ± 192
`Walker2d-v2`	1290 ± 325	3709 ± 159
`Ant-v2`	1001 ± 3	3663 ± 247

(of the same size) collected using the `Pure` strategy. The first uses a partially trained policy π_p (called `Pure-partial`), which is the same as the `Pure` dataset studied in Table 3.1. The second dataset is collected using an untrained random Gaussian policy (called `Pure-random`). Table 3.3 compares the results of `MOReL` using these two datasets. We observe that the value of policy learned with `Pure-partial` dataset far exceeds the value with the `Pure-random` dataset. Thus, the value of policy used for data logging plays a crucial role in the performance achievable with offline RL.

Transfer from P-MDP to environment Finally, we study how the learning progress in P-MDP relates to the progress in the environment. Our theoretical results (Theorem 1) suggest that the value of a policy in the P-MDP cannot substantially exceed the value in the environment. This makes the value in the P-MDP an approximate lower bound on the true performance, and a good surrogate for optimization. In Figure 3.4, we plot the value or return of the policy in the P-MDP and environment over the course of learning. Note that the policy is being learned in the P-MDP, and as a result we observe a clear monotonic learning curve for value in the P-MDP, consistent with the monotonic improvement theory of policy gradient methods [55, 33]. We observe that the value in the true environment closely correlates with the value in P-MDP. In particular, the P-MDP value never substantially exceeds the true performance, suggesting that the pessimism helps to avoid model exploitation.

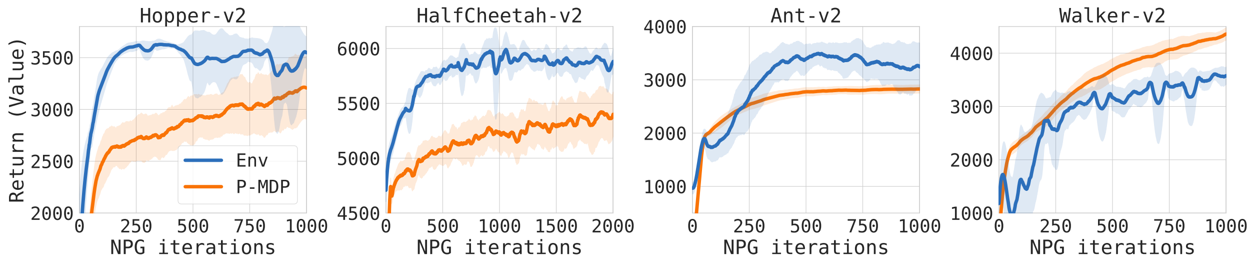


Figure 3.4: Learning curve using the `Pure-partial` dataset, see paper text for details. The policy is learned using the pessimistic MDP (P-MDP), and we plot the performance in both the P-MDP and the real environment over the course of learning. We observe that the performance in the P-MDP closely tracks the true performance and never substantially exceeds it, as predicted in section 3.5. This shows that the policy value in the P-MDP serves as a good surrogate for the purposes of offline policy evaluation and learning.

3.8 Model-Based Offline RL from Vision

We also studied extending our model-based RL approach to offline RL from images. Due to the high dimensionality of the observation space, learning predictive forward dynamics models in the image space is not straightforward. To address the scalability challenge and also to learn models that are amenable to policy learning, we develop a new algorithm – LOMPO – that learns variational dynamics models with a compact latent state that serves as an information bottleneck. Our key insight is that latent space variational models are particularly well suited because: (a) they provide a rich auxiliary objective for representation learning which also enables efficient policy learning; (b) the learned models can be simulated efficiently in the latent space over long horizons without requiring any explicit image reconstruction; (c) the model uncertainty can be estimated directly in the compact latent space, thereby imparting the necessary conservatism needed for offline RL.

We model the setting as a partially observable Markov decision process (POMDP), which can be described with the tuple: $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{X}, \mathcal{R}, T, \mathcal{U}, \gamma)$, where $\boldsymbol{s} \in \mathcal{S}$ is the state space, $\boldsymbol{a} \in \mathcal{A}$ is the action space, $\boldsymbol{x} \in \mathcal{X}$ is the observation space and $r = \mathcal{R}(\boldsymbol{s}, \boldsymbol{a})$ is a reward function. The state evolution is Markovian and governed by the dynamics as $\boldsymbol{s}' \sim \mathcal{T}(\cdot|\boldsymbol{s}, \boldsymbol{a})$. Finally, the observations are generated through the observation model $\boldsymbol{x} \sim \mathcal{U}(\cdot|\boldsymbol{s})$. With such a framework, the probability of a trajectory can be described by:

$$\log P(\boldsymbol{x}_{1:T}|\boldsymbol{a}_{1:T}) = \log \int \prod_{t=1}^{T} \mathcal{U}(\boldsymbol{x}_t|\boldsymbol{s}_t)\mathcal{T}(\boldsymbol{s}_t|\boldsymbol{a}_{t-1}, \boldsymbol{s}_{t-1})d\boldsymbol{s}_{1:T} \tag{3.4}$$

We can introduce the belief distribution $q(\boldsymbol{z}_{1:T}|\boldsymbol{x}_{1:T}, \boldsymbol{a}_{1:T-1}) = \prod_{t=1}^{T} q(\boldsymbol{z}_t|\boldsymbol{x}_t, \boldsymbol{z}_{t-1}, \boldsymbol{a}_{t-1})$, which uses $\boldsymbol{z}_t$ as an approximation for a sufficient statistic for the history till time t. With this latent belief distribution, we can construct the evidence lowerbound (ELBO) for the trajectory likelihood as [131, 132]:

$$\log P(\boldsymbol{x}_{1:T}|\boldsymbol{a}_{1:T}) \geq \mathbb{E}_{q(\boldsymbol{z}_{1:T}|\boldsymbol{x}_{1:T}, \boldsymbol{a}_{1:T-1})}\left[\log \prod_{t=1}^{T} \mathcal{U}(\boldsymbol{x}_t|\boldsymbol{z}_t)\frac{\mathcal{T}(\boldsymbol{z}_t|\boldsymbol{a}_{t-1}, \boldsymbol{z}_{t-1})}{q(\boldsymbol{z}_t|\boldsymbol{x}_t, \boldsymbol{z}_{t-1}, \boldsymbol{a}_{t-1})}\right].$$

After simplification of right hand side on the above bound, we can optimize the following quantity as a lower bound for the log likelihood:

$$\max_{\theta} \quad \mathbb{E}_{q_\theta}\Big[\sum_{t=1}^{T} \underbrace{\log D_\theta(\boldsymbol{x}_t|\boldsymbol{z}_t)}_{\text{reconstruction}} - \underbrace{\mathbb{D}_{KL}(q_\theta(\boldsymbol{z}_t|\boldsymbol{x}_t, \boldsymbol{z}_{t-1}, \boldsymbol{a}_{t-1})||\mathcal{T}_\theta(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}, \boldsymbol{a}_{t-1}))}_{\text{forward model}}\Big]. \tag{3.5}$$

The first term is a reconstruction term, which requires a decoder D_θ to be able to reconstruct the image given the latent state. This encourages the latent state to have sufficient information to be able to perform this reconstruction. The second forward model term encourages the

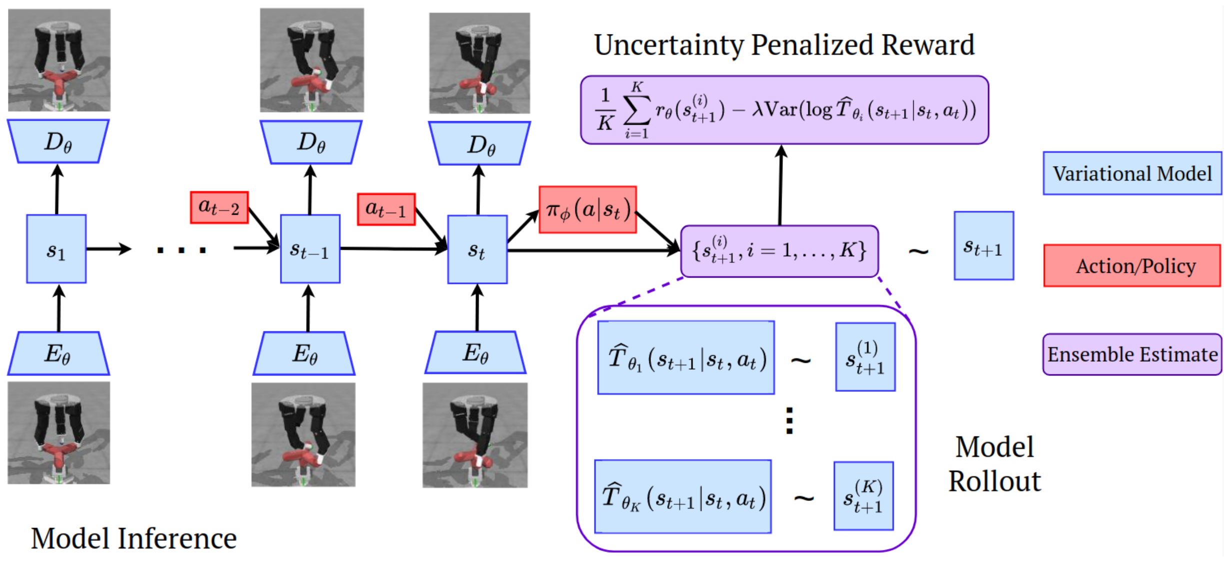

Uncertainty Penalized Reward
$\frac{1}{K}\sum_{i=1}^{K} r_\theta(s_{t+1}^{(i)}) - \lambda \mathrm{Var}(\log \widehat{T}_{\theta_i}(s_{t+1}|s_t,a_t))$
D_θ
E_θ
s_1
s_{t-1}
s_t
a_{t-2}
a_{t-1}
$\pi_\phi(a|s_t)$
$\{s_{t+1}^{(i)}, i=1,\dots,K\}$
s_{t+1}
$\widehat{T}_{\theta_1}(s_{t+1}|s_t,a_t)$
$s_{t+1}^{(1)}$
$\widehat{T}_{\theta_K}(s_{t+1}|s_t,a_t)$
$s_{t+1}^{(K)}$
Variational Model
Action/Policy
Ensemble Estimate
Model Rollout
Model Inference

Figure 3.5: Images are passed through a convolutional encoder E_θ to form a compact representation which are then used along with previous state to infer the current state s_t. We have used the latent state of the POMDP and our state representation interchangeably for simplicity. The model is trained by reconstructing the images from the latent states through the decoder network D_θ. Latent rollouts are carried by choosing a random learned transition model and rewards are penalized based on ensemble disagreement.

learned latent space model to be consistent with the encoding of the next observation. This ensures that the learned forward model propagates information forward over time that is consistent with the belief distribution. At the end of this exercise, we would obtain an encoder, a forward dynamics model, and a decoder. We will perform offline policy learning entirely in the latent space. Finally, this policy can be composed with the image encoder to recover a visuo-motor control policy. We can also learn ensembles of models in the latent space and use discripancy between the members of the ensemble as a proxy for model uncertainty and error. A schematic description of the algorithm architecture is provided in Figure 3.5. In our implementation, for simplicity, we use an additive uncertainty penalty $\tilde{r}(s,a) = r(s,a) - \lambda u(s,a)$ where $u(s,a)$ is the uncertainty detector. This is in contrast to the threshold penalty used in MOReL, although both lead to similar effects for appropriate choices of the uncertainty weight and the threshold hyperparameters.

3.8.1 Results

We experimentally evaluate our algorithm LOMPO along the following questions: (1) Can offline RL reliably scale to realistic robot environments with complex dynamics and interactions? (2) How does LOMPO compare to prior offline model-free RL algorithms and online

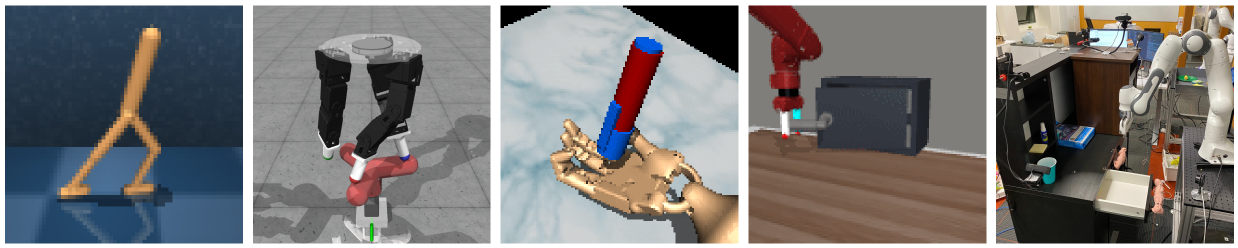


Figure 3.6: Test environments: DeepMind Control Walker task - the observations are raw 64×64 images. Robel D'Claw Screw and Adroit Pen tasks observations are raw 128×128 images and robot proprioception. Sawyer Door open environment - the observation space is raw 128×128 images. The observations for the real robot environment are raw 64×64 images from the overhead camera.

model-based RL algorithms when learning vision-based control tasks from offline data? (3) How does the quality and size of the dataset affect performance? (4) Can LOMPO be applied to an offline RL task on a real robot with raw camera image observations? To answer those three questions, we design a suite of four simulated image-based offline RL problems visualized in Figure 3.6, as well as a real-world drawer opening task.

Comparisons. We compare our proposed method to both model-free and model-based learning algorithms. Our first comparison is direct behavior cloning (BC) from raw image observations, which has proved to be a strong baseline in the past ([133]). We also benchmark to Conservative Q-Learning ([134]), which is a state of the art offline learning algorithm in the low-dimensional case ([133]); however, we again train it from raw image observations. We evaluate an MBPO based model ([88]), which also carries out policy rollouts in latent space similar to LOMPO, but does not apply an uncertainty penalty. The performance of this method is indicative of online model-based methods [135]. We also train the Stochastic Latent Actor Critic (SLAC) model ([136]), a state of the art online learning algorithm from images, however we train fully online. The goal of this benchmark is to evaluate the need for representation learning in offline RL.

Results. Results are reported in Table 3.4. We see that LOMPO achieves high-scores across most high-fidelity simulation environments, using raw observations. Moreover, our proposed model outperforms other model-based learning algorithms across the board and is the only model-based learning algorithm that achieves any success on several environments. Comparing to model-free algorithms, LOMPO still outperforms CQL and behaviour cloning across most environments, with the exception of learning on the expert dataset on the Door Open task. This is a well-known phenomenon when learning dynamics models from narrow expert data. On the other hand, given the thin data distribution and relatively simple dynamics of the task, direct behavior cloning from images performs well on both the medium-expert and expert dataset. We hypothesize that LMOPO performs well on the D'Claw and Adroit expert datasets, as these environments are relatively stationary, as compared to a robot

Environment	**Dataset**	**LOMPO (ours)**	**LMBRL**	**Offline SLAC**	**CQL**	**BC**
Walker Walk	medium-replay	**74.9**	44.7	-0.1	14.7	5.3
Walker Walk	medium-expert	**91.7**	76.3	32.8	45.1	15.6
Walker Walk	expert	**75.8**	24.5	11.3	40.3	11.8
D'Claw Screw	medium-replay	**71.8**	**72.4**	65.9	26.3	11.7
D'Claw Screw	medium-expert	**100.4**	**96.2**	76.3	30.3	27.6
D'Claw Screw	expert	**99.2**	90.8	63.4	24.2	25.2
Adroit Pen	medium-replay	**82.8**	5.2	5.4	25.8	46.7
Adroit Pen	medium-expert	**94.6**	0.0	-1.7	43.5	41.8
Adroit Pen	expert	**96.1**	0.2	-0.4	51.4	45.4
Door Open	medium-expert	**95.7**	0.0	0.0	0.0	72.2
Door Open	expert	0.0	0.0	0.0	0.0	**97.4**

Table 3.4: Results for the DeepMind Control Walker task, the Robel D'Claw Screw task, Adroit Pen task and the Sawyer Door Open task. The scores are undiscounted average returns normalized to roughly lie between 0 and 100, where a score of 0 corresponds to a random policy, and 100 corresponds to an expert. LOMPO consistently outperforms LMBRL, offline SLAC, CQL, and behavioral cloning in almost all settings.

arm manipulation task, and even actions from a stochastic expert cover a wide range of the environment dynamics.

3.8.2 Real Robot Experiments

To answer question (4), we deploy LOMPO on a real Franka Emika Panda robot arm.
Task. The environment consists of a Panda arm mounted in front of an Ikea desk cluttered with random distractor objects. The robot arm is initialized randomly above the desk and the drawer is initialized randomly in a open position. The goal of the robot is to navigate to the handle, hook it, and close the drawer. Observations are raw RGB images from a single overhead camera. The complete setup is shown in the rightmost picture in Figure 3.6.
Dataset. We use a pre-existing dataset of 1000 trajectories that was collected using a semi-supervised batch exploration algorithm [137]. A small balanced dataset of 200 images (0.2% of the full dataset) is manually labeled with whether the drawer is open or closed. Following the set up by [137], we use this dataset to train a classifier to predict whether the drawer is open or closed. We use the classifier probability as a reward for RL, which leads to a sparse, noisy, and unstable reward signal, which is reflective of one challenge of real-world RL. Since the dataset was collected and labeled in the context of a different paper [137], this experiment evaluates the ability to reuse existing offline datasets, which further exemplifies real-world problems.
Comparisons. We compare LOMPO, LMBRL, Offline SLAC, and visual foresight [138] using an SV2P model [139] and a CEM planner. As CQL did not achieve competitive performance on the simulated environments, we did not deploy it on the real robot. Moreover the offline dataset has high variance and consists of mostly non-task centric exploration, which is not suitable for imitation; hence we also did not evaluate behavioral cloning.

Results. We carry out 25 evaluation rollouts on the real robot and summarize results in Table 3.5. Overall, 24/25 of the LOMPO agent rollouts successfully navigate to the drawer handle, hook it, and push the drawer in; however the agent fully closes the drawer in only 19 of the rollouts for a final success rate of **76%**. We hypothesize that the agent does not always close the door as the classifier reward incorrectly predicts the drawer as closed when the drawer is slightly open. In contrast, the LMBRL, Offline SLAC, and visual foresight agents are unable to successfully navigate to the correct handle location, hence achieving a success rate of **0%**. These experiments suggest that LOMPO's uncertainty estimation and pessimism are critical for good offline RL performance. Finally, we note that [137] evaluate visual foresight in the same environment but on an easier version of this task, where the robot arm is initialized near the drawer handle. In this shorter-horizon problem, visual foresight achieves a success rate of **65%** (Figure 8 of [137]), which is still lower than LOMPO's success rate in the more difficult setting. Hence, this suggests that LOMPO is better at solving problems with longer time horizons by incorporating pessimism and a learned value function.

Table 3.5: Results for the Franka desk drawer-closing task.

Method	Success
LOMPO (ours)	**76.0%**
LMBRL	0.0%
Offline SLAC	0.0%
Visual Foresight	0.0%

3.9 Chapter Summary

We introduced MOReL, a new model-based framework for offline RL. MOReL incorporates both *generalization* and *pessimism* (or conservatism). This enables MOReL to perform policy improvement in known states that may not directly occur in the static offline dataset, but can nevertheless be predicted using the dataset by leveraging the power of generalization. At the same time, due to the use of pessimism, MOReL ensures that the agent does not drift to unknown states where the agent cannot predict accurately using the static dataset. We also presented LOMPO, an extension of MOReL to image spaces through the use of varaitional dynamics models that support a compact latent state representation.

Theoretically, we obtain bounds on the suboptimality of MOReL which improve over those in prior work. We further showed that this suboptimality bound cannot be improved upon by *any* offline RL algorithm in the worst case. Experimentally, we evaluated MOReL in the standard continuous control benchmarks in OpenAI gym and showed that it achieves state of the art results. The modular structure of MOReL comprising of model learning, uncertainty estimation, and model-based planning allows the use of a variety of approaches such as multi-step prediction for model learning, abstention for uncertainty estimation, or model-predictive control for action selection. In future work, we hope to explore these directions.

Chapter 4

A GAME THEORETIC FRAMEWORK FOR MODEL-BASED RL

4.1 Introduction

In the previous chapters, we introduced broader notions of generalization and associated abstractions applicable for simulation to reality transfer and offline RL. In this chapter, we study the traditional paradigm of online or interactive RL, where the agent can repeatedly interact with the environment and use this feedback for improving the policy. In particular, we study RL through the lens of model-based RL (MBRL) methods. Surprisingly, we find that broader notions of generalization and abstractions to deal with non-stationarity are required even in this classical paradigm. This is due to the inherent non-stationary nature of RL, where the induced distributions of the policy change as the agent is learning. Despite the recent surge of interest in MBRL, a clear algorithmic framework to understand MBRL and unify insights from recent works has been lacking. To bridge this gap, and to facilitate the design of stable and efficient algorithms, in this chapter, we develop a new framework that casts MBRL as a two-player game.

Classical frameworks for MBRL, adaptive control [140], and dynamic programming [141], are often confined to simple linear models or tabular representations. They also rely on building global models through ideas like persistent excitation [142] or tabular generative models [143]. Such settings and assumptions are often limiting for modern applications. To obtain a globally accurate model, we need the ability to collect data from all parts of the state space [144], which is often impossible. Furthermore, learning globally accurate models may be unnecessary, unsafe, and inefficient. For example, to make an autonomous car drive on the road, we should not require accurate models in situations where it tumbles and crashes in different ways. This motivates a class of *incremental* methods for MBRL that interleave policy and model learning to gradually construct and refine models in the task-relevant parts of the state space. This is in sharp contrast to a two-stage approach of first building a model of the world, and subsequently planning in it.

A unifying framework for incremental MBRL can connect insights from different approaches and help simplify the algorithm design process from the lens of abstraction. As an example, *distribution* or *domain shift* is known to be a major challenge for incremental MBRL. When improving the policy using the learned model, the policy will attempt to shift the distribution over visited states. The learned model may be inaccurate for this modified distribution, resulting in a greatly biased policy update. A variety of approaches have been developed to mitigate this issue. One class of approaches [145, 146, 55], inspired by trust region methods, make conservative changes to the policy to constrain the distribution between successive iterates. In sharp contrast, an alternate set of approaches do not constrain the policy updates in any way, but instead rely on data aggregation to mitigate distribution shift [50, 147, 148]. Our game-theoretic framework for MBRL reveals that these two seemingly disparate approaches are essentially dual approaches to solve the same game.

4.1.1 Highlights and contributions of this chapter

1. We develop a novel framework that casts MBRL as a game between: (a) a *policy player*, which maximizes rewards in the learned model; and (b) a *model player*, which minimizes prediction error of data collected by policy player. Theoretically, we establish that at equilibrium, the policy is near-optimal for the environment.
2. Developing learning algorithms for general continuous games is well known to be challenging. To develop stable and convergent algorithms, we setup a *Stackelberg game* [149] between the two players, which can be solved efficiently through (approximate) bi-level optimization.
3. Stackelberg games are asymmetric games where players make decisions in a pre-specified order. The leader plays first and subsequently the follower. Due to the asymmetric nature, the MBRL game can take two forms depending on choice of leader player. This gives rise to two natural families of algorithms that have complementary strengths. Together, they unify and generalize many prior MBRL algorithms.
4. Experimentally, we show that our algorithms outperform prior model-based and model-free algorithms in sample efficiency; match the asymptotic performance of model-free policy gradient algorithms; and scale gracefully to high-dimensional tasks like dexterous manipulation.

4.2 Background and Notations

We treat the environment as an infinite horizon MDP characterized by: $\boldsymbol{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{R}, P, \gamma, \rho\}$. Per usual notation, $\mathcal{S} \subseteq \mathbb{R}^n$ and $\mathcal{A} \subseteq \mathbb{R}^m$ represent the continuous state and action spaces. The transition dynamics is described by $s' \sim P(\cdot|s,a)$. $\mathcal{R} : \mathcal{S} \to [0, R_{\max}]$, $\gamma \in [0,1)$, and ρ represents the reward, discount, and initial state distribution respectively. Policy is a mapping from states to a probability distribution over actions, i.e. $\pi : \mathcal{S} \to P(\mathcal{A})$, and in practice we typically consider parameterized policies. The goal is to optimize the objective:

$$\max_{\pi} \; J(\pi, \boldsymbol{M}) := \mathbb{E}_{\boldsymbol{M},\pi}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t)\right] \tag{4.1}$$

Model-free methods solve this optimization by directly estimating a gradient using collected samples or through value functions. Model-based methods, in contrast, construct an explicit world model to aid policy optimization.

4.2.1 Model-Based Reinforcement Learning

We represent the world model with another tuple: $\widehat{\boldsymbol{M}} = \{\mathcal{S}, \mathcal{A}, \mathcal{R}, \widehat{P}, \gamma, \rho\}$. The model has the same state-action space, reward function, discount, and initial state distribution. We parameterize the transition dynamics of the model $\widehat{P}$ (as a neural network) and learn the parameters so that it approximates the environment transition dynamics, i.e. $\widehat{P} \approx P$. For simplicity, we assume that the reward function and initial state distribution are known. This is a benign assumption for many applications in control, robotics, and operations research. If required, these quantities can also be learned from data, and are typically easier to learn than $\widehat{P}$. Enormous quantities of experience can be cheaply generated by simulating the model, without interacting with the world, and can be used for policy optimization. Thus, model-based methods tend to be sample efficient.

Idealized Global Model Setting To motivate challenges in MBRL, we first consider the idealized setting of an approximate *global* model. This corresponds to the case where $\widehat{\boldsymbol{M}}$ is sufficiently expressive and approximates $\boldsymbol{M}$ everywhere. Lemma 2 relates the performance of a policy in the model and environment.

Lemma 2. *(Simulation Lemma) Suppose $\widehat{\boldsymbol{M}}$ is such that $D_{TV}\left(P(\cdot|s,a), \widehat{P}(\cdot|s,a)\right) \leq \epsilon_{\boldsymbol{M}}$, $\forall(s,a)$. Then, for any policy π, we have*

$$\left| J(\pi, \boldsymbol{M}) - J(\pi, \widehat{\boldsymbol{M}}) \right| \leq O\left(\frac{\epsilon_{\boldsymbol{M}}}{(1-\gamma)^2}\right) \quad \forall \pi. \tag{4.2}$$

The proof is provided in the appendix. Since Lemma 2 provides a uniform bound applicable to all policies, we can expect good performance in the environment by optimizing the policy in the model, i.e. $\max_\pi J(\pi, \widehat{\boldsymbol{M}})$.

Beyond global models A global modeling approach as above is often impractical. To obtain a globally accurate model, we need the ability to collect data from all parts of the state space [78, 150], which can be difficult. More importantly, learning globally accurate models may be unnecessary, unsafe, and inefficient. For example, to make a robot walk, we should not require accurate models in situations where it falls and crashes in different ways. This motivates the need for *incremental* MBRL, where models are gradually constructed and refined in the task-relevant parts of the state space. To formalize this intuition, we consider the below notion of model quality.

Definition 4. *(Model approximation loss) Given $\widehat{\boldsymbol{M}}$ and distribution $\mu(s,a)$, the model approximation loss is*

$$\mathcal{L}(\widehat{\boldsymbol{M}}, \mu) = \mathbb{E}_{(s,a)\sim\mu}\left[D_{KL}\big(P(\cdot|s,a), \widehat{P}(\cdot|s,a)\big)\right]. \tag{4.3}$$

We use D_{KL} to refer to the KL divergence which can be optimized using samples from $\boldsymbol{M}$, and is closely related to D_{TV} through Pinsker's inequality. In the case of isotropic Gaussian distributions, as typically considered in continuous control applications, D_{KL} reduces to the familiar ℓ_2 loss. Importantly, the loss is intimately tied to the sampling distribution μ. In general, models that are accurate in some parts of the state space need not generalize/transfer to other parts. As a result, a more conservative policy learning procedure is required, in contrast to the global model case.

4.3 Model Based RL as a Two Player Game

In order to capture the interactions between model and policy learning, we formulate MBRL as the following two-player general sum game (ref. as MBRL game)

$$\overbrace{\max_{\pi} \; J(\pi, \widehat{\boldsymbol{M}})}^{\text{policy}-\text{player}} \quad , \quad \overbrace{\min_{\widehat{\boldsymbol{M}}} \; \mathcal{L}(\widehat{\boldsymbol{M}}, \mu^{\pi}_{\boldsymbol{M}})}^{\text{model}-\text{player}} \tag{4.4}$$

We use $\mu^{\pi}_{\boldsymbol{M}} = \frac{1}{T}\sum_{t=0}^{T} P(s_t = s, a_t = a)$ to denote the average state visitation distribution. The policy player maximizes performance in the learned model, while the model player minimizes prediction error under policy player's induced state distribution. This is a game since the objective of each player depends on the parameters of both players.

The above formulation separates MBRL into the constituent components of policy learning (planning) and generative model learning. At the same time, it exposes that the two components are closely intertwined and must be considered together in order to succeed in MBRL. We discuss algorithms for solving the game in Section 4.4, and first focus on the equilibrium properties of the MBRL game. Our results establish that at (approximate) Nash equilibrium of the MBRL game: (1) the model can accurately simulate and predict the performance of the policy; (2) the policy is near-optimal.

Theorem 2. *(Global perf. of equilibrium pair; informal) Suppose we have a pair of policy and model, $(\pi, \widehat{\boldsymbol{M}})$, such that simultaneously*

$$\mathcal{L}(\widehat{\boldsymbol{M}}, \mu^{\pi}_{\boldsymbol{M}}) \leq \epsilon_{\boldsymbol{M}} \quad \text{and} \quad J(\pi, \widehat{\boldsymbol{M}}) \geq J(\pi', \widehat{\boldsymbol{M}}) - \epsilon_{\pi} \; \forall \pi'.$$

For an optimal policy π^*, we have

$$\begin{aligned} & J(\pi^*, \boldsymbol{M}) - J(\pi, \boldsymbol{M}) \leq \\ & O\left(\epsilon_{\pi} + \frac{\sqrt{\epsilon_{\boldsymbol{M}}}}{(1-\gamma)^2} + \frac{1}{1-\gamma} D_{TV}\left(\mu^{\pi^*}_{\boldsymbol{M}}, \mu^{\pi^*}_{\widehat{\boldsymbol{M}}}\right)\right). \end{aligned} \tag{4.5}$$

Proof. A more formal version of the theorem and proof is provided in appendix C.1. □

Remarks: We now make some remarks about the Theorem 2 and its implications.

1. The first two terms are related to sub-optimality in policy optimization and model learning, and can be made small with more compute and data, assuming sufficient capacity.
2. There may be multiple Nash equilibrium for the MBRL game, and the third *domain adaptation* or *transfer learning* term in the bound captures the quality of an equilibrium. It captures the idea that model is trained under distribution of π, i.e. $\mu_{\boldsymbol{M}}^{\pi}$, but evaluated under the distribution of π^*, i.e. $\mu_{\boldsymbol{M}}^{\pi^*}$. If the model can accurately simulate π^*, we can expect to find it in the planning phase, since it would obtain high rewards. This domain adaptation term is a consequence of the exploration problem, and is unavoidable if we desire globally optimal policies. Indeed, even purely model-free algorithms suffer from an analogous divergence term [55, 87]. However, Theorem 2 also applies to locally optimal policies (see appendix C.1) for which we may expect better model transfer.
3. The domain adaptation term can be minimized by considering a wide initial state distribution [55, 12]. This ensures the learned model is more broadly accurate. However, in some applications, the initial state distribution may not be under our control. In such a case, we may draw upon advances in domain adaptation [151, 152] to learn state-action representations better suited for transfer across different policies.

4.4 Algorithms

So far, we have established how MBRL can be viewed as a game that couples policy and model learning. We now turn to developing algorithms for solving the game. Unlike common deep learning settings (e.g. supervised learning), there are no standard workhorses for continuous games. Direct extensions of optimization workhorses (e.g. SGD) are unstable for games due to non-stationarity [153, 154]. We first review some of these extensions before presenting our final algorithms.

4.4.1 Independent simultaneous learners

We first consider a class of algorithms where each player individually optimize their own objectives using gradient descent. Thus, each player treats the setting as stochastic optimization unaware of potential drifts in their objectives due to the two-player nature. These algorithms are sometimes called independent learners, simultaneous learners, or naive learners [153, 155].

Gradient Descent Ascent (GDA) In GDA, each player performs an improvement step holding the parameters of the other player fixed. The resulting updates are given below.

$$\pi_{k+1} = \pi_k + \alpha_k \nabla_{\pi} J(\pi_k, \widehat{\boldsymbol{M}}_k) \tag{4.6}$$
$$\widehat{\boldsymbol{M}}_{k+1} = \widehat{\boldsymbol{M}}_k - \beta_k \nabla_{\widehat{\boldsymbol{M}}} \mathcal{L}(\widehat{\boldsymbol{M}}_k, \mu_{\boldsymbol{M}}^{\pi_k}) \tag{4.7}$$

Both the players update their parameters simultaneously from iteration k to $k+1$. For simplicity, we consider standard gradient descent, which can be equivalently replaced with momentum, Adam, natural gradient etc. Variants of GDA have been used to solve min-max games arising in deep learning such as GANs. However, for certain problems, it can exhibit poor convergence and require very small learning rates [156] or domain-specific heuristics. Furthermore, it makes sub-optimal use of data, since it is desirable to take multiple policy improvement steps to fully reap the benefits of model learning.

Best Response (BR) The BR algorithm aims to mititage the above drawback, where each player computes the best response while fixing the parameters of other players. The best response can be approximated in practice using a large number of gradient steps.

$$\pi_{k+1} = \arg\max_{\pi} \; J(\pi, \widehat{\boldsymbol{M}}_k) \tag{4.8}$$

$$\widehat{\boldsymbol{M}}_{k+1} = \arg\min_{\widehat{\boldsymbol{M}}} \; \mathcal{L}(\widehat{\boldsymbol{M}}, \mu^{\pi_k}_{\boldsymbol{M}}) \tag{4.9}$$

Again, both players simultaneously update their parameters. It is known from a large body of work in online learning that aggressive changes can destabilize learning in non-stationary settings [157]. Large changes to the policy can dramatically alter the sampling distribution, which renders the model incompetent. Similarly, large changes in the model can bias policy learning. In Section 4.5 we experimentally study the performance of GDA and BR on a suite of control tasks and verify that they inefficient (slow) or unstable.

4.4.2 Stackelberg formulation and algorithms

To achieve stable and sample efficient learning, we require algorithms that take the game structure into account. While good workhorses are lacking for general games, Stackelberg games [149] are an exception. They are asymmetric games where we impose a specific playing order and are a generalization of min-max games. We cast the MBRL game in the Stackelberg form, and derive gradient based algorithms to solve the resulting game.

First, we briefly review continuous Stackelberg games. Consider a two player game with players A and B. Let $\boldsymbol{\theta}_A, \boldsymbol{\theta}_B$ be their parameters, and $\mathcal{L}_A(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B)$, $\mathcal{L}_B(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B)$ be their losses. Each player would like their losses minimized. With player A as the leader, the Stackelberg game corresponds to the following nested optimization:

$$\begin{aligned} &\min_{\boldsymbol{\theta}_A} \; \mathcal{L}_A\big(\boldsymbol{\theta}_A, \boldsymbol{\theta}^*_B(\boldsymbol{\theta}_A)\big) \\ &\text{subject to} \;\; \boldsymbol{\theta}^*_B(\boldsymbol{\theta}_A) = \arg\min_{\tilde{\boldsymbol{\theta}}} \; \mathcal{L}_B(\boldsymbol{\theta}_A, \tilde{\boldsymbol{\theta}}) \end{aligned} \tag{4.10}$$

Since the follower chooses the best response, the follower's parameters are implicitly a function of the leader's parameters. The leader is aware of this, and can utilize this information when updating its parameters. The Stackelberg formulation has a number of appealing properties.

- **Algorithm design based on optimization:** From the leader's viewpoint, the Stackelberg formulation transforms a game with complex interactions into a more familiar albeit complex bi-level optimization, for which we have gradient based workhorses [158].
- **Notion of stability and progress:** In general games, there exists no single function that can be used to check if an iterative algorithm makes progress towards the equilibrium. This makes algorithm design and diagnosis difficult. By reducing the game to an optimization, the leader's loss $\mathcal{L}_A(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B)$ can be used to track progress.

For simplicity of exposition, we assume that the best-response is unique for the follower. We later remark on the possibility of multiple minimizers. To solve the nested optimization, it suffices to focus on $\boldsymbol{\theta}_A$ since the follower parameters $\boldsymbol{\theta}_B^*(\boldsymbol{\theta}_A)$ are implicitly a function of $\boldsymbol{\theta}_A$. We can iteratively optimize $\boldsymbol{\theta}_A$ as: $\boldsymbol{\theta}_A \leftarrow \boldsymbol{\theta}_A - \alpha_A \left(\mathrm{d}\mathcal{L}_A(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B^*(\boldsymbol{\theta}_A))/\mathrm{d}\boldsymbol{\theta}_A\right)$, where the gradient is described in Eq. 4.11. The key to solving a Stackelberg game is to make the follower learn very quickly to approximate the best response, while the leader learns slowly.

$$\begin{aligned}\frac{\mathrm{d}\mathcal{L}_A\left(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B^*(\boldsymbol{\theta}_A)\right)}{\mathrm{d}\boldsymbol{\theta}_A} &= \frac{\mathrm{d}\boldsymbol{\theta}_B^*}{\mathrm{d}\boldsymbol{\theta}_A}\, \frac{\partial \mathcal{L}_A(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B)}{\partial \boldsymbol{\theta}_B}\bigg|_{\boldsymbol{\theta}_B = \boldsymbol{\theta}_B^*} \\ &\quad + \frac{\partial \mathcal{L}_A(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B)}{\partial \boldsymbol{\theta}_A}\bigg|_{\boldsymbol{\theta}_B = \boldsymbol{\theta}_B^*}\end{aligned} \tag{4.11}$$

The implicit Jacobian term $(\mathrm{d}\boldsymbol{\theta}_B^*/\mathrm{d}\boldsymbol{\theta}_A)$ can be obtained using the implicit function theorem [159, 17]. Thus, in principle, we can compute the gradient with respect to the leader parameters and solve the nested optimization (to at least a local minimizer). To develop a practical algorithm based on these ideas, we use a few relaxations and approximations. First, we approximate the best response with multiple steps of an iterative optimization algorithm. Secondly, we drop the implicit Jacobian term and use a "first-order" approximation of the gradient. Such an approximation has proven effective in applications like meta-learning [160], GANs [161, 162], and multiple timescale actor-critic methods [163]. Finally, since the Stackelberg game is asymmetric, we can cast the MBRL game in two forms based on which player we choose as the leader.

Policy As Leader (PAL): Choosing the policy player as leader results in the following optimization:

$$\max_{\pi} \left\{ J(\pi, \widehat{\boldsymbol{M}}^{\pi}) \;\; s.t. \;\; \widehat{\boldsymbol{M}}^{\pi} \in \arg\min_{\widehat{\boldsymbol{M}}} \; \ell(\widehat{\boldsymbol{M}}, \mu_{\boldsymbol{M}}^{\pi}) \right\}.$$

We solve this nested optimization using the first order gradient approximation, resulting in updates:

$$\widehat{\boldsymbol{M}}_{k+1} \approx \arg\min_{\widehat{\boldsymbol{M}}} \mathcal{L}(\widehat{\boldsymbol{M}}, \mu_{\boldsymbol{M}}^{\pi_k}) \tag{4.12}$$

$$\pi_{k+1} = \pi_k + \alpha_k \nabla_\pi J(\pi, \widehat{\boldsymbol{M}}_{k+1}) \tag{4.13}$$

We first aggressively improve the model to minimize the loss under current visitation distribution. Subsequently we take a conservative policy. The algorithmic template is described further in Algorithm 3. Note that the PAL updates are different from GDA even if a single gradient step is used to approximate the $\arg\min$. In PAL, the model is first updated using the current visitation distribution from $\widehat{\boldsymbol{M}}_k$ to $\widehat{\boldsymbol{M}}_{k+1}$. The policy subsequently uses $\widehat{\boldsymbol{M}}_{k+1}$ for improvement. In contrast, GDA uses $\widehat{\boldsymbol{M}}_k$ for improving the policy. Finally, suppose we find an approximate solution to the PAL optimization such that $J(\pi, \widehat{\boldsymbol{M}}^{\pi}) \geq \sup_{\tilde{\pi}} J(\tilde{\pi}, \widehat{\boldsymbol{M}}^{\tilde{\pi}}) - \epsilon_{\pi}$. Since the model is optimal for the policy by constriction, we inherit the guarantees of Theorem 2.

Algorithm 3 Policy as Leader (PAL) meta-algorithm

```
1: Initialize: policy $\pi_0$, model $\widehat{\boldsymbol{M}}_0$, data buffer $\mathcal{D} = \{\}$
2: for $k = 0, 1, 2, \ldots$ forever do
3:     Collect data $\mathcal{D}_k$ by executing $\pi_k$ in the environment
4:     Build local (policy-specific) dynamics model: $\widehat{\boldsymbol{M}}_{k+1} = \arg\min \; \mathcal{L}(\widehat{\boldsymbol{M}}, \mathcal{D}_k)$
5:     Improve policy: $\pi_{k+1} = \pi_k + \alpha \nabla_{\pi} J(\pi_k, \widehat{\boldsymbol{M}}_{k+1})$ with a conservative algorithm like NPG
       or TRPO.
6: end for
```

Model as Leader (MAL): Conversely, choosing model as the leader results in the optimization

$$\min_{\widehat{\boldsymbol{M}}} \left\{ \mathcal{L}(\widehat{\boldsymbol{M}}, \mu_{\boldsymbol{M}}^{\pi_{\widehat{\boldsymbol{M}}}}) \quad s.t. \quad \pi_{\widehat{\boldsymbol{M}}} \in \arg\max_{\pi} J(\pi, \widehat{\boldsymbol{M}}) \right\}. \tag{4.14}$$

Similar to PAL, using first order approximation to the bi-level gradient results in:

$$\pi_{k+1} \approx \arg\max_{\pi} J(\pi, \widehat{\boldsymbol{M}}_k) \tag{4.15}$$

$$\widehat{\boldsymbol{M}}_{k+1} = \widehat{\boldsymbol{M}}_k - \beta_k \nabla_{\widehat{\boldsymbol{M}}} \mathcal{L}(\widehat{\boldsymbol{M}}, \mu_{\boldsymbol{M}}^{\pi_{k+1}}) \tag{4.16}$$

We first optimize a policy for the current model. Subsequently, we conservatively improve the model using the data collected with the optimized policy. In practice, instead of a single conservative model improvement step, we aggregate all the historical data and perform a few epochs of training. This has an effect similar to conservative model improvement in a follow the regularized leader interpretation [164, 50, 165]. The algorithmic template is described in Algorithm 4. Similar to the PAL case, we again inherit the guarantees from Theorem 2.

On distributionally robust models and policies Finally, we illustrate how the Stackelberg framework is consistent with commonly used robustification heuristics. We now consider

Algorithm 4 Model as Leader (MAL) meta-algorithm

```
1: Initialize: policy $\pi_0$, model $\widehat{\boldsymbol{M}}_0$, data buffer $\mathcal{D} = \{\}$
2: for $k = 0, 1, 2, \ldots$ forever do
3:     Optimize $\pi_{k+1} = \arg\max_\pi J(\pi, \widehat{\boldsymbol{M}}_k)$ using any algorithm (RL, MPC, planning etc.)
4:     Collect environment data $\mathcal{D}_{k+1}$ using $\pi_{k+1}$
5:     Improve model $\widehat{\boldsymbol{M}}_{k+1} = \widehat{\boldsymbol{M}}_k - \beta \nabla_{\widehat{\boldsymbol{M}}} \mathcal{L}(\widehat{\boldsymbol{M}}, \mathcal{D}_{k+1})$ using any conservative algorithm like
       mirror descent, data aggregation etc.
6: end for
```

the case where there could be multiple best responses to the leader eq. 4.10. For instance, in PAL, there could be multiple models that achieve low error for the policy. Similarly, in MAL, there could be multiple policies that achieve high rewards for the specified model. In such cases, the standard notion of Stackelberg equilibrium is to optimize under the worst case realization [154], which results in:

$$\begin{aligned} &\min_{\boldsymbol{\theta}_A} \max_{\boldsymbol{\theta}_B \in R(\boldsymbol{\theta}_A)} \quad \mathcal{L}_A(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B), \quad \text{where} \\ &R(\boldsymbol{\theta}_A) \overset{\text{def}}{=} \left\{ \tilde{\boldsymbol{\theta}} \mid \mathcal{L}_B(\boldsymbol{\theta}_A, \tilde{\boldsymbol{\theta}}) \leq \mathcal{L}_B(\boldsymbol{\theta}_A, \boldsymbol{\theta}_B) \ \forall \boldsymbol{\theta}_B \right\}. \end{aligned} \tag{4.17}$$

In PAL, model ensemble approaches correspond to approximating the best response set with a finite collection (ensemble) of models. Algorithms inspired by robust or risk-averse control [39, 27, 13] explicitly improve against the adversarial choice in the ensemble, consistent with the Stackelberg setting. Similarly, in the MAL formulation, entropy regularization [166, 167] and disagreement based reward bonuses [150, 168] lead to adversarial best response by encouraging the policy to visit parts of the state space where the model is likely to be inaccurate. Our Stackelberg formulation provides a principled foundation for these important components, which have thus far been viewed as heuristics.

4.5 Experiments

In our experiemental evaluation, we aim to primarily answer the following questions:

1. Do independent learning algorithms (GDA and BR) learn slowly or suffer from instabilities?

2. Do the Stackelberg-style algorithms of PAL and MAL enable stable, monotonic, and sample efficient learning?

3. Do MAL and PAL exhibit different learning characteristics and strengths? Can we characterize the situations where one is more preferable than the other?

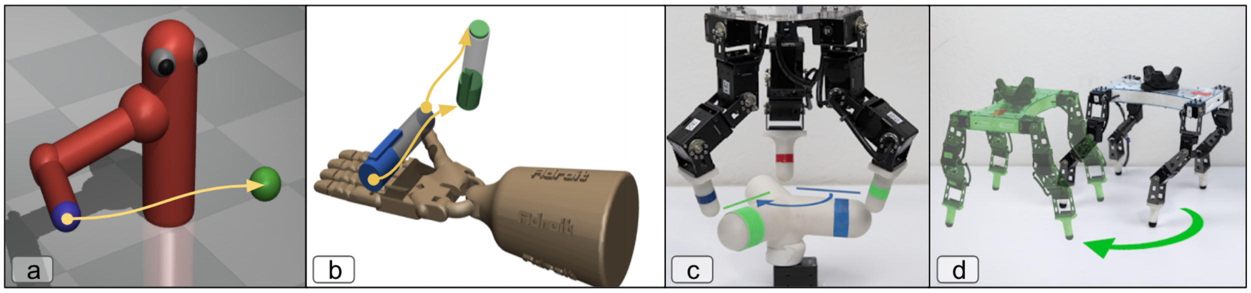


Figure 4.1: (a) Reacher task with a 7DOF arm. (b) In-hand manipulation task with a 24DOF dexterous hand. (c) DClaw-Turn task with a 3 fingered "claw". (d) DKitty-Orient task with a quadrupedal robot. In all the tasks, the desired goal configurations are randomized every episode, which forces the RL agent to learn generalizable policies. We measure and use success rate for our experimental evaluations.

Task Suite We study the behavior of algorithms on a suite of continuous control tasks consisting of: `DClaw-Turn`, `DKitty-Orient`, `7DOF-Reacher`, and `InHand-Pen`. The tasks are illustrated in Figure 4.1 and further details are provided in Appendix C.2.1. The DClaw and DKitty tasks use physically accurate models of robots [169, 65]. The Reacher task is a representative whole arm manipulation task, while the in-hand dexterous manipulation task [170] serves as a representative high-dimensional control task. In addition, we also present results with our algorithms in the OpenAI gym tasks in Appendix C.2.2.

Algorithm Details For all the algorithms of interest (GDA, BR, PAL, MAL), we represent the policy as well as the dynamics model with fully connected neural networks. We instantiate all of these algorithm families with model-based natural policy gradient. Details about the implementation are provided in Appendix C.2. We use ensembles of dynamics models and entropy regularization to encourage robustness.

Comparison of learning algorithms We first study the performance of Stackelberg-style algorithms (PAL, MAL) and compare against the performance of independent algorithms (GDA and BR). Our results, summarized in Figure 4.2, suggest that PAL and MAL can learn all the tasks efficiently. We observe near monotonic improvement, suggesting that the Stackelberg formulation enables stable learning. We also observe that PAL learns faster than MAL for the tasks we study. While GDA eventually achieves near-100% success rate, it leads to considerably slower learning. As outlined in Section 4.4, this is likely due to conservative nature of updates for both the policy and the model. Furthermore, the performance fluctuates rapidly during course of learning, since it does not correspond to stable optimization of any objective. Finally, we observe that BR is unable to make consistent progress. As suggested earlier in Section 4.4, BR makes rapid changes to both model and policy which exacerbates the challenge of distribution mismatch.

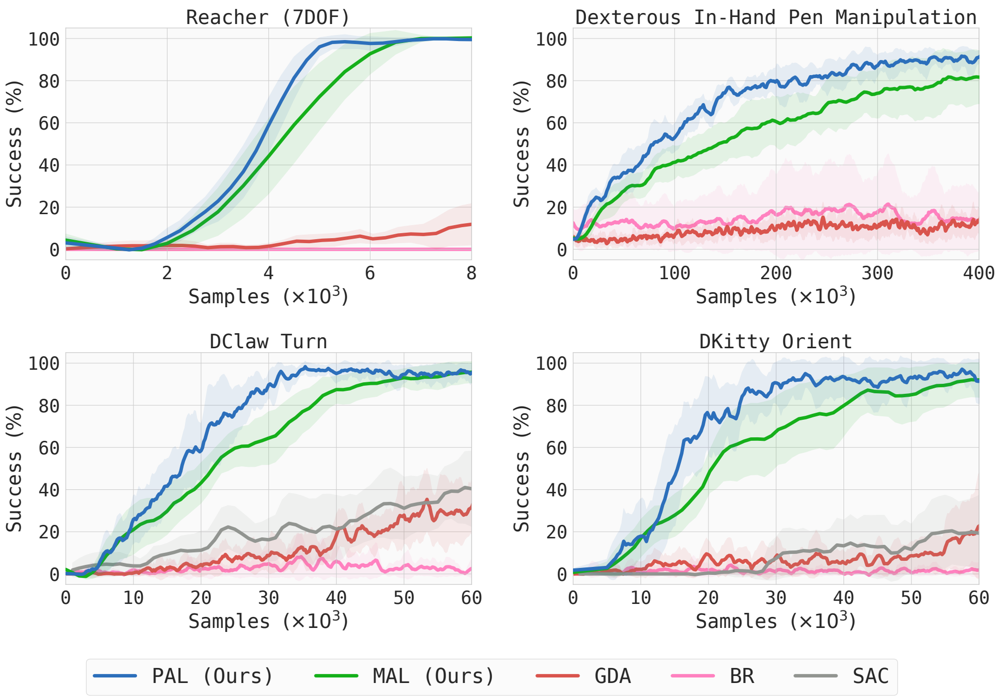


Figure 4.2: Comparison of the learning algorithms. We report results based on 5 random seeds, with solid lines representing the average performance, and shaded regions indicate standard deviation across seeds. PAL and MAL exhibit stable and sample efficient learning. GDA learns very slowly due to sub-optimal use of data. BR does not lead to stable learning due to aggressive changes to both policy and model. For the ROBEL tasks, as a point of comparison, we also include results of SAC a state of the art model-free algorithm.

As a point of comparison, we also plot results of SAC [166], a leading model-free algorithm for the ROBEL tasks (results taken from Ahn et al. [65]). Although SAC is able to solve these tasks, its sample efficiency is comparable to GDA, and substantially slower than PAL and MAL. To compare against other model-based algorithms, we turn to published results from prior work on OpenAI gym tasks. In Figure 4.3, we show that PAL and MAL significantly outperforms prior algorithms. In particular, PAL and MAL are 10 times as efficient as other model-based and model-free methods. PAL is also twice as efficient as MBPO [88], a state of the art hybrid model-based and model-free algorithm. Further details about this comparison are provided in Appendix C.2.2.

Overall our results indicate that PAL and MAL: (a) are substantially more sample efficient than prior model-based and model-free algorithms; (b) achieve the asymptotic performance of their model-free counterparts; (c) can scale to high-dimensional tasks with complex dynamics like dexterous manipulation; (d) can scale to tasks requiring extended rollout horizons (e.g. the OpenAI gym tasks).

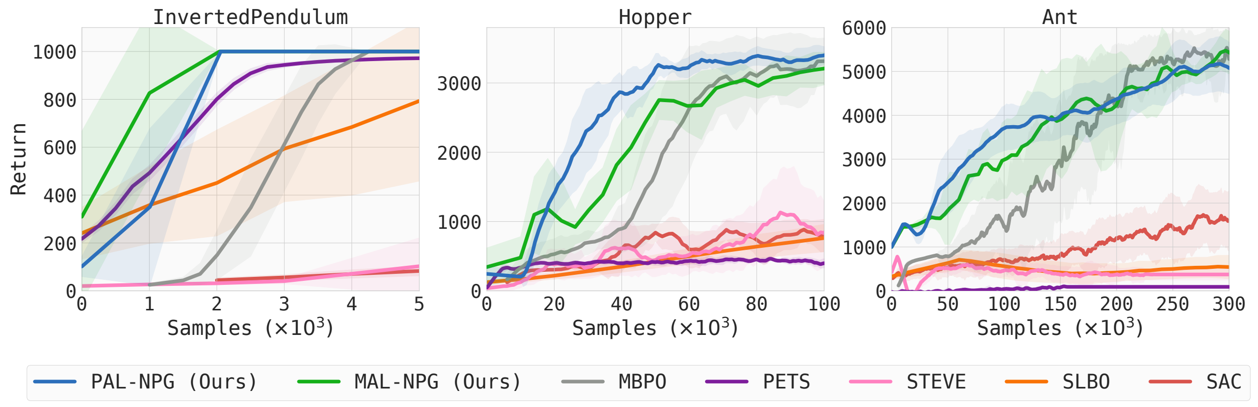


Figure 4.3: Comparison of results on the OpenAI gym benchmark tasks. Results for the baselines are reproduced from [88]. Solid lines are the average performance curves over 5 random seeds, while shaded region represents the standard deviation over these 5 runs. We observe that PAL and MAL show near-monotonic improvement, and substantially outperform the baselines.

Choosing between PAL and MAL Finally, we turn to studying relative strengths of PAL and MAL. For this, we consider two variations of the 7DOF reacher task (from Figure 4.1) corresponding to environment perturbations at an intermediate point of training. In the first

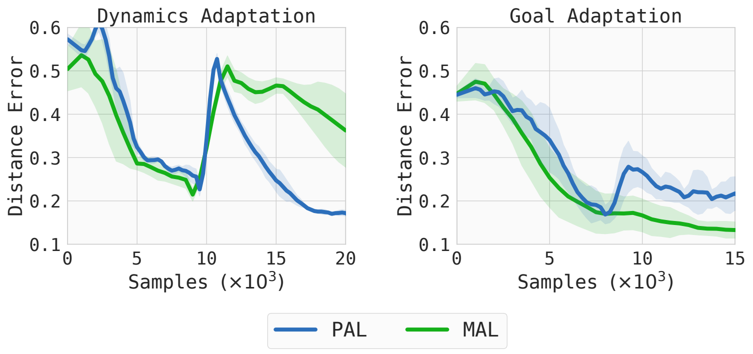


Figure 4.4: PAL vs MAL in non-stationary learning environments. Y axis is the distance between end effector and goal, averaged over the trajectory (lower is better). The left plot corresponds to the case where the dynamics of $\boldsymbol{M}$ is changed after 10^4 samples, while the right plot corresponds to the case where we change the goal distribution after 8×10^3 samples. We observe that PAL recovers quickly from dynamics perturbations, while MAL recovers quickly from goal perturbations.

case, we perturb the dynamics by changing the length of the forearm. In the second case, halfway through the training, we change the goal distribution to a different region of 3D space. Training curves are presented in Figure 4.4. Note that there is a performance drop at the time of introducing the perturbation.

For the first case of dynamics perturbation, we observe that PAL recovers faster. Since PAL learns the model aggressively using recent data, it can forget old inconsistent data and improve the policy using an accurate model. In contrast, MAL adapts the model conservatively, taking longer to forget old inconsistent data, ultimately biasing and slowing the policy learning. In the second experiment, the dynamics is stationary but the goal distribution changes midway. Note that the policy does not generalize zero-shot to the new goal distribution, and requires additional learning or fine-tuning. Since MAL learns a more broadly accurate model, it quickly adapts to the new goal distribution. In contrast, PAL conservatively changes the policy and takes longer to adapt to the new goal distribution.

Thus, in summary, we find that PAL is better suited for situations where the dynamics of the world can drift over time. In contrast, MAL is better suited for situations where the task or goal distribution can change over time, and related settings like multi-task learning.

4.6 Related Work

MBRL and the closely related fields of adaptive control and system identification have a long and rich history (see Åström and Wittenmark [140], Ljung [171] for overview). Early works in MBRL primarily focused on tabular reinforcement learning in a known *generative model* setting [143, 172]. However, this setting assumes access to a highly exploratory policy to collect data, which is often not available in practice. Subsequent works like E3 [111] and R-MAX [112] attempt to lift this limitation, but rely heavily on tabular representations which are inadequate for modern applications like robotics. Coupled with advances in deep learning, there has been a surge of interest in incremental MBRL algorithms with rich function approximation. They generally fall into two sets of approaches, as we outline below.

The first set of approaches are largely inspired by trust region methods, and are similar to the PAL family from our work. A highly accurate "local" model is constructed around the visitation distribution of the current policy, which is subsequently used to conservatively improve the policy. The trust region is intended to ensure that the model is accurate for all policies within it, thereby enabling monotonic performance improvement. GPS [145, 173], DPI [146], and related approaches [174] learn a time varying linear model and perform a KL-constrained policy improvement step. Such a model representation is convenient for an iLQG based policy update [84], but might be restrictive for complex dynamics beyond trajectory-centric RL. To remove these limitations, recent works have started to consider neural networks to represent both policy and the dynamics model. However, somewhat surprisingly, a clean version from the PAL family has not been studied with neural network models. The motivations presented by Xu et al. [175] and Kurutach et al. [89] resemble PAL, however their practical implementations do not strongly enforce the conservative nature of the policy update.

An alternate set of MBRL approaches take a view similar to MAL. Models are updated conservatively through data aggregation, while policies are aggressively optimized. Ross and Bagnell [50] explicitly studied the role of data aggregation in MBRL. They presented an agnostic online learning view of MBRL and showed that data aggregation can lead to a no-regret algorithm for learning the model, even with aggressive policy optimization. Subsequent works have used data augmentation and proposed additional components to enhance efficiency and stability, such as the use of model predictive control [120, 127, 148], uncertainty quantification through Bayesian models [176], and ensembles of dynamics models [13, 147, 148]. We refer readers to Wang et al. [177] for overview of recent MBRL advances.

While specific instances of PAL and MAL have been studied in the past, an overarching framework around them has been lacking. Our descriptions of the PAL and MAL families generalize and unify core insights from prior work and simplify them from the lens of abstraction. Furthermore, the game theoretic formulation enables us to form a connection between the PAL and MAL frameworks. We also note that the PAL and MAL families have similarities to multiple timescale algorithms [163, 178, 179] studied for actor-critic temporal difference learning. These ideas have also been extended to study min-max games like GANs [161]. However, they have not been extended to study model-based RL.

We presented a model-based setting where the model is used to directly improve the policy through rollout based optimization. However, models can be utilized in other ways too. Dyna [180] and MBPO [88] use a learned model to provide additional learning targets for an actor-critic algorithm through short-horizon synthetic trajectories. MBVE [181], STEVE [182], and doubly-robust methods [183, 184, 185] use model-based rollouts to obtain more favorable bias-variance trade-offs for off-policy evaluation. Some of these works have noted that long horizon rollouts can exacerbate model bias. However, in our experiments, we were able to successfully perform rollouts of hundreds of steps. This is likely due to our practical implementation closely following the game theoretic algorithms designed explicitly to mitigate distribution shift and enable effective simulation. It is straightforward to extend PAL and MAL to a hybrid model-based and model-free algorithm, which is likely to provide further performance gains. Similarly, approaches that bootstrap from the model's predictions can improve multi-step simulation [117, 118]. We leave exploration of these directions for future work.

4.7 Summary and Conclusion

In this chapter, we presented a new framework for MBRL that casts it as a game between a policy player and a model player. We established that at equilibrium: (1) the model accurately simulates the policy and predicts its performance; (2) the policy is near-optimal. We derived sub-optimality bounds and made a connection to domain adaptation to characterize the equilibrium quality.

In order to solve the MBRL game, we constructed the Stackelberg version of the game. This has the advantage of: (1) effective gradient based workhorses to solve the Stackelberg optimization problem; (2) an effective objective function to track learning progress towards equilibrium. General continuous games possess neither of these characteristics. The Stackelberg game can take two forms based on which player we choose as the leader, resulting in two natural algorithm families, which we named PAL and MAL. Together they encompass, generalize, and unify a large collection of prior MBRL works. This greatly simplifies MBRL and particularly algorithm design from the lens of abstraction.

We developed practical versions of PAL and MAL using model-based natural policy gradient. We demonstrated stable and sample efficient learning on a suite of control tasks, including state of the art results on OpenAI gym benchmarks. These results suggest that our practical variants of PAL and MAL:

- are more sample efficient compared to prior model-based and model-free algorithms,
- can achieve the same asymptotic performance as model-free counterparts,
- can scale to high-dimensional tasks with complex dynamics like dexterous manipulation,
- can scale to tasks requiring long horizon rollouts (e.g. OpenAI gym tasks which have a 1000 timestep horizon).

More broadly, our work adds to a growing body of recent work which suggests that MBRL can be stable, sample efficient, and more generalizable or adaptable to new tasks and non-stationary settings. For future work, we hope to study alternate ways to solve the Stackelberg optimization; such as using the full implicit gradient term and unrolled optimization. Finally, although we presented our game theoretic framework in the context of MBRL, it is more broadly applicable for any surrogate based optimization including actor-critic methods. It would make for interesting future work to study broader extensions and implications.

Chapter 5
META LEARNING WITH IMPLICIT GRADIENTS

5.1 Introduction

A core aspect of intelligence is the ability to quickly learn new tasks by drawing upon prior experience from related tasks. Recent work has studied how meta-learning algorithms [186, 187, 188] can acquire such a capability by learning to efficiently learn a range of tasks, thereby enabling learning of a new task with as little as a single example [189, 190, 191]. Meta-learning algorithms can be framed in terms of recurrent [192, 189, 193] or attention-based [190, 194] models that are trained via a meta-learning objective, to essentially encapsulate the learned learning procedure in the parameters of a neural network. An alternative formulation is to frame meta-learning as a bi-level optimization procedure [195, 191], where the "inner" optimization represents adaptation to a given task, and the "outer" objective is the meta-training objective. Such a formulation can be used to learn the initial parameters of a model such that optimizing from this initialization leads to fast adaptation and generalization. In this work, we focus on this class of optimization-based methods, and in particular the model-agnostic meta-learning (MAML) formulation [191]. MAML has been shown to be as expressive as black-box approaches [196], is applicable to a broad range of settings [197, 198, 199, 18], and recovers a convergent and consistent optimization procedure [200].

Despite its appealing properties, meta-learning an initialization requires backpropagation through the inner optimization process. As a result, the meta-learning process requires higher-order derivatives, imposes a non-trivial computational and memory burden, and can suffer from vanishing gradients. These limitations make it harder to scale optimization-based meta learning methods to tasks involving medium or large datasets, or those that require many inner-loop optimization steps. Our goal is to develop an algorithm that addresses these limitations.

The main contribution of our work is the development of the implicit MAML (iMAML) algorithm, an approach for optimization-based meta-learning with deep neural networks that removes the need for differentiating through the optimization path. Our algorithm aims to learn a set of parameters such that an optimization algorithm that is initialized at and regularized to this parameter vector leads to good generalization for a variety of learning tasks. By leveraging the implicit differentiation approach, we derive an analytical expression for the meta (or outer level) gradient that depends only on the solution to the inner optimization and not the path taken by the inner optimization algorithm, as depicted in Figure 1. This decoupling of meta-gradient computation and choice of inner level optimizer has a number of

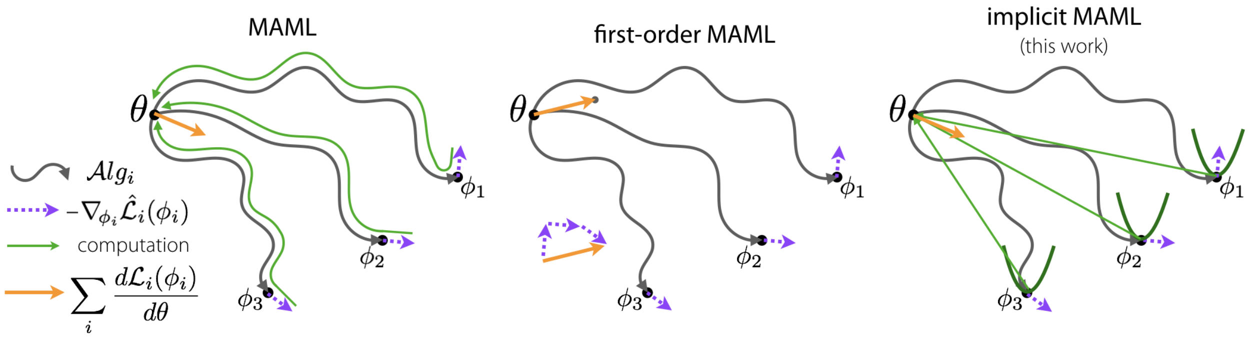


Figure 5.1: To compute the meta-gradient $\sum_i \frac{d\mathcal{L}_i(\phi_i)}{d\theta}$, the MAML algorithm differentiates through the optimization path, as shown in green, while first-order MAML computes the meta-gradient by approximating $\frac{d\boldsymbol{\phi}_i}{d\boldsymbol{\theta}}$ as $\boldsymbol{I}$. Our implicit MAML approach derives an analytic expression for the exact meta-gradient without differentiating through the optimization path by estimating local curvature.

appealing properties.

First, the inner optimization path need not be stored nor differentiated through, thereby making implicit MAML memory efficient and scalable to a large number of inner optimization steps. Second, implicit MAML is agnostic to the inner optimization method used, as long as it can find an approximate solution to the inner-level optimization problem. This permits the use of higher-order methods, and in principle even non-differentiable optimization methods or components like sample-based optimization, line-search, or those provided by proprietary software (e.g. Gurobi). Finally, we also provide the first (to our knowledge) non-asymptotic theoretical analysis of bi-level optimization. We show that an ϵ–approximate meta-gradient can be computed via implicit MAML using $\tilde{O}(\log(1/\epsilon))$ gradient evaluations and $\tilde{O}(1)$ memory, meaning the memory required does not grow with number of gradient steps.

5.2 Problem Formulation

We first present the meta-learning problem in the context of few-shot supervised learning, and then generalize the notation to aid the rest of the exposition in the paper.

5.2.1 Review of Few-Shot Supervised Learning and MAML

In few shot learning, we have a collection of meta-training tasks $\{\mathcal{T}_i\}_{i=1}^M$ drawn from $P(\mathcal{T})$. Each task $\mathcal{T}_i$ is associated with a dataset $\mathcal{D}_i$, from which we can sample two disjoint sets: $\mathcal{D}_i^{\text{tr}}$ and $\mathcal{D}_i^{\text{test}}$. These datasets each consist of K input-output pairs. Let $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{Y}$ denote inputs and outputs, respectively. The datasets take the form $\mathcal{D}_i^{\text{tr}} = \{(\mathbf{x}_i^k, \mathbf{y}_i^k)\}_{k=1}^K$, and similarly for $\mathcal{D}_i^{\text{test}}$. We are interested in learning models of the form $h_{\boldsymbol{\phi}}(\mathbf{x}) : \mathcal{X} \to \mathcal{Y}$, parameterized by $\boldsymbol{\phi} \in \Phi \equiv \mathbb{R}^d$. Performance on a task is specified by a loss function, such as the cross entropy or squared error loss. We will write the loss function in the form $\mathcal{L}(\boldsymbol{\phi}, \mathcal{D})$, as a function of a parameter vector and dataset. The goal for task $\mathcal{T}_i$ is to learn task-specific parameters $\boldsymbol{\phi}_i$ using $\mathcal{D}_i^{\text{tr}}$ such that we minimize the test loss, $\mathcal{L}(\boldsymbol{\phi}_i, \mathcal{D}_i^{\text{test}})$.

In the general bi-level meta-learning setup, we consider a space of algorithms that compute task-specific parameters using a set of meta-parameters $\boldsymbol{\theta} \in \Theta \equiv \mathbb{R}^d$ and the training dataset from the task, such that $\boldsymbol{\phi}_i = \mathcal{A}lg(\boldsymbol{\theta}, \mathcal{D}_i^{\text{tr}})$ for task $\mathcal{T}_i$. The goal of meta-learning is to learn meta-parameters that produce good task specific parameters after adaptation:

$$\overbrace{\boldsymbol{\theta}^*_{\text{ML}} := \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmin}}\, F(\boldsymbol{\theta})}^{\text{outer}-\text{level}}, \text{ where } F(\boldsymbol{\theta}) = \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}\Big(\overbrace{\mathcal{A}lg\big(\boldsymbol{\theta}, \mathcal{D}_i^{\text{tr}}\big)}^{\text{inner}-\text{level}},\ \mathcal{D}_i^{\text{test}} \Big). \tag{5.1}$$

We view this as a bi-level optimization problem since we typically interpret $\mathcal{A}lg\big(\boldsymbol{\theta}, \mathcal{D}_i^{\text{tr}}\big)$ as either explicitly or implicitly solving an underlying optimization problem. At meta-test (deployment) time, when presented with a dataset $\mathcal{D}_j^{\text{tr}}$ corresponding to a new task $\mathcal{T}_j \sim P(\mathcal{T})$, we can achieve good generalization performance (i.e., low test error) by using the adaptation procedure with the meta-learned parameters as $\boldsymbol{\phi}_j = \mathcal{A}lg(\boldsymbol{\theta}^*_{\text{ML}}, \mathcal{D}_j^{\text{tr}})$.

In the case of MAML [191], $\mathcal{A}lg(\boldsymbol{\theta}, \mathcal{D})$ corresponds to one or multiple steps of gradient descent initialized at $\boldsymbol{\theta}$. For example, if one step of gradient descent is used, we have:

$$\boldsymbol{\phi}_i \equiv \mathcal{A}lg(\boldsymbol{\theta}, \mathcal{D}_i^{\text{tr}}) = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_i^{\text{tr}}). \quad \text{(inner-level of MAML)} \tag{5.2}$$

Typically, α is a scalar hyperparameter, but can also be a learned vector [201]. Hence, for MAML, the meta-learned parameter ($\boldsymbol{\theta}^*_{\text{ML}}$) has a learned inductive bias that is particularly well-suited for fine-tuning on tasks from $P(\mathcal{T})$ using K samples. To solve the outer-level problem with gradient-based methods, we require a way to differentiate through $\mathcal{A}lg$. In the case of MAML, this corresponds to backpropagating through the dynamics of the gradient descent learning algorithm.

5.2.2 Proximal Regularization in the Inner Level

To have sufficient learning in the inner level while also avoiding over-fitting, $\mathcal{A}lg$ needs to incorporate some form of regularization. Since MAML uses a small number of gradient steps, this corresponds to early stopping and can be interpreted as a form of regularization and Bayesian prior [202]. In cases like ill-conditioned optimization landscapes and medium-shot learning, we may want to take many gradient steps, which poses two challenges for MAML. First, we need to store and differentiate through the long optimization path of $\mathcal{A}lg$, which imposes a considerable computation and memory burden. Second, the dependence of the model-parameters $\{\boldsymbol{\phi}_i\}$ on the meta-parameters ($\boldsymbol{\theta}$) shrinks and vanishes as the number of gradient steps in $\mathcal{A}lg$ grows, making meta-learning difficult. To overcome these limitations, we consider a more explicitly regularized algorithm:

$$\mathcal{A}lg^{\star}(\boldsymbol{\theta}, \mathcal{D}_i^{\text{tr}}) = \underset{\boldsymbol{\phi}' \in \Phi}{\operatorname{argmin}}\ \mathcal{L}(\boldsymbol{\phi}', \mathcal{D}_i^{\text{tr}}) + \frac{\lambda}{2}\ ||\boldsymbol{\phi}' - \boldsymbol{\theta}||^2. \tag{5.3}$$

The proximal regularization term in Eq. 5.3 encourages $\boldsymbol{\phi}_i$ to remain close to $\boldsymbol{\theta}$, thereby retaining a strong dependence throughout. The regularization strength (λ) plays a role similar to the learning rate (α) in MAML, controlling the strength of the prior ($\boldsymbol{\theta}$) relative to the data ($\mathcal{D}^{\text{tr}}_{\mathcal{T}}$). Like α, the regularization strength λ may also be learned. Furthermore, both α and λ can be scalars, vectors, or full matrices. For simplicity, we treat λ as a scalar hyperparameter. In Eq. 5.3, we use $\mathcal{A}lg^\star$ to denote that the optimization problem is solved exactly. In practice, we use iterative algorithms (denoted by $\mathcal{A}lg$) for finite iterations, which return approximate minimizers. We explicitly consider the discrepancy between approximate and exact solutions in our analysis.

5.2.3 The Bi-Level Optimization Problem

For notation convenience, we will sometimes express the dependence on task $\mathcal{T}_i$ using a subscript instead of arguments, e.g. we write:

$$\mathcal{L}_i(\boldsymbol{\phi}) := \mathcal{L}\big(\boldsymbol{\phi},\ \mathcal{D}_i^{\text{test}}\big), \quad \hat{\mathcal{L}}_i(\boldsymbol{\phi}) := \mathcal{L}\big(\boldsymbol{\phi}, \mathcal{D}_i^{\text{tr}}\big), \quad \mathcal{A}lg_i\big(\boldsymbol{\theta}\big) := \mathcal{A}lg\big(\boldsymbol{\theta}, \mathcal{D}_i^{\text{tr}}\big).$$

With this notation, the bi-level meta-learning problem can be written more generally as:

$$\begin{aligned} \boldsymbol{\theta}^*_{\text{ML}} &= \operatorname*{argmin}_{\boldsymbol{\theta}\in\Theta} F(\boldsymbol{\theta}) := \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}_i\big(\mathcal{A}lg_i^\star(\boldsymbol{\theta})\big), \text{ and} \\ \mathcal{A}lg_i^\star(\boldsymbol{\theta}) &= \operatorname*{argmin}_{\boldsymbol{\phi}'\in\Phi} G_i(\boldsymbol{\phi}', \boldsymbol{\theta}) := \hat{\mathcal{L}}_i(\boldsymbol{\phi}') + \frac{\lambda}{2}\ ||\boldsymbol{\phi}' - \boldsymbol{\theta}||^2. \end{aligned} \tag{5.4}$$

5.2.4 Total and Partial Derivatives

We use $\boldsymbol{d}$ to denote the total derivative and ∇ to denote partial derivative. For nested function of the form $\mathcal{L}_i(\boldsymbol{\phi}_i)$ where $\boldsymbol{\phi}_i = \mathcal{A}lg_i(\boldsymbol{\theta})$, we have from chain rule

$$\boldsymbol{d}_{\boldsymbol{\theta}}\mathcal{L}_i(\mathcal{A}lg_i(\boldsymbol{\theta})) = \frac{d\mathcal{A}lg_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}} \nabla_{\boldsymbol{\phi}}\mathcal{L}_i(\boldsymbol{\phi}) \mid_{\boldsymbol{\phi}=\mathcal{A}lg_i(\boldsymbol{\theta})} = \frac{d\mathcal{A}lg_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}} \nabla_{\boldsymbol{\phi}}\mathcal{L}_i(\mathcal{A}lg_i(\boldsymbol{\theta}))$$

Note the important distinction between $\boldsymbol{d}_{\boldsymbol{\theta}}\mathcal{L}_i(\mathcal{A}lg_i(\boldsymbol{\theta}))$ and $\nabla_{\boldsymbol{\phi}}\mathcal{L}_i(\mathcal{A}lg_i(\boldsymbol{\theta}))$. The former passes derivatives through $\mathcal{A}lg_i(\boldsymbol{\theta})$ while the latter does not. $\nabla_{\boldsymbol{\phi}}\mathcal{L}_i(\mathcal{A}lg_i(\boldsymbol{\theta}))$ is simply the gradient function, i.e. $\nabla_{\boldsymbol{\phi}}\mathcal{L}_i(\boldsymbol{\phi})$, evaluated at $\boldsymbol{\phi} = \mathcal{A}lg_i(\boldsymbol{\theta})$. Also note that $\boldsymbol{d}_{\boldsymbol{\theta}}\mathcal{L}_i(\mathcal{A}lg_i(\boldsymbol{\theta}))$ and $\nabla_{\boldsymbol{\phi}}\mathcal{L}_i(\mathcal{A}lg_i(\boldsymbol{\theta}))$ are d–dimensional vectors, while $\frac{d\mathcal{A}lg_i(\boldsymbol{\theta})}{d\boldsymbol{\theta}}$ is a $(d \times d)$–size Jacobian matrix. Throughout this text, we will also use $\boldsymbol{d}_{\boldsymbol{\theta}}$ and $\frac{d}{d\boldsymbol{\theta}}$ interchangeably.

5.3 The Implicit MAML Algorithm

Our aim is to solve the bi-level meta-learning problem in Eq. 5.4 using an iterative gradient based algorithm of the form $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\ \boldsymbol{d}_{\boldsymbol{\theta}}F(\boldsymbol{\theta})$. Although we derive our method based on standard gradient descent for simplicity, any other optimization method, such as quasi-Newton or Newton methods, Adam [203], or gradient descent with momentum can also be used without modification. The gradient descent update be expanded using the chain rule as

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \, \frac{1}{M} \sum_{i=1}^{M} \frac{d\mathcal{A}lg_i^\star(\boldsymbol{\theta})}{d\boldsymbol{\theta}} \; \nabla_\phi \mathcal{L}_i(\mathcal{A}lg_i^\star(\boldsymbol{\theta})). \tag{5.5}$$

Here, $\nabla_{\boldsymbol{\phi}} \mathcal{L}_i(\mathcal{A}lg_i^\star(\boldsymbol{\theta}))$ is simply $\nabla_{\boldsymbol{\phi}} \mathcal{L}_i(\boldsymbol{\phi}) \mid_{\boldsymbol{\phi}=\mathcal{A}lg_i^\star(\boldsymbol{\theta})}$ which can be easily obtained in practice via automatic differentiation. For this update rule, we must compute $\frac{d\mathcal{A}lg_i^\star(\boldsymbol{\theta})}{d\boldsymbol{\theta}}$, where $\mathcal{A}lg_i^\star$ is implicitly defined as an optimization problem (Eq. 5.4), which presents the primary challenge. We now present an efficient algorithm (in compute and memory) to compute the meta-gradient..

5.3.1 Meta-Gradient Computation

If $\mathcal{A}lg_i^\star(\boldsymbol{\theta})$ is implemented as an iterative algorithm, such as gradient descent, then one way to compute $\frac{d\mathcal{A}lg_i^\star(\boldsymbol{\theta})}{d\boldsymbol{\theta}}$ is to propagate derivatives through the iterative process, either in forward mode or reverse mode. However, this has the drawback of depending explicitly on the path of the optimization, which has to be fully stored in memory, quickly becoming intractable when the number of gradient steps needed is large. Furthermore, for second order optimization methods, such as Newton's method, third derivatives are needed which are difficult to obtain. Furthermore, this approach becomes impossible when non-differentiable operations, such as line-searches, are used. However, by recognizing that $\mathcal{A}lg_i^\star$ is implicitly defined as the solution to an optimization problem, we may employ a different strategy that does not need to consider the path of the optimization but only the final result. This is derived in the following Lemma.

Lemma 3. *(Implicit Jacobian) Consider $\mathcal{A}lg_i^\star(\boldsymbol{\theta})$ as defined in Eq. 5.4 for task $\mathcal{T}_i$. Let $\boldsymbol{\phi}_i = \mathcal{A}lg_i^\star(\boldsymbol{\theta})$ be the result of $\mathcal{A}lg_i^\star(\boldsymbol{\theta})$. If $\left(\boldsymbol{I} + \frac{1}{\lambda}\nabla^2_{\boldsymbol{\phi}} \hat{\mathcal{L}}_i(\boldsymbol{\phi}_i)\right)$ is invertible, then the derivative Jacobian is*

$$\frac{d\mathcal{A}lg_i^\star(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = \left(\boldsymbol{I} + \frac{1}{\lambda}\, \nabla^2_{\boldsymbol{\phi}} \hat{\mathcal{L}}_i(\boldsymbol{\phi}_i)\right)^{-1}. \tag{5.6}$$

Please see appendix for the proof. Note that the derivative (Jacobian) depends only on the final result of the algorithm, and not the path taken by the algorithm. Thus, in principle any approach of algorithm can be used to compute $\mathcal{A}lg_i^\star(\boldsymbol{\theta})$, thereby decoupling meta-gradient computation from choice of inner level optimizer.

Practical Algorithm: While Lemma 3 provides an idealized way to compute the $\mathcal{A}lg_i^\star$ Jacobians and thus by extension the meta-gradient, it may be difficult to directly use it in practice. Two issues are particularly relevant. First, the meta-gradients require computation

Algorithm 5 Implicit Model-Agnostic Meta-Learning (iMAML)

```
1: Require: Distribution over tasks $P(\mathcal{T})$, outer step size $\eta$, regularization strength $\lambda$,
2: while not converged do
3:     Sample mini-batch of tasks $\{\mathcal{T}_i\}_{i=1}^{B} \sim P(\mathcal{T})$
4:     for Each task $\mathcal{T}_i$ do
5:         Compute task meta-gradient $\boldsymbol{g}_i = \texttt{Implicit-Meta-Gradient}(\mathcal{T}_i, \boldsymbol{\theta}, \lambda)$
6:     end for
7:     Average above gradients to get $\hat{\nabla} F(\boldsymbol{\theta}) = (1/B) \sum_{i=1}^{B} \boldsymbol{g}_i$
8:     Update meta-parameters with gradient descent: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \hat{\nabla} F(\boldsymbol{\theta})$   // (or Adam)
9: end while
```

Algorithm 6 Implicit Meta-Gradient Computation

```
1: Input: Task $\mathcal{T}_i$, meta-parameters $\boldsymbol{\theta}$, regularization strength $\lambda$
2: Hyperparameters: Optimization accuracy thresholds $\delta$ and $\delta'$
3: Obtain task parameters $\boldsymbol{\phi}_i$ using iterative optimization solver such that: $\|\boldsymbol{\phi}_i - \mathcal{A}lg_i^{\star}(\boldsymbol{\theta})\| \leq \delta$
4: Compute partial outer-level gradient $\boldsymbol{v}_i = \nabla_{\boldsymbol{\phi}} \mathcal{L}_{\mathcal{T}}(\boldsymbol{\phi}_i)$
5: Use an iterative solver (e.g. CG) along with reverse mode differentiation (to compute Hessian vector products) to compute $\boldsymbol{g}_i$ such that: $\|\boldsymbol{g}_i - \left(\boldsymbol{I} + \frac{1}{\lambda}\nabla^2 \hat{\mathcal{L}}_i(\boldsymbol{\phi}_i)\right)^{-1} \boldsymbol{v}_i\| \leq \delta'$
6: Return: $\boldsymbol{g}_i$
```

of $\mathcal{A}lg_i^{\star}(\boldsymbol{\theta})$, which is the exact solution to the inner optimization problem. In practice, we may be able to obtain only approximate solutions. Second, explicitly forming and inverting the matrix in Eq. 5.6 for computing the Jacobian may be intractable for large deep neural networks. To address these difficulties, we consider approximations to the idealized approach that enable a practical algorithm. First, we consider an approximate solution to the inner optimization problem, that can be obtained with iterative optimization algorithms like gradient descent.

Definition 5. *(δ–approx. algorithm) Let $\mathcal{A}lg_i(\boldsymbol{\theta})$ be a δ–accurate approximation of $\mathcal{A}lg_i^{\star}(\boldsymbol{\theta})$,*

$$\|\mathcal{A}lg_i(\boldsymbol{\theta}) - \mathcal{A}lg_i^{\star}(\boldsymbol{\theta})\| \leq \delta$$

Second, we will perform a partial or approximate matrix inversion given by:

Definition 6. *(δ'–approximate Jacobian-vector product) Let $\boldsymbol{g}_i$ be a vector such that*

$$\|\boldsymbol{g}_i - \left(\boldsymbol{I} + \frac{1}{\lambda}\, \nabla_{\boldsymbol{\phi}}^2 \hat{\mathcal{L}}_i(\boldsymbol{\phi}_i)\right)^{-1} \nabla_{\boldsymbol{\phi}} \mathcal{L}_i(\boldsymbol{\phi}_i)\| \leq \delta'$$

where $\boldsymbol{\phi}_i = \mathcal{A}lg_i(\boldsymbol{\theta})$ and $\mathcal{A}lg_i$ is based on definition 5.

Note that $\boldsymbol{g}_i$ in definition 6 is an approximation of the meta-gradient for task $\mathcal{T}_i$. Observe that $\boldsymbol{g}_i$ can be obtained as an approximate solution to the optimization problem:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\, \boldsymbol{w}^\top \left(\boldsymbol{I} + \frac{1}{\lambda}\, \nabla^2_{\boldsymbol{\phi}} \hat{\mathcal{L}}_i(\boldsymbol{\phi}_i) \right) \boldsymbol{w} - \boldsymbol{w}^\top \nabla_{\boldsymbol{\phi}} \mathcal{L}_i(\boldsymbol{\phi}_i) \tag{5.7}$$

The conjugate gradient (CG) algorithm is particularly well suited for this problem due to its excellent iteration complexity and requirement of only Hessian-vector products of the form $\nabla^2 \hat{\mathcal{L}}_i(\boldsymbol{\phi}_i)\boldsymbol{v}$. Such hessian-vector products can be obtained cheaply without explicitly forming or storing the Hessian matrix (as we discuss in Appendix D.3). This CG based inversion has been successfully deployed in Hessian-free or Newton-CG methods for deep learning [204, 205] and trust region methods in reinforcement learning [33, 12]. Algorithm 5 presents the full practical algorithm. Note that these approximations to develop a practical algorithm introduce errors in the meta-gradient computation. We analyze the impact of these errors in Section 5.3.2 and show that they are controllable. See Appendix D.1 for how iMAML generalizes prior gradient optimization based meta-learning algorithms.

5.3.2 Theoretical Analysis

In Section 5.3.1, we outlined a practical algorithm that makes approximations to the idealized update rule of Eq. 5.5. Here, we attempt to analyze the impact of these approximations, and also understand the computation and memory requirements of iMAML. We find that iMAML can match the minimax computational complexity of backpropagating through the path of the inner optimizer, but is substantially better in terms of memory usage. This work to our knowledge also provides the first non-asymptotic result that analyzes approximation error due to implicit gradients. Theorem 3 provides the computational and memory complexity for obtaining an ϵ–approximate meta-gradient. We assume $\mathcal{L}_i$ is smooth but do not require it to be convex. We assume that G_i in Eq. 5.4 is strongly convex, which can be made possible by appropriate choice of λ. The key to our analysis is a second order Lipshitz assumption, i.e. $\hat{\mathcal{L}}_i(\cdot)$ is ρ-Lipshitz Hessian. This assumption and setting has received considerable attention in recent optimization and deep learning literature [206, 207].

Table 5.1 summarizes our complexity results and compares with MAML and truncated backpropagation [208] through the path of the inner optimizer. We use κ to denote the condition number of the inner problem induced by G_i (see Equation 5.4), which can be viewed as a measure of hardness of the inner optimization problem. Mem$(\nabla\hat{\mathcal{L}}_i)$ is the memory taken to compute a single derivative $\nabla\hat{\mathcal{L}}_i$. Under the assumption that Hessian vector products are computed with the reverse mode of auto-differentiation, we will have that both: the compute time and memory used for computing a Hessian vector product are with a (universal) constant factor of the compute time and memory used for computing $\nabla\hat{\mathcal{L}}_i$ itself (see Appendix D.3). This allows us to measure the compute time in terms of the number of $\nabla\hat{\mathcal{L}}_i$ computations.

Table 5.1: Compute and memory for computing the meta-gradient when using a δ–accurate $\mathcal{A}lg_i$, and the corresponding approximation error. Our compute time is measured in terms of the number of $\nabla\hat{\mathcal{L}}_i$ computations. All results are in $\tilde{O}(\cdot)$ notation, which hide additional log factors; the error bound hides additional problem dependent Lipshitz and smoothness parameters (see the respective Theorem statements). $\kappa \geq 1$ is the condition number for inner objective G_i (see Equation 5.4), and D is the diameter of the search space. The notions of error are subtly different: we assume all methods solve the inner optimization to error level of δ (as per definition 5). For our algorithm, the error refers to the ℓ_2 error in the computation of $\boldsymbol{d_\theta}\mathcal{L}_i(\mathcal{A}lg_i^\star(\boldsymbol{\theta}))$. For the other algorithms, the error refers to the ℓ_2 error in the computation of $\boldsymbol{d_\theta}\mathcal{L}_i(\mathcal{A}lg_i(\boldsymbol{\theta}))$. We use Prop 3.1 of Shaban et al. [208] to provide the guarantee we use. See Appendix D.4 for additional discussion.

Algorithm	Compute	Memory	Error
MAML (GD + full back-prop)	$\kappa \log\left(\frac{D}{\delta}\right)$	$\text{Mem}(\nabla\hat{\mathcal{L}}_i) \cdot \kappa \ \log\left(\frac{D}{\delta}\right)$	$0^\dagger$
MAML (Nesterov's AGD + full back-prop)	$\sqrt{\kappa} \log\left(\frac{D}{\delta}\right)$	$\text{Mem}(\nabla\hat{\mathcal{L}}_i) \cdot \sqrt{\kappa} \ \log\left(\frac{D}{\delta}\right)$	$0^\dagger$
Truncated back-prop (GD) [2]	$\kappa \log\left(\frac{D}{\delta}\right)$	$\text{Mem}(\nabla\hat{\mathcal{L}}_i) \cdot \kappa \ \log\left(\frac{1}{\epsilon}\right)$	$\epsilon^\dagger$
Implicit MAML (this work)	$\sqrt{\kappa} \log\left(\frac{D}{\delta}\right)$	$\text{Mem}(\nabla\hat{\mathcal{L}}_i)$	δ^*

We refer readers to Appendix D.4 for additional discussion about the algorithms and their trade-offs. Our main theorem is as follows:

Theorem 3. *(Informal Statement; Approximation error in Algorithm 6) Suppose that: $\mathcal{L}_i(\cdot)$ is B Lipshitz and L smooth function; that $G_i(\cdot, \boldsymbol{\theta})$ (in Eq. 5.4) is a μ-strongly convex function with condition number κ; that D is the diameter of search space for $\boldsymbol{\phi}$ in the inner optimization problem (i.e. $\|\mathcal{A}lg_i^\star(\boldsymbol{\theta})\| \leq D$); and $\hat{\mathcal{L}}_i(\cdot)$ is ρ-Lipshitz Hessian.*

Let $\boldsymbol{g}_i$ be the task meta-gradient returned by Algorithm 6. For any task i and desired accuracy level ϵ, Algorithm 6 computes an approximate task-specific meta-gradient with the following guarantee:

$$||\boldsymbol{g}_i - \boldsymbol{d_\theta}\mathcal{L}_i(\mathcal{A}lg_i^\star(\boldsymbol{\theta}))|| \leq \epsilon\,.$$

Furthermore, under the assumption that the Hessian vector products are computed by the reverse mode of auto-differentiation, Algorithm 6 can be implemented using at most $\tilde{O}\left(\sqrt{\kappa}\log\left(\frac{poly(\kappa,D,B,L,\rho,\mu,\lambda)}{\epsilon}\right)\right)$ gradient computations of $\hat{\mathcal{L}}_i(\cdot)$ and $2 \cdot \text{Mem}(\nabla\hat{\mathcal{L}}_i)$ memory.

The formal statement of the theorem and the proof are provided the appendix. Importantly, the algorithm's memory requirement is equivalent to the memory needed for Hessian-vector products which is a small constant factor over the memory required for gradient computations, assuming the reverse mode of auto-differentiation is used. Finally, based on the above, we also present corollary 2 which shows that iMAML efficiently finds a stationary point of $F(\cdot)$, due to iMAML having controllable approximation error in gradient computation.

Corollary 2. *(iMAML finds stationary points) Suppose the conditions of Theorem 3 hold and that $F(\cdot)$ is an L_F smooth function. Then the implicit MAML algorithm (Algorithm 5), when the batch size is M (so that we are doing gradient descent), will find a point $\boldsymbol{\theta}$ such that: $\|\nabla F(\boldsymbol{\theta})\| \leq \epsilon$ in a number of calls to* `Implicit-Meta-Gradient` *that is at most $\frac{4ML_f(F(0)-\min_{\boldsymbol{\theta}} F(\boldsymbol{\theta}))}{\epsilon^2}$. Furthermore, the total number of gradient computations (of $\nabla\hat{\mathcal{L}}_i$) is at most*

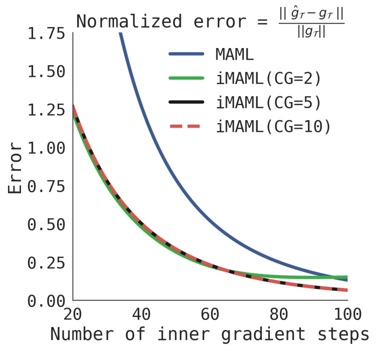
$$\tilde{O}\left(M\sqrt{\kappa}\frac{L_f(F(0)-\min_\theta F(\theta))}{\epsilon^2}\log\left(\frac{poly(\kappa, D, B, L, \rho, \mu, \lambda)}{\epsilon}\right)\right),$$

and only $\tilde{O}(\mathrm{Mem}(\nabla\hat{\mathcal{L}}_i))$ memory is required throughout.

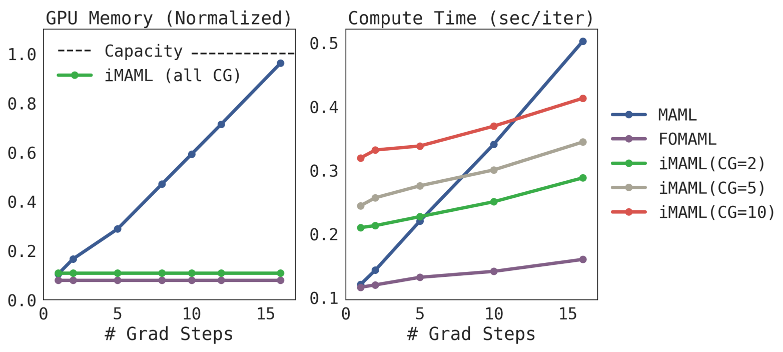
5.4 Experimental Results and Discussion

In our experimental evaluation, we aim to answer the following questions empirically: (1) Does the iMAML algorithm asymptotically compute the exact meta-gradient? (2) With finite iterations, does iMAML approximate the meta-gradient more accurately compared to MAML? (3) How does the computation and memory requirements of iMAML compare with MAML? (4) Does iMAML lead to better results in realistic meta-learning problems? We have answered (1) - (3) through our theoretical analysis, and now attempt to validate it through numerical simulations. For (1) and (2), we will use a simple synthetic example for which we can compute the exact meta-gradient and compare against it (exact-solve error, see definition 7). For (3) and (4), we will use the common few-shot image recognition domains of Omniglot and Mini-ImageNet.

To study the question of meta-gradient accuracy, Figure 5.2 considers a synthetic regression example, where the predictions are linear in parameters. This provides an analytical expression



(a)



(b)

Figure 5.2: Accuracy, Computation, and Memory tradeoffs of iMAML, MAML, and FOMAML. (a) Meta-gradient accuracy level in synthetic example. Computed gradients are compared against the exact meta-gradient per Def 7. (b) Computation and memory trade-offs with 4 layer CNN on 20-way-5-shot Omniglot task. We implemented iMAML in PyTorch, and for an apples-to-apples comparison, we use a PyTorch implementation of MAML from: https://github.com/dragen1860/MAML-Pytorch

Table 5.2: Omniglot results. MAML results are taken from the original work of Finn et al. [191], and first-order MAML and Reptile results are from Nichol et al. [160]. iMAML with gradient descent (GD) uses 16 and 25 steps for 5-way and 20-way tasks respectively. iMAML with Hessian-free uses 5 CG steps to compute the search direction and performs line-search to pick step size. Both versions of iMAML use $\lambda = 2.0$ for regularization, and 5 CG steps to compute the task meta-gradient.

Algorithm	5-way 1-shot	5-way 5-shot	20-way 1-shot	20-way 5-shot
MAML [191]	98.7 ± 0.4%	**99.9 ± 0.1%**	95.8 ± 0.3%	98.9 ± 0.2%
first-order MAML [191]	98.3 ± 0.5%	99.2 ± 0.2%	89.4 ± 0.5%	97.9 ± 0.1%
Reptile [160]	97.68 ± 0.04%	99.48 ± 0.06%	89.43 ± 0.14%	97.12 ± 0.32%
iMAML, GD (ours)	99.16 ± 0.35%	99.67 ± 0.12%	94.46 ± 0.42%	98.69 ± 0.1%
iMAML, Hessian-Free (ours)	**99.50 ± 0.26%**	99.74 ± 0.11%	**96.18 ± 0.36%**	**99.14 ± 0.1%**

for $\mathcal{A}lg_i^\star$ allowing us to compute the true meta-gradient. We fix gradient descent (GD) to be the inner optimizer for both MAML and iMAML. The problem is constructed so that the condition number (κ) is large, thereby necessitating many GD steps. We find that both iMAML and MAML asymptotically match the exact meta-gradient, but iMAML computes a better approximation in finite iterations. We observe that with 2 CG iterations, iMAML incurs a small terminal error. This is consistent with our theoretical analysis. In Algorithm 6, δ is dominated by δ' when only a small number of CG steps are used. However, the terminal error vanishes with just 5 CG steps. The computational cost of 1 CG step is comparable to 1 inner GD step with the MAML algorithm, since both require 1 hessian-vector product (see section D.3 for discussion). Thus, the computational cost as well as memory of iMAML with 100 inner GD steps is significantly smaller than MAML with 100 GD steps.

To study (3), we turn to the Omniglot dataset [209] which is a popular few-shot image recognition domain. Figure 5.2 presents compute and memory trade-off for MAML and iMAML (on 20-way, 5-shot Omniglot). Memory for iMAML is based on Hessian-vector products and is independent of the number of GD steps in the inner loop. The memory use is also independent of the number of CG iterations, since the intermediate computations need not be stored in memory. On the other hand, memory for MAML grows linearly in grad steps, reaching the capacity of a 12 GB GPU in approximately 16 steps. First-order MAML (FOMAML) does not back-propagate through the optimization process, and thus the computational cost is only that of performing gradient descent, which is needed for all the algorithms. The computational cost for iMAML is also similar to FOMAML along with a constant overhead for CG that depends on the number of CG steps. Note however, that FOMAML does not compute an accurate meta-gradient, since it ignores the Jacobian. Compared to FOMAML, the compute cost of MAML grows at a faster rate. FOMAML requires only gradient computations, while backpropagating through GD (as done in MAML) requires a Hessian-vector products at each iteration, which are more expensive.

Finally, we study empirical performance of iMAML on the Omniglot and Mini-ImageNet domains. Following the few-shot learning protocol in prior work [190], we run the iMAML algorithm on the dataset for different numbers of class labels and shots (in the N-way, K-shot setting), and compare two variants of iMAML with published results of the most closely related algorithms: MAML, FOMAML, and Reptile. While these methods are not state-of-the-art on this benchmark, they provide an apples-to-apples comparison for studying the use of implicit gradients in optimization-based meta-learning. For a fair comparison, we use the identical convolutional architecture as these prior works. Note however that architecture tuning can lead to better results for all algorithms [210].

The first variant of iMAML we consider involves solving the inner level problem (the regularized objective function in Eq. 5.4) using gradient descent. The meta-gradient is computed using conjugate gradient, and the meta-parameters are updated using Adam. This presents the most straightforward comparison with MAML, which would follow a similar procedure, but backpropagate through the path of optimization as opposed to invoking implicit differentiation. The second variant of iMAML uses a second order method for the inner level problem. In particular, we consider the Hessian-free or Newton-CG [205, 204] method. This method makes a local quadratic approximation to the objective function (in our case, $G(\boldsymbol{\phi}', \boldsymbol{\theta})$ and approximately computes the Newton search direction using CG. Since CG requires only Hessian-vector products, this way of approximating the Newton search direction is scalable to large deep neural networks. The step size can be computed using regularization, damping, trust-region, or linesearch. We use a linesearch on the training loss in our experiments to also illustrate how our method can handle non-differentiable inner optimization loops. We refer the readers to Nocedal & Wright [205] and Martens [204] for a more detailed exposition of this optimization algorithm. Similar approaches have also gained prominence in reinforcement learning [33, 12].

Tables 5.2 and 5.3 present the results on Omniglot and Mini-ImageNet, respectively. On the Omniglot domain, we find that the GD version of iMAML is competitive with the full MAML algorithm, and substatially better than its approximations (i.e., first-order MAML and Reptile), especially for the harder 20-way tasks. We also find that iMAML with Hessian-free optimization performs substantially better than the other methods, suggesting that powerful optimizers in the inner loop can offer benefits to meta-learning. In the Mini-ImageNet domain, we

Table 5.3: Mini-ImageNet 5-way-1-shot accuracy

Algorithm	5-way 1-shot
MAML	48.70 ± 1.84 %
first-order MAML	48.07 ± 1.75 %
Reptile	49.97 ± 0.32 %
iMAML GD (ours)	48.96 ± 1.84 %
iMAML HF (ours)	49.30 ± 1.88 %

find that iMAML performs better than MAML and FOMAML. We used $\lambda = 0.5$ and 10 gradient steps in the inner loop. We did not perform an extensive hyperparameter sweep, and expect that the results can improve with better hyperparameters. 5 CG steps were used to compute the meta-gradient. The Hessian-free version also uses 5 CG steps for the search direction. Additional experimental details are Appendix D.6.

5.5 *Related Work*

Our work considers the general meta-learning problem [186, 187, 188], including few-shot learning [209, 190]. Meta-learning approaches can generally be categorized into metric-learning approaches that learn an embedding space where non-parametric nearest neighbors works well [211, 190, 212, 213, 214], black-box approaches that train a recurrent or recursive neural network to take datapoints as input and produce weight updates [192, 215, 216, 193] or predictions for new inputs [189, 217, 218, 219, 194], and optimization-based approaches that use bi-level optimization to embed learning procedures, such as gradient descent, into the meta-optimization problem [191, 200, 220, 221, 201, 222, 223, 224]. Hybrid approaches have also been considered to combine the benefits of different approaches [225, 226]. We build upon optimization-based approaches, particularly the MAML algorithm [191], which meta-learns an initial set of parameters such that gradient-based fine-tuning leads to good generalization. Prior work has considered a number of inner loops, ranging from a very general setting where all parameters are adapted using gradient descent [191], to more structured and specialized settings, such as ridge regression [220], Bayesian linear regression [224], and simulated annealing [227]. The main difference between our work and these approaches is that we show how to analytically derive the gradient of the outer objective without differentiating through the inner learning procedure.

Mathematically, we view optimization-based meta-learning as a bi-level optimization problem. Such problems have been studied in the context of few-shot meta-learning (as discussed previously), gradient-based hyperparameter optimization [195, 228, 229, 230, 231], and a range of other settings [232, 233]. Some prior works have derived implicit gradients for related problems [228, 230, 232] while others propose innovations to aid back-propagation through the optimization path for specific algorithms [195, 229, 234], or approximations like truncation [208]. While the broad idea of implicit differentiation is well known, it has not been empirically demonstrated in the past for learning more than a few parameters (e.g., hyperparameters), or highly structured settings such as quadratic programs [232]. In contrast, our method meta-trains deep neural networks with thousands of parameters. Closest to our setting is the recent work of Lee et al. [235], which uses implicit differentiation for quadratic programs in a final SVM layer. In contrast, our formulation allows for adapting the full network for generic objectives (beyond hinge-loss), thereby allowing for wider applications.

We also note that prior works involving implicit differentiation make a strong assumption of an exact solution in the inner level, thereby providing only asymptotic guarantees. In contrast, we provide finite time guarantees which allows us to analyze the case where the inner level is solved approximately. In practice, the inner level is likely to be solved using iterative optimization algorithms like gradient descent, which only return approximate solutions with finite iterations. Thus, this paper places implicit gradient methods under a strong theoretical footing for practically use.

5.6 Chapter Summary

In this chapter, we presented a new method for optimization-based meta-learning that removes the need for differentiating through the inner optimization path, allowing us to decouple the outer meta-gradient computation from the choice of inner optimization algorithm. We showed how this gives us significant gains in compute and memory efficiency, and also conceptually allows us to use a variety of inner optimization methods. While we focused on developing the foundations and theoretical analysis of this method, we believe that this work opens up a number of interesting avenues for future study.

Broader classes of inner loop procedures. While we studied different gradient-based optimization methods in the inner loop, iMAML can in principle be used with a variety of inner loop algorithms, including dynamic programming methods such as Q-learning, two-player adversarial games such as GANs, energy-based models [236], and actor-critic RL methods, and higher-order model-based trajectory optimization methods. This significantly expands the kinds of problems that optimization-based meta-learning can be applied to.

More flexible regularizers. We explored one very simple regularization, ℓ_2 regularization to the parameter initialization, which already increases the expressive power over the implicit regularization that MAML provides through truncated gradient descent. To further allow the model to flexibly regularize the inner optimization, a simple extension of iMAML is to learn a vector- or matrix-valued λ, which would enable the meta-learner model to co-adapt and co-regularize various parameters of the model. Regularizers that act on parameterized density functions would also enable meta-learning to be effective for few-shot density estimation.

Chapter 6

META LEARNING UNDER NON-STATIONARITY

6.1 Introduction

In the previous chapter, we considered meta-learning, which provided a paradigm to achieve generalization in new tasks using a small amount of data. This was achieved by constructing an agent that learns how to learn by utilizing prior tasks to learn priors that are amenable to fast adaptation. This question of how to use prior tasks or experience to inform future learning has actually been studied by two distinct paradigms. As we discussed, meta-learning [186, 190, 191] casts this as the problem of *learning to learn*, where past experience is used to acquire a prior over model parameters or a learning procedure, and typically studies a setting where a set of meta-training tasks are made available together upfront.

In contrast, online learning [237, 157] considers a sequential setting where tasks are revealed one after another, but aims to attain zero-shot generalization without any task-specific adaptation. We argue that neither setting is ideal for studying continual lifelong learning, as experienced by humans in the real world. Meta-learning deals with learning to learn, but neglects the sequential and non-stationary aspects of the problem. Online learning offers an appealing theoretical framework, but does not generally consider how past experience can accelerate adaptation to a new task. In this chapter, we motivate and present the *online meta-learning* problem setting, where the agent simultaneously uses past experiences in a sequential setting to learn good priors, and also adapt quickly to the current task at hand.

As a motivating example, Figure 6.1 shows a family of sinusoids. Imagine that each task is a regression problem from x to y corresponding to *one* sinusoid. When presented with data from a large collection of such tasks, a naïve approach that does not consider the task structure would collectively use all the data, and learn a prior that corresponds to the model $y = 0$. An algorithm that understands the underlying structure would recognize that each curve in the family is a (different) sinusoid, and would therefore attempt to identify, for a new batch of data, which sinusoid it corresponds to. As another example where naïve training on prior tasks fails, Figure 6.1 also shows colored MNIST digits with different backgrounds. Suppose we've seen MNIST digits with various colored backgrounds, and then observe a "7" on a new color. We might conclude from training on all of the data seen so far that all digits with that color must all be "7." In fact, this is an optimal conclusion from a purely statistical standpoint. However, if we understand that the data is divided into different tasks, and take note of the fact that each task has a different color, a better conclusion is that the color is irrelevant.

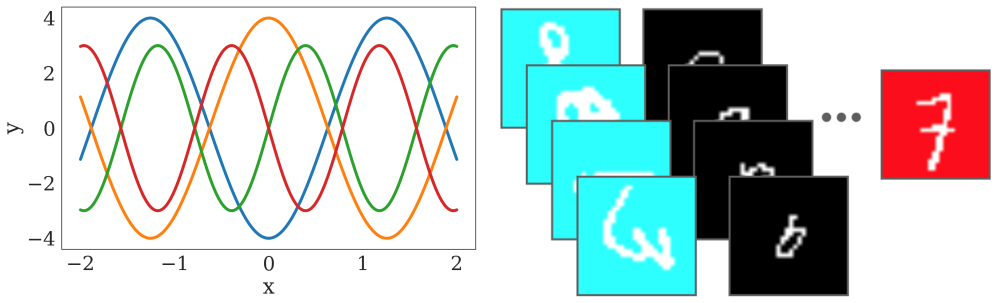


Figure 6.1: (left) sinusoid functions and (right) colored MNIST

Meta-learning offers an appealing solution: by learning how to learn from past tasks, we can make use of task structure and extract information from the data that both allows us to succeed on the current task and adapt to new tasks more quickly. However, typical meta learning approaches assume that a sufficiently large set of tasks are made available upfront for meta-training. In the real world, tasks are likely available only sequentially, as the agent is learning in the world, and also from a non-stationary distribution. By recasting meta-learning in a sequential or online setting, that does not make strong distributional assumptions, we can enable faster learning on new tasks as they are presented.

Contributions: In this chapter, we formulate the online meta-learning problem setting and present the *follow the meta-leader (FTML)* algorithm. This extends the MAML algorithm to the online meta-learning setting, and is analogous to follow the leader in online learning. We analyze FTML and show that it enjoys a $\mathcal{O}(\log T)$ regret guarantee when competing with the best meta-learner in hindsight. We also develop a practical form of FTML that can be used effectively with deep neural networks on large scale tasks, and show that it significantly outperforms prior methods. The experiments involve vision-based sequential learning tasks with the MNIST, CIFAR-100, and PASCAL 3D+ datasets.

6.2 Foundations

Before introducing online meta-learning, we first briefly summarize the foundations of meta-learning, the model-agnostic meta-learning (MAML) algorithm, and online learning. To illustrate the differences in setting and algorithms, we will use the running example of few-shot learning, which we describe below first. We emphasize that online learning, MAML, and the online meta-learning formulations have a broader scope than few-shot supervised learning. We use the few-shot supervised learning example primarily for illustration.

6.2.1 Few-Shot Learning

In the few-shot supervised learning setting [189], we are interested in a family of tasks, where each task $\mathcal{T}$ is associated with a notional and infinite-size population of input-output pairs. In the few-shot learning, the goal is to learn a task while accessing only a small, finite-size labeled dataset $\mathcal{D}_i := \{\mathbf{x}_i, \mathbf{y}_i\}$ corresponding to task $\mathcal{T}_i$. If we have a predictive model, $\boldsymbol{h}(\cdot; \boldsymbol{\phi})$, with parameters $\boldsymbol{\phi}$, the population risk of the model is

$$\mathcal{L}_i(\boldsymbol{\phi}) := \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{T}_i}[\ell(\mathbf{x}, \mathbf{y}, \boldsymbol{\phi})],$$

where the expectation is defined over the task population and ℓ is a loss function, such as the square loss or cross-entropy between the model prediction and the correct label. An example of ℓ corresponding to squared error loss is

$$\ell(\mathbf{x}, \mathbf{y}, \boldsymbol{\phi}) = ||\mathbf{y} - \boldsymbol{h}(\mathbf{x}; \boldsymbol{\phi})||^2.$$

Let $\mathcal{L}(\mathcal{D}_i, \boldsymbol{\phi})$ represent the average loss on the dataset $\mathcal{D}_i$. Being able to effectively minimize $\mathcal{L}_i(\boldsymbol{\phi})$ is likely hard if we rely only on $\mathcal{D}_i$ due to the small size of the dataset. However, we are exposed to many such tasks from the family — either in sequence or as a batch, depending on the setting. By being able to draw upon the multiplicity of tasks, we may hope to perform better, as for example demonstrated in the meta-learning literature.

6.2.2 Meta-Learning and MAML

Meta-learning, or learning to learn [186], aims to effectively bootstrap from a set of tasks to learn faster on a new task. It is assumed that tasks are drawn from a fixed distribution, $\mathcal{T} \sim \mathbb{P}(\mathcal{T})$. At meta-training time, M tasks $\{\mathcal{T}_i\}_{i=1}^M$ are drawn from this distribution and datasets corresponding to them are made available to the agent. At deployment time, we are faced with a new test task $\mathcal{T}_j \sim \mathbb{P}(\mathcal{T})$, for which we are again presented with a small labeled dataset $\mathcal{D}_j := \{\mathbf{x}_j, \mathbf{y}_j\}$. Meta-learning algorithms attempt to find a model using the M training tasks, such that when $\mathcal{D}_j$ is revealed from the test task, the model can be quickly updated to minimize $\mathcal{L}_j(\boldsymbol{\phi})$.

Model-agnostic meta-learning (MAML) [191] does this by learning an initial set of parameters $\boldsymbol{\phi}_{\text{MAML}}$, such that at meta-test time, performing a few steps of gradient descent from $\boldsymbol{\phi}_{\text{MAML}}$ using $\mathcal{D}_j$ minimizes $\mathcal{L}_j(\cdot)$. To get such an initialization, at meta-training time, MAML solves the optimization problem:

$$\boldsymbol{\phi}_{\text{MAML}} := \arg\min_{\boldsymbol{\phi}} \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}_i\big(\boldsymbol{\phi} - \alpha \nabla \hat{\mathcal{L}}_i(\boldsymbol{\phi})\big). \tag{6.1}$$

The inner gradient $\nabla \hat{\mathcal{L}}_i(\boldsymbol{\phi})$ is based on a small mini-batch of data from $\mathcal{D}_i$. Hence, MAML optimizes for few-shot generalization. Note that the optimization problem is subtle: we have a gradient descent update step embedded in the actual objective function. Regardless, Finn et al. [191] show that gradient-based methods can be used on this optimization objective with existing automatic differentiation libraries. Stochastic optimization techniques are used to solve the optimization problem in Eq. 6.1 since the population risk is not known directly. At meta-test time, the solution to Eq. 6.1 is fine-tuned as: $\boldsymbol{\phi}_j \leftarrow \boldsymbol{\phi}_{\text{MAML}} - \alpha \nabla \hat{\mathcal{L}}_j(\boldsymbol{\phi}_{\text{MAML}})$ with the gradient obtained using $\mathcal{D}_j$. Meta-training can be interpreted as learning a prior over model parameters, and fine-tuning as inference [202].

MAML and other meta-learning algorithms (see Section 6.7) are not directly applicable to sequential settings for two reasons. First, they have two distinct phases: meta-training and meta-testing or deployment. We would like the algorithms to work in a continuous learning fashion. Second, meta-learning methods generally assume that the tasks come from some fixed distribution, whereas we would like methods that work for non-stationary task distributions.

6.2.3 Online Learning

In the online learning setting, an agent faces a sequence of loss functions $\{\mathcal{L}_t\}_{t=1}^{\infty}$, one in each round t. These functions need not be drawn from a fixed distribution, and could even be chosen adversarially over time. The goal for the learner is to sequentially decide on model parameters $\{\boldsymbol{\phi}_t\}_{t=1}^{\infty}$ that perform well on the loss sequence. In particular, the standard objective is to minimize some notion of regret defined as the difference between our learner's loss, $\sum_{t=1}^{T} \mathcal{L}_t(\boldsymbol{\phi}_t)$, and the best performance achievable by some family of methods (comparator class). The most standard notion of regret is to compare to the cumulative loss of the best *fixed* model in hindsight:

$$\text{Regret}_T = \sum_{t=1}^{T} \mathcal{L}_t(\boldsymbol{\phi}_t) - \min_{\boldsymbol{\phi}} \sum_{t=1}^{T} \mathcal{L}_t(\boldsymbol{\phi}). \tag{6.2}$$

The goal in online learning is to design algorithms such that this regret grows with T as slowly as possible. In particular, an agent (algorithm) whose regret grows sub-linearly in T is non-trivially learning and adapting. One of the simplest algorithms in this setting is follow the leader (FTL) [237], which updates the parameters as:

$$\boldsymbol{\phi}_{t+1} = \arg\min_{\boldsymbol{\phi}} \sum_{k=1}^{t} \mathcal{L}_k(\boldsymbol{\phi}).$$

FTL enjoys strong performance guarantees depending on the properties of the loss function, and some variants use additional regularization to improve stability [164]. For the few-shot supervised learning example, FTL would consolidate all the data from the prior stream of tasks into a single large dataset and fit a single model to this dataset. As alluded to in Section 6.1, and as we find in our empirical evaluation in Section 6.6, such a "joint training" approach may not learn effective models. To overcome this issue, we may desire a more adaptive notion of a comparator class, and algorithms that have low regret against such a comparator, as we will discuss next.

6.3 The Online Meta-Learning Problem

We consider a general sequential setting where an agent is faced with tasks one after another. Each of these tasks correspond to a *round*, denoted by t. In each round, the goal of the learner is to determine model parameters $\boldsymbol{\phi}_t$ that perform well for the corresponding task at that round. This is monitored by $\mathcal{L}_t : \boldsymbol{\phi} \in \mathcal{W} \to \mathbb{R}$, which we would like to be minimized. Crucially, we consider a setting where the agent can perform some local *task-specific* updates to the model before it is deployed and evaluated in each round. This is realized through an update procedure, which at every round t is a mapping $\mathcal{A}lg_t : \boldsymbol{\phi} \in \mathcal{W} \to \boldsymbol{\phi} \in \mathcal{W}$. This procedure takes as input $\boldsymbol{\phi}$ and returns $\boldsymbol{\phi}$ that performs better on $\mathcal{L}_t$. One example for $\mathcal{A}lg_t$ is a step of gradient descent [191]:

$$\mathcal{A}lg_t(\boldsymbol{\phi}) = \boldsymbol{\phi} - \alpha \nabla \hat{\mathcal{L}}_t(\boldsymbol{\phi}).$$

Here, as specified in Section 6.2, $\nabla \hat{\mathcal{L}}_t$ is potentially an approximate gradient of $\mathcal{L}_t$, as for example obtained using a mini-batch of data from the task at round t. The overall protocol for the setting is as follows:

1. At round t, the agent chooses a model defined by $\boldsymbol{\phi}_t$.
2. The world simultaneously chooses task defined by f_t.
3. The agent obtains access to the update procedure $\mathcal{A}lg_t$, and uses it to update parameters as $\boldsymbol{\phi}_t = \mathcal{A}lg_t(\boldsymbol{\phi}_t)$.
4. The agent incurs loss $\mathcal{L}_t(\boldsymbol{\phi}_t)$. Advance to round $t + 1$.

The goal for the agent is to minimize regret over the rounds. A highly ambitious comparator is the best meta-learned model in hindsight. Let $\{\boldsymbol{\phi}_t\}_{t=1}^{T}$ be the sequence of models generated by the algorithm. Then, the regret we consider is:

$$\text{Regret}_T = \sum_{t=1}^{T} \mathcal{L}_t\big(\mathcal{A}lg_t(\boldsymbol{\phi}_t)\big) - \min_{\boldsymbol{\phi}} \sum_{t=1}^{T} \mathcal{L}_t\big(\mathcal{A}lg_t(\boldsymbol{\phi})\big). \tag{6.3}$$

Notice that we allow the comparator to adapt locally to each task at hand; thus the comparator has strictly more capabilities than the learning agent, since it is presented with all the task functions in batch mode. Here, again, achieving sublinear regret suggests that the agent is improving over time and is competitive with the best meta-learner in hindsight. As discussed earlier, in the batch setting, when faced with multiple tasks, meta-learning performs significantly better than training a single model for all the tasks. Thus, we may hope that learning sequentially, but still being competitive with the best meta-learner in hindsight, provides a significant leap in continual learning.

6.4 Algorithm and Analysis

In this section, we outline the *follow the meta leader* (FTML) algorithm and provide an analysis of its behavior.

6.4.1 Follow the Meta Leader

One of the most successful algorithms in online learning is follow the leader [237, 238], which enjoys strong performance guarantees for smooth and convex functions. Taking inspiration from its form, we propose the FTML algorithm template:

$$\boldsymbol{\phi}_{t+1} = \arg\min_{\boldsymbol{\phi}} \sum_{k=1}^{t} \mathcal{L}_k\big(\mathcal{A}lg_k(\boldsymbol{\phi})\big). \tag{6.4}$$

This can be interpreted as the agent playing the best meta-learner in hindsight if the learning process were to stop at round t. In the remainder of this section, we will show that under standard assumptions on the losses, and just one additional assumption on higher order smoothness, this algorithm has strong regret guarantees. In practice, we may not have full access to $\mathcal{L}_k(\cdot)$, such as when it is the population risk and we only have a finite size dataset. In such cases, we will draw upon stochastic approximation algorithms to solve the optimization problem in Eq. (6.4).

6.4.2 Assumptions

We make the following assumptions about each loss function in the learning problem for all t. Let $\mathbf{u}$ and $\mathbf{v}$ represent two arbitrary choices of *model parameters*.

Assumption A1. *(C^2-smoothness)*

1. *(Lipschitz in function value) $\mathcal{L}$ has gradients bounded by G, i.e. $||\nabla\mathcal{L}(\mathbf{u})|| \leq G \ \forall \ \mathbf{u}$. This is equivalent to $\mathcal{L}$ being $G-$Lipschitz.*
2. *(Lipschitz gradient) $\mathcal{L}$ is $\beta-$smooth, i.e.* $||\nabla\mathcal{L}(\mathbf{u}) - \nabla\mathcal{L}(\mathbf{v})|| \leq \beta||\mathbf{u}-\mathbf{v}|| \ \forall(\mathbf{u},\mathbf{v})$.
3. *(Lipschitz Hessian) $\mathcal{L}$ has $\rho-$Lipschitz Hessians, i.e.* $||\nabla^2\mathcal{L}(\mathbf{u}) - \nabla^2\mathcal{L}(\mathbf{v})|| \leq \rho||\mathbf{u}-\mathbf{v}|| \ \forall(\mathbf{u},\mathbf{v})$.

Assumption A2. *(Strong convexity) Suppose that $\mathcal{L}$ is convex. Furthermore, suppose $\mathcal{L}$ is $\mu-$strongly convex, i.e. $||\nabla\mathcal{L}(\mathbf{u}) - \nabla\mathcal{L}(\mathbf{v})|| \geq \mu||\mathbf{u}-\mathbf{v}||$.*

These assumptions are largely standard in online learning, in various settings [157], except 1.3. Examples where these assumptions hold include logistic regression and $L2$

regression over a bounded domain. Assumption 1.3 is a statement about the higher order smoothness of functions which is common in non-convex analysis [207, 206]. In our setting, it allows us to characterize the landscape of the MAML-like function which has a gradient update step embedded within it. Importantly, these assumptions *do not* trivialize the meta-learning setting. A clear difference in performance between meta-learning and joint training can be observed even in the case where $\mathcal{L}(\cdot)$ are quadratic functions, which correspond to the simplest strongly convex setting. See Appendix E.1 for an example illustration.

6.4.3 Analysis

We analyze the FTML algorithm when the update procedure is a single step of gradient descent, as in the formulation of MAML. Concretely, the update procedure we consider is $\mathcal{A}lg_t(\boldsymbol{\phi}) = \boldsymbol{\phi} - \alpha \nabla \hat{\mathcal{L}}_t(\boldsymbol{\phi})$. For this update rule, we first state our main theorem below.

Theorem 4. *Suppose $\mathcal{L}$ and $\hat{\mathcal{L}} : \mathbb{R}^d \to \mathbb{R}$ satisfy assumptions 1 and 2. Let $\tilde{\mathcal{L}}$ be the function evaluated after a one step gradient update procedure, i.e.*

$$\tilde{\mathcal{L}}(\boldsymbol{\phi}) := \mathcal{L}\big(\boldsymbol{\phi} - \alpha \nabla \hat{\mathcal{L}}(\boldsymbol{\phi})\big).$$

If the step size is selected as $\alpha \leq \min\{\frac{1}{2\beta}, \frac{\mu}{8\rho G}\}$, then $\tilde{\mathcal{L}}$ is convex. Furthermore, it is also $\tilde{\beta} = 9\beta/8$ smooth and $\tilde{\mu} = \mu/8$ strongly convex.

Corollary 3. *(inherited convexity for the MAML objective) If $\{\mathcal{L}_i, \hat{\mathcal{L}}_i\}_{i=1}^K$ satisfy assumptions 1 and 2, then the MAML optimization problem,*

$$\underset{\boldsymbol{\phi}}{\textit{minimize}} \; \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}_i\big(\boldsymbol{\phi} - \alpha \nabla \hat{\mathcal{L}}_i(\boldsymbol{\phi})\big),$$

with $\alpha \leq \min\{\frac{1}{2\beta}, \frac{\mu}{8\rho G}\}$ is convex. Furthermore, it is $9\beta/8$-smooth and $\mu/8$-strongly convex.

The proofs are provided in the appendix. Since the objective function is convex, we may expect first-order optimization methods to be effective, since gradients can be efficiently computed with standard automatic differentiation libraries (as discussed in Finn et al. [191]). In fact, this work was chronologically the first to show that MAML-like objective functions can be provably and efficiently optimized under any set of assumptions. Another immediate corollary of our main theorem is that FTML now enjoys the same regret guarantees (up to constant factors) as FTL does in the comparable setting (with strongly convex losses).

Corollary 4. *(inherited regret bound for FTML) Suppose that for all t, $\mathcal{L}_t$ and $\hat{\mathcal{L}}_t$ satisfy assumptions 1 and 2. Suppose that the update procedure in FTML (Eq. 6.4) is chosen as $\mathcal{A}lg_t(\boldsymbol{\phi}) = \boldsymbol{\phi} - \alpha \nabla \hat{\mathcal{L}}_t(\boldsymbol{\phi})$ with $\alpha \leq \min\{\frac{1}{2\beta}, \frac{\mu}{8\rho G}\}$. Then, FTML enjoys the following regret guarantee*

$$\sum_{t=1}^{T} \mathcal{L}_t\big(\mathcal{A}lg_t(\boldsymbol{\phi}_t)\big) - \min_{\boldsymbol{\phi}} \sum_{t=1}^{T} \mathcal{L}_t\big(\mathcal{A}lg_t(\boldsymbol{\phi})\big) = \mathcal{O}\left(\frac{32G^2}{\mu} \log T\right)$$

Proof. From Theorem 4, we have that each function $\tilde{\mathcal{L}}_t(\boldsymbol{\phi}) = \mathcal{L}_t(\mathcal{A}lg_t(\boldsymbol{\phi}))$ is $\tilde{\mu} = \mu/8$ strongly convex. The FTML algorithm is identical to FTL on the sequence of loss functions $\{\tilde{\mathcal{L}}_t\}_{t=1}^T$, which has a $\mathcal{O}(\frac{4G^2}{\tilde{\mu}} \log T)$ regret guarantee (see Cesa-Bianchi and Lugosi [157] Theorem 3.1). Using $\tilde{\mu} = \mu/8$ completes the proof. □

More generally, our main theorem implies that there exists a large family of online meta-learning algorithms that enjoy sub-linear regret, based on the inherited smoothness and strong convexity of $\tilde{\mathcal{L}}(\cdot)$. See Hazan [239], Shalev-Shwartz [164], Shalev-Shwartz and Kakade [240] for algorithmic templates to derive sub-linear regret based algorithms.

6.5 Practical Online Meta-Learning Algorithm

In the previous section, we derived a theoretically principled algorithm for convex losses. However, many problems of interest in machine learning and deep learning have a non-convex loss landscape, where theoretical analysis is challenging. Nevertheless, algorithms originally developed for convex losses like gradient descent and AdaGrad [241] have shown promising results in practical non-convex settings. Taking inspiration from these successes, we describe a practical instantiation of FTML in this section, and empirically evaluate its performance in Section 6.6.

The main considerations when adapting the FTML algorithm to few-shot supervised learning with high capacity neural network models are: (a) the optimization problem in Eq. (6.4) has no closed form solution, and (b) we do not have access to the population risk $\mathcal{L}_t$ but only a subset of the data. To overcome both these limitations, we can use iterative stochastic optimization algorithms. Specifically, by adapting the MAML algorithm [191], we can use stochastic gradient descent with a minibatch $\mathcal{D}_k^{\text{tr}}$ as the update rule, and stochastic gradient descent with an independently-sampled minibatch $\mathcal{D}_k^{\text{val}}$ to optimize the parameters. The gradient computation is described below:

$$\begin{aligned} \mathbf{g}_t(\boldsymbol{\phi}) = \nabla_{\boldsymbol{\phi}}\, \mathbb{E}_{k\sim\nu^t} \mathcal{L}\big(\mathcal{D}_k^{\text{val}}, \mathcal{A}lg_k(\boldsymbol{\phi})\big), \quad \text{where} \\ \mathcal{A}lg_k(\boldsymbol{\phi}) \equiv \boldsymbol{\phi} - \alpha\, \nabla_{\boldsymbol{\phi}}\, \mathcal{L}\big(\mathcal{D}_k^{\text{tr}}, \boldsymbol{\phi}\big) \end{aligned} \tag{6.5}$$

Here, $\nu^t(\cdot)$ denotes a sampling distribution for the previously seen tasks $\mathcal{T}_1, ..., \mathcal{T}_t$. In our experiments, we use the uniform distribution, $\nu^t \equiv P(k) = 1/t\ \forall k = \{1, 2, \ldots t\}$, but other sampling distributions can be used if required. Recall that $\mathcal{L}(\mathcal{D}, \boldsymbol{\phi})$ is the loss function (e.g. cross-entropy) averaged over the datapoints $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ for the model with parameters $\boldsymbol{\phi}$. Using independently sampled minibatches $\mathcal{D}^{\text{tr}}$ and $\mathcal{D}^{\text{val}}$ minimizes interaction between the inner gradient update $\mathcal{A}lg_t$ and the outer optimization (Eq. 6.4), which is performed using the gradient above ($\mathbf{g}_t$) in conjunction with Adam [203]. While $\mathcal{A}lg_t$ in Eq. 6.5 includes only one gradient step, we observed that it is beneficial to take multiple gradient steps in the inner loop (i.e., in $\mathcal{A}lg_t$), which is consistent with prior works [191, 202, 242, 208].

Now that we have derived the gradient, the overall algorithm proceeds as follows. We first initialize a task buffer $\mathcal{B} = [\ ]$. When presented with a new task at round t, we add task $\mathcal{T}_t$ to $\mathcal{B}$ and initialize a task-specific dataset $\mathcal{D}_t = [\ ]$, which is appended to as data incrementally arrives for task $\mathcal{T}_t$. As new data arrives for task $\mathcal{T}_t$, we iteratively compute and apply the gradient in Eq. 6.5, which uses data from all tasks seen so far. Once all of the data (finite-size) has arrived for $\mathcal{T}_t$, we move on to task $\mathcal{T}_{t+1}$. This procedure is further described in Algorithm 7, including the evaluation, which we discuss next.

Algorithm 7 Online Meta-Learning with FTML

```
1: **Input:** Performance threshold of proficiency, $\gamma$
2: randomly initialize $\boldsymbol{\phi}_1$
3: initialize the task buffer as empty, $\mathcal{B} \leftarrow [\ ]$
4: **for** $t = 1, \ldots$ **do**
5:     initialize $\mathcal{D}_t = \emptyset$
6:     Add $\mathcal{B} \leftarrow \mathcal{B} + [\ \mathcal{T}_t\ ]$
7:     **while** $|\mathcal{D}_{\mathcal{T}_t}| < N$ **do**
8:         Append batch of $n$ new datapoints $\{(\mathbf{x}, \mathbf{y})\}$ to $\mathcal{D}_t$
9:         $\boldsymbol{\phi}_t \leftarrow$ Meta-Update$(\boldsymbol{\phi}_t, \mathcal{B}, t)$
10:        $\boldsymbol{\phi}_t \leftarrow$ Update-Procedure $(\boldsymbol{\phi}_t, \mathcal{D}_t)$
11:        **if** $\mathcal{L}(\mathcal{D}_t^{\text{test}}, \boldsymbol{\phi}_t) < \gamma$ **then**
12:            Record efficiency for task $\mathcal{T}_t$ as $|\mathcal{D}_t|$ datapoints
13:        **end if**
14:    **end while**
15:    Record final performance of $\boldsymbol{\phi}_t$ on test set $\mathcal{D}_t^{\text{test}}$ for task $t$.
16:    $\boldsymbol{\phi}_{t+1} \leftarrow \boldsymbol{\phi}_t$
17: **end for**
```

Algorithm 8 FTML Subroutine: `Meta-Update`$(\boldsymbol{\phi}, \mathcal{B}, t)$

```
1: **Hyperparameters:** $\alpha, \eta$, $N_{\text{meta}}$, $N_{\text{grad}}$
2: **for** $n_{\text{m}} = 1, \ldots, N_{\text{meta}}$ steps **do**
3:     Sample task $\mathcal{T}_k$: $k \sim \nu^t(\cdot)$   // (or a minibatch of tasks)
4:     Sample minibatches $\mathcal{D}_k^{\text{tr}}$, $\mathcal{D}_k^{\text{val}}$ uniformly from $\mathcal{D}_k$
5:     Compute gradient $\mathbf{g}_t$ using $\mathcal{D}_k^{\text{tr}}$, $\mathcal{D}_k^{\text{val}}$, and Eq. 6.5
6:     Update parameters $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \eta\ \mathbf{g}_t$   // (or use Adam)
7: **end for**
8: Return $\boldsymbol{\phi}$
```

Algorithm 9 FTML Subroutine: `Update-Procedure`($\boldsymbol{\phi}$, $\mathcal{D}$)

```
1: Hyperparameters: $\alpha, \eta$, $N_{\text{meta}}$, $N_{\text{grad}}$
2: Initialize $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi}$
3: for $n_{\text{g}} = 1, \ldots, N_{\text{grad}}$ steps do
4:     $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \alpha \nabla \mathcal{L}(\mathcal{D}, \boldsymbol{\phi})$
5: end for
6: Return $\boldsymbol{\phi}$
```

To evaluate performance of the model at any point within a particular round t, we first update the model as using all of the data ($\mathcal{D}_t$) seen so far within round t. This is outlined in the `Update-Procedure` subroutine of Algorithm 8. Note that this is different from the update $\mathbf{U}_t$ used within the meta-optimization, which uses a fixed-size minibatch since many-shot meta-learning is computationally expensive and memory intensive. In practice, we meta-train with update minibatches of size at-most 25, whereas evaluation may use hundreds of datapoints for some tasks. After the model is updated, we measure the performance using held-out data $\mathcal{D}_t^{\text{test}}$ from task $\mathcal{T}_t$. This data is not revealed to the online meta-learner at any time. Further, we also evaluate task learning efficiency, which corresponds to the size of $\mathcal{D}_t$ required to achieve a specified performance threshold γ, e.g. $\gamma = 90\%$ classification accuracy or γ corresponds to a certain loss value. If less data is sufficient to reach the threshold, then priors learned from previous tasks are being useful and we have achieved positive transfer.

6.6 Experimental Evaluation

Our experimental evaluation studies the practical FTML algorithm (Section 6.5) in the context of vision-based online learning problems. These problems include synthetic modifications of the MNIST dataset, pose detection with synthetic images based on PASCAL3D+ models [243], and realistic online image classification experiments with the CIFAR-100 dataset. The aim of our experimental evaluation is to study the following questions: (1) can online meta-learning (and specifically FTML) be successfully applied to multiple non-stationary learning problems? and (2) does online meta-learning (FTML) provide empirical benefits over prior methods?

To this end, we compare to the following algorithms: (a) Train on everything (TOE) trains on all available data so far (including $\mathcal{D}_t$ at round t) and trains a single predictive model. This model is directly tested without any specific adaptation since it has already been trained on $\mathcal{D}_t$. (b) Train from scratch, which initializes $\boldsymbol{\phi}_t$ randomly, and finetunes it using $\mathcal{D}_t$. (c) Joint training with fine-tuning, which at round t, trains on all the data jointly till round $t-1$, and then finetunes it specifically to round t using only $\mathcal{D}_t$. This corresponds to the standard online learning approach where FTL is used (without any meta-learning objective), followed by task-specific fine-tuning.

We note that TOE is a very strong point of comparison, capable of reusing representations across tasks, as has been proposed in a number of prior continual learning works [244, 245, 246].

However, unlike FTML, TOE does not explicitly learn the structure across tasks. Thus, it may not be able to fully utilize the information present in the data, and will likely not be able to learn new tasks with only a few examples. Further, the model might incur negative transfer if the new task differs substantially from previously seen ones, as has been observed in prior work [247]. Training on each task independently from scratch avoids negative transfer, but also precludes any reuse between tasks. When the amount of data for a given task is large, we may expect training from scratch to perform well since it can avoid negative transfer and can learn specifically for the particular task. Finally, FTL with fine-tuning represents a natural online learning comparison, which in principle should combine the best parts of learning from scratch and TOE, since this approach adapts specifically to each task *and* benefits from prior data. However, in contrast to FTML, this method does not explicitly meta-learn and hence may not fully utilize any structure in the tasks.

6.6.1 Rainbow MNIST

In this experiment, we create a sequence of tasks based on the MNIST character recognition dataset. We transform the digits in a number of ways to create different tasks, such as 7 different colored backgrounds, 2 scales (half size and original size), and 4 rotations of 90 degree intervals. As illustrated in Figure 6.2, a task involves correctly classifying digits with a randomly sampled background, scale, and rotation. This leads to 56 total tasks. We partitioned the MNIST training dataset into 56 batches of examples, each with 900 images and applied the corresponding task transformation to each batch of images. The ordering of tasks was selected at random and we set 90% classification accuracy as the proficiency threshold. The learning curves in Figure 6.3 show that FTML learns tasks more and more quickly, with each new task added. We also observe that FTML substantially outperforms

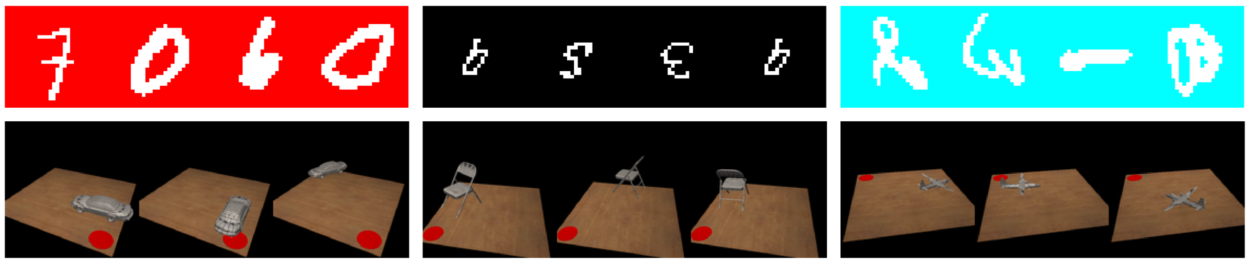


Figure 6.2: Illustration of three tasks for Rainbow MNIST (top) and pose prediction (bottom). CIFAR images not shown. Rainbow MNIST includes different rotations, scaling factors, and background colors. For the pose prediction tasks, the goal is to predict the global position and orientation of the object on the table. Cross-task variation includes varying 50 different object models within 9 object classes, varying object scales, and different camera viewpoints.

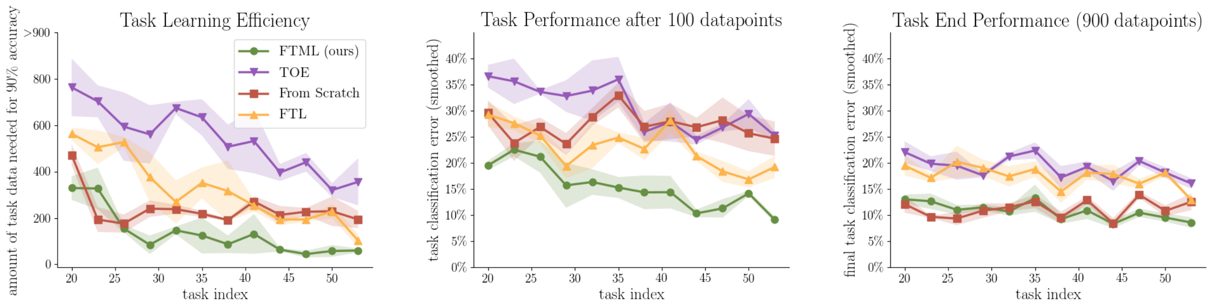


Figure 6.3: Rainbow MNIST results. Left: amount of data needed to learn each new task. Center: task performance after 100 datapoints on the current task. Right: The task performance after all 900 datapoints for the current task have been received. Lower is better for all plots, while shaded regions show standard error computed using three random seeds. FTML can learn new tasks more and more efficiently as each new task is received, demonstrating effective forward transfer.

the alternative approaches in both efficiency and final performance. FTL performance better than TOE since it performs task-specific adaptation, but its performance is still inferior to FTML. We hypothesize that, while the prior methods improve in efficiency over the course of learning as they see more tasks, they struggle to prevent negative transfer on each new task. Our last observation is that training independent models does not learn efficiently, compared to models that incorporate data from other tasks; but, their final performance with 900 data points is similar.

6.6.2 Five-Way CIFAR-100

In this experiment, we create a sequence of 5-way classification tasks based on the CIFAR-100 dataset, which contains more challenging and realistic RGB images than MNIST. Each classification problem involves a newly-introduced class from the 100 classes in CIFAR-100. Thus, different tasks correspond to different labels spaces. The ordering of tasks is selected at random, and we measure performance using classification accuracy. Since it is less clear what the proficiency threshold should be for this task, we evaluate the accuracy on each task after varying numbers of datapoints have been seen. Since these tasks are mutually exclusive (as label space is changing), it makes sense to train the TOE model with a different final layer for each task. An extremely similar approach to this is to use our meta-learning approach but to only allow the final layer parameters to be adapted to each task. Further, such a meta-learning approach is a more direct comparison to our full FTML method, and the comparison can provide insight into whether online meta-learning is simply learning features and performing training on the last layer, or if it is adapting the features to each task. Thus, we compare to this last layer online meta-learning approach instead of TOE

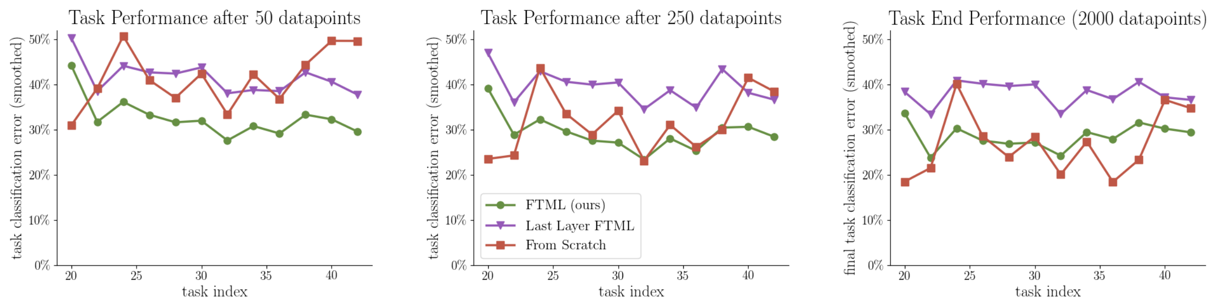


Figure 6.4: Online CIFAR-100 results, evaluating task performance after 50, 250, and 2000 datapoints have been received for a given task. We see that FTML learns each task much more efficiently than models trained from scratch, while both achieve similar asymptotic performance after 2000 datapoints. We also observe that FTML benefits from adapting all layers rather than learning a shared feature space across tasks while adapting only the last layer.

with multiple heads. The results (see Figure 6.4) indicate that FTML learns more efficiently than independent models and a model with a shared feature space. The results on the right indicate that training from scratch achieves good performance with 2000 datapoints, reaching similar performance to FTML. However, the last layer variant of FTML seems to not have the capacity to reach good performance on all tasks.

6.6.3 Sequential Object Pose Prediction

In our final experiment, we study a 3D pose prediction problem. Each task involves learning to predict the global position and orientation of an object in an image. We construct a dataset of synthetic images using 50 object models from 9 different object classes in the PASCAL3D+ dataset [243], rendering the objects on a table using the renderer accompanying the MuJoCo [30] (see Figure 6.2). To place an object on the table, we select a random 2D location, as well as a random azimuthal angle. Each task corresponds to a different object with a randomly sampled camera angle. We place a red dot on one corner of the table to provide a global reference point for the position. Using this setup, we construct 90 tasks (with an average of about 2 camera viewpoints per object), with 1000 datapoints per task. All models are trained to regress to the global 2D position and the sine and cosine of the azimuthal angle (the angle of rotation along the z-axis). For the loss functions, we use mean-squared error, and set the proficiency threshold to an error of 0.05. We show the results of this experiment in Figure 6.5. The results demonstrate that meta-learning can improve both efficiency and performance of new tasks over the course of learning, solving many of the tasks with only 10 datapoints. Unlike the previous settings, TOE substantially outperforms training from scratch, indicating that it can effectively make use of the previous data from other tasks, likely due to the greater structural similarity between the pose detection tasks. However,

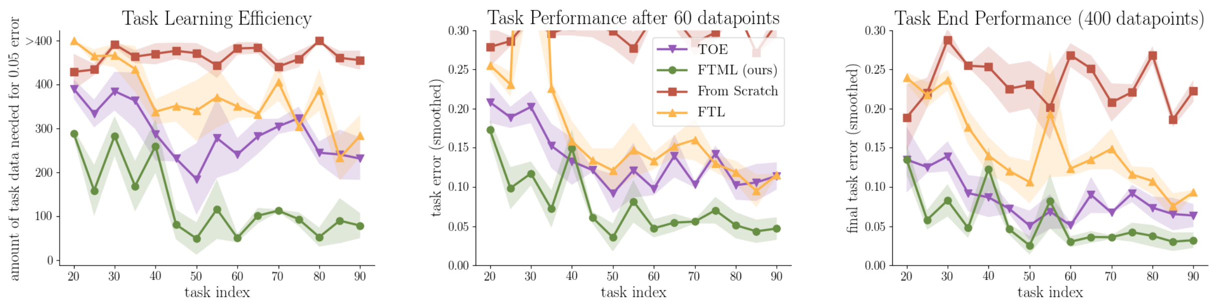


Figure 6.5: Object pose prediction results. Left: we observe that online meta-learning generally leads to faster learning as more and more tasks are introduced, learning with only 10 datapoints for many of the tasks. Center & right, we see that meta-learning enables transfer not just for faster learning but also for more effective performance when 60 and 400 datapoints of each task are available. The order of tasks is randomized, leading to spikes when more difficult tasks are introduced. Shaded regions show standard error across three random seeds

the superior performance of FTML suggests that even better transfer can be accomplished through meta-learning. Finally, we find that FTL performs comparably or worse than TOE, indicating that task-specific fine-tuning can lead to overfitting when the model is not explicitly trained for the ability to be fine-tuned effectively.

6.7 Connections to Related Work

Our work proposes to use meta-learning or learning to learn [187, 186, 188], in the context of online (regret-based) learning. We reviewed the foundations of these approaches in Section 6.2, and we summarize additional related work along different axis.

Meta-learning: Prior works have proposed learning update rules, selective copying of weights, or optimizers for fast adaptation [192, 248, 215, 216, 249, 250, 251], as well as recurrent models that learn by ingesting datasets directly [189, 217, 218, 219, 252]. While some meta-learning works have considered online learning settings at *meta-test time* [189, 199, 253], nearly all prior meta-learning algorithms assume that the *meta-training tasks* come from a stationary distribution. Furthermore, most prior work has not evaluated versions of meta-learning algorithms when presented with a continuous stream of tasks. One exception is work that adapts hyperparameters online [254, 255, 256]. In contrast, we consider a more flexible approach that allows for adapting all of the parameters in the model for each task. More recent work has considered handling non-stationary task distributions in meta-learning using Dirichlet process mixture models over meta-learned parameters [257]. Unlike this prior work, we introduce a simple extension onto the MAML algorithm without mixtures over parameters, and provide theoretical guarantees.

Continual learning: Our problem setting is related to (but distinct from) continual, or lifelong learning [258, 259]. In lifelong learning, a number of recent papers have focused on avoiding forgetting and negative backward transfer [260, 261, 262, 263, 264, 265, 266, 267, 268]. Other papers have focused on maintaining a reasonable model capacity as new tasks are added [269, 270]. In this paper, we sidestep the problem of catastrophic forgetting by maintaining a buffer of all the observed data [271]. In future work, we hope to understand the interplay between limited memory and catastrophic forgetting for variants of the FTML algorithm. Here, we instead focuses on the problem of forward transfer: maximizing the efficiency of learning new tasks within a non-stationary learning setting. Prior works have also considered settings that combine joint training across tasks (in a sequential setting) with task-specific adaptation [272, 127], but have not explicitly employed meta-learning. Furthermore, unlike prior works [273, 244, 245, 246], we also focus on the setting where there are several tens or hundreds of tasks. This setting is interesting since there is significantly more information that can be transferred from previous tasks and we can employ more sophisticated techniques such as meta-learning for transfer, enabling the agent to move towards few-shot learning after experiencing a large number of tasks.

Online learning: Similar to continual learning, online learning deals with a sequential setting with streaming tasks. It is well known in online learning that FTL is computationally expensive, with a large body of work dedicated to developing cheaper algorithms [157, 274, 275, 164]. Again, in this work, we sidestep the computational considerations to first study if the meta-learning analog of FTL can provide performance gains. For this, we derived the FTML algorithm which has low regret when compared to a powerful adaptive comparator class that performs task-specific adaptation. We leave the design of more computationally efficient versions of FTML to future work.

To avoid the pitfalls associated with a single best model in hindsight, online learning literature has also studied alternate notions of regret, with the closest settings being dynamic regret and adaptive or tracking regret. In the dynamic regret setting [276, 277, 278], the performance of the online learner's model sequence is compared against the sequence of optimal solutions corresponding to each loss function in the sequence. Unfortunately, lower-bounds [277] suggest that the comparator class is too powerful and may not provide for any non-trivial learning in the general case. To overcome these limitations, prior work has placed restrictions on how quickly the loss functions or the comparator model can change [279, 280, 276]. In contrast, we consider a different notion of adaptive regret, where the learner and comparator both have access to an update procedure. The update procedures allow the comparator to produce different models for different loss functions, thereby serving as a powerful comparator class (in comparison to a fixed model in hindsight). For this setting, we derived sublinear regret algorithms without placing any restrictions on the sequence of loss functions. Recent and concurrent works have also studied algorithms related to MAML and its first order variants using theoretical tools from online learning literature [281, 282, 283].

These works also derive regret and generalization bounds, but these algorithms have not yet been empirically studied in large scale domains or in non-stationary settings. We believe that our online meta-learning setting captures the spirit and practice of continual lifelong learning, and also shows promising empirical results.

6.8 Discussions and Conclusion

In this paper, we introduced the online meta-learning problem statement, with the aim of connecting the fields of meta-learning and online learning. Online meta-learning provides, in some sense, a more natural perspective on the ideal real-world learning procedure: an intelligent agent interacting with a constantly changing environment should utilize streaming experience to both master the task at hand, and become more proficient at learning new tasks in the future. We analyzed the proposed FTML algorithm and showed that it achieves logarithmic regret. We then illustrated how FTML can be adapted to a practical algorithm. Our experimental evaluations demonstrated that the proposed algorithm outperforms prior methods. In the rest of this section, we reiterate a few salient features of the online meta learning setting (Section 6.3), and outline avenues for future work.

More powerful update procedures. In this work, we concentrated our analysis on the case where the update procedure $\mathcal{A}lg_t$, inspired by MAML, corresponds to one step of gradient descent. However, in practice, many works with MAML (including our experimental evaluation) use multiple gradient steps in the update procedure, and back-propagate through the entire path taken by these multiple gradient steps. Analyzing this case, and potentially higher order update rules will also make for exciting future work.

Memory and computational constraints. In this work, we primarily aimed to discern if it is possible to meta-learn in a sequential setting. For this purpose, we proposed the FTML template algorithm which draws inspiration from FTL in online learning. As discussed in Section 6.7, it is well known that FTL has poor computational properties, since the computational cost of FTL grows over time as new loss functions are accumulated. Further, in many practical online learning problems, it is challenging (and sometimes impossible) to store all datapoints from previous tasks. While we showed that our method can effectively learn nearly 100 tasks in sequence without significant burdens on compute or memory, scalability remains a concern. Can a more streaming algorithm like mirror descent that does not store all the past experiences be successful as well? Our main theoretical results suggests that there exist a large family of online meta-learning algorithms that enjoy sublinear regret. Tapping into the large body of work in online learning, particularly mirror descent, to develop computationally cheaper algorithms would make for exciting future work.

Chapter 7
CONCLUSION

In this thesis we developed abstractions and algorithms to enable domain transfer – the transfer of models, inductive biases, and representations – from one or more source domains to a desired target domain. Successful domain transfer would allow agents to be either directly proficient in the target domain without any domain-specific learning (zero-shot transfer), or be capable of quickly learning in the target domain with minimal data (few-shot adaptation). Such capabilities would likely allow AI systems to move beyond current success in narrowly defined tasks, to be broadly competent in unstructured real-world settings.

In Chapter 2, we studied domain transfer in the context of simulation to reality transfer in deep RL. For this case, we drew upon the principle of risk-aversion to develop the EPOpt algorithm, which learns to successful transfer policies from the simulated source domain to the target domain. In Chapter 2.6, we presented a case study that demonstrates the success of such an approach on a physical robot.

In Chapter 3, we studied offline RL where an agent must learn effective control policies using a fixed static dataset. We outlined the major challenges in this paradigm including the distribution/domain shift between the training data distribution and the induced distribution of the trained policy. We introduced the concept of pessimism or conservatism in the face of uncertainty, and used this to derive MOReL, a model-based offline RL algorithm. We showed that MOReL enjoys strong theoretical guarentees, including minimax optimality, and also achieves state of the art experiment results in offline RL benchmarks. In Chapter 3.8, we extend this approach to offline RL from high dimensional observations like pixels. This is especially challenging since it simultaneously requires representation learning and uncertainty quantification to impart the necessary conservatism. We addressed all of these challenges through the use of variational latent state-space dynamics models which provide a rich auxiliary task for representation learning as well as the ability to perform model rollouts and uncertainty quantification in the compact latent space of the model.

In Chapter 4, we studied model-based RL in the more standard interactive RL paradigm. We illustrated how domain/distribution shift may manifest even in this well studied setting, due to simultaneous policy and model learning that can introduce bias and nonstationarity. By drawing upon insights from game theory, we developed stable algorithms that mitigate against the impact of distribution shift. This enabled the development of model-based RL algorithms that can: (a) be highly sample efficient; (b) match the asymptotic performance of model-free policy gradient; (c) scale gracefully to high dimensional tasks like dexterous manipulation; and (d) handle extended rollout horizons of several hundred timesteps.

In Chapter 5, we moved from transferring from a single source domain to a collection of source domains, by considering the setting of meta-learning. This involves the learning of inductive biases and adaptable representations for accelerating the learning of a new task. We demonstrated how meta-learning can be formalized as a bi-level optimization problem, and developed the implicit MAML algorithm. We showed that implicit MAML is provably convergent, provably compute and memory efficient compared to prior approaches, and demonstrates strong empirical gains over baselines in benchmark tasks.

Finally, in Chapter 6, we studied meta-learning in the continual and non-stationary setting. This allows the agent to continuously accelerate the learning process in new tasks as it encounters more and more tasks in a sequence. We established a new regret metric that is suitable for this setting, and showed that the algorithm we develop (FTML) enjoys a no-regret property. We validated this algorithm on few-shot vision domains, and showed that FTML can enable CNNs to recognize and predict pose of new objects more efficiently than conventional online learning approaches.

While this work presents first steps towards algorithm development for domain transfer in a variety of scenarios, a number of additional steps are required before we can realize the ideal of broadly competent and generalizable AI systems. We outline a few directions for future work below.

Learning multiple skills from offline datasets In this thesis, we showed how model-based offline RL algorithms like MOReL can learn policies for downstream tasks different from those encountered in the offline dataset. However, when we are interested in multiple downstream tasks, this approach of drawing upon the dataset every-time for policy learning can be computationally expensive, repetitive, and may not exhibit the best generalization. One potential way to overcome this is to compress the offline dataset into a library of skills that can be readily drawn upon and re-purposed for a multitude of downstream tasks. While there have been early explorations in this direction [284, 285, 286, 287], development of more capable algorithms and model-based methods may allow for extraction of more diverse skills and better ways to utilize them.

Larger offline datasets and high-capacity models The fields of computer vision and NLP have seen the success of high-capacity self-supervised models like SimCLR, BERT, and GPT, when trained on large and diverse datasets. In the future, it is clear that we will move towards larger and more diverse datasets for offline RL as well. For such high-data regimes, how can we develop offline RL algorithms that scale gracefully with data and compute? Our recent work on decision transformers [288] takes first steps towards this question by demonstrating successful results with the transformer architecture for offline RL benchmarks. Explorations along similar directions would make for exciting future work.

Scaling up meta-learning Similar to above, research in meta-learning has also primarily focused on algorithmic development in the past half decade. As the algorithms mature and we work with larger datasets for real-world applications, the scaling up of meta-learning to larger datasets may open the doors for adaptable representations that can support the learning of thousands of tasks in both perception and control. The study of meta-learning at scale and understanding if qualitatively different representations emerge when training at scale would make for important future work.

BIBLIOGRAPHY

[1] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press, 2016.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6), 2017.

[3] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel rahman Mohamed, Navdeep Jaitly, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath, and Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition, November 2012.

[4] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[5] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *CoRR*, abs/1802.05365, 2018.

[6] Yonghui Wu et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *ArXiv*, abs/1609.08144, 2016.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, 2015.

[8] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016.

[9] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. In *CoRL*, 2018.

[10] OpenAI et al. Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018.

[11] Christian Szegedy, W. Zaremba, Ilya Sutskever, Joan Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2014.

[12] Aravind Rajeswaran, Kendall Lowrey, Emanuel Todorov, and Sham Kakade. Towards generalization and simplicity in continuous control. In *NIPS*, 2017.

[13] Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. Epopt: Learning robust neural network policies using model ensembles. In *ICLR*, 2017.

[14] Kendall Lowrey, Svetoslav Kolev, Jeremy Dao, Aravind Rajeswaran, and Emanuel Todorov. Reinforcement learning for non-prehensile manipulation: Transfer from simulation to physical system. In *IEEE SIMPAR*, 2018.

[15] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel : Model-based offline reinforcement learning. In *NeurIPS*, 2020.

[16] Aravind Rajeswaran, Igor Mordatch, and Vikash Kumar. A game theoretic framework for model-based reinforcement learning. In *ICML*, 2020.

[17] Aravind Rajeswaran, Chelsea Finn, Sham M. Kakade, and Sergey Levine. Meta-learning with implicit gradients. In *NeurIPS*, 2019.

[18] Chelsea Finn, Aravind Rajeswaran, Sham Kakade, and Sergey Levine. Online meta-learning. *International Conference on Machine Learning (ICML)*, 2019.

[19] Volodymyr Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[20] David Silver et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016.

[21] Oriol Vinyals et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, pages 1–5, 2019.

[22] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

[23] Igor Mordatch, Kendall Lowrey, Galen Andrew, Zoran Popovic, and Emanuel Todorov. Interactive control of diverse complex characters with neural networks. In *NIPS*, 2015.

[24] Bowen Baker, I. Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. In *ICLR*, 2020.

[25] Xue Bin Peng, Glen Berseth, and Michiel van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2016)*, 2016.

[26] Sham Kakade. *On the Sample Complexity of Reinforcement Learning*. PhD thesis, University College London, 2003.

[27] Javier García and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 2015.

[28] Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. Bayesian reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 8 (5-6):359–483, 2015.

[29] Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10:1633–1685, December 2009.

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

[31] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

[32] Sham M Kakade. A natural policy gradient. In *NIPS*, 2002.

[33] John Schulman, Sergey Levine, Pieter Abbeel, Michael I Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, 2015.

[34] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[35] Aviv Tamar, Yonatan Glassner, and Shie Mannor. Optimizing the cvar via sampling. In *AAAI Conference on Artificial Intelligence*, 2015.

[36] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning (ICML)*, 2016.

[37] Tom Erez, Yuval Tassa, and Emanuel Todorov. Infinite-horizon model predictive control for periodic tasks with contacts. In *Proceedings of Robotics: Science and Systems*, 2011.

[38] Pawel Wawrzynski. Real-time reinforcement learning by sequential actor-critics and experience replay. *Neural Networks*, 22:1484–1497, 2009.

[39] Kemin Zhou, John C. Doyle, and Keith Glover. *Robust and Optimal Control*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996. ISBN 0-13-456567-3.

[40] Arnab Nilim and Laurent El Ghaoui. Robust control of markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005.

[41] Shiau Hong Lim, Huan Xu, and Shie Mannor. Reinforcement learning in robust markov decision processes. In *NIPS*. 2013.

[42] Nikos Vlassis, Mohammad Ghavamzadeh, Shie Mannor, and Pascal Poupart. *Bayesian Reinforcement Learning*, pages 359–386. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[43] Pascal Poupart, Nikos A. Vlassis, Jesse Hoey, and Kevin Regan. An analytic solution to discrete bayesian reinforcement learning. In *ICML*, 2006.

[44] S. Ross, B. Chaib-draa, and J. Pineau. Bayesian reinforcement learning in continuous pomdps with application to robot navigation. In *ICRA*, 2008.

[45] Michael O. Duff. Design for an optimal probe. In *ICML*, 2003.

[46] Josep M. Porta, Nikos A. Vlassis, Matthijs T. J. Spaan, and Pascal Poupart. Point-based value iteration for continuous pomdps. *Journal of Machine Learning Research*, 7: 2329–2367, 2006.

[47] Erick Delage and Shie Mannor. Percentile optimization for markov decision processes with parameter uncertainty. *Operations Research*, 58(1):203–213, 2010.

[48] Lennart Ljung. *System Identification*, pages 163–173. Birkhäuser Boston, Boston, MA, 1998.

[49] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

[50] Stéphane Ross and Drew Bagnell. Agnostic system identification for model-based reinforcement learning. In *ICML*, 2012.

[51] I. Mordatch, K. Lowrey, and E. Todorov. Ensemble-CIO: Full-body dynamic motion planning that transfers to physical humanoids. In *IROS*, 2015.

[52] Jack M. Wang, David J. Fleet, and Aaron Hertzmann. Optimizing walking controllers for uncertain inputs and environments. *ACM Trans. Graph.*, 2010.

[53] Bruno Castro da Silva, George Konidaris, and Andrew G. Barto. Learning parameterized skills. In *ICML*, 2012.

[54] Philip Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High-confidence off-policy evaluation. In *AAAI Conference on Artificial Intelligence*. 2015.

[55] Sham M. Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *ICML*, 2002.

[56] Sergey Levine and Vladlen Koltun. Guided policy search. In *ICML*, JMLR Workshop and Conference Proceedings, pages 1–9. JMLR.org, 2013.

[57] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[58] Aviv Tamar, Dotan Di Castro, and Ron Meir. Integrating a partial model into model free reinforcement learning. *Journal of Machine Learning Research*, 2012.

[59] Pieter Abbeel, Morgan Quigley, and Andrew Y. Ng. Using inaccurate models in reinforcement learning. In *ICML*, 2006.

[60] Svetoslav Kolev and Emanuel Todorov. Physically consistent state estimation and system identification for contacts. In *IEEE Humanoid Robots (Humanoids)*, 2015.

[61] Joshua Tobin, Rachel Fong, Alex Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017.

[62] Joshua Tobin, W. Zaremba, and P. Abbeel. Domain randomization and generative models for robotic grasping. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3482–3489, 2018.

[63] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *CoRR*, abs/1710.06537, 2017.

[64] Yevgen Chebotar, A. Handa, Viktor Makoviychuk, M. Macklin, J. Issac, Nathan D. Ratliff, and D. Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979, 2019.

[65] Michael Ahn, Henry Zhu, Kristian Hartikainen, Hugo Ponte, Abhishek Gupta, Sergey Levine, and Vikash Kumar. Robel: Robotics benchmarks for learning with low-cost robots. In *Conference on Robot Learning (CoRL)*, 2019.

[66] Ofir Nachum, Michael J. Ahn, Hugo Ponte, Shixiang Gu, and Vikash Kumar. Multi-agent manipulation via locomotion using hierarchical sim2real. *ArXiv*, abs/1908.05224, 2019.

[67] OpenAI et al. Solving rubik's cube with a robot hand. *ArXiv*, abs/1910.07113, 2019.

[68] J. Deng, W. Dong, R. Socher, L. J. Li, K.Li, and L. Fei Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009.

[69] W. Fisher, G. Doddington, and K. Goudie-Marshall. The DARPA Speech Recognition Research Database: Specification and Status. In *Proceedings of the DARPA Workshop*, pages 93–100, 1986.

[70] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, and Phillipp Koehn. One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, abs/1312.3005, 2013.

[71] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.

[72] David Silver et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362:1140–1144, 2018.

[73] Sascha Lange, Thomas Gabel, and Martin A. Riedmiller. Batch reinforcement learning. In *Reinforcement Learning*, volume 12. Springer, 2012.

[74] Philip S Thomas. Safe reinforcement learning. *PhD Thesis*, 2014. URL `http://scholarworks.umass.edu/dissertations_2/514`.

[75] Philip S. Thomas, Bruno Castro da Silva, Andrew G. Barto, Stephen Giguere, Yuriy Brun, and Emma Brunskill. Preventing undesirable behavior of intelligent machines. *Science*, 366(6468):999–1004, 2019.

[76] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. *CoRR*, abs/1812.02900, 2018.

[77] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *CoRR*, abs/1906.00949, 2019.

[78] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. Striving for simplicity in off-policy deep reinforcement learning. *CoRR*, abs/1907.04543, 2019.

[79] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *CoRR*, arXiv:1911.11361, 2019.

[80] Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Àgata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind W. Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *CoRR*, abs/1907.00456, 2019.

[81] Romain Laroche and Paul Trichelair. Safe policy improvement with baseline bootstrapping. *CoRR*, abs/1712.06924, 2017.

[82] Ofir Nachum, Bo Dai, Ilya Kostrikov, Yinlam Chow, Lihong Li, and Dale Schuurmans. Algaedice: Policy gradient from arbitrary experience. *CoRR*, arXiv:1912.02074, 2019.

[83] Andy Zeng, Shuran Song, Johnny Lee, Alberto Rodríguez, and Thomas A. Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *ArXiv*, abs/1903.11239, 2019.

[84] Emanuel Todorov and Weiwei Li. A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems. In *ACC*, 2005.

[85] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 4906–4913. IEEE, 2012.

[86] Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4:1–43, 2012.

[87] Rémi Munos and Csaba Szepesvari. Finite-time bounds for fitted value iteration. *J. Mach. Learn. Res.*, 9:815–857, 2008.

[88] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, pages 12498–12509, 2019.

[89] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. In *ICLR*, 2018.

[90] Ignasi Clavera, Jonas Rothfuss, John Schulman, Yasuhiro Fujita, Tamim Asfour, and Pieter Abbeel. Model-based reinforcement learning via meta-policy optimization. *ArXiv*, abs/1809.05214, 2018.

[91] Omer Gottesman, Fredrik D. Johansson, Joshua Meier, Jack Dent, Donghun Lee, Srivatsan Srinivasan, Linying Zhang, Yi Ding, David Wihl, Xuefeng Peng, Jiayu Yao,

Isaac Lage, Christopher Mosch, Li-Wei H. Lehman, Matthieu Komorowski, Aldo Faisal, Leo Anthony Celi, David A. Sontag, and Finale Doshi-Velez. Evaluating reinforcement learning algorithms in observational health settings. *CoRR*, abs/1805.12298, 2018.

[92] Lu Wang, Wei Zhang 0056, Xiaofeng He, and Hongyuan Zha. Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation. In Yike Guo and Faisal Farooq, editors, *KDD*, pages 2447–2456. ACM, 2018.

[93] Chao Yu, Guoqi Ren, and Jiming Liu 0001. Deep inverse reinforcement learning for sepsis treatment. In *ICHI*, pages 1–3. IEEE, 2019. ISBN 978-1-5386-9138-0.

[94] Alexander L. Strehl, John Langford, and Sham M. Kakade. Learning from logged implicit exploration data. *CoRR*, abs/1003.0120, 2010.

[95] Adith Swaminathan and Thorsten Joachims. Batch learning from logged bandit feedback through counterfactual risk minimization. *J. Mach. Learn. Res*, 16:1731–1755, 2015.

[96] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *RecSys*. ACM, 2016.

[97] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H. Chi. Top-k off-policy correction for a reinforce recommender system. *CoRR*, abs/1812.02353, 2018.

[98] Li Zhou, Kevin Small, Oleg Rokhlenko, and Charles Elkan. End-to-end offline goal-oriented dialog policy learning via policy gradient. *CoRR*, abs/1712.02838, 2017.

[99] Nikos Karampatziakis, Sebastian Kochman, Jade Huang, Paul Mineiro, Kathy Osborne, and Weizhu Chen. Lessons from real-world reinforcement learning in a customer support bot. *CoRR*, abs/1905.02219, 2019.

[100] Ahmad El Sallab, Mohammed Abdou, Etienne Perot, and Senthil Kumar Yogamani. Deep reinforcement learning framework for autonomous driving. *CoRR*, abs/1704.02532, 2017.

[101] Lihong Li, Wei Chu, John Langford, and Xuanhui Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms, 2010. Comment: 10 pages, 7 figures, revised from the published version at the WSDM 2011 conference.

[102] Yao Liu, Adith Swaminathan, Alekh Agarwal, and Emma Brunskill. Off-policy policy gradient with state distribution correction. *CoRR*, abs/1904.08473, 2019.

[103] Assaf Hallak and Shie Mannor. Consistent on-line off-policy evaluation. *CoRR*, abs/1702.07121, 2017.

[104] Carles Gelada and Marc G. Bellemare. Off-policy deep reinforcement learning by bootstrapping the covariate shift. In *AAAI*, pages 3647–3655. AAAI Press, 2019.

[105] Ofir Nachum, Yinlam Chow, Bo Dai, and Lihong Li. Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. *CoRR*, abs/1906.04733, 2019.

[106] Chris Watkins. Learning from delayed rewards. *PhD Thesis, Cambridge University*, 1989.

[107] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018.

[108] Jinglin Chen and Nan Jiang. Information-theoretic considerations in batch reinforcement learning. In *ICML*, 2019.

[109] Mohammad Ghavamzadeh, Marek Petrik, and Yinlam Chow. Safe policy improvement by minimizing robust baseline regret. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2298–2306, 2016.

[110] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *CoRR*, abs/2005.13239, 2020.

[111] Michael Kearns and Satinder Singh. Near optimal reinforcement learning in polynomial time. *Machine Learning*, 49(2-3):209–232, 2002.

[112] Ronen I. Brafman and Moshe Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *J. Mach. Learn. Res.*, 3:213–231, 2001.

[113] Sham M. Kakade, Michael J. Kearns, and John Langford. Exploration in metric state spaces. In *ICML*, 2003.

[114] Nan Jiang. Pac reinforcement learning with an imperfect model. In *AAAI*, pages 3334–3341, 2018.

[115] Anirudh Vemula, Yash Oza, J. Andrew Bagnell, and Maxim Likhachev. Planning and execution using inaccurate models with provable guarantees, 2020.

[116] Samuel K. Ainsworth, Matt Barnes, and Siddhartha S. Srinivasa. Mo' states mo' problems: Emergency stop mechanisms from observation. *CoRR*, abs/1912.01649, 2019.

[117] Arun Venkatraman, Martial Hebert, and J. Andrew Bagnell. Improving multi-step prediction of learned time series models. In *AAAI*, 2015.

[118] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. *ArXiv*, abs/1506.03099, 2015.

[119] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *ArXiv*, abs/1706.03762, 2017.

[120] Grady Williams, Nolan Wagener, Brian Goldfain, Paul Drews, James M. Rehg, Byron Boots, and Evangelos Theodorou. Information theoretic mpc for model-based reinforcement learning. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1714–1721, 2017.

[121] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning, 1998.

[122] Rasool Fakoor, Pratik Chaudhari, and Alexander J. Smola. P3o: Policy-on policy-off policy optimization. *CoRR*, abs/1905.01756, 2019.

[123] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2015.

[124] Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. *CoRR*, abs/1806.03335, 2018.

[125] Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. Efficient exploration through bayesian deep q-networks. In *ITA*, pages 1–9. IEEE, 2018.

[126] Yuri Burda, Harrison Edwards, Amos J. Storkey, and Oleg Klimov. Exploration by random network distillation. In *ICLR*, 2019.

[127] Kendall Lowrey, Aravind Rajeswaran, Sham Kakade, Emanuel Todorov, and Igor Mordatch. Plan online, learn offline: Efficient learning and exploration via model-based control. In *International Conference on Learning Representations (ICLR)*, 2019.

[128] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[129] Justin Fu, Aviral Kumar, Ofir Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning. *ArXiv*, abs/2004.07219, 2020.

[130] Aviral Kumar, Aurick Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *ArXiv*, abs/2006.04779, 2020.

[131] D. Blei, A. Kucukelbir, and J. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112:859–877, 2016.

[132] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[133] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[134] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

[135] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *International Conference on Learning Representations*, 2020.

[136] Alex X. Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *arXiv preprint arxiv:1907.00953.pdf*, 2020.

[137] Annie S. Chen, HyunJi Nam, Suraj Nair, and Chelsea Finn. Batch exploration with examples for scalable robotic reinforcement learning, 2020.

[138] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE, 2017.

[139] Mohammad Babaeizadeh, Chelsea Finn, Dumitru Erhan, Roy H. Campbell, and Sergey Levine. Stochastic variational video prediction. In *ICML*, 2018.

[140] Karl Johan Åström and Björn Wittenmark. Adaptive control. 1989.

[141] Martin L. Puterman. Markov decision processes: Discrete stochastic dynamic programming. In *Wiley Series in Probability and Statistics*, 1994.

[142] Kumpati S. Narendra and Anuradha M. Annaswamy. Persistent excitation in adaptive systems. *International Journal of Control*, 1987.

[143] Michael Kearns and Satinder P. Singh. Finite-sample convergence rates for q-learning and indirect algorithms. In *NIPS*, 1998.

[144] Alekh Agarwal, Sham M. Kakade, and Lin F. Yang. On the optimality of sparse model-based planning for markov decision processes. *ArXiv*, abs/1906.03804, 2019.

[145] Sergey Levine and Pieter Abbeel. Learning neural network policies with guided policy search under unknown dynamics. In *NIPS*, 2014.

[146] Wen Sun, Geoffrey J. Gordon, Byron Boots, and J. Andrew Bagnell. Dual policy iteration. *CoRR*, abs/1805.10755, 2018.

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

[148] Anusha Nagabandi, Kurt Konoglie, Sergey Levine, and Vikash Kumar. Deep dynamics models for learning dexterous manipulation. *ArXiv*, abs/1909.11652, 2019.

[149] Heinrich von Stackelberg. Market structure and equilibrium. 1934.

[150] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017.

[151] Shai Ben-David, John Blitzer, Koby Crammer, and Fernando C Pereira. Analysis of representations for domain adaptation. In *NIPS*, 2006.

[152] Baochen Sun, Jiashi Feng, and Kate Saenko. Return of frustratingly easy domain adaptation. In *AAAI*, 2015.

[153] Yuanhao Wang, Guodong Zhang, and Jimmy Ba. On solving minimax optimization locally: A follow-the-ridge approach. *ArXiv*, abs/1910.07512, 2019.

[154] Tanner Fiez, Benjamin Chasnov, and Lillian J. Ratliff. Convergence of learning dynamics in stackelberg games. *ArXiv*, abs/1906.01217, 2019.

[155] Jakob N. Foerster, Richard Y. Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. In *AAMAS*, 2017.

[156] Florian Schäfer and Anima Anandkumar. Competitive gradient descent. In *NeurIPS*, 2019.

[157] Nicolò Cesa-Bianchi and Gábor Lugosi. Prediction, learning, and games. 2006.

[158] Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of Operations Research*, 153:235–256, 2007.

[159] Steven G. Krantz and Harold R. Parks. The implicit function theorem: History, theory, and applications. 2002.

[160] Alex Nichol, Joshua Achiam, and John Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[161] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NIPS*, 2017.

[162] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. Unrolled generative adversarial networks. *ArXiv*, abs/1611.02163, 2017.

[163] Vijaymohan Konda and Vivek S. Borkar. Actor-critic - type learning algorithms for markov decision processes. *SIAM J. Control and Optimization*, 38:94–123, 1999.

[164] Shai Shalev-Shwartz. Online learning and online convex optimization. *"Foundations and Trends in Machine Learning"*, 2012.

[165] H. Brendan McMahan. Follow-the-regularized-leader and mirror descent: Equivalence theorems and l1 regularization. In *AISTATS*, 2011.

[166] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *ArXiv*, abs/1812.05905, 2018.

[167] Elad Hazan, Sham M. Kakade, Karan Singh, and Abby Van Soest. Provably efficient maximum entropy exploration. In *ICML*, 2018.

[168] Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. *ArXiv*, abs/1906.04161, 2019.

[169] Henry Zhu, Abhishek Gupta, Aravind Rajeswaran, Sergey Levine, and Vikash Kumar. Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost. *2019 International Conference on Robotics and Automation (ICRA)*, pages 3651–3657, 2018.

[170] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

[171] Lennart Ljung. System identification: Theory for the user. 1987.

[172] Alekh Agarwal, Sham M. Kakade, Jason D. Lee, and Gaurav Mahajan. Optimality and approximation with policy gradient methods in markov decision processes. *ArXiv*, abs/1908.00261, 2019.

[173] Igor Mordatch and Emanuel Todorov. Combining the benefits of function approximation and trajectory optimization. In *RSS*, 2014.

[174] Vikash Kumar, Emanuel Todorov, and Sergey Levine. Optimal control with learned local models: Application to dexterous manipulation. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 378–383, 2016.

[175] Huazhe Xu, Yuanzhi Li, Yuandong Tian, Trevor Darrell, and Tengyu Ma. Algorithmic framework for model-based reinforcement learning with theoretical guarantees. *ArXiv*, abs/1807.03858, 2018.

[176] Marc Peter Deisenroth and Carl E. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *ICML*, 2011.

[177] Tingwu Wang, Xuchan Bao, Ignasi Clavera, Jerrick Hoang, Yeming Wen, Eric Langlois, S. Zhang, Guodong Zhang, Pieter Abbeel, and Jimmy Ba. Benchmarking model-based reinforcement learning. *ArXiv*, abs/1907.02057, 2019.

[178] Vijay R. Konda and John N. Tsitsiklis. Convergence rate of linear two-time-scale stochastic approximation. In *The Annals of Applied Probability*, 2004.

[179] Prasenjit Karmakar and Shalabh Bhatnagar. Two time-scale stochastic approximation with controlled markov noise and off-policy temporal-difference learning. *Mathematics of Operations Research*, 43:130–151, 2015.

[180] Richard S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *ICML*, 1990.

[181] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I. Jordan, Joseph Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *CoRR*, abs/1803.00101, 2018.

[182] Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. *arXiv preprint arXiv:1807.01675*, 2018.

[183] Nan Jiang and Lihong Li. Doubly robust off-policy value evaluation for reinforcement learning. In *ICML*, 2015.

[184] Philip S. Thomas and Emma Brunskill. Data-efficient off-policy policy evaluation for reinforcement learning. *ArXiv*, abs/1604.00923, 2016.

[185] Mehrdad Farajtabar, Yinlam Chow, and Mohammad Ghavamzadeh. More robust doubly robust off-policy evaluation. In *ICML*, 2018.

[186] Jurgen Schmidhuber. Evolutionary principles in self-referential learning. *Diploma thesis, Institut f. Informatik, Tech. Univ. Munich*, 1987.

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

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

[189] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International Conference on Machine Learning (ICML)*, 2016.

[190] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Neural Information Processing Systems (NIPS)*, 2016.

[191] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *International Conference on Machine Learning (ICML)*, 2017.

[192] Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, 2001.

[193] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[194] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. *arXiv preprint arXiv:1707.03141*, 2017.

[195] Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015.

[196] Chelsea Finn and Sergey Levine. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. *arXiv:1710.11622*, 2017.

[197] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-shot visual imitation learning via meta-learning. *arXiv preprint arXiv:1709.04905*, 2017.

[198] Fei Mi, Minlie Huang, Jiyong Zhang, and Boi Faltings. Meta-learning for low-resource natural language generation in task-oriented dialogue systems. *arXiv preprint arXiv:1905.05644*, 2019.

[199] Maruan Al-Shedivat, Trapit Bansal, Yuri Burda, Ilya Sutskever, Igor Mordatch, and Pieter Abbeel. Continuous adaptation via meta-learning in nonstationary and competitive environments. *CoRR*, abs/1710.03641, 2017.

[200] Chelsea Finn. *Learning to Learn with Gradients.* PhD thesis, UC Berkeley, 2018.

[201] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-sgd: Learning to learn quickly for few-shot learning. *arXiv preprint arXiv:1707.09835*, 2017.

[202] Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. *International Conference on Learning Representations (ICLR)*, 2018.

[203] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015.

[204] James Martens. Deep learning via hessian-free optimization. In *ICML*, 2010.

[205] Jorge Nocedal and Stephen J. Wright. Numerical optimization (springer series in operations research and financial engineering). 2000.

[206] Chi Jin, Rong Ge, Praneeth Netrapalli, Sham M. Kakade, and Michael I. Jordan. How to escape saddle points efficiently. In *ICML*, 2017.

[207] Yurii Nesterov and Boris T. Polyak. Cubic regularization of newton method and its global performance. *Math. Program.*, 108:177–205, 2006.

[208] Amirreza Shaban, Ching-An Cheng, Olivia Hirschey, and Byron Boots. Truncated back-propagation for bilevel optimization. *CoRR*, abs/1810.10667, 2018.

[209] Brenden M Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua B Tenenbaum. One shot learning of simple visual concepts. In *Conference of the Cognitive Science Society (CogSci)*, 2011.

[210] Jaehong Kim, Youngduck Choi, Moonsu Cha, Jung Kwon Lee, Sangyeul Lee, Sungwan Kim, Yongseok Choi, and Jiwon Kim. Auto-meta: Automated gradient based meta learner search. *arXiv preprint arXiv:1806.06927*, 2018.

[211] Gregory Koch. Siamese neural networks for one-shot image recognition. *ICML Deep Learning Workshop*, 2015.

[212] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 2017.

[213] Boris Oreshkin, Pau Rodríguez López, and Alexandre Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems*, pages 721–731, 2018.

[214] Kelsey R Allen, Evan Shelhamer, Hanul Shin, and Joshua B Tenenbaum. Infinite mixture prototypes for few-shot learning. *arXiv preprint arXiv:1902.04552*, 2019.

[215] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Neural Information Processing Systems (NIPS)*, 2016.

[216] Ke Li and Jitendra Malik. Learning to optimize. *International Conference on Learning Representations (ICLR)*, 2017.

[217] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl2: Fast reinforcement learning via slow reinforcement learning. *arXiv:1611.02779*, 2016.

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

[219] Tsendsuren Munkhdalai and Hong Yu. Meta networks. *International Conference on Machine Learning (ICML)*, 2017.

[220] Luca Bertinetto, Joao F Henriques, Philip HS Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. *arXiv preprint arXiv:1805.08136*, 2018.

[221] Luisa M Zintgraf, Kyriacos Shiarlis, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. Fast context adaptation via meta-learning. *arXiv preprint arXiv:1810.03642*, 2018.

[222] Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*, pages 9516–9527, 2018.

[223] Fengwei Zhou, Bin Wu, and Zhenguo Li. Deep meta-learning: Learning to learn in the concept space. *arXiv preprint arXiv:1802.03596*, 2018.

[224] James Harrison, Apoorva Sharma, and Marco Pavone. Meta-learning priors for efficient online bayesian regression. *arXiv preprint arXiv:1807.08912*, 2018.

[225] Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*, 2018.

[226] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, and Hugo Larochelle. Meta-dataset: A dataset of datasets for learning to learn from few examples. *arXiv preprint arXiv:1903.03096*, 2019.

[227] Ferran Alet, Tomás Lozano-Pérez, and Leslie P Kaelbling. Modular meta-learning. *arXiv preprint arXiv:1806.10166*, 2018.

[228] Fabian Pedregosa. Hyperparameter optimization with approximate gradient. *arXiv preprint arXiv:1602.02355*, 2016.

[229] Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1165–1173. JMLR. org, 2017.

[230] Justin Domke. Generic methods for optimization-based modeling. In *AISTATS*, 2012.

[231] Chuong B. Do, Chuan-Sheng Foo, and Andrew Y. Ng. Efficient multiple hyperparameter learning for log-linear models. In *NIPS*, 2007.

[232] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 136–145. JMLR. org, 2017.

[233] Benoit Landry, Zachary Manchester, and Marco Pavone. A differentiable augmented lagrangian method for bilevel nonlinear optimization. *arXiv preprint arXiv:1902.03319*, 2019.

[234] Laurent Hascoët and Mauricio Araya-Polo. Enabling user-driven checkpointing strategies in reverse-mode automatic differentiation. *CoRR*, abs/cs/0606042, 2006.

[235] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. *arXiv preprint arXiv:1904.03758*, 2019.

[236] Igor Mordatch. Concept learning with energy-based models. *CoRR*, abs/1811.02486, 2018.

[237] James Hannan. Approximation to bayes risk in repeated play. *Contributions to the Theory of Games*, 1957.

[238] Adam Tauman Kalai and Santosh Vempala. Efficient algorithms for online decision problems. *J. Comput. Syst. Sci.*, 71:291–307, 2005.

[239] Elad Hazan. Introduction to online convex optimization. 2016.

[240] Shai Shalev-Shwartz and Sham M. Kakade. Mind the duality gap: Logarithmic regret algorithms for online optimization. In *NIPS*, 2008.

[241] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 2011.

[242] Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your maml. *arXiv preprint arXiv:1810.09502*, 2018.

[243] Yu Xiang, Roozbeh Mottaghi, and Silvio Savarese. Beyond pascal: A benchmark for 3d object detection in the wild. In *Conference on Applications of Computer Vision (WACV)*, 2014.

[244] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv:1606.04671*, 2016.

[245] 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.

[246] 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.

[247] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv:1511.06342*, 2015.

[248] Samy Bengio, Yoshua Bengio, Jocelyn Cloutier, and Jan Gecsei. On the optimization of a synaptic learning rule. In *Optimality in Artificial and Biological Neural Networks*, 1992.

[249] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations (ICLR)*, 2017.

[250] Jürgen Schmidhuber. Optimal ordered problem solver. *Machine Learning*, 54:211–254, 2002.

[251] David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *International Conference on Learning Representations (ICLR)*, 2017.

[252] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. Meta-learning with temporal convolutions. *arXiv preprint arXiv:1707.03141*, 2017.

[253] Anusha Nagabandi, Chelsea Finn, and Sergey Levine. Deep online learning via meta-learning: Continual adaptation for model-based rl. *arXiv preprint arXiv:1812.07671*, 2018.

[254] Stefan Elfwing, Eiji Uchibe, and Kenji Doya. Online meta-learning by parallel algorithm competition. *arXiv:1702.07490*, 2017.

[255] Franziska Meier, Daniel Kappler, and Stefan Schaal. Online learning of a memory for learning rates. *arXiv:1709.06709*, 2017.

[256] Atilim Gunes Baydin, Robert Cornish, David Martinez Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. *arXiv:1703.04782*, 2017.

[257] Erin Grant, Ghassen Jerfel, Katherine Heller, and Thomas L. Griffiths. Modulating transfer between tasks in gradient-based meta-learning, 2019.

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

[259] Jieyu Zhao and Jurgen Schmidhuber. Incremental self-improvement for life-time multi-agent reinforcement learning. In *From Animals to Animats 4: International Conference on Simulation of Adaptive Behavior*, 1996.

[260] Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv:1312.6211*, 2013.

[261] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 2017.

[262] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, 2017.

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

[264] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *Advances in Neural Information Processing Systems*, 2017.

[265] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. Incremental learning of object detectors without catastrophic forgetting. *arXiv:1708.06977*, 2017.

[266] David Lopez-Paz et al. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, 2017.

[267] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *arXiv:1710.10628*, 2017.

[268] Jürgen Schmidhuber. Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem. In *Front. Psychol.*, 2013.

[269] Jeongtae Lee, Jaehong Yun, Sungju Hwang, and Eunho Yang. Lifelong learning with dynamically expandable networks. *arXiv:1708.01547*, 2017.

[270] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. *arXiv:1711.05769*, 2017.

[271] David Isele and Akansel Cosgun. Selective experience replay for lifelong learning. *arXiv preprint arXiv:1802.10269*, 2018.

[272] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artif. Intell.*, 72:81–138, 1995.

[273] Paul Ruvolo and Eric Eaton. Ella: An efficient lifelong learning algorithm. In *International Conference on Machine Learning*, pages 507–515, 2013.

[274] Elad Hazan, Adam Tauman Kalai, Satyen Kale, and Amit Agarwal. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69:169–192, 2006.

[275] Martin Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *ICML*, 2003.

[276] Mark Herbster and Manfred K. Warmuth. Tracking the best expert. *Machine Learning*, 32:151–178, 1995.

[277] Tianbao Yang, Lijun Zhang, Rong Jin, and Jinfeng Yi. Tracking slowly moving clairvoyant: Optimal dynamic regret of online learning with true and noisy gradient. In *ICML*, 2016.

[278] Omar Besbes, Yonatan Gur, and Assaf J. Zeevi. Non-stationary stochastic optimization. *Operations Research*, 63:1227–1244, 2015.

[279] Elad Hazan and Seshadhri Comandur. Efficient learning algorithms for changing environments. In *ICML*, 2009.

[280] Eric C. Hall and Rebecca M Willett. Online convex optimization in dynamic environments. *IEEE Journal of Selected Topics in Signal Processing*, 9:647–662, 2015.

[281] Pierre Alquier, The Tien Mai, and Massimiliano Pontil. Regret bounds for lifelong learning. In *AISTATS*, 2016.

[282] Giulia Denevi, Carlo Ciliberto, Riccardo Grazzi, and Massimiliano Pontil. Learning-to-learn stochastic gradient descent with biased regularization. *CoRR*, abs/1903.10399, 2019.

[283] Mikhail Khodak, Maria-Florina Balcan, and Ameet S. Talwalkar. Provable guarantees for gradient-based meta-learning. *CoRR*, abs/1902.10644, 2019.

[284] Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. Opal: Offline primitive discovery for accelerating offline reinforcement learning. *ArXiv*, abs/2010.13611, 2020.

[285] Avi Singh, Huihan Liu, Gaoyue Zhou, Albert Yu, Nicholas Rhinehart, and Sergey Levine. Parrot: Data-driven behavioral priors for reinforcement learning. *ArXiv*, abs/2011.10024, 2020.

[286] Tanmay Shankar and Abhinav Gupta. Learning robot skills with temporal variational inference. *ArXiv*, abs/2006.16232, 2020.

[287] Karl Pertsch, Youngwoon Lee, and Joseph J. Lim. Accelerating reinforcement learning with learned skill priors. *ArXiv*, abs/2010.11944, 2020.

[288] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *ArXiv*, abs/2106.01345, 2021.

[289] Anusha Nagabandi, Gregory Kahn, Ronald S. Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *ICRA*, 2018.

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

[291] Pieter Abbeel and Andrew Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *ICML*, 2005.

[292] Evan Greensmith, Peter L. Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *JMLR*, 2001.

[293] Cathy Wu, Aravind Rajeswaran, Yan Duan, Vikash Kumar, Alexandre M. Bayen, Sham M. Kakade, Igor Mordatch, and Pieter Abbeel. Variance reduction for policy gradient with action-dependent factorized baselines. In *ICLR*, 2018.

[294] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *ICLR*, 2016.

[295] Sebastien Bubeck. *Convex optimization: Algorithms and complexity.* Foundations and Trends in Machine Learning, 2015.

[296] Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theoretical Computer Science*, 22:317–330, 1983.

[297] Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2008.

[298] Atilim Gunes Baydin, Barak A. Pearlmutter, and Alexey Radul. Automatic differentiation in machine learning: a survey. *CoRR*, abs/1502.05767, 2015.

[299] Andreas Griewank. Some bounds on the complexity of gradients, jacobians, and hessians. 1993.

[300] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[301] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[302] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end learning of deep visuomotor policies. *Journal of Machine Learning Research (JMLR)*, 2016.

[303] Avi Singh, Larry Yang, and Sergey Levine. Gplac: Generalizing vision-based robotic skills using weakly labeled images. *arXiv:1708.02313*, 2017.

Appendix A

EPOPT: ADDITIONAL ABLATIONS AND EXPERIMENTS

A.1 Hyperparameter and model details

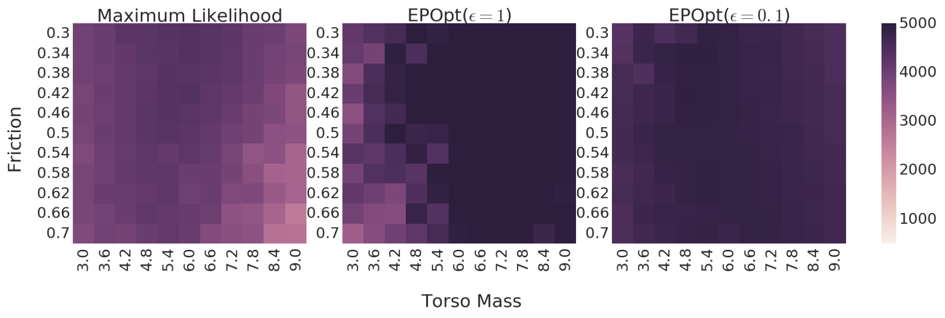
1. Neural network architecture: We used a neural network with two hidden layers, each with 64 units and *tanh* non-linearity. The policy updates are implemented using TRPO.
2. Trust region size in TRPO: The maximum KL divergence between sucessive policy updates are constrained to be 0.01
3. Number and length of trajectory rollouts: In each iteration, we sample $N = 240$ models from the ensemble, one rollout is performed on each such model. This was implemented in parallel on multiple (6) CPUs. Each trajectory is of length 1000 – same as the standard implimentations of these tasks in gym and rllab.

The results in Fig 2.1 and Fig 2.2 were generated after 150 and 200 iterations of TRPO respectively, with each iteration consisting of 240 trajectories as specified in (3) above.

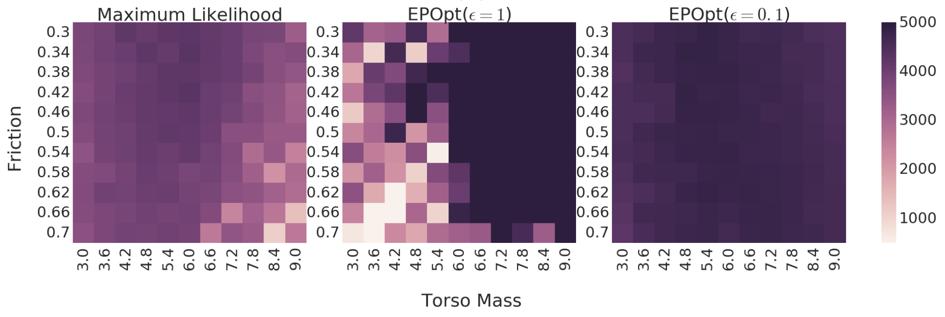
A.2 Robustness results

We present the results for the Half-Cheetah domain in Figure A.1. This experiment is similar to the experiment presented in Figure 2.2, where we evaluate the performance of the trained policy on different physical parameter configurations. In particular, Figure 2.2 illustrates the performance of the three considered policies: viz. TRPO on mean parameters, EPOpt($\epsilon = 1$), and EPOpt($\epsilon = 0.1$). We repeat the same experiment in the half-cheetah domain, and again find that the EPOpt($\epsilon = 0.1$) policy is significantly more robust than training on a single source domain.

To analyze a stronger degree of robustness, we also analyze the 10$^{\text{th}}$ percentile of the return distribution as a proxy for worst-case performance, which is important for a robust control policy (here, distribution of returns for a given model instance is due to variations in initial conditions). The corresponding results are presented below in Figure A.2. We again find that the soft-adversarial approach with EPOpt($\epsilon = 0.1$) leads to the best results.



(a)



(b)

Figure A.1: Performance of policies for various model instances for the half-cheetah domain, similar to Figure 2.2. Again, it is observed that the adversarial trained policy is robust and generalizes well to all models in the source distribution.

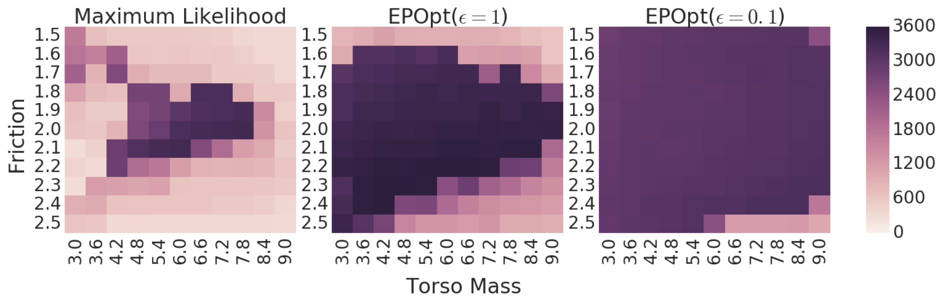


Figure A.2: 10th percentile of return distribution for the hopper task. EPOpt($\epsilon = 0.1$) clearly outperforms the other approaches. The 10th of return distribution for EPOpt($\epsilon = 0.1$) also nearly overlaps with the expected return, indicating that the policies trained using EPOpt($\epsilon = 0.1$) are highly robust and reliable.

A.3 Different settings for ϵ

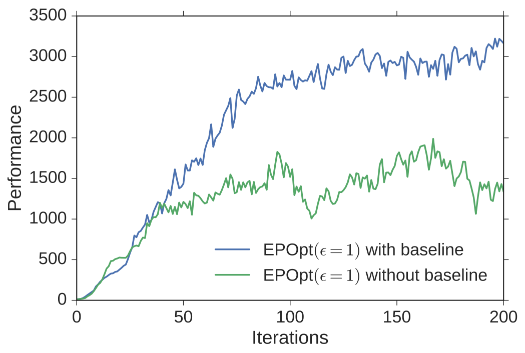
Here, we analyze how different settings for ϵ influences the robustness of learned policies. The policies in this section have been trained for 200 iterations with 240 trajectory samples per iteration. Similar to the description in Section 3.1, the first 100 iterations use $\epsilon = 1$, and the final 100 iterations use the desired ϵ. The source distribution is described in Table 1. We test the performance on a grid over the model parameters. Our results, summarized in Table A.1, indicate that decreasing ϵ decreases the variance in performance, along with a small decrease in average performance, and hence enhances robustness.

Table A.1: Performance statistics for different ϵ settings for the hopper task

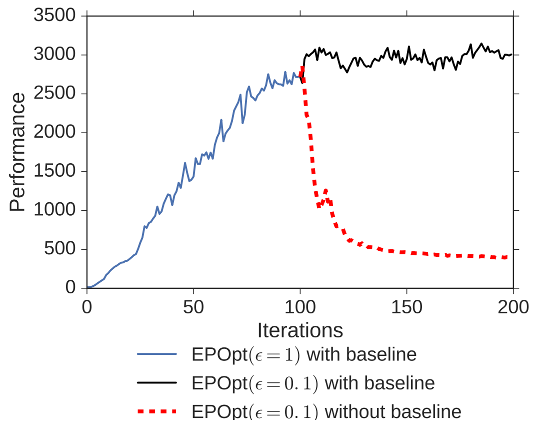
	Performance (Return)							
ϵ	**mean**	**std**	**Percentiles**					
			5	10	25	50	75	90
0.05	2889	502	1662	2633	2841	2939	2966	3083
0.1	3063	579	1618	2848	3223	3286	3336	3396
0.2	3097	665	1527	1833	3259	3362	3423	3483
0.3	3121	706	1461	1635	3251	3395	3477	3513
0.4	3126	869	1013	1241	3114	3412	3504	3546
0.5	3122	1009	984	1196	1969	3430	3481	3567
0.75	3133	952	1005	1516	2187	3363	3486	3548
1.0	3224	1060	1198	1354	1928	3461	3557	3604
Max-Lik	1710	1140	352	414	646	1323	3088	3272

A.4 Importance of the baseline for policy gradient

As described in Section 3.1, it is important to use a good baseline estimate for the value function for the policy optimization step. When optimizing for the expected return, we can interpret the baseline as a variance reduction technique. Intuitively, policy gradient methods adjust parameters of the policy to improve probability of trajectories in proportion to their performance. By using a baseline for the value function, we make updates that increase probability of trajectories that perform better than average and vice versa. In practice, this variance reduction is essential for getting policy gradients to work. For the CVaR case, [35] showed that without using a baseline, the policy gradient is biased. To study importance of the baseline, we first consider the case where we do not employ the adversarial sub-sampling step, and fix $\epsilon = 1$. We use a linear baseline with a time-varying feature vector as described in



(a)



(b)

Figure A.3: (a) depicts the learning curve for EPOpt($\epsilon = 1$) with and without baselines. The learning curves indicate that use of a baseline provides a better ascent direction, thereby enabling faster learning. Figure A.3(b) depicts the learning curve when using the average return and CVaR objectives. For the comparison, we "pre-train" for 100 iterations with $\epsilon = 1$ setting and using a baseline. The results indicates that a baseline is very important for the CVaR objective ($\epsilon < 1$), without which the performance drops very quickly. Here, performance is the average return in the source distribution.

Section 3.1. Figure A.3(a) presents the learning curve for the source distribution in Table 2.1. The results indicate that baselines are crucial to make policy gradients work in practice.

Next, we turn to the case of $\epsilon < 1$. As mentioned in section 3.1, setting a low ϵ from the start leads to unstable learning. The adversarial nature encourages penalizing poor trajectories more, which constrains the initial exploration needed to find promising trajectories. Thus we will "pre-train" by using $\epsilon = 1$ for some iterations, before switching to the desired ϵ setting. From Figure A.3(a), it is clear that pre-training without a baseline is unlikely to help, since the performance is poor. Thus, we use the following setup for comparison: for 100 iterations, EPOpt($\epsilon = 1$) is used with the baseline. Subsequently, we switch to EPOpt($\epsilon = 0.1$) and run for another 100 iterations, totaling 200 iterations. The results of this experiment are depicted in Figure A.3(b). This result indicates that use of a baseline is crucial for the CVaR case, without which the performance degrades very quickly. We repeated the experiment with 100 iterations of pre-training with $\epsilon = 1$ and without baseline, and observed the same effect. These empirical results reinforce the theoretical findings of [35].

Appendix B

MOREL: PROOFS AND ADDITIONAL EXPERIMENT DETAILS

B.1 Theoretical Results: Proofs For Section 3.5

In this section, we present the proofs of our main results Theorem 1 and Proposition 1.

Proof of Theorem 1. We wish to show the following two inequalities.

$$J_{\hat{\rho}_0}(\pi, \hat{\mathcal{M}}_p) \geq J_{\rho_0}(\pi, \mathcal{M}) - \frac{2R_{\max}}{1-\gamma} \cdot D_{TV}(\rho_0, \hat{\rho}_0) - \frac{2\gamma R_{\max}}{(1-\gamma)^2} \cdot \alpha - \frac{2R_{\max}}{1-\gamma} \cdot \mathbb{E}\left[\gamma^{T^\pi_{\mathcal{U}}}\right], \text{ and}$$
$$J_{\hat{\rho}_0}(\pi, \hat{\mathcal{M}}_p) \leq J_{\rho_0}(\pi, \mathcal{M}) + \frac{2R_{\max}}{1-\gamma} \cdot D_{TV}(\rho_0, \hat{\rho}_0) + \frac{2\gamma R_{\max}}{(1-\gamma)^2} \cdot \alpha.$$

The proof of this theorem is inspired by the simulation lemma of [111], with some additional modifications due to pessimism, and goes through the pessimistic MDP $\mathcal{M}_p$, which is the same as $\hat{\mathcal{M}}_p$ except that the starting state distribution is ρ_0 instead of $\hat{\rho}_0$ and the transition probability from a known state-action pair (s, a) is $P(s'|s, a)$ instead of $\widehat{P}(s'|s, a)$. More concretely, $\mathcal{M}_p$ is described by $\{S \cup \text{HALT}, A, r_p, P_p, \rho_0, \gamma\}$, where HALT is an additional absorbing state we introduce similar to what we did for $\hat{\mathcal{M}}_p$. The modified reward and transition dynamics are given by:

$$P_p(s'|s,a) = \begin{cases} \delta(s' = \text{HALT}) & \text{if } U^\alpha(s,a) = \text{TRUE} \\ & \text{or } \; s = \text{HALT} \\ P(s'|s,a) & \text{otherwise.} \end{cases} \qquad r_p(s,a) = \begin{cases} -\kappa & \text{if } s = \text{HALT} \\ r(s,a) & \text{otherwise} \end{cases}$$

We first show that

$$J_{\hat{\rho}_0}(\pi, \hat{\mathcal{M}}_p) \geq J_{\rho_0}(\pi, \mathcal{M}_p) - \frac{2R_{\max}}{1-\gamma} \cdot D_{TV}(\rho_0, \hat{\rho}_0) - \frac{2\gamma R_{\max}}{(1-\gamma)^2} \cdot \alpha, \text{ and}$$
$$J_{\hat{\rho}_0}(\pi, \hat{\mathcal{M}}_p) \leq J_{\rho_0}(\pi, \mathcal{M}_p) + \frac{2R_{\max}}{1-\gamma} \cdot D_{TV}(\rho_0, \hat{\rho}_0) + \frac{2\gamma R_{\max}}{(1-\gamma)^2} \cdot \alpha,$$

The main idea is to couple the evolutions of any given policy on the pessimistic MDP $\mathcal{M}_p$ and the model-based pessimistic MDP $\hat{\mathcal{M}}_p$ so that $(s_{t-1}, a_{t-1}) \overset{\text{def}}{=} (s^{\mathcal{M}_p}_{t-1}, a^{\mathcal{M}_p}_{t-1}) = (s^{\hat{\mathcal{M}}_p}_{t-1}, a^{\hat{\mathcal{M}}_p}_{t-1})$.

Assuming that such a coupling can be performed in the first step, since $\left\| P(s,a) - \hat{P}(s,a) \right\|_1 \leq \alpha$, this coupling can be performed at each subsequent step with probability $1 - \alpha$. The probability that the coupling is not valid at time t is at most

$1-(1-\alpha)^t$. So the total difference in the values of the policy π on the two MDPs can be upper bounded as:

$$\begin{aligned}\left|J_{\hat{\rho_0}}(\pi,\hat{\mathcal{M}}_p)-J_{\rho_0}(\pi,\mathcal{M}_p)\right| &\leq \frac{2R_{\max}}{1-\gamma}\cdot D_{TV}(\rho_0,\hat{\rho_0})+\sum_t \gamma^t\left(1-(1-\alpha)^t\right)\cdot 2\cdot R_{\max}\\ &\leq \frac{2R_{\max}}{1-\gamma}\cdot D_{TV}(\rho_0,\hat{\rho_0})+\frac{2\gamma R_{\max}}{(1-\gamma)^2}\cdot\alpha.\end{aligned}$$

We now argue that

$$\begin{aligned}&J_{\rho_0}(\pi,\mathcal{M}_p)\geq J_{\rho_0}(\pi,\mathcal{M})-\frac{2R_{\max}}{1-\gamma}\cdot\mathbb{E}\left[\gamma^{T^\pi_u}\right],\ \text{and}\\ &J_{\rho_0}(\pi,\mathcal{M}_p)\leq J_{\rho_0}(\pi,\mathcal{M}).\end{aligned}$$

For the first part, we see that the evolution of any policy π on the pessimistic MDP $\mathcal{M}_p$, can be coupled with the evolution of π on the actual MDP $\mathcal{M}$ until π encounters an unknown state. From this point, the total rewards obtained on the pessimistic MDP $\mathcal{M}_p$ will be $\frac{-R_{\max}}{1-\gamma}$, while the maximum total reward obtained by π on $\mathcal{M}$ from that point on is $\frac{R_{\max}}{1-\gamma}$. Multiplying by the discount factor $\mathbb{E}\left[\gamma^{T^\pi_u}\right]$ proves the first part.

For the second part, consider any policy π and let it evolve on the MDP $\mathcal{M}$ as $(s,a,s'_{\mathcal{M}})$. Simulate an evolution of the same policy π on $\mathcal{M}_p$, $\left(s,a,s'_{\mathcal{M}_p}\right)$, as follows: if $(s,a)\in SA_k$, then $s'_{\mathcal{M}_p}=s'_{\mathcal{M}}$ and if $(s,a)\in\mathcal{U}$, then $s'_{\mathcal{M}_p}=$ HALT. We see that the rewards obtained by π on each transition in $\mathcal{M}_p$ is less than or equal to that obtained by π on the same transition in $\mathcal{M}$. This proves the second part of the lemma. □

Lemma 4. *(Hitting time and visitation distribution) For any set $\mathcal{S}\subseteq S\times A$, and any policy π, we have $\mathbb{E}\left[\gamma^{T^\pi_{\mathcal{S}}}\right]\leq\frac{1}{1-\gamma}\cdot d^{\pi,\mathcal{M}}(\mathcal{S})$.*

Proof of Lemma 4. The proof is rather straightforward. We have

$$\begin{aligned}\mathbb{E}\left[\gamma^{T^\pi_{\mathcal{U}}}\right]&\leq\sum_{(s',a')\in\mathcal{U}}\mathbb{E}\left[\gamma^{T^\pi_{(s',a')}}\right]\leq\sum_{(s',a')\in\mathcal{U}}\sum_{t=0}^{\infty}\gamma^t P(s_t=s',a_t=a'|s_0\sim\rho_0,\pi,\mathcal{M})\\ &=\frac{1}{1-\gamma}\sum_{(s',a')\in\mathcal{U}}d^{\pi,\mathcal{M}}(s',a')=\frac{1}{1-\gamma}\cdot d^{\pi,\mathcal{M}}(\mathcal{U}).\end{aligned}$$

□

Proof of Proposition 1. We consider the MDP in Figure B.1, where we set $k=10\log\frac{1}{1-\gamma}$. The MDP has $k+1$ states, with three actions a_1,a_2 and a_3 at each state. The rewards (shown on the transition arrows) are all 0 except for the action a_1 taken in state $k+1$, in which case it is 1. Note that the rewards can be scaled by $R_{\max}$ but for simplicity, we consider the setting with

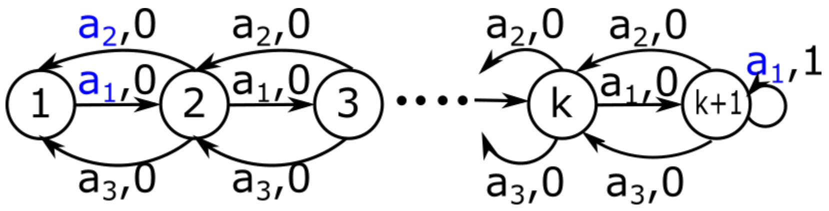


Figure B.1: This example shows that the suboptimality of any offline RL algorithm is at least $\frac{R_{\max}}{4(1-\gamma)^2} \times \frac{d^{\pi^*,\mathcal{M}}(\mathcal{U}_D)}{\log\frac{1}{1-\gamma}}$ in the worst case and hence Corollary 1 is tight. The states $1, 2, \cdots, k+1$ in the MDP are depicted under the circles. The actions a_1, a_2, a_3, rewards and transitions are depicted on the arrows connecting the states. The actions taken by the behavior (i.e. the data collection) policy are depicted in blue. See Proposition 1 and its proof for more details.

$R_{\max} = 1$. It is clear that the optimal policy π^* is to take the action a_1 in all the states. The starting state distribution ρ_0 is state 1 with probability $p_0 \overset{\text{def}}{=} \frac{\epsilon}{(1-\gamma)\log\frac{1}{1-\gamma}}$ and state $k+1$ with probability $1 - p_0$. The actions taken by the data collection policy are shown in blue. Since the dataset consists only of (state, action, reward, next state) pairs $(1, a_1, 0, 2), (2, a_2, 0, 1)$ and $(k+1, a_1, 1, k+1)$ we see that $\mathcal{U}_D = (S \times A) \setminus \{(1, a_1), (2, a_2), (k+1, a_1)\}$ and $d^{\pi^*,\mathcal{M}}(\mathcal{U}_D) = (1-\gamma)\cdot\sum_{t=1}^{k-1}\gamma^t\cdot p_0 \leq (1-\gamma)\cdot(k-1)\cdot p_0 \leq \epsilon$ proving the first claim. Since none of the states and actions in $\mathcal{U}_D$ are seen in the dataset, after permuting the actions if necessary, the expected time taken by any policy learned from the dataset, to reach state $k+1$ starting from state 1 is at least $\exp(k/5) \geq (1-\gamma)^{-2}$. So, the value of any policy $\hat{\pi}$ learned from the dataset is at most $\frac{1-p_0}{1-\gamma} + \frac{p_0\cdot\gamma^{(1-\gamma)^{-2}}}{1-\gamma} = \frac{1}{1-\gamma} - p_0 \cdot \frac{1-\gamma^{(1-\gamma)^{-2}}}{1-\gamma} \leq \frac{1}{1-\gamma} - \frac{3p_0}{4(1-\gamma)}$, where we used $\gamma \in [0.95, 1)$ in the last step. On the other hand, the value of π^* is at least $\frac{1-p_0}{1-\gamma} + p_0 \cdot \left(\frac{1}{1-\gamma} - k\right)$. So the suboptimality of any learned policy is at least $p_0 \cdot \left(\frac{3}{4(1-\gamma)} - k\right) = p_0 \cdot \left(\frac{3}{4(1-\gamma)} - 10\log\frac{1}{1-\gamma}\right) \geq \frac{p_0}{4(1-\gamma)}$, where we again used $\gamma \in [0.95, 1)$ in the last step. Substituting the value of p_0 proves the proposition. □

Proof of Lemma 1. We first note that the empirical starting distribution $\hat{\rho}_0$ satisfies $D_{TV}(\rho_0, \hat{\rho}_0) \leq \frac{C}{\rho_{0,\min}} \cdot \sqrt{\frac{\log\frac{1}{\delta\rho_{0,\min}}}{n}}$, for a large enough constant C. This is because for each state s in the support of ρ_0, its empirical frequency in $\mathcal{D}$ satisfies:

$$|\hat{\rho}_0(s) - \rho_0(s)| \leq C\sqrt{\frac{\log\frac{1}{\delta\rho_{0,\min}}}{n}},$$

with probability at least $1 - \delta\rho_{0,\min}$ using Chernoff's bound, where C is an absolute numerical constant. Using union bound over at most $\frac{1}{\rho_{0,\min}}$ states in the support of ρ_0, we see that with probability at least $1-\delta$, we have $D_{TV}(\rho_0, \hat{\rho_0}) \leq \frac{C}{\rho_{0,\min}} \cdot \sqrt{\frac{\log \frac{1}{\delta \rho_{0,\min}}}{n}}$.

Similarly, for any state action pair (s,a), denoting $n_{(s,a)}$ as the number of times (s,a) appears in $\mathcal{D}$, we have that:

$$\frac{n_{(s,a)}}{n} - d^{\pi_b,\mathcal{M}}(s,a) \geq -C\sqrt{\frac{\log \frac{1}{\delta d^{\pi_b}_{\min}}}{n}},$$

with probability at least $1 - d^{\pi_b}_{\min}\delta$. Again using a union bound over all state-action pairs in the support of $d^{\pi_b,\mathcal{M}}(\cdot)$, we see that:

$$n_{(s,a)} \geq d^{\pi_b}_{\min} \cdot n - C\sqrt{n \cdot \log \frac{1}{\delta d^{\pi_b}_{\min}}},$$

for every (s,a) in the support of $d^{\pi_b,\mathcal{M}}(\cdot)$ with probability at least $1-\delta$. The asssumption on the size of n then implies that $n_{(s,a)} \geq \frac{d^{\pi_b}_{\min} \cdot n}{2}$. Using a similar Chernoff bound argument, we see that $D_{TV}(P(\cdot|s,a), \hat{P}(\cdot|s,a)) \leq \frac{C}{p_{\min}} \cdot \sqrt{\frac{\log \frac{1}{\delta p_{\min} d^{\pi_b}_{\min}}}{n_{(s,a)}}}$ for every (s,a) in the support of $d^{\pi_b,\mathcal{M}}(\cdot)$ with probability at least $1-\delta$. By choosing $\alpha = \frac{C}{p_{\min}} \cdot \sqrt{\frac{\log \frac{1}{\delta p_{\min} d^{\pi_b}_{\min}}}{n_{(s,a)}}}$, we see that $\mathcal{U} \cap \mathrm{Supp}(d^{\pi_b,\mathcal{M}}) = \emptyset$ and hence $T^{\pi_b}_{\mathcal{U}} = \infty$. By Theorem 1, we have that for any policy π, we have:

$$\begin{aligned}
J_{\rho_0}(\pi_{\text{out}}, \mathcal{M}) &\geq J_{\hat{\rho_0}}(\pi_{\text{out}}, \hat{\mathcal{M}}_p) - \frac{2R_{\max}}{1-\gamma} \cdot D_{TV}(\rho_0, \hat{\rho_0}) - \frac{2\gamma R_{\max}}{(1-\gamma)^2} \cdot \alpha \\
&\geq J_{\hat{\rho_0}}(\pi, \hat{\mathcal{M}}_p) - \epsilon_\pi - \frac{2R_{\max}}{1-\gamma} \cdot D_{TV}(\rho_0, \hat{\rho_0}) - \frac{2\gamma R_{\max}}{(1-\gamma)^2} \cdot \alpha \\
&\geq J_{\rho_0}(\pi, \mathcal{M}) - \epsilon_\pi - \frac{4R_{\max}}{1-\gamma} \cdot D_{TV}(\rho_0, \hat{\rho_0}) - \frac{4\gamma R_{\max}}{(1-\gamma)^2} \cdot \alpha - \frac{2R_{\max}}{1-\gamma} \cdot \mathbb{E}\left[\gamma^{T^\pi_{\mathcal{U}}}\right].
\end{aligned}$$

Plugging $\pi = \pi_b$ gives us the first assertion and plugging $\pi = \pi^*$ and using Lemma 4 gives us the second assertion. □

B.2 *Additional Experimental Details And Results*

B.2.1 *Environment Details And Setup*

As mentioned before, following recent efforts in offline RL [76, 77, 79], we consider four continuous control tasks: `Hopper-v2`, `HalfCheetah-v2`, `Ant-v2`, `Walker2d-v2` from OpenAI gym [128] simulated with MuJoCo [30]. As normally done in MBRL literature with OpenAI gym tasks [89, 289, 290, 16], we reduce the planning horizon for the environments to 400 or 500. Similar to [290, 16], we append our state parameterization with center of mass velocity to compute the reward from observations. Mirroring realistic settings, we assume access to data collected using a partially trained (sub-optimal) policy interacting with the environment. To obtain a partially trained policy π_p [76, 77, 79], we run (online) TRPO [33] until the policy reaches a value of 1000, 4000, 1000, 1000 respectively for these environments. This policy in conjunction with exploration strategies are used to collect the datasets (see below for more details). All our results are obtained by averaging runs of five random seeds (for the planning algorithm), with the seed values being $123, 246, 369, 492, 615$. Each of our experiments are run with 1 NVidia GPU and 2 CPUs using a total of 16GB of memory.

B.2.2 *Dynamics Model, Policy Network And Evaluation*

We use 2 hidden layer MLPs with 512 (for `Hopper-v2`, `Walker2d-v2`, `Ant-v2`) or 1024 (for `HalfCheetah-v2`) ReLU activated nodes each for representing the dynamics model, use an ensemble of four such models for building the USAD, and our policy is represented with a 2 hidden layer MLP with 32 tanh activated nodes in each layer. The dynamics model is learnt using Adam [123] and the policy parameters are learnt using model-based NPG steps [16]. We set hyper-parameters and track policy learning curve by performing rollouts in the real environment; these rollouts aren't used for other purposes in the learning procedure. Similar protocols are used in prior work[76, 77, 79].

B.2.3 *Details about the BRAC datasets*

We build off the experimental setup of [79]. Towards this, we first go over some notation. Firstly, let π_b represent the behavior policy, π_r is a random policy that picks actions according to a certain probability distribution (for e.g., Gaussian π_r^g/Uniform π_r^u etc.), π_p a partially-trained policy, which one can assume is better than a random policy in value. Let $\pi_b^u(q)$ be a policy that plays random actions with probability q, and sampled actions from π_b with probability $1-q$. Let $\pi_b^g(\beta)$ be a policy that adds zero-mean Gaussian noise with standard deviation β to actions sampled from π_b. Consider a behavior policy which, for instance, can be a partially trained data logging policy π_b. We consider five different exploration strategies, each corresponding to adding different kinds of exploratory noise to π_b, as described below.

For each environment, we use a combination of a behavior policy π_b, a noisy behavior policy $\tilde{\pi}_b$ (see below), and a pure random stochastic process π_r to collect several datasets, following Wu et al. [79]. Each dataset contains the equivalent of 1 million timesteps of interactions with the environment. See below for detailed instructions.

($\mathcal{E}$**1**) `Pure`: The entire dataset is collected with the data logging (behavioral) policy π_b.

($\mathcal{E}$**2**) `Eps-1`: 40% of the dataset is collected with π_b, another 40% collected with $\pi_b^u(0.1)$, and the final 20% is collected with a random policy π_r.

($\mathcal{E}$**3**) `Eps-3`: 40% of the dataset is collected with π_b, another 40% collected with $\pi_b^u(0.3)$, and the final 20% is collected with a random policy π_r.

($\mathcal{E}$**4**) `Gauss-1`: 40% of the dataset is collected with π_b, another 40% collected with $\pi_b^g(0.1)$, and the final 20% is collected with a random policy π_r.

($\mathcal{E}$**5**) `Gauss-3`: 40% of the dataset is collected with π_b, another 40% collected with $\pi_b^g(0.3)$, and the final 20% is collected with a random policy π_r.

B.2.4 Hyperparameter Guidelines and Ablations

We did not have resources to perform a thorough hyperparamter search, and largely used our intuitions to guide the choice of hyperparameters. We believe that better results are possible with hyperparameter optimization. First, we present the influence of the discrepancy threshold for differentiating known and unknown states. We first define the maximum discipancy in the dataset:

$$\text{disc}_{\mathcal{D}} = \max_{(s,a)\in\mathcal{D}} \max_{i,j} \|f_i(s,a) - f_j(s,a)\|$$

where $\mathcal{D}$ denotes offline dataset, and f_i denotes i^{th} dynamics model in the ensemble.

Table B.1: Influence of discrepancy threshold on the `Hopper-v2` task. We use a penalty of 0.0 along with episode termination for visiting unknown regions in these experiments. We train all the cases for 1000 iterations, and report the average value over the last 100 iterations.

Discrepancy Threshold	Value in P-MDP	Value in true MDP
$0.1 \times \text{disc}_{\mathcal{D}}$	1315.16	2082.21
$0.2 \times \text{disc}_{\mathcal{D}}$	2479.92	3244.48
$0.5 \times \text{disc}_{\mathcal{D}}$	3074.75	3359.66
$1.0 \times \text{disc}_{\mathcal{D}}$	3543.23	3595.60
$5.0 \times \text{disc}_{\mathcal{D}}$	3245.66	3027.59
Naive-MBRL	3656.08	2809.66

Our general observations and guidelines for hyperparameters are:

1. In Table B.1, we first note that $0.1 \times \text{disc}_{\mathcal{D}}$ has the most amount of pessimism and Naive-MBRL has the least/no amount of pessimism. We observe that we obtain best results in the true MDP with an intermediate level of pessimism. Having either too much pessimism or no pessimism both lead to poor results, but for very different reasons that we outline below.

2. A high degree of pessimism makes policy optimization in the P-MDP difficult. The optimization process may be slow or highly noisy. This is due to non-smoothness introduced in the dynamics and reward due to abrupt changes involving early episode terminations. If difficulty in policy optimization is observed in the P-MDP, we recommend considering reducing the degree of pessimism.

3. With a lack or low degree of pessimism, policy optimization is typically easier, but the performance in the true MDP might degrade. If it is observed that the value in the P-MDP overestimates the value in the true MDP substantially, then we recommend increasing the degree of pessimism.

4. For the tasks considered in this work, positive rewards indicate progress towards the goal. Most of the locomotion tasks involve forward velocity as the primary component of the the reward term. In these cases, we observed that the choice of reward penalty for going into unknown regions did not play a crucial role, as long as it was ≤ 0. The degree of influence of this parameter in other environments is yet to be determined, and beyond the scope of our empirical study.

Appendix C

GAME-MBRL: PROOFS AND EXPERIMENT DETAILS

C.1 Theoretical Results

Lemma 1 restated. *(Simulation lemma) Suppose we have a model* $\widehat{\boldsymbol{M}}$ *such that*

$$D_{TV}(P_{\boldsymbol{M}}(\cdot|s,a), P_{\widehat{\boldsymbol{M}}}(\cdot|s,a)) \leq \epsilon_{\boldsymbol{M}} \quad \forall (s,a),$$

and the reward function is such that $|\mathcal{R}(s)| \leq R_{\max}$ $\forall s \in \mathcal{S}$*. Then, we have*

$$\left| J(\pi, \boldsymbol{M}) - J(\pi, \widehat{\boldsymbol{M}}) \right| \leq \frac{2\gamma \epsilon_{\boldsymbol{M}} R_{\max}}{(1-\gamma)^2} \quad \forall \pi$$

Proof. Let $V^\pi(s, \boldsymbol{M})$ and $V^\pi(s, \widehat{\boldsymbol{M}})$ denote the value of policy π starting from an arbitrary state $s \in \mathcal{S}$ in $\boldsymbol{M}$ and $\widehat{\boldsymbol{M}}$ respectively. For simplicity of notation, we also define

$$P^\pi_{\boldsymbol{M}}(s'|s) := \mathbb{E}_{a\sim\pi(\cdot|s)}\left[P_{\boldsymbol{M}}(s'|s,a)\right] \quad \text{and} \quad P^\pi_{\widehat{\boldsymbol{M}}}(s'|s) := \mathbb{E}_{a\sim\pi(\cdot|s)}\left[P_{\widehat{\boldsymbol{M}}}(s'|s,a)\right].$$

Before the proof, we note the following useful observations.

1. Since $D_{TV}(P_{\boldsymbol{M}}(\cdot|s,a), P_{\widehat{\boldsymbol{M}}}(\cdot|s,a)) \leq \epsilon_{\boldsymbol{M}}$ $\forall (s,a)$, the inequality also holds for an average over actions, i.e. $D_{TV}(P^\pi_{\boldsymbol{M}}(\cdot|s), P^\pi_{\widehat{\boldsymbol{M}}}(\cdot|s)) \leq \epsilon_{\boldsymbol{M}}$ $\forall s$.

2. Since the rewards are bounded, we can achieve a maximum reward of $R_{\max}$ in each time step. Using a geometric summation with discounting γ, we have

$$\max_{s\in\mathcal{S}} V^\pi(s, \boldsymbol{M}) \leq \frac{R_{\max}}{1-\gamma} \quad \forall \pi, s$$

3. Let $f(x) : x \in \mathcal{X} \to [-f_{\max}, f_{\max}]$ be a real-valued function with bounded range, i.e. $0 \leq f_{\max} < \infty$. Let $P_1(x)$ and $P_2(x)$ be two probability distribution (density) over the space $\mathcal{X}$. Then, we have

$$\left|\mathbb{E}_{x\sim P_1(\cdot)}[f(x)] - \mathbb{E}_{x\sim P_2(\cdot)}[f(x)]\right| \leq 2 f_{\max}\, D_{TV}(P_1, P_2)$$

Using the above observations, we have the following inequalities:

$$
\begin{aligned}
&\left|V^\pi(s,\boldsymbol{M}) - V^\pi(s,\widehat{\boldsymbol{M}})\right| \\
&= \left|\mathcal{R}(s) + \gamma \mathbb{E}_{s'\sim P^\pi_{\boldsymbol{M}}(\cdot|s)}\left[V^\pi(s',\boldsymbol{M})\right] - \mathcal{R}(s) - \gamma \mathbb{E}_{s'\sim P^\pi_{\widehat{\boldsymbol{M}}}(\cdot|s)}\left[V^\pi(s',\widehat{\boldsymbol{M}})\right]\right| \\
&\leq \gamma \left|\mathbb{E}_{s'\sim P^\pi_{\boldsymbol{M}}(\cdot|s)}\left[V^\pi(s',\boldsymbol{M})\right] - \mathbb{E}_{s'\sim P^\pi_{\widehat{\boldsymbol{M}}}(\cdot|s)}\left[V^\pi(s',\boldsymbol{M})\right]\right| + \\
&\quad \gamma \left|\mathbb{E}_{s'\sim P^\pi_{\widehat{\boldsymbol{M}}}(\cdot|s)}\left[V^\pi(s',\boldsymbol{M}) - V^\pi(s',\widehat{\boldsymbol{M}})\right]\right| \\
&\leq 2\gamma \left(\max_{s'\in\mathcal{S}} \; V^\pi(s',\boldsymbol{M})\right) D_{TV}(P^\pi_{\boldsymbol{M}}(\cdot|s), P^\pi_{\widehat{\boldsymbol{M}}}(\cdot|s)) + \gamma \max_{s'\in\mathcal{S}} \left|V^\pi(s',\boldsymbol{M}) - V^\pi(s',\widehat{\boldsymbol{M}})\right|
\end{aligned}
$$

Since the above bound holds for all states, we have that $\forall \pi$

$$
\begin{aligned}
(1-\gamma)\max_{s'\in\mathcal{S}} \left|V^\pi(s',\boldsymbol{M}) - V^\pi(s',\widehat{\boldsymbol{M}})\right| &\leq 2\gamma \left(\max_{s' in \mathcal{S}} \; V^\pi(s',\boldsymbol{M})\right) D_{TV}(P^\pi_{\boldsymbol{M}}(\cdot|s), P^\pi_{\widehat{\boldsymbol{M}}}(\cdot|s)) \\
&\leq \frac{2\gamma R_{\max}}{1-\gamma} D_{TV}(P^\pi_{\boldsymbol{M}}(\cdot|s), P^\pi_{\widehat{\boldsymbol{M}}}(\cdot|s)) \\
&\leq \frac{2\gamma \epsilon_{\boldsymbol{M}} R_{\max}}{1-\gamma}
\end{aligned}
$$

Stated alternatively, the above inequality implies

$$
\left|V^\pi(s,\boldsymbol{M}) - V^\pi(s,\widehat{\boldsymbol{M}})\right| \leq \frac{2\gamma \epsilon_{\boldsymbol{M}} R_{\max}}{(1-\gamma)^2} \quad \forall s, \pi
$$

Finally, note that the performance criteria $J(\pi, \widehat{\boldsymbol{M}})$ and $J(\pi, \boldsymbol{M})$ are simply the average of the value function over the initial state distribution. Since the above inequality holds for all states, it also holds for the average over initial state distribution. □

We note that the above simulation lemma (or closely related forms) have been proposed and proved several times in prior literature (e.g. see [113, 291]). We present the proof largely for completeness and also to motivate the proof techniques we will use for our main theoretical result (Theorem 2).

C.1.1 Performance with Task-Driven Local Models

We now relax the global model requirement and consider the case where we have more local models, as well as the case of a policy-model equilibrium pair. We first provide a lemma that characterizes error amplification in local simulation.

Lemma 5. *(Error amplification in local simulation) Let $P_1(\cdot|s)$ and $P_2(\cdot|s)$ be two Markov chains with the same initial state distribution. Let $P_1^t(s)$ and $P_2^t(s)$ be the marginal distributions over states at time t when following P_1 and P_2 respectively. Suppose*

$$
\mathbb{E}_{s\sim P_1^t}\left[D_{TV}(P_1(\cdot|s), P_2(\cdot|s))\right] \leq \epsilon \;\; \forall \, t
$$

then, the marginal distributions are bounded as:

$$D_{TV}(P_1^t, P_2^t) \leq \epsilon t \quad \forall\ t$$

Proof. Let us fix a state $s \in \mathcal{S}$, and let $\bar{s} \in \mathcal{S}$ denote a "dummy" state variable. Then,

$$\begin{aligned}
\left|P_1^t(s) - P_2^t(s)\right| &= \left|\sum_{\bar{s}\in\mathcal{S}} P_1(s|\bar{s}) P_1^{t-1}(\bar{s}) - \sum_{\bar{s}\in\mathcal{S}} P_2(s|\bar{s}) P_2^{t-1}(\bar{s})\right| \\
&\leq \sum_{\bar{s}\in\mathcal{S}} \left|P_1(s|\bar{s}) P_1^{t-1}(\bar{s}) - P_2(s|\bar{s}) P_2^{t-1}(\bar{s})\right| \\
&\leq \sum_{\bar{s}\in\mathcal{S}} \left|P_1^{t-1}(\bar{s})\big(P_1(s|\bar{s}) - P_2(s|\bar{s})\big)\right| + \left|P_2(s|\bar{s})\big(P_1^{t-1}(\bar{s}) - P_2^{t-1}(\bar{s})\big)\right|
\end{aligned}$$

Using the above inequality, we have

$$\begin{aligned}
2D_{TV}(P_1^t, P_2^t) &= \sum_{s\in\mathcal{S}} \left|P_1^t(s) - P_2^t(s)\right| \\
&\leq \sum_{\bar{s}\sim\mathcal{S}} P_1^{t-1}(\bar{s}) \sum_{s\in\mathcal{S}} \left|P_1(s|\bar{s}) - P_2(s|\bar{s})\right| + \sum_{\bar{s}\in\mathcal{S}} \left|P_1^{t-1}(\bar{s}) - P_2^{t-1}(\bar{s})\right| \\
&\leq 2\epsilon + 2D_{TV}(P_1^{t-1}, P_2^{t-1}) \\
&\leq 2t\epsilon
\end{aligned}$$

where the last step uses the previous inequality recursively till $t = 0$, where the Markov chains have the same (initial) state distribution. $\square$

The above lemma considers the error between two Markov chains. Note that fixing a policy in an MDP results in a Markov chain transition dynamics. Thus, fixing the policy, we can use the above lemma to compare the resulting Markov chains in $\boldsymbol{M}$ and $\widehat{\boldsymbol{M}}$. Consider the following definitions:

$$\begin{aligned}
\mu^{\pi}_{\widehat{\boldsymbol{M}}}(s,a) &= \frac{1}{T_\infty} \sum_{t=0}^{T_\infty} P(s_t = s, a_t = a) \\
\tilde{\mu}^{\pi}_{\widehat{\boldsymbol{M}}}(s,a) &= (1-\gamma) \sum_{t=0}^{\infty} \gamma^t P(s_t = s, a_t = a)
\end{aligned}$$

The first distribution $\mu^{\pi}_{\widehat{\boldsymbol{M}}}$ is the average state visitation distribution when executing π in $\widehat{\boldsymbol{M}}$, and T_∞ is the episode duration (could tend to ∞ in the non-episodic case). The second distribution $\tilde{\mu}^{\pi}_{\widehat{\boldsymbol{M}}}$ is the discounted state visitation distribution when executing π in $\widehat{\boldsymbol{M}}$. Let $\mu^{\pi}_{\boldsymbol{M}}$ and $\tilde{\mu}^{\pi}_{\boldsymbol{M}}$ be their analogues in $\boldsymbol{M}$. When learning the dynamics model, we would minimize the prediction error under $\mu^{\pi}_{\boldsymbol{M}}$, while $J(\pi, \boldsymbol{M})$ is dependent on rewards under $\tilde{\mu}^{\pi}_{\boldsymbol{M}}$. Let

$$\mu^{\pi,t}_{\boldsymbol{M}}(s,a) = P(s_t = s, a_t = a)$$

be the marginal distribution at time t when following π in $\boldsymbol{M}$. Let $\mu^{\pi,t}_{\widehat{\boldsymbol{M}}}(s,a)$ be analogously defined when following π in $\widehat{\boldsymbol{M}}$. Using these definitions, we first characterize the difference in performance of the same policy π under $\boldsymbol{M}$ and $\widehat{\boldsymbol{M}}$.

Lemma 6. *(Performance difference due to model error) Let $\boldsymbol{M}$ and $\widehat{\boldsymbol{M}}$ be two different MDPs differing only in their transition dynamics – $P_{\boldsymbol{M}}$ and $P_{\widehat{\boldsymbol{M}}}$. Let the absolute value of rewards be bounded by $R_{\max}$. Fix a policy π for both $\boldsymbol{M}$ and $\widehat{\boldsymbol{M}}$, and let $P^t_{\boldsymbol{M}}$ and $P^t_{\widehat{\boldsymbol{M}}}$ be the resulting marginal state distributions at time t. If the MDPs are such that*

$$\mathbb{E}_{(s,a)\sim\mu^{\pi,t}_{\boldsymbol{M}}}\left[D_{TV}\big(P_{\boldsymbol{M}}(\cdot|s,a), P_{\widehat{\boldsymbol{M}}}(\cdot|s,a)\big)\right] \leq \epsilon \;\; \forall t$$

then, the performance difference is bounded as:

$$\left|J(\pi,\boldsymbol{M}) - J(\pi,\widehat{\boldsymbol{M}})\right| \leq \frac{2\gamma\epsilon R_{\max}}{(1-\gamma)^2}$$

Proof. Recall that the performance of a policy can be written as:

$$J(\pi,\widehat{\boldsymbol{M}}) = \frac{1}{1-\gamma}\mathbb{E}_{\tilde{\mu}^{\pi}_{\widehat{\boldsymbol{M}}}}\left[\mathcal{R}(s)\right] = \mathbb{E}\left[\sum_{t=0}^{\infty}\gamma^t\mathcal{R}(s_t)\right]$$

where the randomness for the second term is due to $\widehat{\boldsymbol{M}}$ and π. We can analogously write $J(\pi,\boldsymbol{M})$ as well. Thus, the performance difference can be bounded as:

$$\begin{aligned}\left|J(\pi,\boldsymbol{M}) - J(\pi,\widehat{\boldsymbol{M}})\right| &= \left|\frac{1}{1-\gamma}\mathbb{E}_{\tilde{\mu}^{\pi}_{\boldsymbol{M}}}\left[\mathcal{R}(s)\right] - \frac{1}{1-\gamma}\mathbb{E}_{\tilde{\mu}^{\pi}_{\widehat{\boldsymbol{M}}}}\left[\mathcal{R}(s)\right]\right| \\ &\leq \frac{2R_{\max}}{1-\gamma}D_{TV}\big(\tilde{\mu}^{\pi}_{\boldsymbol{M}}, \tilde{\mu}^{\pi}_{\widehat{\boldsymbol{M}}}\big)\end{aligned}$$

Also recall that we have

$$\mu^{\pi,t}_{\widehat{\boldsymbol{M}}}(s,a) = P(s_t=s, a_t=a) = P^t_{\widehat{\boldsymbol{M}}}(s)\pi(a|s)$$

We can bound the discounted state visitation distribution as

$$\begin{aligned}2D_{TV}\big(\tilde{\mu}^{\pi}_{\boldsymbol{M}}, \tilde{\mu}^{\pi}_{\widehat{\boldsymbol{M}}}\big) &= \sum_{s,a}\left|\tilde{\mu}^{\pi}_{\boldsymbol{M}}(s,a) - \tilde{\mu}^{\pi}_{\widehat{\boldsymbol{M}}}(s,a)\right| \\ &= (1-\gamma)\sum_{s,a}\left|\sum_t \gamma^t\mu^{\pi,t}_{\boldsymbol{M}}(s,a) - \gamma^t\mu^{\pi,t}_{\widehat{\boldsymbol{M}}}(s,a)\right| \\ &\leq (1-\gamma)\sum_{s,a}\sum_t\gamma^t\left|\mu^{\pi,t}_{\boldsymbol{M}}(s,a) - \mu^{\pi,t}_{\widehat{\boldsymbol{M}}}(s,a)\right| \\ &= (1-\gamma)\sum_s\sum_t\gamma^t\left|P^t_{\boldsymbol{M}}(s) - P^t_{\widehat{\boldsymbol{M}}}(s)\right| \\ &\leq (1-\gamma)\sum_{t=0}^{\infty}\gamma^t\left(2t\epsilon\right)\end{aligned}$$

where the last inequality uses Lemma 5. Notice that the final summation is an arithmetico-geometric series. When simplified, this results in

$$D_{TV}\left(\tilde{\mu}^{\pi}_{\boldsymbol{M}}, \tilde{\mu}^{\pi}_{\widehat{\boldsymbol{M}}}\right) \leq (1-\gamma)\frac{\epsilon\gamma}{(1-\gamma)^2} \leq \frac{\epsilon\gamma}{1-\gamma}$$

Using this bound for the performance difference yields the desired result. □

Remarks: The performance difference (due to model error) lemma we present is quite distinct and different from the performance difference lemma from Kakade and Langford [55]. Specifically, our lemma bounds the performance difference between the *same policy* in two *different models.* In contrast, the lemma from Kakade and Langford [55] characterizes the performance difference between two *different policies* in the *same model.*

C.1.2 Proof of main theorem

Armed with the above results, we are now ready to prove the main theorem which characterizes the global performance when we have a policy-model pair close to equilibrium.

Theorem 2 restated. *(Global performance of equilibrium pair) Suppose we have policy-model pair* $(\pi, \widehat{\boldsymbol{M}})$ *such that the following conditions hold simultaneously:*

$$\mathcal{L}(\widehat{\boldsymbol{M}}, \mu^{\pi,t}_{\boldsymbol{M}}) \leq \epsilon_{\boldsymbol{M}} \;\forall t \quad \textit{and} \quad J(\pi, \widehat{\boldsymbol{M}}) \geq \sup_{\pi'} J(\pi', \widehat{\boldsymbol{M}}) - \epsilon_{\pi}.$$

Let π^* *be an optimal policy so that* $J(\pi^*, \boldsymbol{M}) \geq J(\pi', \boldsymbol{M})\; \forall \pi'$. *Then, we have*

$$J(\pi^*, \boldsymbol{M}) - J(\pi, \boldsymbol{M}) \leq \frac{2\gamma\sqrt{\epsilon_{\boldsymbol{M}}}R_{\max}}{(1-\gamma)^2} + \epsilon_{\pi} + \frac{2R_{\max}}{1-\gamma} D_{TV}\left(\tilde{\mu}^{\pi^*}_{\boldsymbol{M}}, \tilde{\mu}^{\pi^*}_{\widehat{\boldsymbol{M}}}\right).$$

Proof. We first simplify the performance difference, and subsequently bound the different terms. Let $\pi^*_{\widehat{\boldsymbol{M}}}$ to be an optimal policy in the model, so that $J(\pi^*_{\widehat{\boldsymbol{M}}}, \widehat{\boldsymbol{M}}) \geq J(\pi', \widehat{\boldsymbol{M}})\; \forall \pi'$. We can decompose the performance difference due to various contributions as:

$$\begin{aligned} J(\pi^*, \boldsymbol{M}) - J(\pi, \boldsymbol{M}) &= J(\pi^*, \boldsymbol{M}) - J(\pi^*, \widehat{\boldsymbol{M}}) + J(\pi^*, \widehat{\boldsymbol{M}}) - J(\pi, \boldsymbol{M}) \\ &= \underbrace{J(\pi^*, \boldsymbol{M}) - J(\pi^*, \widehat{\boldsymbol{M}})}_{\text{Term-I}} + \underbrace{J(\pi^*, \widehat{\boldsymbol{M}}) - J(\pi, \widehat{\boldsymbol{M}})}_{\text{Term-II}} + \underbrace{J(\pi, \widehat{\boldsymbol{M}}) - J(\pi, \boldsymbol{M})}_{\text{Term-III}} \end{aligned}$$

Let us first consider **Term-II**, which is related to the sub-optimality in the planning problem. Notice that we have:

$$J(\pi^*, \widehat{\boldsymbol{M}}) - J(\pi, \widehat{\boldsymbol{M}}) = \left(J(\pi^*, \widehat{\boldsymbol{M}}) - J(\pi^*_{\widehat{\boldsymbol{M}}}, \widehat{\boldsymbol{M}})\right) + \left(J(\pi^*_{\widehat{\boldsymbol{M}}}, \widehat{\boldsymbol{M}}) - J(\pi, \widehat{\boldsymbol{M}})\right) \leq 0 + \epsilon_{\pi}$$

We have $J(\pi^*, \widehat{\boldsymbol{M}}) - J(\pi^*_{\widehat{\boldsymbol{M}}}, \widehat{\boldsymbol{M}}) \leq 0$ since $\pi^*_{\widehat{\boldsymbol{M}}}$ is the optimal policy in the model, and we have $J(\pi^*_{\widehat{\boldsymbol{M}}}, \widehat{\boldsymbol{M}}) - J(\pi, \widehat{\boldsymbol{M}}) \leq \epsilon_\pi$ due to the approximate equilibrium condition.

For **Term-III**, we will draw upon the model error performance difference lemma (Lemma 6). Note that the equilibrium condition of low error along with Pinsker's inequality implies

$$\mathbb{E}_{s \sim \mu^{\pi,t}_{\boldsymbol{M}}} \left[D_{TV}\left(P_{\boldsymbol{M}}(\cdot|s,a), P_{\widehat{\boldsymbol{M}}}(\cdot|s,a) \right) \right] \leq \sqrt{\epsilon_{\boldsymbol{M}}}$$

Using this and Lemma 6, we have

$$J(\pi, \widehat{\boldsymbol{M}}) - J(\pi, \boldsymbol{M}) \leq \frac{2\gamma\sqrt{\epsilon_{\boldsymbol{M}}} R_{\max}}{(1-\gamma)^2}$$

Finally, **Term-I** is a transfer learning term that measures the error of $\widehat{\boldsymbol{M}}$ (which has low error under π) under the distribution of π^*. The performance difference can be written as

$$\begin{aligned} J(\pi^*, \boldsymbol{M}) - J(\pi^*, \widehat{\boldsymbol{M}}) &= \frac{1}{1-\gamma} \mathbb{E}_{(s,a) \sim \tilde{\mu}^{\pi^*}_{\boldsymbol{M}}} \left[\mathcal{R}(s) \right] - \frac{1}{1-\gamma} \mathbb{E}_{(s,a) \sim \tilde{\mu}^{\pi^*}_{\widehat{\boldsymbol{M}}}} \left[\mathcal{R}(s) \right] \\ &\leq \frac{2R_{\max}}{1-\gamma} D_{TV}\left(\tilde{\mu}^{\pi^*}_{\boldsymbol{M}}, \tilde{\mu}^{\pi^*}_{\widehat{\boldsymbol{M}}} \right) \end{aligned}$$

Putting all the terms together, we have

$$J(\pi^*, \boldsymbol{M}) - J(\pi, \boldsymbol{M}) \leq \frac{2R_{\max}}{1-\gamma} D_{TV}\left(\tilde{\mu}^{\pi^*}_{\boldsymbol{M}}, \tilde{\mu}^{\pi^*}_{\widehat{\boldsymbol{M}}} \right) + \epsilon_\pi + \frac{2\gamma\sqrt{\epsilon_{\boldsymbol{M}}} R_{\max}}{(1-\gamma)^2}$$

□

Remarks: Tighter bounds on the transfer learning term is not possible without additional assumptions. However, the spirit of the transfer learning issue is captured by the term.

1. It suggests that there is a preference hierarchy between models that achieve similar low error under $\mu^{\pi}_{\boldsymbol{M}}$. The models that can simulate a wider class of policies (i.e. have better transfer) are preferable for MBRL. This establishes a concrete connection between MBRL and domain adaptation, and we hope that various ideas from transfer learning and domain adaptation [151] can benefit MBRL.

2. The structure of $\mu^{\pi^*}_{\boldsymbol{M}}$ provides avenues to achieve better transfer. Note that the start state distribution is the same for $\boldsymbol{M}$ and $\widehat{\boldsymbol{M}}$, and is also shared by all policies. Thus, if we could design or choose the start state distribution to be wide, it automatically ensures good mixing between $\mu^{\pi^*}_{\boldsymbol{M}}$ and $\mu^{\pi^*}_{\widehat{\boldsymbol{M}}}$ by virtue of them sharing the start state distribution. We could obtain such a distribution by training an exploratory policy [150, 167], and executing it for a few steps to construct a starting state distribution.

The theorem assumes that the errors are small at each timestep: $\mathcal{L}(\widehat{\boldsymbol{M}}, \mu_{\boldsymbol{M}}^{\pi,t}) \leq \epsilon_{\boldsymbol{M}}\ \forall t$. This is only a slightly stronger assumption than the average error being small. In practice, by executing the policy, we would have a dataset drawn from $\mu_{\boldsymbol{M}}^{\pi}$. Thus, it should be possible to make the error small under $\mu_{\boldsymbol{M}}^{\pi}$. Recall that $\mu_{\boldsymbol{M}}^{\pi} = (1/T_\infty)\sum_{t=0}^{T_\infty} \mu_{\boldsymbol{M}}^{\pi,t}$. If we use an expressive function approximator, and if there is sufficient concentration of measure, small error over $\mu_{\boldsymbol{M}}^{\pi}$ would lead to small error at each timestep. Furthermore, since we typically store time-indexed trajectories, we can check in practice that the error is small at each timestep.

C.2 Algorithm Implementation Details and Experiments

Our implementation builds on top of MJRL (`https://github.com/aravindr93/mjrl`) for NPG [12, 170] and interfacing with MuJoCo/OpenAI-gym [30, 128]. We adapt the NPG implementation to work with learned models. Our model learning minimizes one-step prediction error using Adam. We first describe the details of these subroutines before describing the full algorithms.

Policy Details We represent the policy as a neural network, and use the learned model for performing synthetic rollouts as specified in Subroutine 1. For the set of initial states, we can either sample from the initial state distribution of MDP (if it is known) or keep track of initial states from the environment in a separate initial state replay buffer. We found both to perform near-identically. Furthermore, the synthetic rollouts can be started from either the initial state distribution of the MDP, or from intermediate states in real rollouts. We found starting 50% of synthetic rollouts from intermediate (real-world) rollout states leads to better asymptotic results for longer horizon gym tasks. This is consistent with prior works that suggest sampling from a wide initial state distribution is beneficial for policy gradient methods [55, 12]. The subroutine is written assuming a reward oracle, which can either be a known function, or can be learned from data. We found both settings to work near-identically, since rewards are often simple functions of the state-action and substantially easier to learn than dynamics. We consider a maximum rollout horizon of 500, which can become shorter if the maximum environment horizon is smaller, or if termination conditions kick in for the rollouts. If the environments have termination conditions, we enforce these for the synthetic rollouts as well. Finally, we use a baseline/value network for the purposes of variance reduction [292, 293] – specifically GAE [294]. We use the default values for most parameters as summarized in Table C.1, and do not tune them.

Model details We model the MDP dynamics with ensembles of neural network dynamics models. Ensembles capture epistemic uncertainty [147] and provide robustness for policy optimization [13]. We are provided with a dataset of tuples $\mathcal{D} = \{(s_t, a_t, s_{t+1})\}$, and we

Subroutine 1 Model-Based Natural Policy Gradient Update Step

```
1: Require: Policy (stochastic) network $\pi_\theta$, value/baseline network $V_\psi$, ensemble of MDP dynamics models $\{\widehat{\boldsymbol{M}}_\phi\}$, reward function $\mathcal{R}$, initial state distribution or buffer.
2: Hyperparameters: Discount factor $\gamma$, GAE $\lambda$, number of trajectories $N_\tau$, rollout horizon $H$, normalized NPG step size $\delta$
3: Initialize trajectory buffer $\mathcal{D}_\tau = \{\}$
4: for $k = 1, 2, \ldots, N_\tau$ do
5:     Sample initial state $s_0^k$ from initial state distribution/buffer
6:     Perform $H$ step rollout from $s_0^k$ with $\pi_\theta$ to get $\tau_j^k = (s_0^k, a_0^k, s_1^k, a_2^k, \ldots s_H^k, a_H^k)$, one for each model $\widehat{\boldsymbol{M}}_\phi^j$ in the ensemble.
7:     Query reward function to obtain rewards for each step of the trajectories
8:     Truncate trajectories if termination/truncation conditions are part of the environment
9:     Aggregate the trajectories in trajectory buffer, $\mathcal{D}_\tau = \mathcal{D}_\tau \cup \{\tau\}$
10: end for
11: Compute advantages for each trajectory using $V_\psi$ and GAE [294].
12: Compute vanilla policy gradient using the dataset
        $$g = \mathbb{E}_{(s,a)\sim\mathcal{D}_\tau}\left[\nabla_\theta \log \pi_\theta(a|s) A^\pi(s,a)\right]$$
13: Perform normalized NPG update ($F$ denotes the Fisher matrix)
        $$\theta = \theta + \sqrt{\frac{\delta}{g^T F^{-1} g}}\; F^{-1} g$$
14: Update value/baseline network $V_\psi$ to fit the computed returns in $\mathcal{D}_\tau$.
15: Return Policy network $\pi_\theta$, value network $V_\psi$
```

parameterize the model as:

$$\widehat{\boldsymbol{M}}_\phi(s_t, a_t) = s_t + \sigma_\Delta \; MLP_\phi\left(\frac{s_t - \mu_s}{\sigma_s}, \frac{a_t - \mu_a}{\sigma_a}\right)$$

where we $\Delta_t = s_{t+1} - s_t$ are the state differences, and mean centering and scaling are performed based on the dataset. We solve the following optimization problem to learn the parameters:

$$\min_\phi \;\; \mathbb{E}_{(s_t, a_t, s_{t+1})\sim\mathcal{D}}\left[\left\|(s_{t+1} - s_t) - \sigma_\Delta \; MLP_\phi\left(\frac{s_t - \mu_s}{\sigma_s}, \frac{a_t - \mu_a}{\sigma_a}\right)\right\|^2\right].$$

We specify the important hyperparameters along with PAL and MAL descriptions. When training, we also ensure that at-least 10^2 gradient steps and at-most 10^5 gradient steps are used, to avoid boundary issues when the buffer size is too small or large.

Table C.1: Hyperparameters used for policy improvement with NPG

Parameter	**Value**
Policy network	MLP (64, 64)
Value/baseline network	MLP (128, 128)
Discount γ	0.995
GAE λ	0.97
# synthetic trajectories (N_τ)	200
Rollout horizon (H)	min (env-horizon, 500, termination)
normalized step size δ	0.05

Policy As Leader: The practical version of the PAL-NPG algorithm is provided below. The algorithm alternates between collecting a small amount of data in each iteration, learning a dynamics model, and conservatively improving the policy. We use a small replay buffer to aggregate data from the past few iterations, but the replay buffer is kept small in size to ensure that the model is primarily trained to be accurate under current state visitation.

Algorithm 10 Policy As Leader (PAL) – Practical Version

```
1: Initialize: Policy network $\pi_0$, model network(s) $\widehat{\boldsymbol{M}}_0$, value network $V_0$.
2: Hyperparameters: Initial samples $N_{init}$, samples per update $N$, buffer size $B \approx N$, number of NPG steps $K \approx 1$
3: Initial Data: Collect $N_{init}$ samples from the environment by interacting with initial policy. Store data in buffer $\mathcal{D}$.
4: for $k = 0, 1, 2, \ldots$ do
5:     Learn dynamics model(s) $\widehat{\boldsymbol{M}}_{k+1}$ using data in the buffer.
6:     Policy updates: $\pi_{k+1}, V_{k+1} =$ Model-Based NPG($\pi_k, V_k, \widehat{\boldsymbol{M}}_{k+1}$) // call K times
7:     Collect dataset of $N$ samples from world by interacting with $\pi_{k+1}$. Add data to replay buffer $\mathcal{D}$, discarding old data if size is larger than $B$.
8: end for
```

For hyperparameter selection, we performed a coarse search on DClaw task and used the same parameters for the remaining tasks with minor changes. The main parameters we focused on were the number of NPG updates per iteration, for which we tried $K = \{1, 2, 4, 8\}$ and found 4 to be best. Similarly, we studied number of samples per iteration $N = \{1, 5, 10, 20\} \times$env-horizon, and found 5 to be ideal for DClaw, DKitty, Reacher, and Hand tasks. For the hand task, $N = 10 \times$horizon produced more stable results, and we report results with this choice. The OpenAI gym tasks are longer horizon and we found fewer samples are sufficient. For the gym tasks, we use $N = 1000$ samples per iteration, which towards the later half of training often amounts to only one trajectory. We use a buffer of size $B = 2500$, which often amounts to using data from the past 2-5 iterations. We also use ensembles of dynamics models and we tried ensemble sizes of $\{1, 2, 4, 8\}$ and found 4 to be a good trade-off between performance and computation. We also found it important to initialize the policy with sufficient small amount of exploratory noise to avoid stability and divergence issues. We consider Gaussian policies with diagonal covariance where the neural network parameterizes the mean, and the diagonal co-variance is also learned. We initialize the standard deviation as $\sigma = \exp(-1)$. We do not add any additional exploratory noise when collecting data, but simply use the learned covariance in the Gaussian policy. We summarize the details in Table C.2.

Table C.2: Hyperparameters used for the PAL-NPG algorithm

Parameter	**Value**
Model network	MLP (512, 512)
Learning algorithm	Adam (default parameters)
No. of epochs	100
Mini-batch size	200
Ensemble size	4
Buffer size B	2500
Initial samples (N_{init})	2500
Samples per iteration (N)	min(5×env-horizon, 1000)
NPG updates (K)	4

Model As Leader: The practical version of the MAL-NPG algorithm is provided in Algorithm 10. The algorithm alternates between optimizing a policy using current model, collecting additional data which is aggregated into a data buffer, and finally improving the model using the aggregated data. For hyperparameter selection, we follow the same overall approach as described in MAL. Compared to PAL, the main differences are that a larger number of initial samples are required, since the policy is optimized aggressively. We tried $K = \{10, 25, 40, 60\}$ and found $K = 25$ to be a good trade-off between performance and computation. We tried $N = \{1, 5, 10, 20\} \times$env-horizon and found $N = 20\times$horizon to provide the best results. For the OpenAI gym tasks, we used $N = 3000$ samples. For the simpler Pendulum task, we use fewer samples which still leads to stable results. We again use an ensemble of 4 models. The hyperparameter details are summarized in Table C.3.

Algorithm 11 Model As Leader (MAL) – Practical Version

```
1: **Initialize:** Policy network $\pi_0$, model network(s) $\widehat{\boldsymbol{M}}_0$, value network $V_0$.
2: **Hyperparameters:** Initial samples $N_{init}$, samples per update $N$, number of NPG steps $K \gg 1$
3: **Initial Data:** Collect $N_{init}$ samples from the environment by interacting with initial policy. Store data in buffer $\mathcal{D}$.
4: **Initial Model:** Learn model(s) $\widehat{\boldsymbol{M}}_0$ using data in $\mathcal{D}$.
5: **for** $k = 0, 1, 2, \ldots$ **do**
6:     Optimize $\pi_{k+1}$ using $\widehat{\boldsymbol{M}}_k$ for $K \gg 1$ steps of model-based NPG (Subroutine 1).
7:     Collect dataset $\mathcal{D}_{k+1}$ of $N$ samples from environment using $\pi_{k+1}$.
8:     Aggregate all collected data so far $\mathcal{D} = \mathcal{D} \cup \mathcal{D}_{k+1}$.
9:     Learn dynamics model(s) $\widehat{\boldsymbol{M}}_{k+1}$ using data in $\mathcal{D}$.
10: **end for**
```

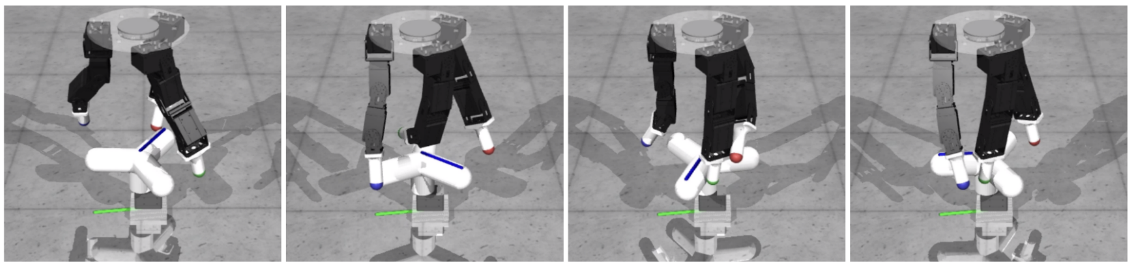
Table C.3: Hyperparameters used for the MAL-NPG algorithm

Parameter	**Value**
Model network	MLP (512, 512)
Learning algorithm	Adam (default parameters)
No. of epochs	10
Mini-batch size	200
Ensemble size	4
Buffer size B	∞
Initial samples (N_{init})	5000
Samples per iteration (N)	min(20×env-horizon, 3000)
NPG updates (K)	25

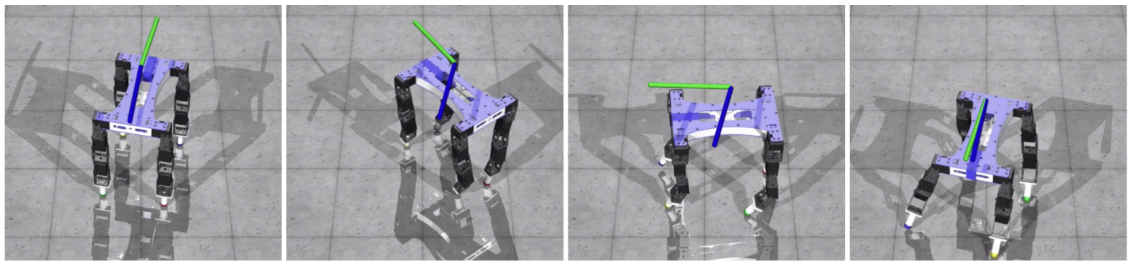
C.2.1 Task Suite

The main tasks we study are `DClaw-Turn`, `DKitty-Orient`, `7DOF-Reacher`, and `InHand-Pen`.

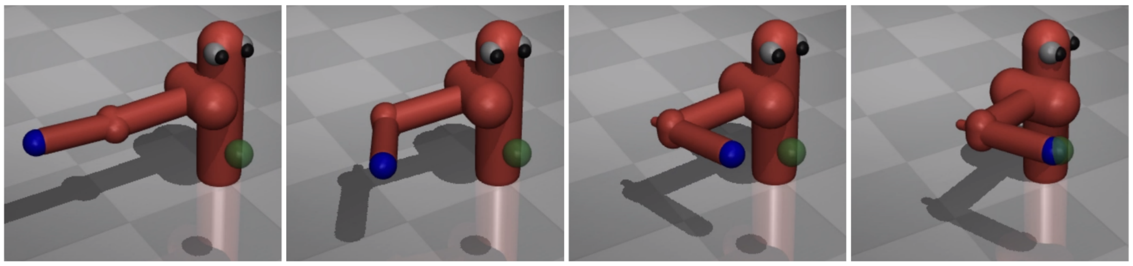
1. The `DClaw-Turn` task requires a 3 fingered "DClaw" to rotate a faucet to a desired orientation (see illustration below). The observations consist of the joint positions and velocities of the claw as well as the faucet; in addition to the desired valve orientation. The reward measures the closeness between the current faucet configuration and desired configuration. For further details about the task, see Ahn et al. [65] (task `DClawTurnRandom-v0`)..



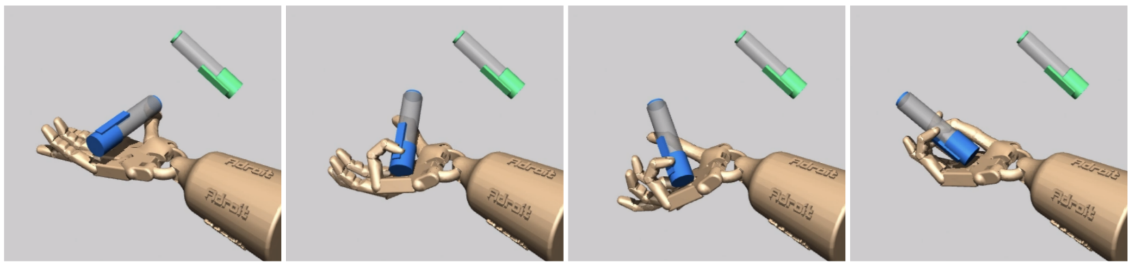
2. The `DKitty-Orient` task requires a quadruped (DKitty) to change its orientation in order to face in a desired direction (as illustrated below). The observations consist of the pose and velocity of various joints in the robot, and the desired orientation. The reward measures the difference between current pose of the robot and the desired pose. For further details about the task, see Ahn et al. [65] (task `DKittyOrientRandom-v0`).



3. The `7DOF-Reacher` reacher task requires a 7DOF robot arm (corresponding to a Sawyer robot) to reach various spatial goals with its end effector (finger tip). The observations consist of the joint pose and velocities of the arm and the desired location for the end effector. The reward measures the distance between the end effector and the goal.



4. Finally, the `InHand-Pen` task requires a 24DOF dexterous hand to manipulate a pen in-hand to point in a desired orientation. The observations consist of the joint pose and velocity for the hand and pen, and the desired pose for the pen. The reward measures the difference between the pose of the pen and the desired pose. For additional details about the task, see Rajeswaran et al. [170] (task `pen-v0`).



C.2.2 **Results on OpenAI gym benchmarks**

We also benchmark the performance of PAL-NPG and MAL-NPG in the OpenAI gym benchmarks [128]. Specifically, we consider three tasks: `InvertedPendulum-v2`, `Hopper-v2`, and `Ant-v2`. For baselines, we consider MBPO [88], PETS [147], STEVE [182], SLBO [175], and SAC [166]. A subset of these algorithms were considered for benchmarking deep RL in the recent work of Wang et al. [177]. MBPO is the current state of the art model-based method, and SAC is a state of the art model-free algorithm. The hyperparameters for PAL and MAL are specified in Tables 1-3 and related discussion. The results are presented in Figure C.1. We find that our methods substantially outperform the baselines. In particular, compared to state of the art MBPO, our method is nearly twice as efficient in InvertedPendulum and Hopper. Our methods are nearly 10× as efficient as other baselines.

In the hopper and ant tasks, we include the velocity of center of mass in the observation space in order to be able to compute the rewards for synthetic rollouts. Similar approaches are followed in prior works as well, e.g. in SLBO [175]. Finally, we note that MBPO is a hybrid model-based and model-free method, while our PAL and MAL implementations are entirely model-based. In MBPO, it was noted that long horizon model-based rollouts were

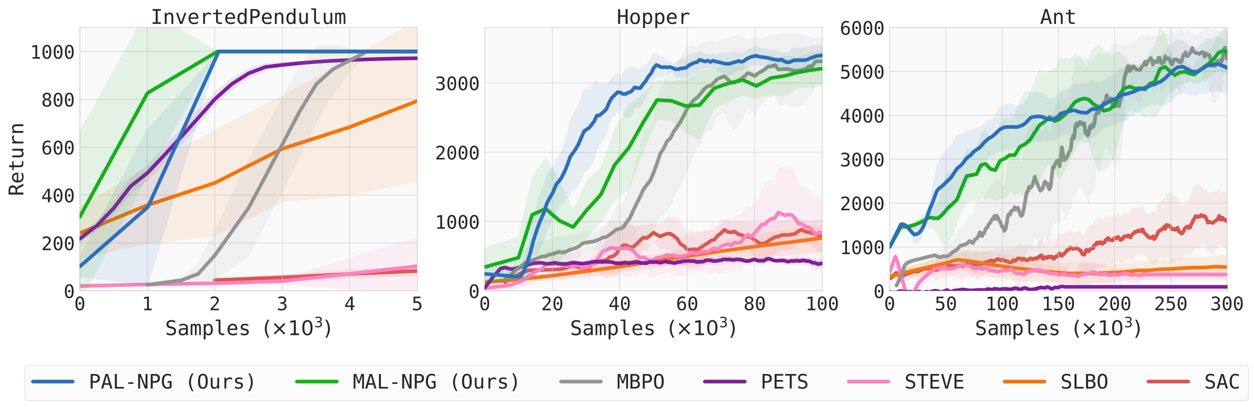


Figure C.1: Comparison of results on the OpenAI gym benchmark tasks. Results for the baselines are reproduced from Janner et al. [88]. We observe that PAL and MAL have stable near-monotonic improvement, and substantially outperform the baselines.

unstable and combining with an off-policy critic was important. We find that through our Stackelberg formulation, which is intended to carefully control the effects of distribution shift, we are able to perform rollouts of hundreds of steps without error amplification. As a result, even though our algorithms are purely model based, they can achieve sample efficient learning without loss in asymptotic performance. It is straightforward to extend our PAL and MAL approaches to the hybrid model-based and model-free setting, and could likely lead to a further increase in efficiency for some tasks. We leave exploration of this to future work.

C.2.3 Model Error Amplification

While the 1-step (prediction) generalization error is easy to measure, it does not provide direct intuitions about the model quality for the purpose of policy improvement. We study error amplification over lookahead horizon to better understand the quality of model for purposes of policy improvement. Let s_0 be the initial state for both $\boldsymbol{M}$ and $\widehat{\boldsymbol{M}}$. We wish to measure $L(t) = \mathbb{E}[||s_t^{\boldsymbol{M}} - s_t^{\widehat{\boldsymbol{M}}}||]$ where $s_t^{\boldsymbol{M}}$ and $s_t^{\widehat{\boldsymbol{M}}}$ are obtained by following the dynamics of $\boldsymbol{M}$ and $\widehat{\boldsymbol{M}}$ respectively. The state evolution depends on actions, and for this we consider two modes: open loop and closed loop.

In **open loop** mode, we first sample an initial state and set it for both $\boldsymbol{M}$ and $\widehat{\boldsymbol{M}}$, i.e. $s_0^{\boldsymbol{M}} = s_0^{\widehat{\boldsymbol{M}}}$. Subsequently, we execute π in $\boldsymbol{M}$ in obtain a trajectory. The action sequence is then executed in open-loop in $\widehat{\boldsymbol{M}}$. Specifically, this makes $a_t^{\widehat{\boldsymbol{M}}} = a_t^{\boldsymbol{M}} = \pi(s_t^{\boldsymbol{M}})$. In **closed loop** mode, we again sample the initial state and set $s_0^{\boldsymbol{M}} = s_0^{\widehat{\boldsymbol{M}}}$. Subsequently, we collect trajectory by indipendetly executing the policy, so that we have: $a_t^{\boldsymbol{M}} = \pi(s_t^{\boldsymbol{M}})$ and $a_t^{\widehat{\boldsymbol{M}}} = \pi(s_t^{\widehat{\boldsymbol{M}}})$.

We study the error for the DClaw task, and plot the error in prediction of faucet angle. This

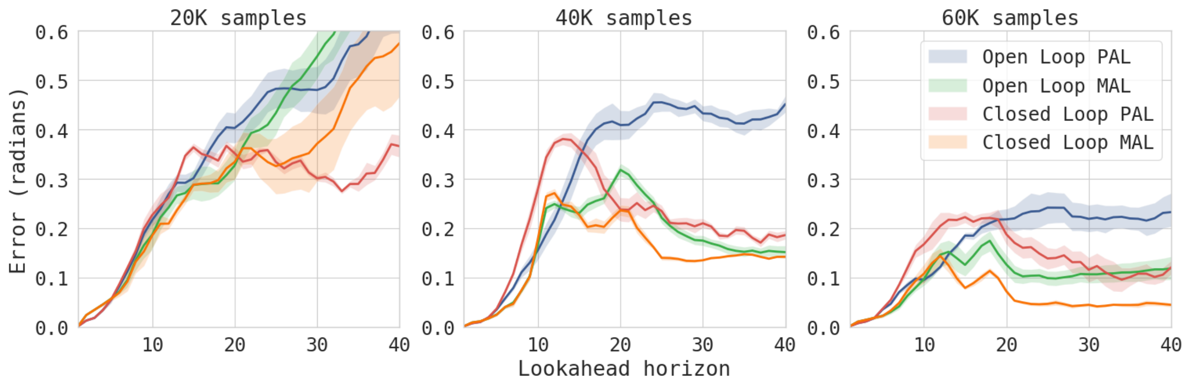


Figure C.2: Error amplification over look-ahead horizon in the DClawTurnRandom task. In this experiment, we measure the error of policy π_k under model $\widehat{\boldsymbol{M}}_k$ for various stages of training (20K, 40K, and 60K samples). The x-axis is the look-ahead horizon, and the y-axis is the error in prediction of the valve orientation (in radians). See main text for explanation of open loop and closed loop.

is the primary quantity of interest, since the task involves turning the faucet to the desired orientation. The results are presented in Figure C.2. We make the following observations:

1. With more training, the entire profile of errors reduce. This is encouraging, since it suggests that the model quality improves with more training.
2. Closed loop prediction errors are smaller than open loop errors. This suggests that the policy shapes the state visitation to regions where the model is more accurate. Thus, the algorithms we consider ensure that the policy and model remain compatible.
3. During initial stages of training, PAL has lower error. However, towards the end of training, MAL learns more accurate models, by improving the model quality at a faster rate. This is likely due to MAL maintaining a larger replay buffer with more diverse transitions obtained over the course of training. This further underscores why MAL can better handle non-stationarities in task and goal distribution.
4. The error does not strictly increase with time. In particular, we observe profiles where the error shrinks towards the end of the horizon. As the policy improves, it turns the faucet to the desired configuration with greater probability. Thus, the long term consequences of the policy are in fact more easily predictable than the intermediate transient effects. This further suggests that the policy-model pair together capture the semantics of the task.

Appendix D

IMPLICIT MAML: PROOFS AND EXPERIMENT DETAILS

D.1 Relationship between iMAML and Prior Algorithms

The presented iMAML algorithm has close connections, as well as notable differences, to a number of related algorithms like MAML [191], first-order MAML, and Reptile [160]. Conventionally, these algorithms do not consider any explicit regularization in the inner-level and instead rely on early stopping, through only a few gradient descent steps. In our problem setting described in Eq. 5.4, we consider an explicitly regularized inner-level problem (refer to discussion in Section 5.2.2). We describe the connections between the algorithms in this explicitly regularized setting below.

MAML The MAML algorithm first invokes an iterative algorithm to solve the inner optimization problem (see definition 5). Subsequently, it backpropagates through the path of the optimization algorithm to update the meta-parameters as:

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \eta \, \frac{1}{M} \sum_{i=1}^{M} \boldsymbol{d}_{\boldsymbol{\theta}} \mathcal{L}_i(\mathcal{A}lg_i(\boldsymbol{\theta}^k)).$$

Since $\mathcal{A}lg_i(\boldsymbol{\theta})$ approximates $\mathcal{A}lg_i^\star(\boldsymbol{\theta})$, it can be viewed that both MAML and iMAML intend to perform the same idealized update in Eq. 5.5. However, they perform the meta-gradient computation very differently. MAML backpropagates through the path of an iterative algorithm, while iMAML computes the meta-gradient through the implicit Jacobian approach outlined in Section 5.3.1 (see Figure 5.1 for a visual depiction). As a result, iMAML can be vastly more efficient in memory while having lesser or comparable computational requirements. It also allows for higher order optimization methods and non-differentiable components.

First-order MAML ignores the effect of meta-parameters $\boldsymbol{\theta}$ on task parameters $\{\boldsymbol{\phi}_i\}$ in the meta-gradient computation and updates the meta-parameters as:

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \eta \, \frac{1}{M} \sum_{i=1}^{M} \nabla_{\boldsymbol{\phi}} \mathcal{L}_i(\boldsymbol{\phi}_i) \mid_{\boldsymbol{\phi}_i = \mathcal{A}lg_i(\boldsymbol{\theta}^k)}$$

Note that iMAML strictly generalizes this, since first-order MAML is simply iMAML when the conjugate gradient procedure is not invoked (or 0 steps of CG). Thus, iMAML allows for an easy way to interpolate from first-order MAML to the full MAML algorithm.

Reptile, similar to first-order MAML, ignores the dependence of task-parameters on meta-parameters [160]. However, instead of following the gradients at $\boldsymbol{\phi}_i = \mathcal{A}lg_i(\boldsymbol{\theta}^k)$, Reptile uses the task-parameters as targets and slowly moves meta-parameters towards them:

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \eta \, \frac{1}{M} \sum_{i=1}^{M} (\boldsymbol{\theta}^k - \boldsymbol{\phi}_i).$$

From the proximal point equation in the proof of Lemma 3, we have $\boldsymbol{\phi}_i = \boldsymbol{\theta}^k - \frac{1}{\lambda} \nabla_{\boldsymbol{\phi}} \mathcal{L}_i(\boldsymbol{\phi}_i)$, using which we see that the Reptile equation becomes: $\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \frac{\eta}{\lambda M} \sum_{i=1}^{M} \nabla_{\boldsymbol{\phi}} \mathcal{L}_i(\boldsymbol{\phi}_i)$. Thus, Reptile and first-order MAML are identical in our problem formulation up to the choice of learning rate. Making the regularization explicit allows us to illustrate this equivalence.

D.2 Optimization Preliminaries

Let $f : \mathbb{R}^d \to \mathbb{R}$. A function f is B Lipschitz (or B-bounded gradient norm) if for all $x \in \mathbb{R}^d$

$$||\nabla f(x)|| \leq B\,.$$

Similarly, we say that a matrix valued function $M : \mathbb{R}^d \times \mathbb{R}^{d'} \to \mathbb{R}$ is ρ-Lipschitz if

$$||M(x) - M(x')|| \leq \rho ||x - x'||\,,$$

where $\|\cdot\|$ denotes the spectral norm. We say that f is L-smooth if for all $x, x' \in \mathbb{R}^d$

$$||\nabla f(x) - \nabla f(x')|| \leq L ||x - x'||$$

and that f is μ-strongly convex if f is convex and if for all $x, x' \in \mathbb{R}^d$,

$$||\nabla f(x) - \nabla f(x')|| \geq \mu ||x - x'||\,.$$

We will make use of the following black-box complexity of first-order gradient methods for minimizing strongly convex and smooth functions.

Lemma 7. *(δ-approximate solver; see [295]) Suppose f is a function that is L-smooth and μ strongly convex. Define $\kappa := L/\mu$, and let $x^\star = \operatorname{argmin} f(x)$. Nesterov's accelerated gradient descent can be used to find a point x such that:*

$$\|x - x^\star\| \leq \delta$$

using a number of gradient computations of f that is bounded as follows:

$$\#\ \textit{gradient computations of } f(\cdot) \leq 2\sqrt{\kappa}\,\log\left(2\kappa \frac{\|x^\star\|}{\delta}\right)\,.$$

D.3 Review: Time and Space Complexity of Hessian-Vector Products

We briefly discuss the time and space complexity of Hessian-vector product computation using the reverse mode of automatic differentiation. The reverse mode of automatic differentiation [296, 297] is the widely used method for automatic differentiation in modern software packages like TensorFlow and PyTorch [298]. Recall that for a differentiable function $f(x)$, the reverse mode of automatic differentiation computes $\nabla f(x)$ in time that is no more than a factor of 5 of the time it takes to compute $f(x)$ itself (see [297] for review). As our algorithm makes use of Hessian vector products, we will make use of the following assumption as to how Hessian vector products will be computed when executing Algorithm 6.

Assumption A3. *(Complexity of Hessian-vector product) We assume that the time to compute the Hessian-vector product $\nabla^2_{\boldsymbol{\phi}} \hat{\mathcal{L}}_i(\boldsymbol{\phi})\boldsymbol{v}$ is no more than a (universal) constant over the time used to compute $\nabla \hat{\mathcal{L}}_i(\boldsymbol{\phi})$ (typically, this constant is 5). Furthermore, we assume that the memory used to compute the Hessian-vector product $\nabla^2_{\boldsymbol{\phi}} \hat{\mathcal{L}}_i(\boldsymbol{\phi})\boldsymbol{v}$ is no more than twice the memory used when computing $\nabla \hat{\mathcal{L}}_i(\boldsymbol{\phi})$. This assumption is valid if the reverse mode of automatic differentiation is used to compute Hessian vector products (see [299]).*

A few remarks about this assumption are in order. With regards to computation, first observe that the gradient of the scalar function $\nabla_{\boldsymbol{\phi}} \hat{\mathcal{L}}_i(\boldsymbol{\phi})^\top \boldsymbol{v}$ is the desired Hessian vector product $\nabla^2_{\boldsymbol{\phi}} \hat{\mathcal{L}}_i(\boldsymbol{\phi})\boldsymbol{v}$. Thus computing the Hessian vector product using the reverse mode is within a constant factor of computing the function itself, which is simply the cost of computing $\nabla \hat{\mathcal{L}}_i(\boldsymbol{\phi})^\top \boldsymbol{v}$. The issue of memory is more subtle (see [299]), which we now discuss. The memory used to compute the gradient of a scalar cost function $f(x)$ using the reverse mode of auto-differentiation is proportional to the size of the computation graph; precisely, the memory required to compute the gradient is equal to the total space required to store all the intermediate variables used when computing $f(x)$. In practice, this is often much larger than the memory required to compute $f(x)$ itself, due to that all intermediate variables need not be simultaneously stored in memory when computing $f(x)$. However, for the special case of computing the gradient of the function $f(\boldsymbol{\phi}) = \nabla_{\boldsymbol{\phi}} \hat{\mathcal{L}}_i(\boldsymbol{\phi})^\top \boldsymbol{v}$, the factor of 2 in the memory bound is a consequence of the following reason: first, using the reverse mode to compute $f(\boldsymbol{\phi})$ means we already have stored the computation graph of $\hat{\mathcal{L}}_i(\boldsymbol{\phi})$ itself. Furthermore, the size of the computation graph for computing $f(\boldsymbol{\phi}) = \nabla_{\boldsymbol{\phi}} \hat{\mathcal{L}}_i(\boldsymbol{\phi})^\top \boldsymbol{v}$ is essentially the same size as the computation graph of $\hat{\mathcal{L}}_i(\boldsymbol{\phi})$. This leads to the factor of 2 memory bound; see Griewank [299] for further discussion.

D.4 Additional Discussion About Compute and Memory Complexity

Our main complexity results are summarized in Table 1. For these results, we consider two notions of error that are subtly different, which we explicitly define below. Let $\boldsymbol{g}_i$ be the computed meta-gradient for task $\mathcal{T}_i$. Then, the errors we consider are:

Definition 7. *Exact-solve error (our notion of error): Our goal is to accurately compute the gradient of $F(\theta)$ as defined in Equation 5.4, where $\mathcal{A}lg_i^\star(\theta)$ is an exact algorithm. Specifically, we seek to compute a $\boldsymbol{g}_i$ such that:*

$$\|\boldsymbol{g}_i - \boldsymbol{d_\theta} \mathcal{L}_i(\mathcal{A}lg_i^\star(\boldsymbol{\theta}))\| \leq \epsilon$$

where ϵ is the error in the gradient computation.

Definition 8. *Approx-solve error: Here we suppose that $\mathcal{A}lg_i$ computes a δ–accurate solution to the inner optimization problem over G_i in Eq. 5.4, i.e. that $\mathcal{A}lg_i$ satisfies $\|\mathcal{A}lg_i(\boldsymbol{\theta}) - \mathcal{A}lg_i^\star(\boldsymbol{\theta})\| \leq \delta$, as per definition 5. Then the objective is to compute a $\boldsymbol{g}$ such that:*

$$\|\boldsymbol{g} - \boldsymbol{d_\theta} \mathcal{L}_i(\mathcal{A}lg_i(\boldsymbol{\theta}))\| \leq \epsilon$$

where ϵ is the error in the gradient computation of $\boldsymbol{d_\theta} \mathcal{L}_i(\mathcal{A}lg_i(\boldsymbol{\theta}))$. Subtly, note that the gradient is with respect to the δ-approximate algorithm, as opposed to using $\mathcal{A}lg_i^\star$.

For the complexity results, we assume that MAML invokes $\mathcal{A}lg_i$ to get a δ-approximate solution for inner problem (recall definition 5). The exact-solve error for MAML is not known in the literature; in particular, even as $\delta \to 0$ it is not evident if the approx-solve solution tends to the exact-solve solution, unless further regularity conditions are imposed. The approx-solve error for MAML is 0, ignoring finite-precision and numerical issues, since it backpropagates through the path. Truncated backprop [208] also invokes $\mathcal{A}lg_i$ to obtain a δ-approximate solution but instead performs a truncated or partial back-propagation so that it uses a smaller number of iterations when computing the gradient through the path of $\mathcal{A}lg_i(\boldsymbol{\theta})$. Exact-solve error for truncated backprop is also not known, but a small approx-solve error can be obtained with less memory than full back-prop. We use Prop 3.1 of Shaban et al. [208] to provide a guarantee that leads to an ϵ–accurate approximation of the full-backprop (i.e. MAML) gradient. It is not evident how accurate the truncated procedure is when an accelerated method is used instead. Finally, our iMAML algorithm also invokes an approximate solver $\mathcal{A}lg_i$ rather than $\mathcal{A}lg_i^\star$. However, importantly, we guarantee a small exact-solve error even though we do not require access to $\mathcal{A}lg_i^\star$. Furthermore, the iMAML algorithm also requires substantially less memory. Up to small constant factors, it only utilizes the memory required for computing a single gradient of $\hat{\mathcal{L}}_i(\cdot)$.

D.5 Theoretical Results and Proofs

Lemma 5.3.1 Consider $\mathcal{A}lg_i^\star(\boldsymbol{\theta})$ as defined in Eq. 5.4 for task $\mathcal{T}_i$. Let $\boldsymbol{\phi}_i = \mathcal{A}lg_i^\star(\boldsymbol{\theta})$ be the result of $\mathcal{A}lg_i^\star(\boldsymbol{\theta})$. If $\left(\boldsymbol{I} + \frac{1}{\lambda}\nabla_\phi^2 \hat{\mathcal{L}}_i(\boldsymbol{\phi}_i)\right)$ is invertible, then the derivative Jacobian is

$$\frac{d\mathcal{A}lg_i^\star(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = \left(\boldsymbol{I} + \frac{1}{\lambda}\, \nabla_\phi^2 \hat{\mathcal{L}}_i(\boldsymbol{\phi}_i)\right)^{-1}.$$

Proof. We drop the task i subscripts in the proof for convenience. Since $\boldsymbol{\phi} = \mathcal{A}lg^{\star}(\boldsymbol{\theta})$ is the minimizer of $G(\boldsymbol{\phi}', \boldsymbol{\theta})$ in Eq. 5.4, the stationary point conditions imply that

$$\nabla_{\boldsymbol{\phi}'} G(\boldsymbol{\phi}', \boldsymbol{\theta}) \mid_{\boldsymbol{\phi}'=\boldsymbol{\phi}} = 0 \implies \nabla \hat{\mathcal{L}}(\boldsymbol{\phi}) + \lambda(\boldsymbol{\phi} - \boldsymbol{\theta}) = 0 \implies \boldsymbol{\phi} = \boldsymbol{\theta} - \frac{1}{\lambda} \nabla \hat{\mathcal{L}}(\boldsymbol{\phi}),$$

which is an implicit equation that often arises in proximal point methods. When the derivative exists, we can differentiate the above equation to obtain:

$$\frac{d\boldsymbol{\phi}}{d\boldsymbol{\theta}} = \boldsymbol{I} - \frac{1}{\lambda} \nabla^2 \hat{\mathcal{L}}(\boldsymbol{\phi}) \frac{d\boldsymbol{\phi}}{d\boldsymbol{\theta}} \implies \left(\boldsymbol{I} + \frac{1}{\lambda} \nabla^2 \hat{\mathcal{L}}(\boldsymbol{\phi}) \right) \frac{d\boldsymbol{\phi}}{d\boldsymbol{\theta}} = \boldsymbol{I}.$$

which completes the proof. □

Assumption A4. *(Regularity conditions) Suppose the following holds for all tasks i:*

1. *$\mathcal{L}_i(\cdot)$ is B Lipshitz and L smooth.*

2. *For all $\boldsymbol{\theta}$, $G_i(\cdot, \boldsymbol{\theta})$ is both a β-smooth function and a μ-strongly convex function. Define:*
$$\kappa := \frac{\beta}{\mu}.$$

3. *$\hat{\mathcal{L}}_i(\cdot)$ is ρ-Lipshitz Hessian, i.e. $\nabla^2 \hat{\mathcal{L}}_i(\cdot)$ is ρ-Lipshitz.*

4. *For all $\boldsymbol{\theta}$, suppose the arg-minimizer of $G_i(\cdot, \boldsymbol{\theta})$ is unique and bounded in a ball of radius D, i.e. for all $\boldsymbol{\theta}$,*
$$\|\mathcal{A}lg_i^{\star}(\boldsymbol{\theta})\| \leq D.$$

Lemma 8. *(Implicit Gradient Accuracy) Suppose Assumption A4 holds. Fix a task i. Suppose that $\boldsymbol{\phi}_i$ satisfies:*
$$\|\boldsymbol{\phi}_i - \mathcal{A}lg_i^{\star}(\boldsymbol{\theta})\| \leq \delta$$
and that $\boldsymbol{g}_i$ satisfies:
$$\|\boldsymbol{g}_i - \left(\boldsymbol{I} + \frac{1}{\lambda} \nabla^2 \hat{\mathcal{L}}_i(\boldsymbol{\phi}) \right)^{-1} \nabla_{\boldsymbol{\phi}} \mathcal{L}_i(\boldsymbol{\phi})\| \leq \delta'.$$
Assuming that $\delta < \mu/(2\rho)$, we have that:
$$\|\boldsymbol{g}_i - \boldsymbol{d}_{\boldsymbol{\theta}} \mathcal{L}_i(\mathcal{A}lg_i^{\star}(\boldsymbol{\theta}))\| \leq \left(2\frac{\lambda\rho}{\mu^2} B + \frac{\lambda L}{\mu} \right) \delta + \delta'$$

Proof. First, observe that:

$$\boldsymbol{d_\theta}\mathcal{L}_i(\mathcal{A}lg_i^\star(\boldsymbol{\theta})) = \left(\boldsymbol{I} + \frac{1}{\lambda}\ \nabla^2\hat{\mathcal{L}}_i(\mathcal{A}lg_i^\star(\boldsymbol{\theta}))\right)^{-1} \nabla_{\boldsymbol{\phi}}\mathcal{L}_i(\mathcal{A}lg_i^\star(\boldsymbol{\theta}))$$

For notational convenience, we drop the i subscripts within the proof. We have:

$$\begin{aligned}
& \|\boldsymbol{d_\theta}\mathcal{L}(\mathcal{A}lg^\star(\boldsymbol{\theta})) - \boldsymbol{g}\| \\
\leq\ & \|\boldsymbol{d_\theta}\mathcal{L}(\mathcal{A}lg^\star(\boldsymbol{\theta})) - \left(\boldsymbol{I} + \frac{1}{\lambda}\ \nabla^2\hat{\mathcal{L}}(\boldsymbol{\phi})\right)^{-1} \nabla_{\boldsymbol{\phi}}\mathcal{L}(\boldsymbol{\phi})\| + \delta' \\
\leq\ & \|\boldsymbol{d_\theta}\mathcal{L}(\mathcal{A}lg^\star(\boldsymbol{\theta})) - \left(\boldsymbol{I} + \frac{1}{\lambda}\ \nabla^2\hat{\mathcal{L}}(\boldsymbol{\phi})\right)^{-1} \nabla_{\boldsymbol{\phi}}\mathcal{L}(\mathcal{A}lg^\star(\boldsymbol{\theta}))\| + \\
& \|\left(\boldsymbol{I} + \frac{1}{\lambda}\ \nabla^2\hat{\mathcal{L}}(\boldsymbol{\phi})\right)^{-1} \left(\nabla_{\boldsymbol{\phi}}\mathcal{L}(\mathcal{A}lg^\star(\boldsymbol{\theta})) - \nabla_{\boldsymbol{\phi}}\mathcal{L}(\boldsymbol{\phi})\right)\| + \delta'
\end{aligned}$$

where the first inequality uses the triangle inequality.

We now bound each of these terms. For the second term,

$$\begin{aligned}
& \|\left(\boldsymbol{I} + \frac{1}{\lambda}\ \nabla^2\hat{\mathcal{L}}(\boldsymbol{\phi})\right)^{-1} \left(\nabla_{\boldsymbol{\phi}}\mathcal{L}(\mathcal{A}lg^\star(\boldsymbol{\theta})) - \nabla_{\boldsymbol{\phi}}\mathcal{L}(\boldsymbol{\phi})\right)\| \\
\leq\ & \|\left(\boldsymbol{I} + \frac{1}{\lambda}\ \nabla^2\hat{\mathcal{L}}(\boldsymbol{\phi})\right)^{-1}\|\|\nabla_{\boldsymbol{\phi}}\mathcal{L}(\mathcal{A}lg^\star(\boldsymbol{\theta})) - \nabla_{\boldsymbol{\phi}}\mathcal{L}(\boldsymbol{\phi})\| \\
\leq\ & \lambda L\|\left(\lambda\boldsymbol{I} + \nabla^2\hat{\mathcal{L}}(\boldsymbol{\phi})\right)^{-1}\|\|\mathcal{A}lg^\star(\boldsymbol{\theta}) - \boldsymbol{\phi}\| \\
=\ & \lambda L\|\nabla^2_{\boldsymbol{\phi}}G(\boldsymbol{\phi},\boldsymbol{\theta})^{-1}\|\|\mathcal{A}lg^\star(\boldsymbol{\theta}) - \boldsymbol{\phi}\| \\
\leq\ & \frac{\lambda L}{\mu}\delta
\end{aligned}$$

where we the second inequality uses that $\nabla_{\boldsymbol{\phi}}\mathcal{L}$ is L-smooth and the final inequality uses that G is μ strongly convex.

For the first term, we have:

$$\begin{aligned}
& \|\boldsymbol{d_\theta}\mathcal{L}(\mathcal{A}lg^\star(\boldsymbol{\theta})) - \left(\boldsymbol{I} + \frac{1}{\lambda}\ \nabla^2\hat{\mathcal{L}}(\boldsymbol{\phi})\right)^{-1} \nabla_{\boldsymbol{\phi}}\mathcal{L}(\mathcal{A}lg^\star(\boldsymbol{\theta}))\| \\
=\ & \|\left(\left(\boldsymbol{I} + \frac{1}{\lambda}\ \nabla^2\hat{\mathcal{L}}(\mathcal{A}lg^\star(\boldsymbol{\theta}))\right)^{-1} - \left(\boldsymbol{I} + \frac{1}{\lambda}\ \nabla^2\hat{\mathcal{L}}(\boldsymbol{\phi})\right)^{-1}\right) \nabla_{\boldsymbol{\phi}}\mathcal{L}(\mathcal{A}lg^\star(\boldsymbol{\theta}))\| \\
\leq\ & \lambda\|\left(\lambda\boldsymbol{I} + \nabla^2\hat{\mathcal{L}}(\mathcal{A}lg^\star(\boldsymbol{\theta}))\right)^{-1} - \left(\lambda\boldsymbol{I} + \nabla^2\hat{\mathcal{L}}(\boldsymbol{\phi})\right)^{-1}\|B,
\end{aligned}$$

using that $\nabla_{\boldsymbol{\phi}}\mathcal{L}$ is B Lipshitz. Now let

$$\Delta := \nabla^2\hat{\mathcal{L}}(\mathcal{A}lg^\star(\boldsymbol{\theta})) - \nabla^2\hat{\mathcal{L}}(\boldsymbol{\phi}), \quad M := \nabla^2_{\boldsymbol{\phi}}G(\boldsymbol{\phi},\boldsymbol{\theta}) = \lambda\boldsymbol{I} + \nabla^2\hat{\mathcal{L}}(\boldsymbol{\phi})$$

Due to that $\nabla^2 \hat{\mathcal{L}}(\cdot)$ is Lipshitz Hessian, $\|\Delta\| \le \rho\delta$. Also, by our assumption on δ, we have that:

$$\|M^{-1}\Delta\| \le \|\Delta\|/\mu \le \rho\delta/\mu \le 1/2,$$

which implies that $\| \left(\boldsymbol{I} + M^{-1}\Delta\right)^{-1} \| \le 2$. Hence,

$$\begin{aligned}
& \| \left(\lambda \boldsymbol{I} + \nabla^2 \hat{\mathcal{L}}(\mathcal{A}lg^\star(\boldsymbol{\theta}))\right)^{-1} - \left(\lambda \boldsymbol{I} + \nabla^2 \hat{\mathcal{L}}(\boldsymbol{\phi})\right)^{-1} \| \\
= \;& \| \left(M + \Delta\right)^{-1} - M^{-1} \| \\
\le \;& \|M^{-1}\| \| \left(\boldsymbol{I} + M^{-1}\Delta\right)^{-1} - \boldsymbol{I}\| \\
= \;& \|M^{-1}\| \| \left(\boldsymbol{I} + M^{-1}\Delta\right)^{-1} \left(\boldsymbol{I} - \left(\boldsymbol{I} + M^{-1}\Delta\right)\right) \| \\
\le \;& \|M^{-1}\| \| \left(\boldsymbol{I} + M^{-1}\Delta\right)^{-1} \| \|M^{-1}\Delta\| \\
\le \;& \frac{1}{\mu} \cdot 2 \cdot \frac{\rho\delta}{\mu} = 2\frac{\rho}{\mu^2}\delta.
\end{aligned}$$

The proof is completed by substitution. □

Theorem 5. *(Approximate Implicit Gradient Computation) Suppose Assumption A4 holds. Fix a task i. Let*

$$B_1 \;\; := \;\; 2\frac{\lambda\rho}{\mu^2}B + \frac{\lambda L}{\mu}$$

Suppose Nesterov's accelerated gradient descent algorithm is used to compute $\boldsymbol{\phi}$ (as desired in Algorithm 6), using a number of iterations that is:

$$2\sqrt{\kappa}\,\log\left(8\kappa D\left(\frac{B_1}{\epsilon} + \frac{\rho}{\mu}\right)\right)$$

and suppose Nesterov's accelerated gradient descent algorithm (or the conjugate gradient algorithm) is used to compute $\boldsymbol{g}_i$ using a number of iterations that is:

$$2\sqrt{\kappa}\,\log\left(4\kappa\frac{(\lambda/\mu)B}{\epsilon}\right).$$

We have that:

$$\|\boldsymbol{g}_i - \boldsymbol{d}_{\boldsymbol{\theta}}\mathcal{L}_i(\mathcal{A}lg_i^\star(\boldsymbol{\theta}))\| \le \epsilon.$$

Proof. The result will follow from the guarantees in Lemma 7. Specifically, let us set $\delta = \min\{\epsilon/(2B_1), \mu/(2\rho)\}$ and $\delta' = \epsilon/2$. To ensure the bound of δ, by Lemma 8, it suffices to use a number of iterations that is bounded by:

$$2\log\left(2\kappa\frac{\|D\|}{\delta}\right) \le 2\sqrt{\kappa}\,\log\left(8\kappa D\left(\frac{B_1}{\epsilon} + \frac{\rho}{\mu}\right)\right)$$

To ensure the bound of δ', the algorithm will be solving the sub-problem in Equation 5.7. First observe that in the context of in Lemma 7, note that $\|x^\star\| = \| \left(\boldsymbol{I} + \frac{1}{\lambda} \nabla^2 \hat{\mathcal{L}}_i(\boldsymbol{\phi}) \right)^{-1} \nabla \mathcal{L}_i(\boldsymbol{\phi}) \| \leq (\lambda/\mu)B$, and so it suffices to use a number of iterations that is bounded by:

$$2 \log \left(2\kappa \frac{\|x^\star\|}{\delta} \right) \leq 2 \log \left(4\kappa \frac{(\lambda/\mu)B}{\epsilon} \right),$$

which completes the proof. □

Finally, we present a corollary of previous theorem that shows that iMAML finds approximate stationary points due to controllable error in gradient computation.

Corollary 5. *(iMAML finds stationary points) Suppose the conditions of Theorem 3 hold and that $F(\cdot)$ is an L_F smooth function. Then the implicit MAML algorithm (Algorithm 5), when the batch size is M (so that we are doing gradient descent), will find a point $\boldsymbol{\theta}$ such that $\|\nabla F(\boldsymbol{\theta})\| \leq \epsilon$, in a number of calls to* **`Implicit-Meta-Gradient`** *that is at most $\frac{4ML_f(F(0) - \min_{\boldsymbol{\theta}} F(\boldsymbol{\theta}))}{\epsilon^2}$. Furthermore, the total number of gradient computations (of $\nabla \hat{\mathcal{L}}_i$) is at most $\tilde{O}\left(M\sqrt{\kappa} \frac{L_f(F(0) - \min_\theta F(\theta))}{\epsilon^2} \log \left(\frac{poly(\kappa, D, B, L, \rho, \mu, \lambda)}{\epsilon} \right) \right)$, and only $\tilde{O}(\mathrm{Mem}(\nabla \hat{\mathcal{L}}_i))$ memory is required throughout.*

D.6 *Experiment Details*

Here, we provide additional details of the experimental set-up for the experiments in Section 5.4. All training runs were conducted on a single NVIDIA (Titan Xp) GPU.

D.6.1 *Synthetic Experiments*

For the synthetic experiments, we consider a linear regression problem. We consider parametric models of the form $h_{\boldsymbol{\phi}}(\mathbf{x}) = \boldsymbol{\phi}^T \mathbf{x}$, where $\mathbf{x}$ can either be the raw inputs or features (e.g. Fourier features) of the input. For task $\mathcal{T}_i$, we can equivalently write a quadratic objective that represents the task loss as:

$$\hat{\mathcal{L}}_i(\boldsymbol{\phi}) = \frac{1}{2} \mathbb{E}_{(\mathbf{x},\mathbf{y}) \sim \mathcal{D}_i^{\mathrm{tr}}} \left[\|h_{\boldsymbol{\phi}}(\mathbf{x}) - \mathbf{y}\|^2 \right] = \frac{1}{2} \boldsymbol{\phi}^T A_i \boldsymbol{\phi} + \boldsymbol{\phi}^T b_i,$$

where $A_i = \mathbb{E}_{(\mathbf{x},\mathbf{y}) \sim \mathcal{D}_i^{\mathrm{tr}}} \left[\mathbf{x}\mathbf{x}^T \right]$ and $b_i = \mathbb{E}_{(\mathbf{x},\mathbf{y}) \sim \mathcal{D}_i^{\mathrm{tr}}} \left[\mathbf{x}^T \mathbf{y} \right]$. Thus, the inner level objective and corresponding minimizer can be written as:

$$G_i(\boldsymbol{\phi}', \boldsymbol{\theta}) = \frac{1}{2} \boldsymbol{\phi}'^T A_i \boldsymbol{\phi}' + \boldsymbol{\phi}'^T b_i + \frac{\lambda}{2} (\boldsymbol{\phi}' - \boldsymbol{\theta})^T (\boldsymbol{\phi}' - \boldsymbol{\theta})$$

$$\mathcal{A}lg_i^\star(\boldsymbol{\theta}) = \left(A_i + \lambda \boldsymbol{I} \right)^{-1} (\lambda \boldsymbol{\theta} - b_i)$$

Thus, the exact meta-gradient can be written as

$$\boldsymbol{d_\theta} \mathcal{L}_i(\mathcal{A}lg_i^\star(\boldsymbol{\theta})) = \lambda (A_i + \lambda \boldsymbol{I})^{-1} \nabla_{\boldsymbol{\phi}} \mathcal{L}_i(\boldsymbol{\theta}) \mid_{\boldsymbol{\phi} = \mathcal{A}lg_i^\star(\boldsymbol{\theta})} .$$

We compare this gradient with the gradients computed by the iMAML and MAML algorithms. We considered the case of $\mathbf{x} \in \mathbb{R}^{50}$, $\mathbf{y} \in \mathbb{R}$, $\lambda = 5.0$, and $\kappa = 50$, for the presented results.

D.6.2 Omniglot and Mini-ImageNet experiments

We follow the standard training and evaluation protocol as in prior works [189, 190, 191].

Omniglot Experiments The GD version of iMAML uses 16 gradient steps for 5-way 1-shot and 5-way 5-shot settings, and 25 gradient steps for 20-way 1-shot and 20-way 5-shot settings. A regularization strength of $\lambda = 2.0$ was used for both. 5 steps of conjugate gradient was used to compute the meta-gradient for each task in the mini-batch, and the meta-gradients were averaged before taking a step with the default parameters of Adam in the outer loop.

The Hessian-free version of MAML proceeds by using Hessian-free or Newton-CG method for solving the inner optimization problem (with respect to $\boldsymbol{\phi}$) with objective $G_i(\boldsymbol{\phi}, \boldsymbol{\theta})$. This method proceeds by constructing a local quadratic approximation to the objective and approximately computing the Newton direction with conjugate gradient. 5 CG steps are used for this process in our experiments. This allows us to compute the search direction, following which a step size has to be picked. We pick the step size through line-search. This procedure of computing the approximate Newton direction and linesearch is repeated 3 times in our experiments to solve the inner optimization problem well.

Mini-ImageNet For the GD version of iMAML, 10 GD steps were used with regularization strength of $\lambda = 0.5$. Again, 5 CG steps are used to compute the meta-gradient. Similarly, in the Hessian-Free variant, we again use 5 CG steps to compute the search direction followed by line search. This process is repeated 3 times to solve the inner level optimization. Again, to compute the meta-gradient, 5 steps of CG are used.

Appendix E

ONLINE META-LEARNING: THEORETICAL RESULTS

E.1 Linear Regression Example

Here, we present a simple example of optimizing a collection of quadratic objectives (equivalent to linear regression on fixed set of features), where the solutions to joint training and the meta-learning (MAML) problem are different. The purpose of this example is to primarily illustrate that meta-learning can provide performance gains even in seemingly simple and restrictive settings. Consider a collection of objective functions: $\{\mathcal{L}_i : \boldsymbol{\phi} \in \mathbb{R}^d \rightarrow \mathbb{R}\}_{i=1}^M$ which can be described by quadratic forms. Specifically, each of these functions are of then form

$$\mathcal{L}_i(\boldsymbol{\phi}) = \frac{1}{2}\boldsymbol{\phi}^T \boldsymbol{A}_i \boldsymbol{\phi} + \boldsymbol{\phi}^T \boldsymbol{b}_i.$$

This can represent linear regression problems as follows: let $(\mathbf{x}_{\mathcal{T}_i}, \mathbf{y}_{\mathcal{T}_i})$ represent input-output pairs corresponding to task $\mathcal{T}_i$. Let the predictive model be $\boldsymbol{h}(\mathbf{x}) = \boldsymbol{\phi}^T \mathbf{x}$. Here, we assume that a constant scalar (say 1) is concatenated in $\mathbf{x}$ to subsume the constant offset term (as common in practice). Then, the loss function can be written as:

$$\mathcal{L}_i(\boldsymbol{\phi}) = \frac{1}{2}\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{T}_i}\left[||\boldsymbol{h}(\mathbf{x}) - \mathbf{y}||^2\right]$$

which corresponds to $\boldsymbol{A}_i = \mathbb{E}_{\mathbf{x}\sim\mathcal{T}_i}[\mathbf{x}\mathbf{x}^T]$ and $\boldsymbol{b}_i = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{T}_i}[\mathbf{x}^T\mathbf{y}]$. For these set of problems, we are interested in studying the difference between joint training and meta-learning.

Joint training The first approach of interest is joint training which corresponds to the optimization problem

$$\min_{\boldsymbol{\phi}\in\mathbb{R}^d} F(\boldsymbol{\phi})\,, \text{ where } F(\boldsymbol{\phi}) = \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}_i(\boldsymbol{\phi}). \tag{E.1}$$

Using the form of $\mathcal{L}_i$, we have

$$F(\boldsymbol{\phi}) = \frac{1}{2}\,\boldsymbol{\phi}^T\left(\frac{1}{M}\sum_{i=1}^{M}\boldsymbol{A}_i\right)\boldsymbol{\phi} + \boldsymbol{\phi}^T\left(\frac{1}{M}\sum_{i=1}^{M}\boldsymbol{b}_i\right).$$

Let us define the following:

$$\bar{\boldsymbol{A}} := \frac{1}{M}\sum_{i=1}^{M}\boldsymbol{A}_i \quad\text{and}\quad \bar{\boldsymbol{b}} := \frac{1}{M}\sum_{i=1}^{M}\boldsymbol{b}_i.$$

The solution to the joint training optimization problem (Eq. E.1) is then given by

$$\boldsymbol{\phi}^*_{\text{joint}} = -\bar{\boldsymbol{A}}^{-1}\bar{\boldsymbol{b}}.$$

Meta learning (MAML) The second approach of interest is meta-learning, which as mentioned in Section 6.2 corresponds to the optimization problem:

$$\min_{\boldsymbol{\phi}\in\mathbb{R}^d} \tilde{F}(\boldsymbol{\phi})\,,\ \text{where } \tilde{F}(\boldsymbol{\phi}) = \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}_i(\mathcal{A}lg_i(\boldsymbol{\phi})). \tag{E.2}$$

Here, we specifically concentrate on the 1-step (exact) gradient update procedure: $\mathcal{A}lg_i(\boldsymbol{\phi}) = \boldsymbol{\phi} - \alpha\nabla\mathcal{L}_i(\boldsymbol{\phi})$. In the case of the quadratic objectives, this leads to:

$$\begin{aligned}\mathcal{L}_i(\mathcal{A}lg_i(\boldsymbol{\phi})) = &\frac{1}{2}(\boldsymbol{\phi} - \alpha\boldsymbol{A}_i\boldsymbol{\phi} - \alpha\boldsymbol{b}_i)^T\boldsymbol{A}_i(\boldsymbol{\phi} - \alpha\boldsymbol{A}_i\boldsymbol{\phi} - \alpha\boldsymbol{b}_i)\\ &+ (\boldsymbol{\phi} - \alpha\boldsymbol{A}_i\boldsymbol{\phi} - \alpha\boldsymbol{b}_i)^T\boldsymbol{b}_i\end{aligned}$$

The corresponding gradient can be written as:

$$\begin{aligned}\nabla\mathcal{L}_i(\mathcal{A}lg_i(\boldsymbol{\phi})) &= \Big(\boldsymbol{I} - \alpha\boldsymbol{A}_i\Big)\Big(\boldsymbol{A}_i\big(\boldsymbol{\phi} - \alpha\boldsymbol{A}_i\boldsymbol{\phi} - \alpha\boldsymbol{b}_i\big) + \boldsymbol{b}_i\Big)\\ &= \big(\boldsymbol{I} - \alpha\boldsymbol{A}_i\big)\boldsymbol{A}_i\big(\boldsymbol{I} - \alpha\boldsymbol{A}_i\big)\boldsymbol{\phi} + \big(\boldsymbol{I} - \alpha\boldsymbol{A}_i\big)^2\boldsymbol{b}_i\end{aligned}$$

For notational convenience, we define:

$$\begin{aligned}\boldsymbol{A}_\dagger &:= \frac{1}{M}\sum_{i=1}^{M}\big(\boldsymbol{I} - \alpha\boldsymbol{A}_i\big)^2\boldsymbol{A}_i\\ \boldsymbol{b}_\dagger &:= \frac{1}{M}\sum_{i=1}^{M}\big(\boldsymbol{I} - \alpha\boldsymbol{A}_i\big)^2\boldsymbol{b}_i.\end{aligned}$$

Then, the solution to the MAML optimization problem (Eq. E.2) is given by

$$\boldsymbol{\phi}^*_{\text{MAML}} = -\boldsymbol{A}_\dagger^{-1}\boldsymbol{b}_\dagger.$$

Remarks In general, $\boldsymbol{\phi}^*_{\text{joint}} \neq \boldsymbol{\phi}^*_{\text{MAML}}$ based on our analysis. Note that $\boldsymbol{A}_\dagger$ is a weighed average of different $\boldsymbol{A}_i$, but the weights themselves are a function of $\boldsymbol{A}_i$. The reason for the difference between $\boldsymbol{\phi}^*_{\text{joint}}$ and $\boldsymbol{\phi}^*_{\text{MAML}}$ is the difference in moments of input distributions. The two solutions, $\boldsymbol{\phi}^*_{\text{joint}}$ and $\boldsymbol{\phi}^*_{\text{MAML}}$, coincide when $\boldsymbol{A}_i = \boldsymbol{A}\ \forall i$. Furthermore, since $\boldsymbol{\phi}^*_{\text{MAML}}$ was optimized to explicitly minimize $\tilde{F}(\cdot)$, it would lead to better performance after task-specific adaptation. This analysis reveals that there is a clear separation in performance between joint training and meta-learning even in the case of quadratic loss functions. Improved performance with meta-learning approaches have been noted empirically with non-convex loss landscapes induced by neural networks. Our example illustrates that meta learning can provide non-trivial gains over joint training even in simple convex loss landscapes.

E.2 Theoretical Analysis

We first begin by reviewing some definitions and proving some helper lemmas before proving the main theorem.

E.2.1 Notations, Definitions, and Some Properties

- For a vector $\mathbf{u} \in \mathbb{R}^d$, we use $\|\mathbf{u}\|$ to denote the $L2$ norm, i.e. $\sqrt{\mathbf{u}^T\mathbf{u}}$.
- For a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times d}$, we use $\|\boldsymbol{A}\|$ to denote the matrix norm induced by the vector norm,
$$\|\boldsymbol{A}\| = \sup\left\{\frac{\|\boldsymbol{A}\mathbf{u}\|}{\|\mathbf{u}\|} : \mathbf{u} \in \mathbb{R}^d \text{ with } \|\mathbf{u}\| \neq 0\right\}$$
We briefly review two properties of this norm that are useful for us
  - $\|\boldsymbol{A}+\boldsymbol{B}\| \leq \|\boldsymbol{A}\| + \|\boldsymbol{B}\|$ (triangle inequality)
  - $\|\boldsymbol{A}\mathbf{u}\| \leq \|\boldsymbol{A}\|\|\mathbf{u}\|$ (sub-multiplicative property)

  Both can be seen easily by recognizing that an alternate equivalent definition of the induced norm is $\|\boldsymbol{A}\| \geq \frac{\|\boldsymbol{A}\mathbf{u}\|}{\|\mathbf{u}\|}\ \forall \mathbf{u}$.
- For (symmetric) matrices $\boldsymbol{A} \in \mathbb{R}^{d\times d}$, we use $\boldsymbol{A} \succeq 0$ to denote the positive semi-definite nature of the matrix. Similarly, $\boldsymbol{A} \succeq \boldsymbol{B}$ implies that $\mathbf{u}^T(\boldsymbol{A}-\boldsymbol{B})\mathbf{u} \geq 0\ \forall \mathbf{u}$.

Definition 9. *A function $f(\cdot)$ is $G-$***Lipschitz continuous** *iff:*

$$\|\mathcal{L}(\mathbf{u}) - \mathcal{L}(\mathbf{v})\| \leq G\|\mathbf{u}-\mathbf{v}\| \quad \forall\ \mathbf{u},\mathbf{v}$$

Definition 10. *A differentiable function $\mathcal{L}(\cdot)$ is $\beta-$***smooth** *iff:*

$$\|\nabla\mathcal{L}(\mathbf{u}) - \nabla\mathcal{L}(\mathbf{v})\| \leq \beta\|\mathbf{u}-\mathbf{v}\| \quad \forall\ \mathbf{u},\mathbf{v}$$

Definition 11. *A twice-differentiable function $\mathcal{L}(\cdot)$ is $\rho-$***Hessian Lipschitz** *iff:*

$$\|\nabla^2\mathcal{L}(\mathbf{u}) - \nabla^2\mathcal{L}(\mathbf{v})\| \leq \rho\|\mathbf{u}-\mathbf{v}\| \quad \forall\ \mathbf{u},\mathbf{v}$$

Definition 12. *A twice-differentiable function $\mathcal{L}(\cdot)$ is $\mu-$***strongly convex** *iff:*

$$\nabla^2\mathcal{L}(\mathbf{u}) \succeq \mu\boldsymbol{I} \quad \forall\ \mathbf{u}$$

We also review some properties of convex functions that help with our analysis.

Lemma 9. *(Elementery properties of convex functions; see e.g. Boyd and Vandenberghe [300], Shalev-Shwartz [164])*

1. *If $\mathcal{L}(\cdot)$ is differentiable, convex, and $G-$Lipschitz continuous, then*
$$\|\nabla\mathcal{L}(\mathbf{u})\| \leq G \;\; \forall\, \mathbf{u}.$$

2. *If $\mathcal{L}(\cdot)$ is twice-differentiable, convex, and $\beta-$smooth, then*
$$\nabla^2\mathcal{L}(\mathbf{u}) \preceq \beta\boldsymbol{I} \;\; \forall\, \mathbf{u}.$$
This is also equivalent to $\|\nabla^2\mathcal{L}(\mathbf{u})\| \leq \beta \;\; \forall\, \mathbf{u}$. The strong convexity also implies that
$$\|\mathcal{L}(\mathbf{u}) - \mathcal{L}(\mathbf{v})\| \geq \mu\|\mathbf{u}-\mathbf{v}\| \;\; \forall\, \mathbf{u},\mathbf{v}.$$

E.2.2 **Descent Lemma**

In this sub-section, we prove a useful lemma about the dynamics of gradient descent and its contractive property. For this, we first consider a lemma that provides an inequality analogous to that of mean-value theorem.

Lemma 10. *(Fundamental theorem of line integrals) Let $\varphi : \mathbb{R}^d \to \mathbb{R}$ be a differentiable function. Let $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$ be two arbitrary points, and let $\boldsymbol{r}(\gamma)$ be a curve from $\mathbf{u}$ to $\mathbf{v}$, parameterized by scalar γ. Then,*

$$\varphi(\mathbf{v}) - \varphi(\mathbf{u}) = \int_{\gamma[\mathbf{u},\mathbf{v}]} \nabla\varphi(\boldsymbol{r}(\gamma))d\boldsymbol{r} = \int_{\gamma[\mathbf{u},\mathbf{v}]} \nabla\varphi(\boldsymbol{r}(\gamma))\boldsymbol{r}'(\gamma)d\gamma$$

Lemma 11. *(Mean value inequality) Let $\boldsymbol{\varphi} : \mathbf{u} \in \mathbb{R}^d \to \mathbb{R}^m$ be a differentiable function. Let $\nabla\boldsymbol{\varphi}$ be the function Jacobian, such that $\nabla\boldsymbol{\varphi}_{i,j} = \frac{\partial\varphi_i}{\partial\mathbf{u}_j}$. Let $M = \max_{\mathbf{u}\in\mathbb{R}^d} \|\nabla\boldsymbol{\varphi}(\mathbf{u})\|$. Then, we have*
$$\|\boldsymbol{\varphi}(\mathbf{u}) - \boldsymbol{\varphi}(\mathbf{v})\| \leq M\;\|\mathbf{u}-\mathbf{v}\| \;\; \forall\, \mathbf{u},\mathbf{v} \in \mathbb{R}^d$$

Proof. Let the line segment connecting $\mathbf{u}$ and $\mathbf{v}$ be parameterized as $\mathbf{z}(t) = \mathbf{v} + t(\mathbf{u}-\mathbf{v}),\; t \in [0,1]$. Using Lemma 10 for each coordinate, we have that

$$\boldsymbol{\varphi}(\mathbf{u}) - \boldsymbol{\varphi}(\mathbf{v}) = \left(\int_{t=0}^{t=1} \nabla\boldsymbol{\varphi}\big(\mathbf{z}(t)\big)dt\right)\Big(\mathbf{z}(1) - \mathbf{z}(0)\Big) = \left(\int_{t=0}^{t=1} \nabla\boldsymbol{\varphi}\big(\mathbf{z}(t)\big)dt\right)\Big(\mathbf{u}-\mathbf{v}\Big)$$

$$\begin{aligned}
\|\boldsymbol{\varphi}(\mathbf{u}) - \boldsymbol{\varphi}(\mathbf{v})\| &= \left\| \int_{t=0}^{t=1} \nabla\boldsymbol{\varphi}\big(\mathbf{z}(t)\big)\big(\mathbf{u}-\mathbf{v}\big)dt \right\| \\
&\overset{(a)}{\leq} \int_{t=0}^{t=1} \big\|\nabla\boldsymbol{\varphi}\big(\mathbf{z}(t)\big)\big(\mathbf{u}-\mathbf{v}\big)\big\|\, dt \\
&\overset{(b)}{\leq} \int_{t=0}^{t=1} \big\|\nabla\boldsymbol{\varphi}\big(\mathbf{z}(t)\big)\big\| \, \big\|\big(\mathbf{u}-\mathbf{v}\big)\big\|\, dt \\
&\overset{(c)}{\leq} \int_{t=0}^{t=1} M\,\|\mathbf{u}-\mathbf{v}\|\, dt \\
&= M\|\mathbf{u}-\mathbf{v}\| \int_{t=0}^{t=1} dt = M\|\mathbf{u}-\mathbf{v}\|
\end{aligned}$$

Step (a) follows from the Cauchy-Schwartz inequality. Step (b) is a consequence of the sub-multiplicative property. Finally, step (c) is based on the definition of M. $\square$

Lemma 12. *(Contractive property of gradient descent) Let $\varphi : \mathbb{R}^d \to \mathbb{R}$ be an arbitrary function that is $G-$Lipschitz, $\beta-$smooth, and $\mu-$strongly convex. Let $\mathcal{A}lg(\mathbf{u}) = \mathbf{u} - \alpha\nabla\varphi(\mathbf{u})$ be the gradient descent update rule with $\alpha \leq \frac{1}{\beta}$. Then,*

$$\|\mathcal{A}lg(\mathbf{u}) - \mathcal{A}lg(\mathbf{v})\| \leq (1-\alpha\mu)\|\mathbf{u}-\mathbf{v}\| \;\; \forall \; \mathbf{u},\mathbf{v} \in \mathbb{R}^d.$$

Proof. Firstly, the Jacobian of $\mathcal{A}lg(\cdot)$ is given by $\nabla\mathcal{A}lg(\mathbf{u}) = \boldsymbol{I} - \alpha\nabla^2\varphi(\mathbf{u})$. Since $\mu\boldsymbol{I} \preceq \nabla^2\varphi(\mathbf{u}) \preceq \beta\boldsymbol{I}$ $\forall\mathbf{u} \in \mathbb{R}^d$, we can bound the Jacobian as

$$(1-\alpha\beta)\boldsymbol{I} \preceq \nabla\mathcal{A}lg(\mathbf{u}) \preceq (1-\alpha\mu)\boldsymbol{I} \quad \forall\; \mathbf{u} \in \mathbb{R}^d$$

The upper bound also implies that $\|\nabla\mathcal{A}lg(\mathbf{u})\| \leq (1-\alpha\mu)$ $\forall\mathbf{u}$. Using this and invoking Lemma 11 on $\mathcal{A}lg(\cdot)$, we get that

$$\|\mathcal{A}lg(\mathbf{u}) - \mathcal{A}lg(\mathbf{v})\| \leq (1-\alpha\mu)\|\mathbf{u}-\mathbf{v}\|.$$

$\square$

E.2.3 Main Theorem

Using the previous lemmas, we can prove our main theorem. We restate the assumptions and statement of the theorem and then present the proof.
Assumption 1. *(C^2-smoothness) Suppose that $\mathcal{L}(\cdot)$ is twice differentiable and*

- *$\mathcal{L}(\cdot)$ is $G-$Lipschitz in function value.*
- *$\mathcal{L}(\cdot)$ is $\beta-$smooth, or has $\beta-$Lipschitz gradients.*
- *$\mathcal{L}(\cdot)$ has $\rho-$Lipschitz hessian.*

Assumption 2. *(Strong convexity)* Suppose that $\mathcal{L}(\cdot)$ is convex. Further suppose that it is $\mu-$strongly convex.
Theorem 1. *Suppose $\mathcal{L}$ and $\hat{\mathcal{L}} : \mathbb{R}^d \to \mathbb{R}$ satisfy assumptions 1 and 2. Let $\tilde{\mathcal{L}}$ be the function evaluated after a one step gradient update procedure, i.e.*

$$\tilde{\mathcal{L}}(\boldsymbol{\phi}) := \mathcal{L}\big(\boldsymbol{\phi} - \alpha \nabla \hat{\mathcal{L}}(\boldsymbol{\phi})\big).$$

If the step size is selected as $\alpha \leq \min\{\frac{1}{2\beta}, \frac{\mu}{8\rho G}\}$, then $\tilde{\mathcal{L}}$ is convex. Furthermore, it is also $\tilde{\beta} = 9\beta/8$ smooth and $\tilde{\mu} = \mu/8$ strongly convex.

Proof. Consider two arbitrary points $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$. Using the chain rule and our shorthand of $\tilde{\mathbf{u}} \equiv \mathcal{A}lg(\mathbf{u}), \tilde{\mathbf{v}} \equiv \mathcal{A}lg(\mathbf{u})$, we have

$$\begin{aligned}
\nabla\tilde{\mathcal{L}}(\mathbf{u}) - \nabla\tilde{\mathcal{L}}(\mathbf{v}) &= \nabla\mathcal{A}lg(\mathbf{u})\nabla\mathcal{L}(\tilde{\mathbf{u}}) - \nabla\mathcal{A}lg(\mathbf{v})\nabla\mathcal{L}(\tilde{\mathbf{v}}) \\
&= \left(\nabla\mathcal{A}lg(\mathbf{u}) - \nabla\mathcal{A}lg(\mathbf{v})\right)\nabla\mathcal{L}(\tilde{\mathbf{u}}) + \nabla\mathcal{A}lg(\mathbf{v})\left(\nabla\mathcal{L}(\tilde{\mathbf{u}}) - \nabla\mathcal{L}(\tilde{\mathbf{v}})\right).
\end{aligned}$$

We first show the smoothness of the function. Taking the norm on both sides, for the specified α, we have:

$$\begin{aligned}
\|\nabla\tilde{\mathcal{L}}(\mathbf{u}) - \nabla\tilde{\mathcal{L}}(\mathbf{v})\| &= \|\left(\nabla\mathcal{A}lg(\mathbf{u}) - \nabla\mathcal{A}lg(\mathbf{v})\right)\nabla\mathcal{L}(\tilde{\mathbf{u}}) + \nabla\mathcal{A}lg(\mathbf{v})\left(\nabla\mathcal{L}(\tilde{\mathbf{u}}) - \nabla\mathcal{L}(\tilde{\mathbf{v}})\right)\| \\
&\leq \|\left(\nabla\mathcal{A}lg(\mathbf{u}) - \nabla\mathcal{A}lg(\mathbf{v})\right)\nabla\mathcal{L}(\tilde{\mathbf{u}})\| + \|\nabla\mathcal{A}lg(\mathbf{v})\left(\nabla\mathcal{L}(\tilde{\mathbf{u}}) - \nabla\mathcal{L}(\tilde{\mathbf{v}})\right)\|
\end{aligned}$$

due to triangle inequality. We now bound both the terms on the right hand side. We have

$$\begin{aligned}
\|\left(\nabla\mathcal{A}lg(\mathbf{u}) - \nabla\mathcal{A}lg(\mathbf{v})\right)\nabla\mathcal{L}(\tilde{\mathbf{u}})\| &\overset{(a)}{\leq} \|\nabla\mathcal{A}lg(\mathbf{u}) - \nabla\mathcal{A}lg(\mathbf{v})\|\|\nabla\mathcal{L}(\tilde{\mathbf{u}})\| \\
&= \|\Big(\boldsymbol{I} - \alpha\nabla^2\hat{\mathcal{L}}(\mathbf{u})\Big) - \Big(\boldsymbol{I} - \alpha\nabla^2\hat{\mathcal{L}}(\mathbf{v})\Big)\|\|\nabla\mathcal{L}(\tilde{\mathbf{u}})\| \\
&= \alpha\|\nabla^2\hat{\mathcal{L}}(\mathbf{u}) - \nabla^2\hat{\mathcal{L}}(\mathbf{v})\|\|\nabla\mathcal{L}(\tilde{\mathbf{u}})\| \\
&\overset{(b)}{\leq} \alpha\rho\|\mathbf{u} - \mathbf{v}\|\|\nabla\mathcal{L}(\tilde{\mathbf{u}})\| \\
&\overset{(c)}{\leq} \alpha\rho G\|\mathbf{u} - \mathbf{v}\|
\end{aligned}$$

where (a) is due to the sub-multiplicative property of norms (Cauchy-Schwarz inequality), (b) is due to the Hessian Lipschitz property, and (c) is due to the Lipschitz property in function value. Similarly, we can bound the second term as

$$\begin{aligned}
\|\nabla \mathcal{A}lg(\mathbf{v})\left(\nabla \mathcal{L}(\tilde{\mathbf{u}}) - \nabla \mathcal{L}(\tilde{\mathbf{v}})\right)\| &= \left\|\left(\boldsymbol{I} - \alpha \nabla^2 \hat{\mathcal{L}}(\mathbf{v})\right)\left(\nabla \mathcal{L}(\tilde{\mathbf{u}}) - \nabla \mathcal{L}(\tilde{\mathbf{v}})\right)\right\| \\
&\overset{(a)}{\leq} (1-\alpha\mu)\|\nabla \mathcal{L}(\tilde{\mathbf{u}}) - \nabla \mathcal{L}(\tilde{\mathbf{v}})\| \\
&\overset{(b)}{\leq} (1-\alpha\mu)\beta\|\tilde{\mathbf{u}} - \tilde{\mathbf{v}}\| \\
&\overset{(c)}{=} (1-\alpha\mu)\beta\|\mathcal{A}lg(\mathbf{u}) - \mathcal{A}lg(\mathbf{v})\| \\
&\overset{(d)}{\leq} (1-\alpha\mu)\beta(1-\alpha\mu)\|\mathbf{u} - \mathbf{v}\| \\
&= (1-\alpha\mu)^2\beta\|\mathbf{u} - \mathbf{v}\|
\end{aligned}$$

Here, (a) is due to $\boldsymbol{I} - \alpha \nabla^2 \hat{\mathcal{L}}(\mathbf{v})$ being symmetric, PSD, and $\lambda_{\max}\left(\boldsymbol{I} - \alpha \nabla^2 \hat{\mathcal{L}}(\mathbf{v})\right) \leq 1 - \alpha\mu$. Step (b) is due to smoothness of $\mathcal{L}(\cdot)$. Step (c) is simply using our shorthand of $\tilde{\mathbf{u}} \equiv \mathcal{A}lg(\mathbf{u}), \tilde{\mathbf{v}} \equiv \mathcal{A}lg(\mathbf{u})$. Finally, step (d) is due to Lemma 12 on $\mathcal{A}lg(\cdot)$. Putting the previous pieces together, when $\alpha \leq \min\left\{\frac{1}{2\beta}, \frac{\mu}{8\rho G}\right\}$, we have that

$$\begin{aligned}
\|\nabla \tilde{\mathcal{L}}(\mathbf{u}) - \nabla \tilde{\mathcal{L}}(\mathbf{v})\| &\leq \|\left(\nabla \mathcal{A}lg(\mathbf{u}) - \nabla \mathcal{A}lg(\mathbf{v})\right)\nabla \mathcal{L}(\tilde{\mathbf{u}})\| + \|\nabla \mathcal{A}lg(\mathbf{v})\left(\nabla \mathcal{L}(\tilde{\mathbf{u}}) - \nabla \mathcal{L}(\tilde{\mathbf{v}})\right)\| \\
&\leq \alpha\rho G\|\mathbf{u} - \mathbf{v}\| + (1-\alpha\mu)^2\beta\|\mathbf{u} - \mathbf{v}\| \\
&\leq \left(\frac{\mu}{8} + \beta\right)\|\mathbf{u} - \mathbf{v}\| \\
&\leq \frac{9\beta}{8}\|\mathbf{u} - \mathbf{v}\|
\end{aligned}$$

and thus $\tilde{\mathcal{L}}(\cdot)$ is $\tilde{\beta} = \frac{9\beta}{8}$ smooth. Similarly, for the lower bound, we have

$$\begin{aligned}
\|\nabla \tilde{\mathcal{L}}(\mathbf{u}) - \nabla \tilde{\mathcal{L}}(\mathbf{v})\| &= \|\left(\nabla \mathcal{A}lg(\mathbf{u}) - \nabla \mathcal{A}lg(\mathbf{v})\right)\nabla \mathcal{L}(\tilde{\mathbf{u}}) + \nabla \mathcal{A}lg(\mathbf{v})\left(\nabla \mathcal{L}(\tilde{\mathbf{u}}) - \nabla \mathcal{L}(\tilde{\mathbf{v}})\right)\| \\
&\geq \|\nabla \mathcal{A}lg(\mathbf{v})\left(\nabla \mathcal{L}(\tilde{\mathbf{u}}) - \nabla \mathcal{L}(\tilde{\mathbf{v}})\right)\| - \|\left(\nabla \mathcal{A}lg(\mathbf{u}) - \nabla \mathcal{A}lg(\mathbf{v})\right)\nabla \mathcal{L}(\tilde{\mathbf{u}})\|
\end{aligned}$$

using the triangle inequality. We now again proceed to bound each term. We already derived an upper bound for the second term on the right side, and we require a lower bound for the first term.

$$\begin{aligned}
\|\nabla \mathcal{A}lg(\mathbf{v}) \left(\nabla \mathcal{L}(\tilde{\mathbf{u}}) - \nabla \mathcal{L}(\tilde{\mathbf{v}})\right)\| &= \left\| \Big(\boldsymbol{I} - \alpha \nabla^2 \hat{\mathcal{L}}(\mathbf{v})\Big) \Big(\nabla \mathcal{L}(\tilde{\mathbf{u}}) - \nabla \mathcal{L}(\tilde{\mathbf{v}})\Big) \right\| \\
&\overset{(a)}{\geq} (1-\alpha\beta) \|\nabla \mathcal{L}(\tilde{\mathbf{u}}) - \nabla \mathcal{L}(\tilde{\mathbf{v}})\| \\
&\overset{(b)}{\geq} (1-\alpha\beta)\mu \|\tilde{\mathbf{u}} - \tilde{\mathbf{v}}\| \\
&= (1-\alpha\beta)\mu \|\mathbf{u} - \alpha \nabla \hat{\mathcal{L}}(\mathbf{u}) - \mathbf{v} + \alpha \nabla \hat{\mathcal{L}}(\mathbf{v})\| \\
&\geq \mu(1-\alpha\beta) \Big(\|\mathbf{u} - \mathbf{v}\| - \alpha \|\nabla \hat{\mathcal{L}}(\mathbf{u}) - \nabla \hat{\mathcal{L}}(\mathbf{v})\| \Big) \\
&\overset{(c)}{\geq} \mu(1-\alpha\beta) \Big(\|\mathbf{u} - \mathbf{v}\| - \alpha\beta \|\mathbf{u} - \mathbf{v}\| \Big) \\
&\geq \mu(1-\alpha\beta)^2 \|\mathbf{u} - \mathbf{v}\| \\
&\geq \frac{\mu}{4} \|\mathbf{u} - \mathbf{v}\|
\end{aligned}$$

Here (a) is due to $\lambda_{\min}\Big(\boldsymbol{I} - \alpha \nabla^2 \hat{\mathcal{L}}(\mathbf{v})\Big) \geq 1 - \alpha\beta$, (b) is due to strong convexity, and (c) is due to smoothness of $\hat{\mathcal{L}}(\cdot)$. Using both the terms, we have that

$$\begin{aligned}
\|\nabla \tilde{\mathcal{L}}(\mathbf{u}) - \nabla \tilde{\mathcal{L}}(\mathbf{v})\| &\geq \|\nabla \mathcal{A}lg(\mathbf{v}) \left(\nabla \mathcal{L}(\tilde{\mathbf{u}}) - \nabla \mathcal{L}(\tilde{\mathbf{v}})\right)\| - \|\left(\nabla \mathcal{A}lg(\mathbf{u}) - \nabla \mathcal{A}lg(\mathbf{v})\right) \nabla \mathcal{L}(\tilde{\mathbf{u}})\| \\
&\geq \left(\frac{\mu}{4} - \frac{\mu}{8}\right) \|\mathbf{u} - \mathbf{v}\| \geq \frac{\mu}{8} \|\mathbf{u} - \mathbf{v}\|
\end{aligned}$$

Thus the function $\tilde{\mathcal{L}}(\cdot)$ is $\tilde{\mu} = \frac{\mu}{8}$ strongly convex. □

E.3 Additional Experimental Details

For all experiments, we trained our FTML method with 5 inner batch gradient descent steps with step size $\alpha = 0.1$. We use an inner batch size of 10 examples for MNIST and pose prediction and 25 datapoints for CIFAR. Except for the CIFAR NML experiments, we train the convolutional networks using the Adam optimizer with default hyperparameters [203]. We found Adam to be unstable on the CIFAR setting for NML and instead used SGD with momentum, with a momentum parameter of 0.9 and a learning rate of 0.01 for the first 5 thousand iterations, followed by a learning rate of 0.001 for the rest of learning.

For the MNIST and CIFAR experiments, we use the cross entropy loss, using label smoothing with $\epsilon = 0.1$ as proposed by Szegedy et al. [301]. We also use this loss for the inner loss in the FTML objective.

In the MNIST and CIFAR experiments, we use a convolutional neural network model with 5 convolution layers with 32 3×3 filters interleaved with batch normalization and ReLU nonlinearities. The output of the convolution layers is flattened and followed by a linear layer and a softmax, feeding into the output. In the pose prediction experiment, all models use a convolutional neural network with 4 convolution layers each with 16 5×5 filters. After the convolution layers, we use a spatial soft-argmax to extract learned feature points, an architecture that has previously been shown to be effective for spatial tasks [302, 303]. We then pass the feature points through 2 fully connected layers with 200 hidden units and a linear layer to the output. All layers use batch normalization and ReLU nonlinearities.