# Learning to Learn with Gradients

*Chelsea Finn*

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2018-105
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2018/EECS-2018-105.html

August 8, 2018

**Learning to Learn with Gradients**

by

Chelsea B. Finn

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Assistant Professor Sergey Levine, Chair
Professor Pieter Abbeel, Chair
Professor Trevor Darrell
Professor Thomas Griffiths

Summer 2018

**Learning to Learn with Gradients**

**Abstract**

Learning to Learn with Gradients

by

Chelsea B. Finn

Doctor of Philosophy in Computer Science

University of California, Berkeley

Assistant Professor Sergey Levine, Chair
Professor Pieter Abbeel, Chair

Humans have a remarkable ability to learn new concepts from only a few examples and quickly adapt to unforeseen circumstances. To do so, they build upon their prior experience and prepare for the ability to adapt, allowing the combination of previous observations with small amounts of new evidence for fast learning. In most machine learning systems, however, there are distinct train and test phases: training consists of updating the model using data, and at test time, the model is deployed as a rigid decision-making engine. In this thesis, we discuss algorithms for learning to learn, or meta-learning, which aim to endow machines with flexibility akin to that of humans. Instead of deploying a fixed, non-adaptable system, these meta-learning techniques explicitly train for the ability to quickly adapt so that, at test time, they can learn quickly when faced with new scenarios.

To study the problem of learning to learn, we first develop a clear and formal definition of the meta-learning problem, its terminology, and desirable properties of meta-learning algorithms. Building upon these foundations, we present a class of model-agnostic meta-learning methods that embed gradient-based optimization into the learner. Unlike prior approaches to learning to learn, this class of methods focus on acquiring a transferable *representation* rather than a good learning rule. As a result, these methods inherit a number of desirable properties from using a fixed optimization as the learning rule, while still maintaining full expressivity, since the learned representations can control the update rule.

We show how these methods can be extended for applications in motor control by combining elements of meta-learning with techniques for deep model-based reinforcement learning, imitation learning, and inverse reinforcement learning. By doing so, we build simulated agents that can adapt in dynamic environments, enable real robots to learn to manipulate new objects by watching a video of a human, and allow humans to convey goals to robots with only a few images. Finally, we conclude by discussing open questions and future directions in meta-learning, aiming to identify the key shortcomings and limiting assumptions of our existing approaches.

# ACKNOWLEDGMENTS

First and foremost, I am tremendously grateful for my advisers Sergey Levine and Pieter Abbeel for their continuous support and guidance throughout my PhD, and for providing me the freedom to work on a variety of problems. Thank you to my committee members Trevor Darrell and Tom Griffiths for their support, guidance, and fruitful conversations. I am also grateful for my undergraduate adviser, Seth Teller, for inspiring me to pursue a career in research.

I'm very happy to have had the opportunity to collaborate with an amazing set of students, faculty, and researchers throughout my PhD. For the work in this thesis, I enjoyed working with Kelvin Xu, Tianhe Yu, Ellis Ratner, Ignasi Clavera, Anusha Nagabandi, Anca Dragan, Annie Xie, Avi Singh, Sudeep Dasari, and Tianhao Zhang. I would especially like to thank all of the undergraduate and masters students that I worked with during my PhD, including Tianhe Yu, Frederik Ebert, Annie Xie, Sudeep Dasari, Anurag Ajay, and Xin Yu Tan, for your enthusiasm and hard work in the projects we undertook and your patience as I learned how to best advise you. Beyond the work in this thesis, I had the pleasure of working with a number of other students and faculty, including Trevor Darrell, Justin Fu, Alex Lee, Aravind Srinivas, Coline Devin, Erin Grant, and Eric Tzeng. Thank you to all of the talented researchers in Berkeley AI Research for fostering a collaborative and friendly environment, particularly to my nearby labmates, Sandy Huang, Alex Lee, Greg Kahn, Coline Devin, Abhishek Gupta, and Marvin Zhang.

I'm grateful to have had the opportunity to do an internship at Google Brain under Vincent Vanhoucke, collaborating with Sergey Levine and Ian Goodfellow and working with the original Arm Farm. I was fortunate enough to have tremendous freedom from Vincent during my internship, as well as support from and thoughtful conversations with many, including Peter Pastor, Mrinal Kalakrishnan, George Dahl, and Jon Barron.

My time at Berkeley would not have been the same without the produce of Berkeley Bowl, the beautiful Berkeley hills, and the camaraderie of Team Fuego, including Johan, Suzanne, Maggie, Rosalie, Avi, Nat, Lydia, Jon, Eric, Chris, Chris, Evan, Jessica, Rob, Joe, Neil, Spencer, Michel, Vince, Lance, and many others. I would also like to thank Neil for deadline gnocchi blobs, infinite water refills, and of course, your love and support.

Finally, this thesis is dedicated to my parents, Leslie Garrison and Jeff Finn, for all the years of your love and support.

# CONTENTS


1 INTRODUCTION 1

I FOUNDATIONS 5

2 PROBLEM STATEMENT 6
- 2.1 Meta-Learning Problem and Terminology 6
- 2.2 Design Space of Meta-Learning Problems 8

3 DESIRABLE PROPERTIES OF META-LEARNING ALGORITHMS 9
- 3.1 Expressive Power of Meta-Learning Algorithms 9
- 3.2 Consistent Learning Algorithms 11
- 3.3 Handling Ambiguity in Learning 13

II META-LEARNING WITH GRADIENTS 14

4 A MODEL-AGNOSTIC META-LEARNING ALGORITHM 15
- 4.1 General Algorithm 16
- 4.2 Species of MAML 18
  - 4.2.1 Supervised Regression and Classification 19
  - 4.2.2 Reinforcement Learning 20
- 4.3 Implementation and First-Order Approximation 21
- 4.4 Expressive Power of Model-Agnostic Meta-Learning 23
  - 4.4.1 Universality of the One-Shot Gradient-Based Learner 24
  - 4.4.2 General Universality of the Gradient-Based Learner 29
  - 4.4.3 Loss Functions 30
- 4.5 Related Work 31
- 4.6 Experimental Evaluation 32
  - 4.6.1 Regression 33
  - 4.6.2 Classification 35
  - 4.6.3 Reinforcement Learning 37
  - 4.6.4 Empirical Study of Inductive Bias 39
  - 4.6.5 Effect of Depth 42
- 4.7 Discussion 43

5 A PROBABILISTIC MODEL-AGNOSTIC META-LEARNING ALGORITHM 45
- 5.1 Overview and Preliminaries 45

5.2 Gradient-Based Meta-Learning with Variational Inference 46
5.3 Probabilistic Model-Agnostic Meta-Learning with Hybrid Inference 48
5.4 Adding Additional Dependencies 50
5.5 Related Work 51
5.6 Experiments 52
5.6.1 Discussion and Future Work 56

III EXTENSIONS AND APPLICATIONS 58

6 ONLINE ADAPTIVE CONTROL 59
6.1 Meta-Learning for Adaptive Control 59
6.1.1 Preliminaries: Model-Based RL 61
6.1.2 Overview 61
6.1.3 Online Model Adaptive Control 62
6.1.4 General Algorithm of Meta-Learning for Adaptive Control 64
6.1.5 Recurrence-Based Adaptive Control (RBAC) 65
6.1.6 Gradient-Based Adaptive Control (GBAC) 65
6.1.7 Related Work 66
6.1.8 Experiments 67
6.1.9 Discussion 71

7 FEW-SHOT IMITATION LEARNING 73
7.1 Meta-Imitation Learning 73
7.1.1 Overview 74
7.1.2 Two-Head Architecture: Learning a Loss for Fast Adaptation 75
7.1.3 Learning to Imitate without Expert Actions 76
7.1.4 Model Architectures for Meta-Imitation Learning 77
7.1.5 Related Work 79
7.1.6 Experiments 80
7.1.7 Discussion and Future Work 85
7.2 One-Shot Imitation from Humans 86
7.2.1 Problem Overview 87
7.2.2 Domain-Adaptive Meta-Learning 88
7.2.3 Learned Temporal Adaptation Objectives 89
7.2.4 Probabilistic Interpretation 91
7.2.5 Model Architectures 93
7.2.6 Related Work 95
7.2.7 Experiments 97
7.2.8 Discussion 102

8 FEW-SHOT INTENT INFERENCE 104
8.1 Learning a Prior over Intent via Meta Inverse Reinforcement Learning 105
8.1.1 Preliminaries and Overview 107
8.1.2 Learning to Learn Rewards 108
8.1.3 Related Work 112
8.1.4 Experiments 113
8.1.5 Discussion 117
8.2 Few-Shot Goal Inference for Visuomotor Learning and Planning 117
8.2.1 Overview 119
8.2.2 Problem Set-up 119
8.2.3 Meta-learning for Few-Shot Goal Inference 120
8.2.4 Few-Shot Goal Inference for Learning and Planning 121
8.2.5 Related Work 123
8.2.6 Experiments 124
8.2.7 Discussion 130
9 CONCLUSION 131

IV APPENDICES 134
A MODEL-AGNOSTIC META-LEARNING METHODS 135
A.1 Supplementary Proofs for 1-Shot Universality 135
A.1.1 Proof of Lemma 4.4.1 135
A.1.2 Proof of Lemma A.1.1 137
A.1.3 Form of linear weight matrices 138
A.1.4 Output function 139
A.2 Full K-Shot Proof of Universality 140
A.3 Supplementary Proof for K-Shot Universality 144
A.4 Universality with Deep ReLU Networks 145
A.5 Proof of Theorem 4.4.1 146
A.6 Proof of Theorem 4.4.2 146
A.7 MAML Experimental Details 147
A.7.1 Classification 147
A.7.2 Reinforcement Learning 147
A.7.3 Inductive Bias Experiments 147
A.7.4 Depth Experiments 148
A.8 MAML Additional Sinusoid Results 149
A.9 MAML Additional Comparisons 149
A.9.1 Multi-task baselines 149

A.9.2 Context vector adaptation 152
A.10 Ambiguous CelebA Details 153
A.11 PLATIPUS Experimental Details 153
A.12 PLATIPUS MiniImagenet Comparison 154
B EXTENSIONS TO CONTROL 155
B.1 Online Adaptation: Additional Experiments and Experimental Details 155
B.1.1 Model Prediction Errors: Pre-update vs. Post-update 155
B.1.2 Effect of Meta-Training Distribution 157
B.1.3 Learning Curves 158
B.1.4 Reward functions 158
B.1.5 Hyperparameters 159
B.2 MIL Experimental Details 160
B.2.1 Simulated Reaching 160
B.2.2 Simulated Pushing 161
B.2.3 Real-World Placing 161
B.3 DAML Hyperparameter and Experimental Details 163
B.3.1 Data collection 163
B.3.2 Architecture Choices 163
B.3.3 Placing, Pushing, and Pick-and-Place Experiments 163
B.3.4 Diverse Human Demonstration Experiments 165
B.3.5 Sawyer Robot Experiments 165
B.3.6 Simulated Pushing Experiment 166
B.4 MandRIL Experimental Details 166
B.4.1 Hyperparameters 166
B.4.2 Environment Details 167
B.5 MandRIL Detailed Meta-Objective Derivation 168
B.6 FLO Experimental Details 170
B.6.1 Model Architecture 170
B.6.2 Autoencoder Comparison Details 170

# 1

# INTRODUCTION

*A tree that is unbending, is easily broken.*

— Lao Tzu

Humans have a remarkable ability to learn new concepts from only a few examples and quickly adapt to unforeseen circumstances. To do so, they build upon their prior experience, reusing concepts and abstractions that they have built up over time to efficiently adapt from only small amounts of new evidence. How can we build intelligent systems with the same versatility and flexibility?

One critical piece of the puzzle, we argue, is the form of the data. To see this, consider the standard paradigm in machine learning (ML) of focusing on one particular task — e.g. recognizing speech (Hannun et al., 2014), translating text (Wu et al., 2016), classifying objects in images (Krizhevsky et al., 2012), playing Atari breakout (Mnih et al., 2013), or screwing on a bottle cap (Levine et al., 2016a) — and then training a model or policy, end-to-end, to solve that task from scratch, i.e. from a random initialization. Within the ML community, end-to-end learning from scratch has often been viewed as the gold standard, since it doesn't require substantial domain-specific human knowledge or expertise to "solve" the task. See, for example, Collobert et al. (2011), Yi et al. (2014), and Levine et al. (2016a). Yet, from the perspective of how humans learn, it makes absolutely no sense to have a system learn a single, individual task from scratch. This is like asking a human baby to become an expert in chess before knowing how to pick up a chess piece; or asking an infant to learn to translate English sentences into French before developing an understanding of basic vocabulary in either language. These existing systems are siloed within very narrow environments, often for the sake of practicality but at the expense of narrow experiences that are insufficient for developing common sense. If we want systems that exhibit the generality of human intelligence, they must not require millions of datapoints for each and every new task, concept, or environment.

How can we instead build systems that can quickly and efficiently learn a broad range of new concepts and skills? Perhaps to do so, we can look at the data on which existing systems are trained. For example, the MNIST (LeCun et al., 1998) and CIFAR-10 (Krizhevsky and G. Hinton, 2009) datasets both have 60,000 images, split up evenly within 10 classes. These datasets are relatively small but have arguably driven progress in machine learning research. Notably, these datasets are distinctly different from the experiences of a human. Instead of seeing 6,000 instances of 10 different objects (e.g. 6,000 forks, 6,000 bottles, etc), humans experience data that is much closer to the transpose of that: 10 instances of 6,000 different objects. With this level of diversity, it is not very surprising that humans can generalize so effectively.

While there is no doubt that the distribution and nature of the data play a large role in generalization, training a system on diverse datasets does not, in and of itself, lead to adaptability. We instead need to transfer knowledge from diverse prior experiences when trying to learn new tasks. Transfer learning is a long-standing subfield within machine learning (Caruana, 1993), studying the ability to leverage prior sets of data when learning from new data. Arguably, one of the biggest modern success stories of transfer learning is the technique of pre-training on large amounts of previously available data and then fine-tuning the pre-trained model on data from new tasks. This technique has been wildly successful for training convolutional networks initialized with supervised pretraining on ImageNet (Deng et al., 2009; Donahue et al., 2014; Sharif Razavian et al., 2014; Yosinski et al., 2014; Russakovsky et al., 2015), and more recently, for pre-training language models on large corpora (Dai and Le, 2015; P. Ramachandran et al., 2016; Howard and Ruder, 2018; Radford et al., 2018). Yet, pre-training techniques will only go so far; their performance is limited when fine-tuning with only a handful of examples (Vinyals et al., 2016; Ravi and Larochelle, 2017). This few-shot learning setting is much more challenging, but certainly possible to solve, given the ability of humans to handle such small amounts of data.

In this thesis, we'll be considering an approach to transfer learning that optimizes for transferability and fast learning. This class of methods explicitly trains for the ability to learn new concepts, or *learns how to learn*. While the concept of learning-to-learn, or meta-learning, is not new (Schmidhuber, 1987; Naik and Mammone, 1992; S. Bengio et al., 1992; Thrun and Pratt, 1998), modern techniques in deep learning and gradient-based optimization, along with increased computational power and large datasets, motivate us to revisit this approach in a new light.

Meta-learning can also be viewed as learning the structure among previously seen tasks or concepts such that this learned prior can be combined with small amounts of

new data to make generalizable inferences. From this perspective, there is a close relation between meta-learning and hierarchical Bayesian modeling (Tenenbaum, 1999; Fei-Fei et al., 2003; Lawrence and Platt, 2004; K. Yu et al., 2005; J. Gao et al., 2008; Daumé III, 2009; Lake et al., 2011; Wan et al., 2012; H. Edwards and Storkey, 2017), an approach which has successfully been applied to few-shot learning problems. This concept of meta-learning as learning a prior in hierarchical Bayesian models will be useful when aiming to reason about uncertainty in learning and for developing intuitions about different approaches.

Prior approaches to meta-learning have largely fell into one of two categories – methods that train large *black-box* neural network models to learn from data that is passed in, and those that incorporate *structure* of known optimization procedures into the learner. In the former, a deep neural network model, such as an LSTM (Hochreiter et al., 2001; Andrychowicz et al., 2016; Z. Li et al., 2017; Ravi and Larochelle, 2017), a Neural Turing Machine (Santoro et al., 2016), or a model with another form of memory or recurrence (J. Ba et al., 2016; Munkhdalai and H. Yu, 2017), is trained to "learn" from datapoints that are provided as input. The model either also takes as input a new, unlabeled datapoint and must predict its label (Hochreiter et al., 2001; Santoro et al., 2016), or predicts the weights of another neural network model that can solve the task (Andrychowicz et al., 2016; Z. Li et al., 2017; Ravi and Larochelle, 2017; Ha et al., 2017). This approach has been extended to the reinforcement learning setting by Wang and Hebert (2016) and Duan et al. (2016b), learning a recurrent neural network policy that does not reset its hidden state across episodes so that it can "learn" from previous episodes. These approaches are expressive and can be applied to a wide range of problems. Yet, without any structure, learning these black-box learning procedures from scratch can be difficult and inefficient.

A number of prior works have aimed to incorporate structure into the meta-learning process. In particular, one approach for few-shot classification is to learn to compare examples in a learned metric space using, e.g., Siamese networks (Koch et al., 2015) or recurrent models (Vinyals et al., 2016; Shyam et al., 2017; Snell et al., 2017). These approaches have generated some of the most successful results, but are more difficult to directly extend to other problems, such as reinforcement learning. This prior literature motivates the development of a method with the generality of black-box approaches while incorporating structure, like the latter. Further, these works have been developed largely independently without common terminology or even a common problem statement. If we hope to push research and understanding of these methods forward, it would be helpful to have a set of guiding principles and concrete, yet sufficiently-general problem statement. We hope to work towards such guidelines in this thesis.

The contributions of this thesis are as follows:

- In Chapter 2, we consider the meta-learning problem statement and give examples of different instantiations of this problem statement. Our problem definition and notation encapsulates both meta-supervised and meta-reinforcement learning settings.
- In Chapter 3, we develop a set of measurable properties of desirable meta-learning algorithms, aiming to suggest a set of guiding principles for those developing new meta-learning algorithms.
- In Chapter 4, we present our core contribution, a simple, yet general approach to meta-learning that builds upon the success of fine-tuning from pre-trained initializations. We analyze the theoretical properties of this model-agnostic meta-learning algorithm, and empirically compare it to prior approaches in both few-shot supervised learning problems and fast reinforcement learning. A subset of this work was published previously as Finn et al. (2017a) and Finn and Levine (2018).
- In Chapter 5, we propose a probabilistic version of the algorithm presented in Chapter 4. This method was previously published as Finn et al. (2018).
- Unlike the standard leraning-to-learn setting, we can consider a temporal window as a task. We develop this idea further in Chapter 6 and evaluate our approach for online adaptation to varying simulated environments. This work appeared previously as Clavera et al. (2018)
- In Chapter 7, we develop the notion that the inner optimization can use a *learned* loss function, and show how we can use meta-learning to have a robot learn from a single demonstration, including a demonstration that consists of a raw video of a human performing the task. This work was published previously as Finn et al. (2017b) and T. Yu et al. (2018).
- In Chapter 8, we consider the problem of inferring the intention of a human from the person's behavior by building upon prior experience. We show how this idea can be used for learning rewards and objectives in navigation and robotic manipulation. This work appeared previously as Xu et al. (2018) and Xie et al. (2018).
- Finally, we conclude by discussing open challenges in learning to learn in Chapter 9.

# Part I

# FOUNDATIONS

# 2

# PROBLEM STATEMENT

## 2.1 META-LEARNING PROBLEM AND TERMINOLOGY

The goal of few-shot meta-learning is to train a model that can quickly adapt to a new task using only a few datapoints and training iterations. To accomplish this, the model or learner is trained during a meta-learning phase on a set of tasks, such that the trained model can quickly adapt to new tasks using only a small number of examples or trials. In effect, the meta-learning problem treats entire tasks as training examples. In this section, we formalize this meta-learning problem setting in a general manner, including brief examples of different learning domains.

We consider a model, denoted $f$, that maps observations $\mathbf{x}$ to outputs $\mathbf{y}$. During meta-learning, the model is trained to be able to adapt to a large or infinite number of tasks. Since we would like to apply our framework to a variety of learning problems, from classification to reinforcement learning, we introduce a generic notion of a learning task below. Formally, each task $\mathcal{T} = \{\mathcal{L}(\theta, \mathcal{D}), \rho(\mathbf{x}_1), \rho(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{y}_t), H\}$ consists of a loss function $\mathcal{L}$ that takes as input the model's parameters $\theta$ and a dataset $\mathcal{D}$, a distribution over initial observations $\rho(\mathbf{x}_1)$, a transition distribution $\rho(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{y}_t)$, and an episode length $H$. In i.i.d. supervised learning problems, the length $H = 1$ and a dataset $\mathcal{D}$ consists of labeled input, output pairs, $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1)^{(k)}\}$. Whereas, in reinforcement learning problems, the model $f$ may generate samples of length $H$ by choosing an output $\hat{\mathbf{y}}_t$ at each time $t$; hence, the dataset passed to the loss function consists of trajectories rolled-out by the model: $\mathcal{D} = \{(\mathbf{x}_1, \hat{\mathbf{y}}_1, \ldots, \mathbf{x}_H, \hat{\mathbf{y}}_H)^{(k)}\}$. The loss $\mathcal{L}(\theta, \mathcal{D}) \to \mathbb{R}$, provides task-specific feedback for the model $f_\theta$, which might be in the form of a misclassification loss or a cost function in a Markov decision process.

In our meta-learning scenario, we consider a distribution over tasks $p(\mathcal{T})$ that we want our model to be able to adapt to. In the $K$-shot learning setting, the model is trained to learn a new task $\mathcal{T}_i$ drawn from $p(\mathcal{T})$ from only $K$ samples drawn from $q_i$, denoted as $\mathcal{D}_i^{tr}$,

| symbol | terminology | examples or more details |
|---|---|---|
| $\mathcal{T}$ | task | entity being learned or adapted to, corresponds to an objective, domain, environment, or combinations thereof |
| $p(\mathcal{T})$ | task distribution | distribution of tasks from which the meta-training and meta-testing tasks are drawn. |
| $\{\mathcal{T}_i\} \sim p(\mathcal{T})$ | meta-training tasks | set of tasks used for meta-learning |
| $\{\mathcal{D}_{\mathcal{T}_i}\}$ | meta-training set | set of datasets corresponding to the meta-training tasks; the algorithm will learn to learn from data in these datasets |
| $\{\mathcal{T}_j\} \sim p(\mathcal{T})$ | meta-test tasks | set of tasks used for evaluation; the learned learning procedure will be evaluated on its ability to learn these tasks |
| $\{\mathcal{D}_{\mathcal{T}_j}\}$ | meta-test set | set of datasets corresponding to meta-test tasks |
| $\mathcal{D}_{\mathcal{T}}^{\text{tr}}$ | training set (support set) | training data for task $\mathcal{T}$, usually $K$ datapoints sampled from $\mathcal{D}_{\mathcal{T}}$ |
| $\mathcal{D}_{\mathcal{T}}^{\text{test}}$ | test set (query set) | test data for task $\mathcal{T}$, sampled from $\mathcal{D}_{\mathcal{T}}$ |

Table 1: Summary of meta-learning terminology used in this paper. Terms sometimes used in other literature is shown in parentheses.

and feedback $\mathcal{L}_{\mathcal{T}_i}$ generated by $\mathcal{T}_i$. During meta-training, a task $\mathcal{T}_i$ is sampled from $p(\mathcal{T})$, the model is trained with $K$ samples using feedback from the corresponding loss $\mathcal{L}_{\mathcal{T}_i}$ from $\mathcal{T}_i$, and then tested on new samples from $\mathcal{T}_i$, denoted as $\mathcal{D}_i^{\text{test}}$. The model $f$ is then improved by considering how the *test* error on new data $\mathcal{D}_i^{\text{test}}$ changes with respect to the parameters. In effect, the test error on sampled tasks $\mathcal{T}_i$ serves as the training error of the meta-learning process. At the end of meta-training, new tasks are sampled from $p(\mathcal{T})$, and meta-performance is measured by the model's performance after learning from $K$ samples. Generally, tasks used for meta-testing are held out during meta-training.

In Table 1, we overview meta-learning terminology and notation used in this document. In essence, a meta-learning algorithm learns to learn tasks using data from tasks in the meta-training set. After meta-learning, the learned learning algorithm is evaluated

in its ability to learn new tasks in the meta-test set. We will use the term 'task' broadly to encapsulate a concept to be learned, a domain to be adapted to, or combinations thereof.

## 2.2 DESIGN SPACE OF META-LEARNING PROBLEMS

As we discussed previously, in meta-supervised learning settings, the training dataset $\mathcal{D}_{\mathcal{T}}^{\text{train}}$ and test dataset $\mathcal{D}_{\mathcal{T}}^{\text{test}}$ for each task $\mathcal{T}$ are fully supervised; whereas, in meta-reinforcement learning, each dataset consists of roll-outs from the policy. In Section 8.1.2, we will also explore how the training and test data and their objective can correspond to demonstrations used for inverse reinforcement learning. While in each of these examples, the training and test data (and the objectives on them) are the same, notice that this does not *need* to be true. The only constraints for proper meta-learning are that (a) the training set is informative for solving the test set, and (b) the test set and its corresponding objective allow for optimization of the desired learning outcome.

We take advantage of this observation in a simple way in Section 4.2.2 by using different policy gradient estimators for the train and test data. But, this observation is most interesting because it opens up a wide design space of possible meta-learning instantiations and algorithms: learning to learn in some manner via some supervision. The training data need not even be supervised in a standard way. As we will show in Sections 7.2 and 8.2, we can learn to learn from weak supervision via strong supervision. In this case, the training data for each task is weakly supervised while the test data contains full supervision. With this insight, we can develop meta-imitation learning algorithms that allow a robot learn how to learn from videos of humans using fully-supervised robot demonstrations (see Section 7.2); and we can learn to learn a reward function from only positive examples of success (see Section 8.2). One can also take this one step further and use unsupervised training data with supervised test data to learn unsupervised learning algorithms using supervised data, as proposed by Metz et al. (2018).

# 3

# DESIRABLE PROPERTIES OF META-LEARNING ALGORITHMS

In this chapter, we present three concrete and measurable properties of meta-learning algorithms and in particular, the class of learning procedures that they acquire. The properties that we consider relate to the expressive power of the meta-learning algorithm, the consistency of the acquired learning procedures, and the ability to handle ambiguity. This list of properties is by no means complete: there are other properties that are also desirable, such as optimizability and simplicity. But, we will start with these three because we have concrete means to measure them. Further, the proposed metrics are not necessarily the ideal way to measure the overarching properties; but, we hope that they can serve as a starting point for thinking about how different meta-learning algorithms compare.

## 3.1 EXPRESSIVE POWER OF META-LEARNING ALGORITHMS

An important property of meta-learning algorithms is their expressive power, namely the ability to represent a large number of learning algorithms. More expressive power means that the method can represent more sophisticated learning procedures, which is relevant for scalability. For example, we may want to recover a learning procedure that can learn a new concept when only provided with positive examples of that concept (e.g. see Section 8.2). Or, we may want to recover a reinforcement learning procedure that learns safely and avoid risky states. To recover such learning procedures through meta-learning, we need a sufficiently expressive meta-learning algorithm. Note that measures of expressive power only indicate the ability to represent different functions, which is a prerequisite — but not a guarantee — for the ability to actually recover or learn those functions. To study this property, we need a way to measure such expressive power – we need to formally define a learning procedure and measure the size of the set of learning procedures that can be encoded by a particular algorithm.

One intuitive way to define a learning procedure would be a something that takes as input a dataset $\mathcal{D}$ and outputs a vector of parameters that are used to make predictions about new datapoints. While it is natural to think of learning procedures as such, this definition has a number of downsides. First, it only allows for learning parametric models, while a number of meta-learning algorithms have taken more non-parametric approaches (Vinyals et al., 2016; Snell et al., 2017). Second, it is overcomplete. There is often more than one parameter vectors that can lead to the same underlying function; for example, in a ReLU neural network, a weight matrix of all zeros with any non-positive bias value encodes the same function. We can more specifically define the function that is learned by considering the input/output pairs of that function. We thus choose to characterize a learning algorithm as something that takes as input both a dataset $\mathcal{D}$ and a test observation $\mathbf{x}^\star$ and outputs a prediction $\mathbf{y}^\star$, as defined below.

**Definition 3.1.1.** A *learning algorithm* is a procedure or function that processes data in $\mathcal{D}$ to make predictions $\hat{\mathbf{y}}^\star$ from new inputs $\mathbf{x}^\star$.

This definition encapsulates the notion of learning a particular function that maps from $\mathbf{x}^\star$ to $\hat{\mathbf{y}}^\star$ from data $\mathcal{D}$. Note that this definition is general to any learning problem where your goal is to recover a function from data, including learning classifiers, regressors, and policies.

Now that we have defined a learning procedure, we would like to measure the set of learning procedures that a particular meta-learning algorithm can represent. If we view a learning procedure as a function defined above, $\hat{\mathbf{y}}^\star = f(\mathcal{D}, \mathbf{x}^\star)$, we can develop a simple binary indication for maximal expressive power or not by building upon the notion of a universal function approximator. Concretely, we will define a *universal learning procedure approximator* as a universal function approximator for the function mapping from $\mathcal{D}$ and $\mathbf{x}^\star$ to $\hat{\mathbf{y}}^\star$. If a meta-learning algorithm can represent such a learning procedure, then it has maximal expressive power. While this measure is simple, it's binary nature is a certainly a limitation as it does not allow us to characterize the degree to which a meta-learning algorithm approaches universality. A more continuous measure would be desirable in cases where, for example, we want to study expressive power under constraints like the size of the learner model. We leave the possibility of developing such as measure as a question for future work. In the rest of this section, we will study the expressive power of a few previous black-box meta-learning approaches, focusing on the problem of meta-supervised learning.

We can broadly classify black-box meta-learning methods into two categories. In the first approach (Santoro et al., 2016; Duan et al., 2016b; Wang and Hebert, 2016; Mishra et al., 2018), there is a meta-learner model $g$ with parameters $\phi$ which takes as input the

dataset $\mathcal{D}_{\mathcal{T}}$ for a particular task $\mathcal{T}$ and a new test input $\mathbf{x}^\star$, and outputs the estimated output $\hat{\mathbf{y}}^\star$ for that input:

$$\hat{\mathbf{y}}^\star = g(\mathcal{D}_{\mathcal{T}}, \mathbf{x}^\star; \phi) = g((\mathbf{x}, \mathbf{y})_1, ..., (\mathbf{x}, \mathbf{y})_K, \mathbf{x}^\star; \phi)$$

The meta-learner $g$ is typically a recurrent model that iterates over the dataset $\mathcal{D}$ and the new input $\mathbf{x}^\star$. For a recurrent neural network model that satisfies the UFA theorem, this approach is maximally expressive, as it can represent any function on the dataset $\mathcal{D}_{\mathcal{T}}$ and test input $\mathbf{x}^\star$.

In the second approach (Hochreiter et al., 2001; S. Bengio et al., 1992; K. Li and Malik, 2017b; Andrychowicz et al., 2016; Ravi and Larochelle, 2017; Ha et al., 2017), there is a meta-learner $g$ that takes as input the dataset for a particular task $\mathcal{D}_{\mathcal{T}}$ and the current weights $\theta$ of a learner model $f$, and outputs new parameters $\phi_{\mathcal{T}}$ for the learner model. Then, the test input $\mathbf{x}^\star$ is fed into the learner model to produce the predicted output $\hat{\mathbf{y}}^\star$. The process can be written as follows:

$$\hat{\mathbf{y}}^\star = f(\mathbf{x}^\star; \phi_{\mathcal{T}}) = f(\mathbf{x}^\star; g(\mathcal{D}_{\mathcal{T}}; \phi)) = f(\mathbf{x}^\star; g((\mathbf{x}, \mathbf{y})_{1:K}; \phi))$$

Note that, in the form written above, this approach can be as expressive as the previous approach, since the meta-learner could simply copy the dataset into some of the predicted weights, reducing to a model that takes as input the dataset and the test example.[1] Several versions of this approach, i.e. Ravi and Larochelle (2017) and K. Li and Malik (2017b), have the recurrent meta-learner operate on order-invariant features such as the gradient and objective value averaged over the datapoints in the dataset, rather than operating on the individual datapoints themselves. This induces a potentially helpful inductive bias that disallows coupling between datapoints, ignoring the ordering within the dataset. As a result, the meta-learning process can only produce permutation-invariant functions of the dataset.

We will come back to this property in Section 4.4, when we discuss the expressive power of model-agnostic meta-learning algorithms.

### 3.2 CONSISTENT LEARNING ALGORITHMS

Beyond expressive power and scalability, another important aspect of meta-learning is the ability to recover reasonable solutions even when faced with data from tasks that

1 For this to be possible, the model $f$ must be a neural network with at least two hidden layers, since the dataset can be copied into the first layer of weights and the predicted output must be a universal function approximator of both the dataset and the test input.

are slightly outside of the distribution of training tasks. This ability to generalize well is critical for real world applications where the number of training tasks may not adequately cover the full distribution of training tasks, and lifelong learning settings where the distribution of tasks is continuously shifting. Unfortunately, this notion of generalization is extremely difficult to measure, particularly for non-convex learning problems with limited data. We will relax our assumptions and define a basic property that is more straight-forward to measure, but still relates to finding good solutions to out-of-distribution tasks. In particular, we consider a learning algorithm to be *consistent* if it finds the true function when provided infinite data:

**Definition 3.2.1.** A learning algorithm $f$ is *consistent* if it satisfies the following property:

$$\lim_{|\mathcal{D}|\to\infty} f(\mathcal{D}, \mathbf{x}_i^\star) \to \mathbf{y}_i^\star \; \forall(\mathbf{x}_i^\star, \mathbf{y}_i^\star)$$

This property can only be achieved by meta-learning algorithms for which the true function from  to $\mathbf{y}^\star$ can actually be represented, which is, e.g., satisfied by universal meta-learners[2]. For convex supervised learning problems with separable data, hand-designed learning algorithms such as gradient descent are consistent, modulo numerical issues. While we cannot prove that gradient descent algorithms are consistent in the general case, there is significant empirical evidence that, given enough data and sufficiently large neural networks, stochastic gradient descent can find solutions with good training error (C. Zhang et al., 2017). Therefore, it is not unreasonable to ask that our learned learning procedures are also consistent insofar as gradient descent is consistent.

This property has interesting implications. It means that even if a meta-test task is completely distinct from the distribution tasks in the meta-training set, the learned learning algorithm will do well on the task given enough data: the learned learning procedure can *overcome the prior* if needed. Theoretically, 'enough' data might be infinite data. But, in practice, it is reasonable to expect that consistent learning algorithms will not do substantially worse than learning from scratch in cases where the prior is not completely incorrect, and likely still better than learning from scratch when the prior at least points in the right direction. Hence, we would expect consistent learning algorithms to generalize reasonably well when the tasks are outside of but near the meta-training task distribution. Whereas, with learning procedures that are not consistent, e.g. black-box learning algorithms, one can only hope for good extrapolation performance, even when provided extremely an large dataset on the extrapolated task.

2 It is likely possible to define a notion of consistency for learning algorithms with non-universal function approximators by, for example, considering local optimum, but we leave this for future work.

### 3.3 HANDLING AMBIGUITY IN LEARNING

The final property we consider relates to ambiguity. Even when utilizing prior experience when learning from a few datapoints, there might not be simply enough information in the examples for a new task to resolve the task or concept with high certainty. Therefore, it is desirable to develop few-shot meta-learning methods that can propose multiple potential solutions to an ambiguous few-shot learning problem. Such a method could be used to evaluate uncertainty (by measuring agreement between the samples), perform active learning, or elicit direct human supervision about which sample is preferable. For example, in safety-critical applications, such as few-shot medical image classification, uncertainty is crucial for determining if the learned classifier should be trusted. When learning from such small amounts of data, uncertainty estimation can also help predict if additional data would be beneficial for learning and improving the estimate of the rewards. Finally, while we do not experiment with this in this work, we expect that modeling this ambiguity will be helpful for reinforcement learning problems, where it can be used to aid in exploration.

To model ambiguity, systems should be capable of sampling different possible functions underlying the data. With this ability, the learning procedure can generate different hypotheses about the underlying function, which can be used to better seek out new data to reduce uncertainty. Interestingly, standard learning procedures like gradient descent, linear regression, and nearest neighbors do not satisfy this property, nor does a recurrent neural network, as each of these learning algorithms are deterministic. Instead, we need to build probabilistic meta-learning algorithms that can reason, in some way, over the distribution of functions. Representing distributions over functions is relatively straightforward when using simple function approximators, such as linear functions, and has been done extensively in early few-shot learning approaches using Bayesian models (Tenenbaum, 1999; Fei-Fei et al., 2003). But this problem becomes substantially more challenging when reasoning over high-dimensional function approximators such as deep neural networks, since explicitly representing expressive distributions over thousands or millions of parameters if often intractable. In Chapter 5, we will discuss how we can overcome this challenge to develop meta-learning algorithms that are both scalable and probabilistic.

# Part II

# META-LEARNING WITH GRADIENTS

# 4

## A MODEL-AGNOSTIC META-LEARNING ALGORITHM

In this chapter, we propose a meta-learning algorithm that is general and model-agnostic, in the sense that it can be directly applied to any learning problem and model that is trained with a gradient descent procedure. Our focus is on deep neural network models, but we illustrate how our approach can easily handle different architectures and different problem settings, including classification, regression, and policy gradient reinforcement learning, with minimal modification. The key idea underlying our method is to train the model's initial parameters such that the model has maximal performance on a new task after the parameters have been updated through one or more gradient steps computed with a small amount of data from that new task. Unlike prior meta-learning methods that learn an update function or learning rule (Schmidhuber, 1987; S. Bengio et al., 1992; Andrychowicz et al., 2016; Ravi and Larochelle, 2017), our algorithm does not expand the number of learned parameters nor place constraints on the model architecture (e.g. by requiring a recurrent model (Santoro et al., 2016) or a Siamese network (Koch et al., 2015)), and it can be readily combined with fully connected, convolutional, or recurrent neural networks. It can also be used with a variety of loss functions, including differentiable supervised losses and non-differentiable reinforcement learning objectives.

The process of training a model's parameters such that a few gradient steps, or even a single gradient step, can produce good results on a new task can be viewed from a feature learning standpoint as building an internal representation that is broadly suitable for many tasks. If the internal representation is suitable to many tasks, simply fine-tuning the parameters slightly (e.g. by primarily modifying the top layer weights in a feedforward model) can produce good results. In effect, our procedure optimizes for models that are easy and fast to fine-tune, allowing the adaptation to happen in the right space for fast learning. From a dynamical systems standpoint, our learning process can be viewed as maximizing the sensitivity of the loss functions of new tasks with respect to the parameters: when the sensitivity is high, small local changes to the parameters

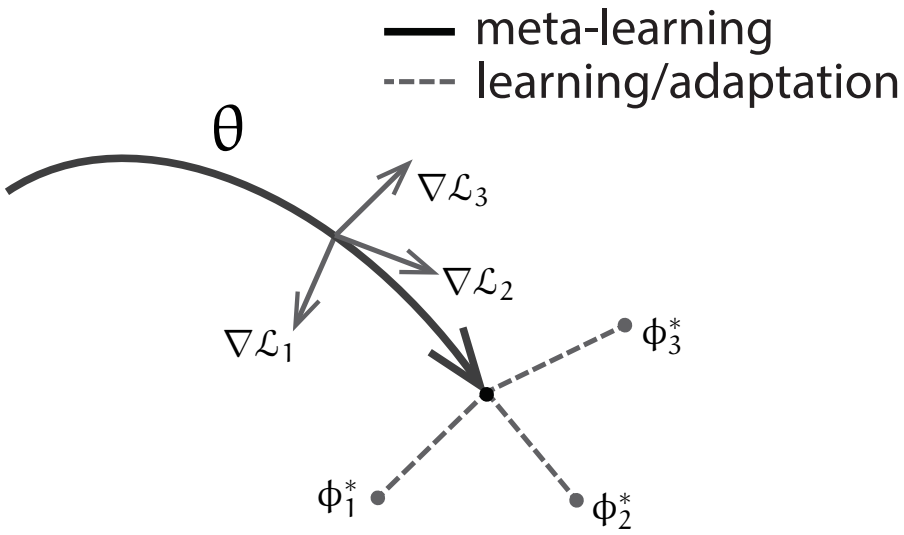


Figure 1: Illustrative diagram of our model-agnostic meta-learning algorithm (MAML), which optimizes for a representation $\theta$ that can quickly adapt to new tasks.

can lead to large improvements in the task loss.

The primary contribution in this chapter is a simple model- and task-agnostic algorithm for meta-learning that trains a model's parameters such that a small number of gradient updates will lead to fast learning on a new task. We demonstrate the algorithm on different model types, including fully connected and convolutional networks, and in several distinct domains, including few-shot regression, image classification, and reinforcement learning. Our evaluation shows that our meta-learning algorithm compares favorably to state-of-the-art one-shot learning methods designed specifically for supervised classification, while using fewer parameters, but that it can also be readily applied to regression and can accelerate reinforcement learning in the presence of task variability, substantially outperforming direct pretraining as initialization.

### 4.1 GENERAL ALGORITHM

In contrast to prior work, which has sought to train recurrent neural networks that ingest entire datasets (Santoro et al., 2016; Duan et al., 2016b) or feature embeddings that can be combined with nonparametric methods at test time (Vinyals et al., 2016; Koch et al., 2015), we propose a method that can learn the parameters of any standard model via meta-

learning in such a way as to prepare that model for fast adaptation. The intuition behind this approach is that some internal representations are more transferrable than others. For example, a neural network might learn internal features that are broadly applicable to all tasks in $p(\mathcal{T})$, rather than a single individual task. How can we encourage the emergence of such general-purpose representations? We take an explicit approach to this problem: since the model will be fine-tuned using a gradient-based learning rule on a new task, we will aim to learn a model in such a way that this gradient-based learning rule can make rapid progress on new tasks drawn from $p(\mathcal{T})$, without overfitting. In effect, we will aim to find model parameters that are *sensitive* to changes in the task, such that small changes in the parameters will produce large improvements on the loss function of any task drawn from $p(\mathcal{T})$, when altered in the direction of the gradient of that loss (see Figure 1). We make no assumption on the form of the model, other than to assume that it is parametrized by some parameter vector $\theta$, and that the loss function is smooth enough in $\theta$ that we can use gradient-based learning techniques.

Formally, we consider a model represented by a parametrized function $f_\theta$ with parameters $\theta$. When adapting to a new task $\mathcal{T}_i$, the model's parameters $\theta$ become $\phi_i$. In our method, the updated parameter vector $\phi_i$ is computed using one or more gradient descent updates on the training data for task $\mathcal{T}_i$. For example, when using one gradient update,

$$\phi_i = \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}^{\text{tr}}_{\mathcal{T}_i}).$$

The step size $\alpha$ may be fixed as a hyperparameter or meta-learned. For simplicity of notation, we will consider one gradient update for the rest of this section, but using multiple gradient updates is a straightforward extension.

The model parameters are trained by optimizing for the performance of $f_{\phi_i}$ with respect to $\theta$ across tasks sampled from $p(\mathcal{T})$. More concretely, the meta-objective is as follows:

$$\min_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}(\phi_i, \mathcal{D}^{\text{test}}_{\mathcal{T}_i}) = \min_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}(\theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}^{\text{tr}}_{\mathcal{T}_i}), \mathcal{D}^{\text{test}}_{\mathcal{T}_i}) \tag{1}$$

Note that the meta-optimization is performed over the model parameters $\theta$, whereas the objective is computed using the updated model parameters $\phi$. In effect, our proposed method aims to optimize the model parameters such that one or a small number of gradient steps on a new task will produce maximally effective behavior on that task.

The meta-optimization across tasks is performed via stochastic gradient descent (SGD),

**Algorithm 1** Model-Agnostic Meta-Learning

**Require:** $p(\mathcal{T})$: distribution over tasks
**Require:** $\alpha$, $\beta$: step size hyperparameters
1: randomly initialize $\theta$
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for** each $\mathcal{T}_i$ **do**
5: Sample $\mathcal{D}^{tr}_{\mathcal{T}_i} \sim \mathcal{D}_{\mathcal{T}_i}$
6: Sample $\mathcal{D}^{test}_{\mathcal{T}_i} \sim \mathcal{D}_{\mathcal{T}_i}$ $\mathcal{D}^{tr}_{\mathcal{T}_i}$
7: Evaluate $\nabla_\theta \mathcal{L}(\theta, \mathcal{D}^{tr}_{\mathcal{T}_i})$ with respect to K examples
8: Compute adapted parameters with gradient descent: $\phi_i = \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}^{tr}_{\mathcal{T}_i})$
9: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}(\phi_i, \mathcal{D}^{test}_{\mathcal{T}_i})$

such that the model parameters $\theta$ are updated as follows:

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}(\phi_i, \mathcal{D}^{test}_{\mathcal{T}_i}) \tag{2}$$

where $\beta$ is the meta step size. The full algorithm, in the general case, is outlined in Algorithm 1.

The gradient of the MAML objective update involves a gradient through a gradient. Computationally, this requires an additional backward pass through f to compute Hessian-vector products, which is supported by standard deep learning libraries such as TensorFlow (Abadi et al., 2016). We discuss this further in Section 4.3. In our experiments, we also include a comparison to dropping this backward pass and using a first-order approximation, which we discuss in Section 4.6.2.

### 4.2 SPECIES OF MAML

In this section, we discuss specific instantiations of our meta-learning algorithm for supervised learning and reinforcement learning. The domains differ in the form of loss function and in how data is generated by the task and presented to the model, but the same basic adaptation mechanism can be applied in both cases.

### 4.2.1 *Supervised Regression and Classification*

Few-shot learning is well-studied in the domain of supervised tasks, where the goal is to learn a new function from only a few input/output pairs for that task, using prior data from similar tasks for meta-learning. For example, the goal might be to classify images of a Segway after seeing only one or a few examples of a Segway, with a model that has previously seen many other types of objects. Likewise, in few-shot regression, the goal is to predict the outputs of a continuous-valued function from only a few datapoints sampled from that function, after training on many functions with similar statistical properties.

To formalize the supervised regression and classification problems in the context of the meta-learning definitions in Section 2.1, we can define the horizon $H = 1$ and drop the timestep subscript on $\mathbf{x}_t$, since the model accepts a single input and produces a single output, rather than a sequence of inputs and outputs. The task $\mathcal{T}_i$ generates $K$ i.i.d. observations $\mathbf{x}$ from $\rho_i$, and the task loss is represented by the error between the model's output for $\mathbf{x}$ and the corresponding target values $\mathbf{y}$ for that observation and task.

Two common loss functions used for supervised classification and regression are cross-entropy and mean-squared error (MSE), which we will describe below; though, other supervised loss functions may be used as well. For regression tasks using mean-squared error, the loss takes the form:

$$\mathcal{L}(\psi, \mathcal{D}_{\mathcal{T}_i}) = \sum_{\mathbf{x}^{(j)}, \mathbf{y}^{(j)} \sim \mathcal{D}_{\mathcal{T}_i}} \| f_\psi(\mathbf{x}^{(j)}) - \mathbf{y}^{(j)} \|_2^2, \tag{3}$$

where $\mathbf{x}^{(j)}, \mathbf{y}^{(j)}$ are an input/output pair sampled from task $\mathcal{T}_i$. In $K$-shot regression tasks, $K$ input/output pairs are provided for learning for each task.

Similarly, for discrete classification tasks with a cross-entropy loss, the loss takes the form:

$$\begin{aligned} \mathcal{L}(\psi, \mathcal{D}_{\mathcal{T}_i}) = \sum_{\mathbf{x}^{(j)}, \mathbf{y}^{(j)} \sim \mathcal{D}_{\mathcal{T}_i}} & \mathbf{y}^{(j)} \log f_\psi(\mathbf{x}^{(j)}) \\ & + (1 - \mathbf{y}^{(j)}) \log(1 - f_\psi(\mathbf{x}^{(j)})) \end{aligned} \tag{4}$$

According to the conventional terminology, $K$-shot classification tasks use $K$ input/output pairs from each class, for a total of $NK$ data points for $N$-way classification. Given a distribution over tasks $p(\mathcal{T}_i)$, these loss functions can be directly inserted into the equations in Section 4.1 to perform meta-learning, as detailed in Algorithm 2.

**Algorithm 2** MAML for Few-Shot Supervised Learning

**Require:** $p(\mathcal{T})$: distribution over tasks
**Require:** $\alpha$, $\beta$: step size hyperparameters
1: randomly initialize $\theta$
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for** each $\mathcal{T}_i$ **do**
5: Sample K datapoints $\mathcal{D}^{tr}_{\mathcal{T}_i} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{D}_{\mathcal{T}_i}$
6: Evaluate $\nabla_\theta \mathcal{L}(\theta, \mathcal{D}^{tr}_{\mathcal{T}_i})$ using $\mathcal{L}$ in Equation (3) or (4)
7: Compute adapted parameters with gradient descent: $\phi_i = \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}^{tr}_{\mathcal{T}_i})$
8: Sample datapoints $\mathcal{D}^{test}_{\mathcal{T}_i} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{D}_{\mathcal{T}_i}$ $\mathcal{D}^{tr}_{\mathcal{T}_i}$ for the meta-update
9: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}(\phi_i, \mathcal{D}^{test}_{\mathcal{T}_i})$ using each $\mathcal{D}^{test}_{\mathcal{T}_i}$ and $\mathcal{L}$ in Equation 3 or 4

#### 4.2.2 *Reinforcement Learning*

In reinforcement learning (RL), the goal of few-shot meta-learning is to enable an agent to quickly acquire a policy for a new test task using only a small amount of experience in the test setting. A new task might involve achieving a new goal or succeeding on a previously trained goal in a new environment. For example, an agent might learn to quickly figure out how to navigate mazes so that, when faced with a new maze, it can determine how to reliably reach the exit with only a few samples. In this section, we will discuss how MAML can be applied to meta-learning for RL, learning to learn a policy $f$ that maps from states $\mathbf{s}$ to actions $\mathbf{a}$.

Each RL task $\mathcal{T}_i$ contains an initial state distribution $\rho_i(\mathbf{s}_1)$ and a transition distribution $\rho_i(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$, and the loss $\mathcal{L}$ corresponds to the (negative) reward function $R$. The entire task is therefore a Markov decision process (MDP) with horizon $H$, where the learner is allowed to query a limited number of sample trajectories for few-shot learning. Any aspect of the MDP may change across tasks in $p(\mathcal{T})$. The model being learned, $f_\theta$, is a policy that maps from states $\mathbf{s}_t$ to a distribution over actions $\mathbf{a}_t$ at each timestep $t \in \{1, ..., H\}$. The loss for task $\mathcal{T}_i$ and model $f_\psi$ takes the form

$$\mathcal{L}(\psi, \mathcal{D}_{\mathcal{T}_i}) = -\mathbb{E}_{\mathbf{s}_t, \mathbf{a}_t \sim f_\psi, \rho_{\mathcal{T}_i}} \left[ \sum_{t=1}^{H} R_i(\mathbf{s}_t, \mathbf{a}_t) \right]. \tag{5}$$

In K-shot reinforcement learning, K rollouts from $f_\theta$ and task $\mathcal{T}_i$, $(\mathbf{s}_1, \mathbf{a}_1, ...\mathbf{s}_H)$, and the

**Algorithm 3** MAML for Reinforcement Learning

**Require:** $p(\mathcal{T})$: distribution over tasks
**Require:** $\alpha$, $\beta$: step size hyperparameters
1: randomly initialize $\theta$
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for** each $\mathcal{T}_i$ **do**
5: Sample K trajectories $\mathcal{D}^{tr}_{\mathcal{T}_i} = \{(\mathbf{s}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ using $f_\theta$ in $\mathcal{T}_i$
6: Evaluate $\nabla_\theta \mathcal{L}(\theta, \mathcal{D}^{tr}_{\mathcal{T}_i})$ using $\mathcal{L}$ in Equation 5
7: Compute adapted parameters with gradient descent: $\phi_i = \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}^{tr}_{\mathcal{T}_i})$
8: Sample trajectories $\mathcal{D}^{test}_{\mathcal{T}_i} = \{(\mathbf{x}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ using $f_{\phi_i}$ in $\mathcal{T}_i$
9: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}(\phi_i, \mathcal{D}^{test}_{\mathcal{T}_i})$ using $\mathcal{L}$ in Equation 5

corresponding rewards $R(\mathbf{s}_t, \mathbf{a}_t)$, may be used for adaptation on a new task $\mathcal{T}_i$. Since the expected reward is generally not differentiable due to unknown dynamics, we use policy gradient methods to estimate the gradient both for the model gradient update(s) and the meta-optimization. Since policy gradients are an on-policy algorithm, each additional gradient step during the adaptation of $f_\theta$ requires new samples from the current policy $f_{\phi_i}$. We detail the algorithm in Algorithm 3. This algorithm has the same structure as Algorithm 2, with the principal difference being that steps 5 and 8 require sampling trajectories from the environment corresponding to task $\mathcal{T}_i$. Practical implementations of this method may also use a variety of improvements recently proposed for policy gradient algorithms, including state or action-dependent baselines and trust regions (Schulman et al., 2015).

### 4.3 Implementation and First-Order Approximation

Performing the MAML optimization in Equation 1 via gradient descent involves computing second-derivatives, since the outer optimization is over $\theta$ and the objective includes a gradient with respect to $\theta$. If more than one gradient step is computed in the inner optimization, the optimization does not involve higher-order derivatives because the additional gradient steps are not computed with respect to $\theta$, but with respect to the updated parameters. Computing the gradients for the MAML objective in meta-supervised learning problems is straightforward when using standard deep learning libraries such as TensorFlow (Abadi et al., 2016) and PyTorch (Paszke et al., 2017), since these libraries

can automatically differentiate through the inner gradient through an extra backward pass. However, it is trickier to implement the gradient of the meta-RL objective: the inner gradient estimate involves an expectation over samples from $f_\theta$ and auto-differentiation methods are ignorant of the dependency of the samples on $\theta$. While it is possible to compute a correction term for the MAML gradient for simple policy gradient estimators such as REINFORCE (Williams, 1992), it is much more difficult to compute for optimizers such as trust-region policy optimization (TRPO) (Schulman et al., 2015), which we use in our experiments. We find that, for the illustrative examples in our experiments, it is okay to ignore this dependency. However, we expect it to be much more important in settings where the data distribution under the initial policy is crucial for good performance, namely task distributions where nontrivial exploration strategies are important for good performance.

Now, one might ask – do we need the second order information at all? Prior work (Ravi and Larochelle, 2017) has found that second-order information with LSTM-based meta-optimizers can achieve good results with these second derivatives omitted. In our experiments, we find that the additional backward pass contributes to around 33% of the computation, and this expense will increase substantially when using more gradient steps within the MAML objective. If we can avoid differentiating *through learning*, then we can achieve better scaling to long inner optimizations involving many gradient steps. We develop a first-order approximation to the MAML algorithm that simply involves stopping the gradient from back-propagating through the inner gradient:

$$\min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}\left(\theta - \alpha \, \text{sg}\left(\nabla_\theta \mathcal{L}(\theta, \mathcal{D}^{\text{tr}}_{\mathcal{T}_i})\right), \mathcal{D}^{\text{test}}_{\mathcal{T}_i}\right)$$

where we use sg to denote a stop gradient operation. Note that this approximation still computes the meta-gradient at the post-update parameter values $\phi_i$, which provides for effective meta-learning. This approximation effectively treats the parameter update as a constant. To implement this approximation, one simply needs to run gradient descent on a sampled task starting from $\theta$ to get to $\phi = \theta + c$, and then backpropagate the test task performance evaluated at $\theta + c$, treating $c$ as a constant that does not depend on $\theta$.

As we discuss in our experiments, we surprisingly find that this first-order approximation works well on few-shot image recognition benchmarks, achieving similar performance to including full second-order information. However, in more complex domains, such as few-shot imitation learning (see Chapter 7), we found the first-order approximation to not work at all.

Because MAML is simply running gradient descent at test time starting from a meta-learning initialization, MAML is guaranteed to acquire *consistent* learning algorithms with enough gradient descent steps, insofar as gradient descent is consistent[1]. But does this consistency come at a cost? A natural question that arises is whether MAML loses representational power by incorporating the structure of gradient descent. Intuitively, we might surmise that learning an update rule is more expressive than simply learning an initialization for gradient descent. In this section, we seek to answer the following question: does simply learning the initial parameters of a deep neural network have the same representational power as arbitrarily expressive meta-learners that directly ingest the training data at meta-test time? Or, more concisely, does representation combined with standard gradient descent have sufficient capacity to constitute any learning algorithm?

We analyze this question from the standpoint of universal learning procedure approximators, defined in Section 3.1, focusing on the setting of meta-supervised learning (as opposed to meta-RL). Recall that we previously defined a universal learning procedure approximator to be a learner that can approximate any function of the set of training datapoints $\mathcal{D}_{\mathcal{T}}$ and the test point $\mathbf{x}^\star$. Unlike black-box meta-learning algorithms, it is not obvious whether or not the MAML update imposes any constraints on the learning procedures that can be acquired. In this section, we will find that, for a sufficiently deep learner model, MAML has the same theoretical representational power as recurrent meta-learners. We therefore conclude that, when using deep, expressive function approximators, there is no theoretical disadvantage in terms of representational power to using MAML over a black-box meta-learner represented, for example, by a recurrent network.

Formally, in model-agnostic meta-learning (MAML), standard gradient descent is used to update the weights of the learner $f$. Following the notation introduced in Section 3.1, the prediction $\hat{\mathbf{y}}^\star$ for a test input $\mathbf{x}^\star$ is:

$$\begin{aligned}\hat{\mathbf{y}}^\star &= f_{\text{MAML}}(\mathcal{D}_{\mathcal{T}}, \mathbf{x}^\star; \theta) \\ &= f(\mathbf{x}^\star; \phi_{\mathcal{T}}) = f(\mathbf{x}^\star; \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_{\mathcal{T}})) = f\left(\mathbf{x}^\star; \theta - \alpha \nabla_\theta \frac{1}{K} \sum_{k=1}^{K} \ell(\mathbf{y}_k, f(\mathbf{x}_k; \theta))\right),\end{aligned}$$

where $\theta$ denotes the initial parameters of the model $f$ and also corresponds to the parameters that are meta-learned, and $\ell$ corresponds to a loss function with respect to the

1 Recall Section 3.2 for a discussion of consistency.

label and prediction. It is clear how $f_{\text{MAML}}$ can approximate any function on $\mathbf{x}^\star$, as per the universal function approximation (UFA) theorem; however, it is not obvious if $f_{\text{MAML}}$ can represent any function of the set of input, output pairs in $\mathcal{D}_{\mathcal{T}}$, since the UFA theorem does not consider the gradient operator.

The first goal of this section is to show that $f_{\text{MAML}}(\mathcal{D}_{\mathcal{T}}, \mathbf{x}^\star; \theta)$ is a universal function approximator of $(\mathcal{D}_{\mathcal{T}}, \mathbf{x}^\star)$ in the one-shot setting, where the dataset $\mathcal{D}_{\mathcal{T}}$ consists of a single datapoint $(\mathbf{x}, \mathbf{y})$. Then, we will consider the case of K-shot learning, showing that $f_{\text{MAML}}(\mathcal{D}_{\mathcal{T}}, \mathbf{x}^\star; \theta)$ is universal in the set of functions that are invariant to the permutation of datapoints. In both cases, we will discuss meta supervised learning problems with both discrete and continuous labels and the loss functions under which universality does or does not hold.

### 4.4.1 *Universality of the One-Shot Gradient-Based Learner*

We first introduce a proof of the universality of gradient-based meta-learning for the special case with only one training point, corresponding to one-shot learning. We denote the training datapoint as $(\mathbf{x}, \mathbf{y})$, and the test input as $\mathbf{x}^\star$. A universal learning algorithm approximator corresponds to the ability of a meta-learner to represent any function $f_{\text{target}}(\mathbf{x}, \mathbf{y}, \mathbf{x}^\star)$ up to arbitrary precision.

We will proceed by construction, showing that there exists a neural network function $\hat{f}(\cdot; \theta)$ such that $\hat{f}(\mathbf{x}^\star; \phi)$ approximates $f_{\text{target}}(\mathbf{x}, \mathbf{y}, \mathbf{x}^\star)$ up to arbitrary precision, where $\phi = \theta - \alpha \nabla_\theta \ell(\mathbf{y}, f(\mathbf{x}))$ and $\alpha$ is the non-zero learning rate. The proof holds for a standard multi-layer ReLU network, provided that it has sufficient depth. As we discuss in Section 4.4.3, the loss function $\ell$ cannot be any loss function, but the standard cross-entropy and mean-squared error objectives are both suitable. In this proof, we will start by presenting the form of $\hat{f}$ and deriving its value after one gradient step. Then, to show universality, we will construct a setting of the weight matrices that enables independent control of the information flow coming forward from $\mathbf{x}$ and $\mathbf{x}^\star$, and backward from $\mathbf{y}$.

We will start by constructing $\hat{f}$, which, as shown in Figure 2 is a generic deep network with $N + 2$ layers and ReLU nonlinearities. Note that, for a particular weight matrix $W_i$ at layer $i$, a single gradient step $W_i - \alpha \nabla_{W_i} \ell$ can only represent a rank-1 update to the matrix $W_i$. That is because the gradient of $W_i$ is the outer product of two vectors, $\nabla_{W_i} \ell = \mathbf{a}_i \mathbf{b}_{i-1}^\intercal$, where $\mathbf{a}_i$ is the error gradient with respect to the pre-synaptic activations at layer $i$, and $\mathbf{b}_{i-1}$ is the forward post-synaptic activations at layer $i - 1$. The expressive power of a single gradient update to a single weight matrix is therefore quite limited. However, if we sequence $N$ weight matrices as $\prod_{i=1}^{N} W_i$, corresponding to multiple linear

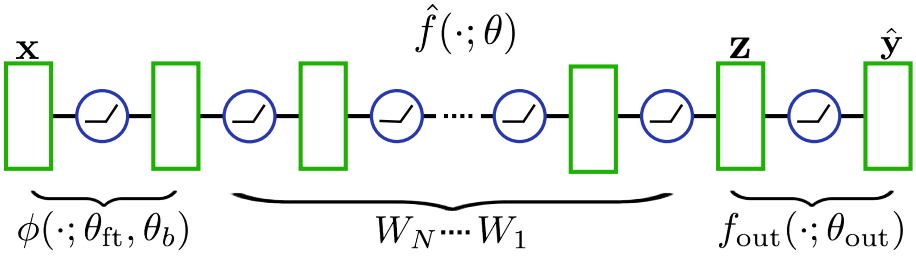


Figure 2: A deep fully-connected neural network with N+2 layers and ReLU nonlinearities. With this generic fully connected network, we prove that, with a single step of gradient descent, the model can approximate any function of the dataset and test input.

layers, it is possible to acquire a rank-N update to the linear function represented by $W = \prod_{i=1}^{N} W_i$. Note that deep ReLU networks act like deep linear networks when the input and pre-synaptic activations are non-negative. Motivated by this reasoning, we will construct $\hat{f}(\cdot;\theta)$ as a deep ReLU network where a number of the intermediate layers act as linear layers, which we ensure by showing that the input and pre-synaptic activations of these layers are non-negative. This allows us to simplify the analysis. The simplified form of the model is as follows:

$$\hat{f}(\cdot;\theta) = f_{\text{out}}\left(\left(\prod_{i=1}^{N} W_i\right)\phi(\cdot;\theta_{\text{ft}},\theta_b);\theta_{\text{out}}\right),$$

where $\phi(\cdot;\theta_{\text{ft}},\theta_b)$ represents an input feature extractor with parameters $\theta_{\text{ft}}$ and a scalar bias transformation variable $\theta_b$, $\prod_{i=1}^{N} W_i$ is a product of square linear weight matrices, $f_{\text{out}}(\cdot,\theta_{\text{out}})$ is a function at the output, and $\theta := \{\theta_{\text{ft}},\theta_b,\{W_i\},\theta_{\text{out}}\}$ are the learned parameters. The input feature extractor and output function can be represented with fully connected neural networks with one or more hidden layers, which we know are universal function approximators, while $\prod_{i=1}^{N} W_i$ corresponds to a set of linear layers with non-negative input and activations.

Next, we derive the form of the model's prediction after one gradient update, $\hat{f}(\mathbf{x}^\star;\phi)$. Let $\mathbf{z} = \left(\prod_{i=1}^{N} W_i\right)\phi(\mathbf{x};\theta_{\text{ft}},\theta_b)$, and the error gradient $\nabla_{\mathbf{z}}\ell = e(\mathbf{x},\mathbf{y})$. Then, the gradient with respect to each weight matrix $W_i$ is:

$$\nabla_{W_i}\ell(\mathbf{y},\hat{f}(\mathbf{x},\theta)) = \left(\prod_{j=1}^{i-1} W_j\right)^{T} e(\mathbf{x},\mathbf{y})\phi(\mathbf{x};\theta_{\text{ft}},\theta_b)^{T}\left(\prod_{j=i+1}^{N} W_j\right)^{T}.$$

Therefore, the post-update value of $\prod_{i=1}^{N} W_i' = \prod_{i=1}^{N}(W_i - \alpha \nabla_{W_i} \ell)$ is given by

$$\prod_{i=1}^{N} W_i - \alpha \sum_{i=1}^{N} \left(\prod_{j=1}^{i-1} W_j\right) \left(\prod_{j=1}^{i-1} W_j\right)^T e(\mathbf{x}, \mathbf{y}) \phi(\mathbf{x}; \theta_{ft}, \theta_b)^T \left(\prod_{j=i+1}^{N} W_j\right)^T \left(\prod_{j=i+1}^{N} W_j\right) - O(\alpha^2),$$

where we will disregard the last term, assuming that $\alpha$ is comparatively small such that $\alpha^2$ and all higher order terms vanish. In general, these terms do not necessarily need to vanish, and likely would further improve the expressiveness of the gradient update, but we disregard them here for the sake of the simplicity of the derivation. Ignoring these terms, we now note that the post-update value of $\mathbf{z}^\star$ when $\mathbf{x}^\star$ is provided as input into $\hat{f}(\cdot; \phi)$ is given by

$$\begin{aligned} \mathbf{z}^\star = &\prod_{i=1}^{N} W_i \phi(\mathbf{x}^\star; \phi_{ft}, \phi_b) \\ &-\alpha \sum_{i=1}^{N} \left(\prod_{j=1}^{i-1} W_j\right) \left(\prod_{j=1}^{i-1} W_j\right)^T e(\mathbf{x}, \mathbf{y}) \phi(\mathbf{x}; \theta_{ft}, \theta_b)^T \left(\prod_{j=i+1}^{N} W_j\right)^T \left(\prod_{j=i+1}^{N} W_j\right) \phi(\mathbf{x}^\star; \phi_{ft}, \phi_b), \end{aligned} \tag{6}$$

and $\hat{f}(\mathbf{x}^\star; \phi) = f_{out}(\mathbf{z}^\star; \phi_{out})$.

Our goal is to show that that there exists a setting of $W_i$, $f_{out}$, and $\phi$ for which the above function, $\hat{f}(\mathbf{x}^\star, \phi)$, can approximate any function of $(\mathbf{x}, \mathbf{y}, \mathbf{x}^\star)$. To show universality, we will aim to independently control information flow from $\mathbf{x}$, from $\mathbf{y}$, and from $\mathbf{x}^\star$ by multiplexing forward information from $\mathbf{x}$ and backward information from $\mathbf{y}$. We will achieve this by decomposing $W_i$, $\phi$, and the error gradient into three parts, as follows:

$$W_i := \begin{bmatrix} \tilde{W}_i & 0 & 0 \\ 0 & \overline{W}_i & 0 \\ 0 & 0 & \check{w}_i \end{bmatrix} \quad \phi(\cdot; \theta_{ft}, \theta_b) := \begin{bmatrix} \tilde{\phi}(\cdot; \theta_{ft}, \theta_b) \\ \mathbf{o} \\ \theta_b \end{bmatrix} \quad \nabla_{\mathbf{z}} \ell(\mathbf{y}, \hat{f}(\mathbf{x}; \theta)) := \begin{bmatrix} \mathbf{o} \\ \overline{e}(\mathbf{y}) \\ \check{e}(\mathbf{y}) \end{bmatrix} \tag{7}$$

where the initial value of $\theta_b$ will be $0$. The top components all have equal numbers of rows, as do the middle components. As a result, we can see that $\mathbf{z}$ will likewise be made up of three components, which we will denote as $\tilde{\mathbf{z}}$, $\overline{\mathbf{z}}$, and $\check{z}$. Lastly, we construct the top component of the error gradient to be $\mathbf{o}$, whereas the middle and bottom components, $\overline{e}(\mathbf{y})$ and $\check{e}(\mathbf{y})$, can be set to be any linear (but not affine) function of $\mathbf{y}$. We will discuss how to achieve this gradient in the latter part of this section when we define $f_{out}$ and in Section 4.4.3.

In Appendix A.1.3, we show that we can choose a particular form of $\tilde{W}_i$, $\overline{W}_i$, and $\check{w}_i$ that will simplify the products of $W_j$ matrices in Equation 6, such that we get the following form for $\overline{\mathbf{z}}^\star$:

$$\overline{\mathbf{z}}^\star = -\alpha \sum_{i=1}^{N} A_i \overline{e}(\mathbf{y}) \tilde{\phi}(\mathbf{x}; \theta_{ft}, \theta_b)^\top B_i^\top B_i \tilde{\phi}(\mathbf{x}^\star; \theta_{ft}, \phi_b), \tag{8}$$

where $A_1 = I$, $B_N = I$, $A_i$ can be chosen to be any symmetric positive-definite matrix, and $B_i$ can be chosen to be any positive definite matrix. In Appendix A.4, we further show that these definitions of the weight matrices satisfy the condition that the activations are non-negative, meaning that the model $\hat{f}$ can be represented by a generic deep network with ReLU nonlinearities.

Finally, we need to define the function $f_{out}$ at the output. When the training input $\mathbf{x}$ is passed in, we need $f_{out}$ to propagate information about the label $\mathbf{y}$ as defined in Equation 7. And, when the test input $\mathbf{x}^\star$ is passed in, we need a different function defined only on $\overline{\mathbf{z}}^\star$. Thus, we will define $f_{out}$ as a neural network that approximates the following multiplexer function and its derivatives (as shown possible by Hornik et al., 1990):

$$f_{out}\left(\begin{bmatrix} \tilde{\mathbf{z}} \\ \overline{\mathbf{z}} \\ \check{z} \end{bmatrix}; \theta_{out}\right) = \mathbb{1}(\overline{\mathbf{z}} = \mathbf{0}) g_{pre}\left(\begin{bmatrix} \tilde{\mathbf{z}} \\ \overline{\mathbf{z}} \\ \check{z} \end{bmatrix}; \theta_g\right) + \mathbb{1}(\overline{\mathbf{z}} \neq \mathbf{0}) h_{post}(\overline{\mathbf{z}}; \theta_h), \tag{9}$$

where $g_{pre}$ is a linear function with parameters $\theta_g$ such that $\nabla_{\mathbf{z}} \ell = e(\mathbf{y})$ satisfies Equation 7 (see Section 4.4.3) and $h_{post}(\cdot; \theta_h)$ is a neural network with one or more hidden layers. As shown in Appendix A.1.4, the post-update value of $f_{out}$ is

$$f_{out}\left(\begin{bmatrix} \tilde{\mathbf{z}}^\star \\ \overline{\mathbf{z}}^\star \\ \check{z}^\star \end{bmatrix}; \phi_{out}\right) = h_{post}(\overline{\mathbf{z}}^\star; \theta_h). \tag{10}$$

Now, combining Equations 8 and 10, we can see that the post-update value is the following:

$$\hat{f}(\mathbf{x}^\star; \phi) = h_{post}\left(-\alpha \sum_{i=1}^{N} A_i \overline{e}(\mathbf{y}) \tilde{\phi}(\mathbf{x}; \theta_{ft}, \theta_b)^\top B_i^\top B_i \tilde{\phi}(\mathbf{x}^\star; \theta_{ft}, \phi_b); \theta_h\right) \tag{11}$$

In summary, so far, we have chosen a particular form of weight matrices, feature extractor, and output function to decouple forward and backward information flow and

recover the post-update function above. Now, our goal is to show that the above function $\hat{f}(\mathbf{x}^\star; \phi)$ is a universal learning algorithm approximator, as a function of $(\mathbf{x}, \mathbf{y}, \mathbf{x}^\star)$. For notational clarity, we will use $k_i(\mathbf{x}, \mathbf{x}^\star) := \tilde{\phi}(\mathbf{x}; \theta_{ft}, \theta_b)^T B_i^T B_i \tilde{\phi}(\mathbf{x}^\star; \theta_{ft}, \phi_b)$ to denote the inner product in the above equation, noting that it can be viewed as a type of kernel with the RKHS defined by $B_i\tilde{\phi}(\mathbf{x}; \theta_{ft}, \theta_b)$.[2] The connection to kernels is not in fact needed for the proof, but provides for convenient notation and an interesting observation. We then define the following lemma:

**Lemma 4.4.1.** *Let us assume that $\bar{e}(\mathbf{y})$ can be chosen to be any linear (but not affine) function of $\mathbf{y}$. Then, we can choose $\theta_{ft}$, $\theta_h$, $\{A_i; i > 1\}$, $\{B_i; i < N\}$ such that the function*

$$\hat{f}(\mathbf{x}^\star; \phi) = h_{post}\left(-\alpha \sum_{i=1}^{N} A_i \bar{e}(\mathbf{y}) k_i(\mathbf{x}, \mathbf{x}^\star); \theta_h\right) \tag{12}$$

*can approximate any continuous function of $(\mathbf{x}, \mathbf{y}, \mathbf{x}^\star)$ on compact subsets of $\mathbb{R}^{\dim(\mathbf{y})}$.*[3]

Intuitively, Equation 12 can be viewed as a sum of basis vectors $A_i\bar{e}(\mathbf{y})$ weighted by $k_i(\mathbf{x}, \mathbf{x}^\star)$, which is passed into $h_{post}$ to produce the output. There are likely a number of ways to prove Lemma 4.4.1. In Appendix A.1.1, we provide a simple though inefficient proof, which we will briefly summarize here. We can define $k_i$ to be a indicator function, indicating when $(\mathbf{x}, \mathbf{x}^\star)$ takes on a particular value indexed by $i$. Then, we can define $A_i\bar{e}(\mathbf{y})$ to be a vector containing the information of $\mathbf{y}$ and $i$. Then, the result of the summation will be a vector containing information about the label $\mathbf{y}$ and the value of $(\mathbf{x}, \mathbf{x}^\star)$ which is indexed by $i$. Finally, $h_{post}$ defines the output for each value of $(\mathbf{x}, \mathbf{y}, \mathbf{x}^\star)$. The bias transformation variable $\theta_b$ plays a vital role in our construction, as it breaks the symmetry within $k_i(\mathbf{x}, \mathbf{x}^\star)$. Without such asymmetry, it would not be possible for our constructed function to represent any function of $\mathbf{x}$ and $\mathbf{x}^\star$ after one gradient step.

In conclusion, we have shown that there exists a neural network structure for which $\hat{f}(\mathbf{x}^\star; \phi)$ is a universal approximator of $f_{target}(\mathbf{x}, \mathbf{y}, \mathbf{x}^\star)$. We chose a particular form of $\hat{f}(\cdot; \theta)$ that decouples forward and backward information flow. With this choice, it is possible to impose any desired post-update function, even in the face of adversarial training datasets and loss functions, e.g. when the gradient points in the wrong direction. If we make the assumption that the inner loss function and training dataset are not chosen adversarially and the error gradient points in the direction of improvement, it is likely

2 Due to the symmetry of kernels, this requires interpreting $\theta_b$ as part of the input, rather than a kernel hyperparameter, so that the left input is $(\mathbf{x}, \theta_b)$ and the right one is $(\mathbf{x}^\star, \phi_b)$.

3 The assumption with regard to compact subsets of the output space is inherited from the UFA theorem.

that a much simpler architecture will suffice that does not require multiplexing of forward and backward information in separate channels. Informative loss functions and training data allowing for simpler functions is indicative of the inductive bias built into gradient-based meta-learners, which is not present in recurrent meta-learners.

Our result in this section implies that a sufficiently deep representation combined with just a single gradient step can approximate any one-shot learning algorithm. In the next section, we will show the universality of MAML for K-shot learning algorithms.

### 4.4.2 *General Universality of the Gradient-Based Learner*

Now, we consider the more general K-shot setting, aiming to show that MAML can approximate any permutation invariant function of a dataset and test datapoint $(\{(\mathbf{x}, \mathbf{y})_i; i \in 1...K\}, \mathbf{x}^\star)$ for $K > 1$. Note that K does not need to be small. To reduce redundancy, we will only overview the differences from the 1-shot setting in this section. We include a full proof in Appendix A.2.

In the K-shot setting, the parameters of $\hat{f}(\cdot, \theta)$ are updated according to the following rule:

$$\phi = \theta - \alpha \frac{1}{K} \sum_{k=1}^{K} \nabla_\theta \ell(\mathbf{y}_k, f(\mathbf{x}_k; \theta))).$$

Defining the form of $\hat{f}$ to be the same as in Section 4.4.1, the post-update function is the following:

$$\hat{f}(\mathbf{x}^\star; \phi) = h_{\text{post}} \left( -\alpha \frac{1}{K} \sum_{i=1}^{N} \sum_{k=1}^{K} A_i \bar{e}(\mathbf{y}_k) k_i(\mathbf{x}_k, \mathbf{x}^\star); \theta_h \right)$$

In Appendix A.3, we show one way in which this function can approximate any function of $(\{(\mathbf{x}, \mathbf{y})_k; k \in 1...K\}, \mathbf{x}^\star)$ that is invariant to the ordering of the training datapoints $\{(\mathbf{x}, \mathbf{y})_k; k \in 1...K\}$. We do so by showing that we can select a setting of $\tilde{\phi}$ and of each $A_i$ and $B_i$ such that $\bar{\mathbf{z}}^\star$ is a vector containing a discretization of $\mathbf{x}^\star$ and frequency counts of the discretized datapoints[4]. If $\bar{\mathbf{z}}^\star$ is a vector that completely describes $(\{(\mathbf{x}, \mathbf{y})_i\}, \mathbf{x}^\star)$ without loss of information and because $h_{\text{post}}$ is a universal function approximator, $\hat{f}(\mathbf{x}^\star; \phi)$ can approximate any continuous function of $(\{(\mathbf{x}, \mathbf{y})_i\}, \mathbf{x}^\star)$ on compact subsets of $\mathbb{R}^{\dim(\mathbf{y})}$. It's also worth noting that the form of the above equation greatly resembles a kernel-based function approximator around the training points, and a substantially more efficient universality proof can likely be obtained starting from this premise.

4 With continuous labels $\mathbf{y}$ and mean-squared error $\ell$, we require the mild assumption that no two datapoints may share the same input value $\mathbf{x}$: the input datapoints must be unique.

### 4.4.3 *Loss Functions*

In the previous sections, we showed that a deep representation combined with gradient descent can approximate any learning algorithm. In this section, we will discuss the requirements that the loss function must satisfy in order for the results in Sections 4.4.1 and 4.4.2 to hold. As one might expect, the main requirement will be for the label to be recoverable from the gradient of the loss.

As seen in the definition of $f_{\text{out}}$ in Equation 9, the pre-update function $\hat{f}(\mathbf{x}, \theta)$ is given by $g_{\text{pre}}(\mathbf{z}; \theta_g)$, where $g_{\text{pre}}$ is used for back-propagating information about the label(s) to the learner. As stated in Equation 7, we require that the error gradient with respect to $\mathbf{z}$ to be:

$$\nabla_{\mathbf{z}} \ell(\mathbf{y}, \hat{f}(\mathbf{x}; \theta)) = \begin{bmatrix} \mathbf{0} \\ \overline{e}(\mathbf{y}) \\ \check{e}(\mathbf{y}) \end{bmatrix}, \quad \text{where } \mathbf{z} = \begin{bmatrix} \tilde{\mathbf{z}} \\ \overline{\mathbf{z}} \\ \theta_b \end{bmatrix} = \begin{bmatrix} \tilde{\phi}(\mathbf{x}; \theta_{\text{ft}}, \theta_b) \\ \mathbf{0} \\ 0 \end{bmatrix},$$

and where $\overline{e}(\mathbf{y})$ and $\check{e}(\mathbf{y})$ must be able to represent [at least] any linear function of the label $\mathbf{y}$.

We define $g_{\text{pre}}$ as follows: $g_{\text{pre}}(\mathbf{z}) := \begin{bmatrix} \tilde{W}_g & \overline{W}_g & \check{\mathbf{w}}_g \end{bmatrix} \mathbf{z} = \tilde{W}_g \tilde{\mathbf{z}} + \overline{W}_g \overline{\mathbf{z}} + \theta_b \check{\mathbf{w}}_g$.

To make the top term of the gradient equal to $\mathbf{0}$, we can set $\tilde{W}_g$ to be $0$, which causes the pre-update prediction $\hat{\mathbf{y}} = \hat{f}(\mathbf{x}, \theta)$ to be $\mathbf{0}$. Next, note that $\overline{e}(\mathbf{y}) = \overline{W}_g^T \nabla_{\hat{\mathbf{y}}} \ell(\mathbf{y}, \hat{\mathbf{y}})$ and $\check{e}(\mathbf{y}) = \check{\mathbf{w}}_g^T \nabla_{\hat{\mathbf{y}}} \ell(\mathbf{y}, \hat{\mathbf{y}})$. Thus, for $e(\mathbf{y})$ to be any linear function of $\mathbf{y}$, we require a loss function for which $\nabla_{\hat{\mathbf{y}}} \ell(\mathbf{y}, \mathbf{0})$ is a linear function $A\mathbf{y}$, where $A$ is invertible. Essentially, $\mathbf{y}$ needs to be recoverable from the loss function's gradient. In Appendix A.5 and A.6, we prove the following two theorems, thus showing that the standard $\ell_2$ and cross-entropy losses allow for the universality of gradient-based meta-learning.

**Theorem 4.4.1.** *The gradient of the standard mean-squared error objective evaluated at $\hat{\mathbf{y}} = \mathbf{0}$ is a linear, invertible function of $\mathbf{y}$.*

**Theorem 4.4.2.** *The gradient of the softmax cross entropy loss with respect to the pre-softmax logits is a linear, invertible function of $\mathbf{y}$, when evaluated at $\mathbf{0}$.*

Now consider other popular loss functions whose gradients do not satisfy the label-linearity property. The gradients of the $\ell_1$ and hinge losses are piecewise constant, and thus do not allow for universality. The Huber loss is also piecewise constant in some areas its domain. These error functions effectively lose information because simply looking at their gradient is insufficient to determine the label. Recurrent meta-learners that

take the gradient as input, rather than the label, e.g. Andrychowicz et al. (2016), will also suffer from this loss of information when using these error functions.

## 4.5 RELATED WORK

The method that we propose in this chapter addresses the general problem of meta-learning (Thrun and Pratt, 1998; Schmidhuber, 1987; Naik and Mammone, 1992), which includes few-shot learning. A popular approach for meta-learning is to train a meta-learner that learns how to update the parameters of the learner's model (S. Bengio et al., 1992; Schmidhuber, 1992; Y. Bengio et al., 1990). This approach has been applied to learning to optimize deep networks (Hochreiter et al., 2001; Andrychowicz et al., 2016; K. Li and Malik, 2017a), as well as for learning dynamically changing recurrent networks (Ha et al., 2017). One recent approach learns both the weight initialization and the optimizer, for few-shot image recognition (Ravi and Larochelle, 2017). Unlike these methods, the MAML learner's weights are updated using the gradient, rather than a learned update; our method does not introduce additional parameters for meta-learning nor require a particular learner architecture.

Few-shot learning methods have also been developed for specific tasks such as generative modeling (H. Edwards and Storkey, 2017; Rezende et al., 2016) and image recognition (Vinyals et al., 2016). One successful approach for few-shot classification is to learn to compare new examples in a learned metric space using e.g. Siamese networks (Koch et al., 2015) or recurrence with attention mechanisms (Vinyals et al., 2016; Shyam et al., 2017; Snell et al., 2017). These approaches have generated some of the most successful results, but are difficult to directly extend to other problems, such as reinforcement learning. Our method, in contrast, is agnostic to the form of the model and to the particular learning task.

Another approach to meta-learning is to train memory-augmented models on many tasks, where the recurrent learner is trained to adapt to new tasks as it is rolled out. Such networks have been applied to few-shot image recognition (Santoro et al., 2016; Munkhdalai and H. Yu, 2017) and learning "fast" reinforcement learning agents (Duan et al., 2016b; Wang and Hebert, 2016). Memory augmentation can also take the form of a recurrent network with fast Hebbian learning updates (J. Ba et al., 2016). Our experiments show that our method outperforms the recurrent approach on few-shot classification. Furthermore, unlike these methods, our approach simply provides a good weight initialization and uses the same gradient descent update for both the learner and meta-update. As a result, it is straightforward to finetune the learner for additional gradient

steps.

Our approach is also related to methods for initialization of deep networks. In computer vision, models pretrained on large-scale image classification have been shown to learn effective features for a range of problems (Donahue et al., 2014). In contrast, our method explicitly optimizes the model for fast adaptability, allowing it to adapt to new tasks with only a few examples. Our method can also be viewed as explicitly maximizing sensitivity of new task losses to the model parameters. A number of prior works have explored sensitivity in deep networks, often in the context of initialization (Saxe et al., 2014; Kirkpatrick et al., 2016). Most of these works have considered good random initializations, though a number of papers have addressed data-dependent initializers (Krähenbühl et al., 2016; Salimans and D. Kingma, 2016), including learned initializations (Husken and Goerick, 2000; Maclaurin et al., 2015). In contrast, our method explicitly trains the parameters for sensitivity on a given task distribution, allowing for extremely efficient adaptation for problems such as K-shot learning and rapid reinforcement learning in only one or a few gradient steps.

## 4.6 EXPERIMENTAL EVALUATION

The goal of our experimental evaluation is to answer the following questions: (1) Can MAML enable fast learning of new tasks? (2) Can MAML be used for meta-learning in multiple different domains, including supervised regression, classification, and reinforcement learning? (3) Can a model learned with MAML continue to improve with additional gradient updates and/or examples? (4) Is there empirical benefit to incorporating the inductive bias of gradient descent into the meta-learning process?

To answer the first two questions, we will first study MAML in three different domains, a toy regression problem, two few-shot image classification problems, and four simple reinforcement learning problems involved continuous control of simulated robots. Then, in Section 4.6.4, we will consider the latter two questions, aiming to empirically study the inductive bias of gradient-based and recurrent meta-learners. Finally, in Section 4.6.5, we will investigate the role of model depth in gradient-based meta-learning, as the theory suggests that deeper networks lead to increased expressive power for representing different learning procedures.

All of the meta-learning problems that we consider require some amount of adaptation to new tasks at test-time. When possible, we compare our results to an oracle that receives the identity of the task (which is a problem-dependent representation) as an additional input, as an upper bound on the performance of the model. All of the

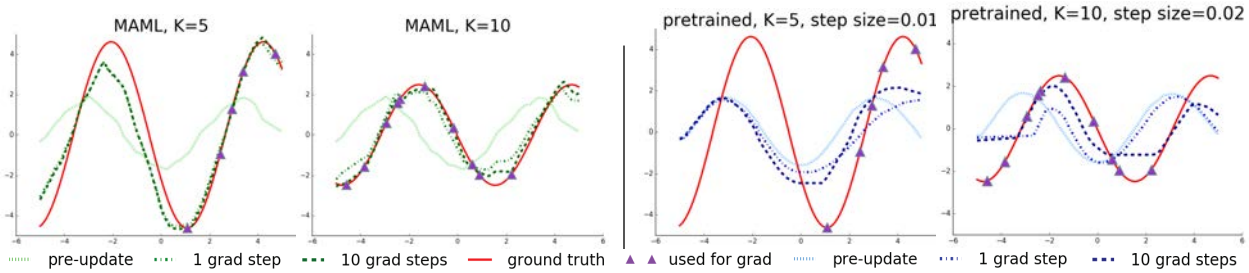


Figure 3: Few-shot adaptation for the simple regression task. Left: Note that MAML is able to estimate parts of the curve where there are no datapoints, indicating that the model has learned about the periodic structure of sine waves. Right: Fine-tuning of a model pretrained on the same distribution of tasks without MAML, with a tuned step size. Due to the often contradictory outputs on the pre-training tasks, this model is unable to recover a suitable representation and fails to extrapolate from the small number of test-time samples.

experiments were performed using TensorFlow (Abadi et al., 2016), which allows for automatic differentiation through the gradient update(s) during meta-learning. The code is available online[5].

#### 4.6.1 *Regression*

We start with a simple regression problem that illustrates the basic principles of MAML. Each task involves regressing from the input to the output of a sine wave, where the amplitude and phase of the sinusoid are varied between tasks. Thus, $p(\mathcal{T})$ is continuous, where the amplitude varies within $[0.1, 5.0]$ and the phase varies within $[0, \pi]$, and the input and output both have a dimensionality of 1. During training and testing, datapoints $\mathbf{x}$ are sampled uniformly from $[-5.0, 5.0]$. The loss is the mean-squared error between the prediction $f(\mathbf{x})$ and true value. The regressor is a neural network model with 2 hidden layers of size 40 with ReLU nonlinearities. When training with MAML, we use one gradient update with $K = 10$ examples with a fixed step size $\alpha = 0.01$, and use Adam as the meta-optimizer (D. Kingma and J. Ba, 2015). The baselines are likewise trained with Adam. To evaluate performance, we fine-tune a single meta-learned model on varying numbers of $K$ examples, and compare performance to two baselines: (a) pretraining on all of the tasks, which entails training a network to regress to random sinusoid functions

5 Code for the regression and supervised experiments is at `github.com/cbfinn/maml` and code for the RL experiments is at `github.com/cbfinn/maml_rl`

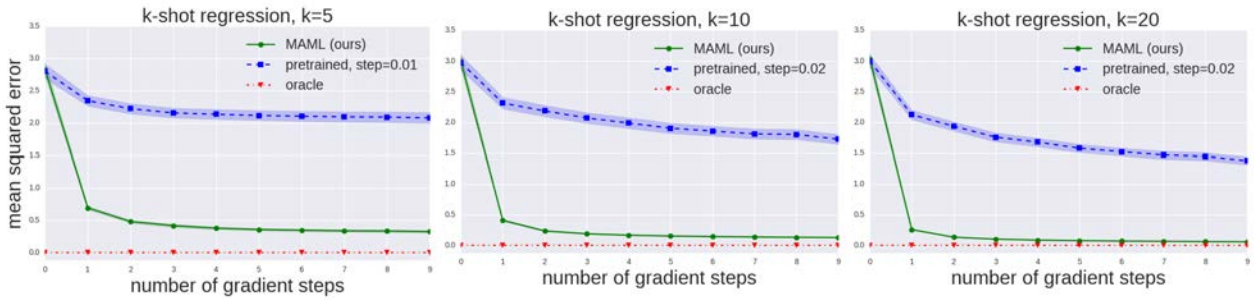


Figure 4: Quantitative sinusoid regression results showing the learning curve at meta test-time. Note that MAML continues to improve with additional gradient steps without overfitting to the extremely small dataset during meta-testing, achieving a loss that is substantially lower than the baseline fine-tuning approach.

and then, at test-time, fine-tuning with gradient descent on the $K$ provided points, using an automatically tuned step size, and (b) an oracle which receives the true amplitude and phase as input. In Appendix A.9, we show comparisons to additional multi-task and adaptation methods.

We evaluate performance by fine-tuning the model learned by MAML and the pre-trained model on $K = \{5, 10, 20\}$ datapoints. During fine-tuning, each gradient step is computed using the same $K$ datapoints. The qualitative results, shown in Figure 3 and further expanded on in Appendix A.8 show that the learned model is able to quickly adapt with only 5 datapoints, shown as purple triangles, whereas the model that is pre-trained using standard supervised learning on all tasks is unable to adequately adapt with so few datapoints without catastrophic overfitting. Crucially, when the $K$ datapoints are all in one half of the input range, the model trained with MAML can still infer the amplitude and phase in the other half of the range, demonstrating that the MAML trained model $f$ has learned to model the periodic nature of the sine wave. Furthermore, we observe both in the qualitative and quantitative results (Figure 4 and Appendix A.8) that the model learned with MAML continues to improve with additional gradient steps, despite being trained for maximal performance after one gradient step. This improvement suggests that MAML optimizes the parameters such that they lie in a region that is amenable to fast adaptation and is sensitive to loss functions from $p(\mathcal{T})$, as discussed in Section 4.1, rather than overfitting to parameters $\theta$ that only improve after one step.

#### 4.6.2 *Classification*

To evaluate MAML in comparison to prior meta-learning and few-shot learning algorithms, we applied our method to few-shot image recognition on the Omniglot (Lake et al., 2011) and MiniImagenet datasets. The Omniglot dataset consists of 20 instances of 1623 characters from 50 different alphabets. Each instance was drawn by a different person. The MiniImagenet dataset was proposed by Ravi and Larochelle (2017), and involves 64 training classes, 12 validation classes, and 24 test classes. The Omniglot and MiniImagenet image recognition tasks are the most common recently used few-shot learning benchmarks (Vinyals et al., 2016; Santoro et al., 2016; Ravi and Larochelle, 2017). We follow the experimental protocol proposed by Vinyals et al. (2016), which involves fast learning of N-way classification with 1 or 5 shots. The problem of N-way classification is set up as follows: select N unseen classes, provide the model with K different instances of each of the N classes, and evaluate the model's ability to classify new instances within the N classes. For Omniglot, we randomly select 1200 characters for training, irrespective of alphabet, and use the remaining for testing. The Omniglot dataset is augmented with rotations by multiples of 90 degrees, as proposed by Santoro et al. (2016).

Our model follows the same architecture as the embedding function used by Vinyals et al. (2016), which has 4 modules with a $3 \times 3$ convolutions and 64 filters, followed by batch normalization (Ioffe and Szegedy, 2015), a ReLU nonlinearity, and $2 \times 2$ max-pooling. The Omniglot images are downsampled to $28 \times 28$, so the dimensionality of the last hidden layer is 64. As in the baseline classifier used by Vinyals et al. (2016), the last layer is fed into a softmax. For Omniglot, we used strided convolutions instead of max-pooling. For MiniImagenet, we used 32 filters per layer to reduce overfitting, as done by (Ravi and Larochelle, 2017). In order to also provide a fair comparison against memory-augmented neural networks (Santoro et al., 2016) and to test the flexibility of MAML, we also provide results for a non-convolutional network. For this, we use a network with 4 hidden layers with sizes 256, 128, 64, 64, each including batch normalization and ReLU nonlinearities, followed by a linear layer and softmax. For all models, the loss function is the cross-entropy error between the predicted and true class. Additional hyperparameter details are included in Appendix A.7.1.

We present the results in Table 2. The convolutional model learned by MAML compares well to the state-of-the-art results on this task, narrowly outperforming the prior methods. Some of these existing methods, such as matching networks, Siamese networks, and memory models are designed with few-shot classification in mind, and are not readily applicable to domains such as reinforcement learning. Additionally, the model

| Omniglot (Lake et al., 2011) | 5-way Accuracy | | 20-way Accuracy | |
|---|---|---|---|---|
| | 1-shot | 5-shot | 1-shot | 5-shot |
| MANN, no conv (Santoro et al., 2016) | 82.8% | 94.9% | – | – |
| **MAML, no conv (ours)** | **89.7 ± 1.1%** | **97.5 ± 0.6%** | – | – |
| Siamese nets (Koch et al., 2015) | 97.3% | 98.4% | 88.2% | 97.0% |
| matching nets (Vinyals et al., 2016) | 98.1% | 98.9% | 93.8% | 98.5% |
| neural statistician (H. Edwards and Storkey, 2017) | 98.1% | 99.5% | 93.2% | 98.1% |
| memory mod. (Kaiser et al., 2017) | 98.4% | 99.6% | 95.0% | 98.6% |
| **MAML (ours)** | **98.7 ± 0.4%** | **99.9 ± 0.1%** | **95.8 ± 0.3%** | **98.9 ± 0.2%** |

| MiniImagenet (Ravi and Larochelle, 2017) | 5-way Accuracy | |
|---|---|---|
| | 1-shot | 5-shot |
| fine-tuning baseline | 28.86 ± 0.54% | 49.79 ± 0.79% |
| nearest neighbor baseline | 41.08 ± 0.70% | 51.04 ± 0.65% |
| matching nets (Vinyals et al., 2016) | 43.56 ± 0.84% | 55.31 ± 0.73% |
| meta-learner LSTM (Ravi and Larochelle, 2017) | 43.44 ± 0.77% | 60.60 ± 0.71% |
| **MAML, first order approx. (ours)** | **48.07 ± 1.75%** | **63.15 ± 0.91%** |
| **MAML (ours)** | **48.70 ± 1.84%** | **63.11 ± 0.92%** |

Table 2: Few-shot classification on held-out Omniglot characters (top) and the MiniImagenet test set (bottom). MAML achieves results that are comparable to or outperform state-of-the-art convolutional and recurrent models. Siamese nets, matching nets, and the memory module approaches are all specific to classification, and are not directly applicable to regression or RL scenarios. The ± shows 95% confidence intervals over tasks. Note that the Omniglot results may not be strictly comparable since the train/test splits used in the prior work were not available. The MiniImagenet evaluation of baseline methods and matching networks is from Ravi and Larochelle (2017).

learned with MAML uses fewer overall parameters compared to matching networks and the meta-learner LSTM, since the algorithm does not introduce any additional parameters beyond the weights of the classifier itself. Compared to these prior methods, memory-augmented neural networks (Santoro et al., 2016) specifically, and recurrent meta-learning models in general, represent a more broadly applicable class of methods that, like MAML, can be used for other tasks such as reinforcement learning (Duan et al., 2016b; Wang and Hebert, 2016). However, as shown in the comparison, MAML signifi-

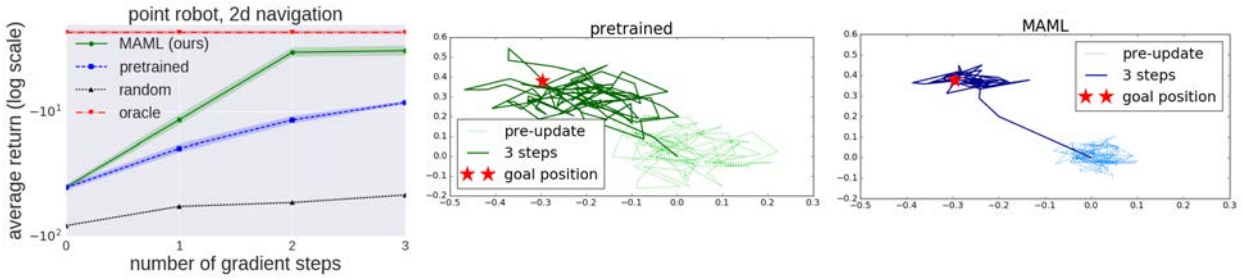


Figure 5: Left: quantitative results from 2D navigation task, Middle & Right: qualitative comparison between model learned with MAML and with fine-tuning from a pretrained network.

cantly outperforms memory-augmented networks and the meta-learner LSTM on 5-way Omniglot and MiniImagenet classification, both in the 1-shot and 5-shot case.

A significant computational expense in MAML comes from the use of second derivatives when backpropagating the meta-gradient through the gradient operator in the meta-objective (see Equation (2)). On MiniImagenet, we show a comparison to a first-order approximation of MAML, where these second derivatives are omitted. Note that the resulting method still computes the meta-gradient at the post-update parameter values $\phi_i$, which provides for effective meta-learning. Surprisingly however, the performance of this method is nearly the same as that obtained with full second derivatives, suggesting that most of the improvement in MAML comes from the gradients of the objective at the post-update parameter values, rather than the second order updates from differentiating through the gradient update. Past work has observed that ReLU neural networks are locally almost linear (Goodfellow et al., 2015), which suggests that second derivatives may be close to zero in most cases, partially explaining the good performance of the first-order approximation. This approximation removes the need for computing Hessian-vector products in an additional backward pass, which we found led to roughly 33% speed-up in network computation.

### 4.6.3 *Reinforcement Learning*

To evaluate MAML on reinforcement learning problems, we constructed several sets of tasks based off of the simulated continuous control environments in the rllab benchmark suite (Duan et al., 2016a). We discuss the individual domains below. In all of the domains, the model trained by MAML is a neural network policy with two hidden layers of size 100, with ReLU nonlinearities. The gradient updates are computed using vanilla policy gradient (REINFORCE) (Williams, 1992), and we use trust-region policy optimization

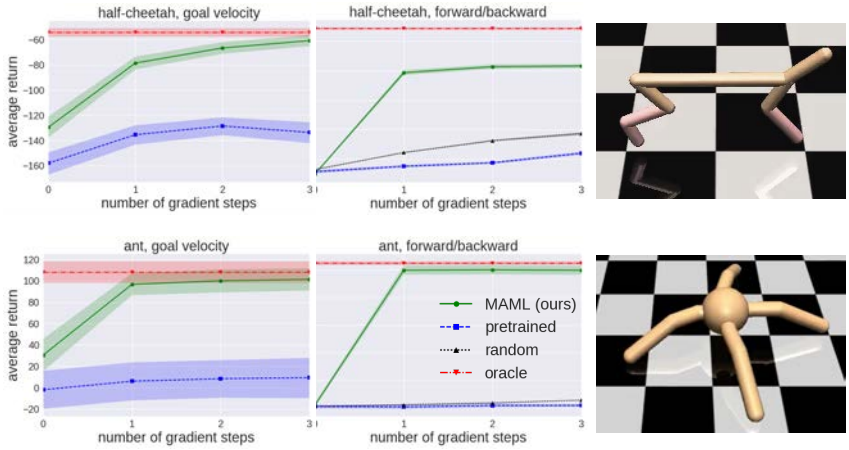


Figure 6: Reinforcement learning results for the half-cheetah (top) and ant (bottom) locomotion tasks, with the environments shown on the right. Each gradient step requires additional samples from the environment, unlike the supervised learning tasks. The results show that MAML can adapt to new goal velocities and directions substantially faster than conventional pretraining or random initialization, achieving good performs in just two or three gradient steps. We exclude the goal velocity, random baseline curves, since the returns are much worse ($< -200$ for cheetah and $< -25$ for ant).

(TRPO) as the meta-optimizer (Schulman et al., 2015). In order to avoid computing third derivatives, we use finite differences to compute the Hessian-vector products for TRPO. For both learning and meta-learning updates, we use the standard linear feature baseline proposed by Duan et al. (2016a), which is fitted separately at each iteration for each sampled task in the batch. We compare to three baseline models: (a) pretraining one policy on all of the tasks and then fine-tuning, (b) training a policy from randomly initialized weights, and (c) an oracle policy which receives the parameters of the task as input, which for the tasks below corresponds to a goal position, goal direction, or goal velocity for the agent. The baseline models of (a) and (b) are fine-tuned with gradient descent with a manually tuned step size. Videos of the learned policies can be viewed at `sites.google.com/view/maml`

**2D Navigation.** In our first meta-RL experiment, we study a set of tasks where a point agent must move to different goal positions in 2D, randomly chosen for each task within a unit square. The observation is the current 2D position, and actions correspond to velocity commands clipped to be in the range $[-0.1, 0.1]$. The reward is the negative squared distance to the goal, and episodes terminate when the agent is within 0.01 of the goal or at the horizon of $H = 100$. The policy was trained with MAML to maximize performance after 1 policy gradient update using 20 trajectories. Additional hyperparameter

settings for this problem and the following RL problems are in Appendix A.7.2. In our evaluation, we compare adaptation to a new task with up to 4 gradient updates, each with 40 samples. The results in Figure 5 show the adaptation performance of models that are initialized with MAML, conventional pretraining on the same set of tasks, random initialization, and an oracle policy that receives the goal position as input. The results show that MAML can learn a model that adapts much more quickly in a single gradient update, and furthermore continues to improve with additional updates.

**Locomotion.** To study how well MAML can scale to more complex deep RL problems, we also study adaptation on high-dimensional locomotion tasks with the MuJoCo simulator (Todorov et al., 2012). The tasks require two simulated robots – a planar cheetah and a 3D quadruped (the "ant") – to run in a particular direction or at a particular velocity. In the goal velocity experiments, the reward is the negative absolute value between the current velocity of the agent and a goal, which is chosen uniformly at random between 0.0 and 2.0 for the cheetah and between 0.0 and 3.0 for the ant. In the goal direction experiments, the reward is the magnitude of the velocity in either the forward or backward direction, chosen at random for each task in $p(\mathcal{T})$. The horizon is $H = 200$, with 20 rollouts per gradient step for all problems except the ant forward/backward task, which used 40 rollouts per step. The results in Figure 6 show that MAML learns a model that can quickly adapt its velocity and direction with even just a single gradient update, and continues to improve with more gradient steps. The results also show that, on these challenging tasks, the MAML initialization substantially outperforms random initialization and pretraining. In fact, pretraining is in some cases worse than random initialization, a fact observed in prior RL work (Parisotto et al., 2016).

#### 4.6.4 *Empirical Study of Inductive Bias*

In this section, we aim to empirically explore the differences between gradient-based and recurrent meta-learners. In particular, we aim to answer the following questions: (1) can a learner trained with MAML further improve from additional gradient steps when learning new tasks at test time, or does it start to overfit? and (2) does the inductive bias of gradient descent enable better few-shot learning performance on tasks outside of the training distribution, compared to learning algorithms represented as recurrent networks?

To study both questions, we will consider two simple few-shot learning domains. The first is 5-shot regression on a family of sine curves with varying amplitude and phase. We trained all models on a uniform distribution of tasks with amplitudes $A \in [0.1, 5.0]$,

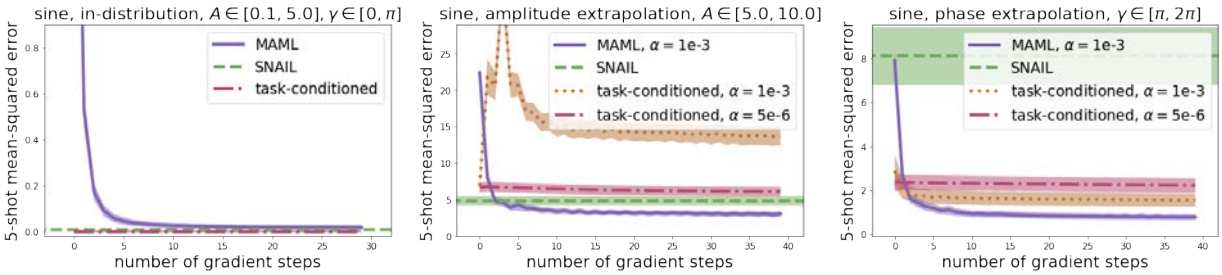


Figure 7: The effect of additional gradient steps at test time when attempting to solve new tasks. The MAML model, trained with 5 inner gradient steps, can further improve with more steps. All methods are provided with the same data – 5 examples – where each gradient step is computed using the same 5 datapoints.

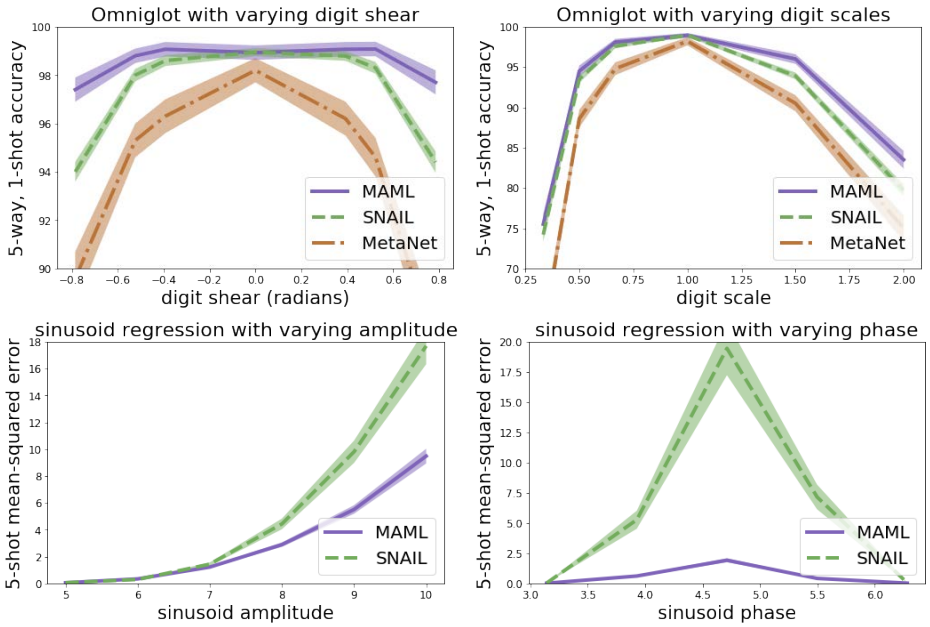


Figure 8: Learning performance on out-of-distribution tasks as a function of the task variability. There is a clear trend that recurrent meta-learners such as SNAIL and MetaNet acquire learning strategies that are less generalizable than those learned with gradient-based meta-learning.

and phases $\gamma \in [0, \pi]$. The second domain is 1-shot character classification using the Omniglot dataset (Lake et al., 2011), following the training protocol introduced by Santoro et al. (2016). In our comparisons to recurrent meta-learners, we will use two state-of-the-art

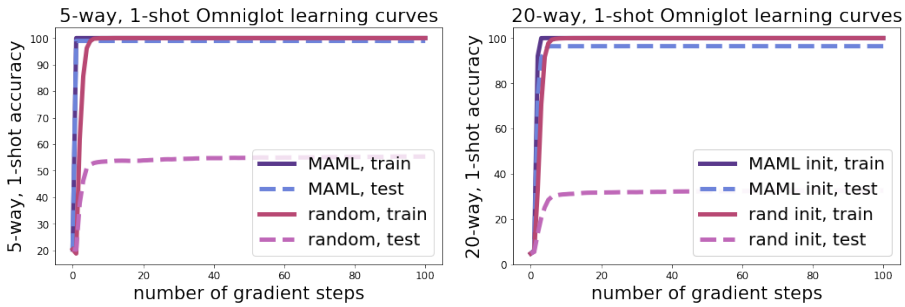


Figure 9: Comparison of finetuning from a MAML-initialized network and a network initialized randomly, trained from scratch. Both methods achieve about the same training accuracy. But, MAML also attains good test accuracy, while the network trained from scratch overfits catastrophically to the 5 or 20 examples. Interestingly, the MAML-initialized model does not begin to overfit, even though meta-training used 1 and 5 steps respectively while the graph shows up to 100 gradient steps.

meta-learning models: SNAIL (Mishra et al., 2018) and meta-networks (Munkhdalai and H. Yu, 2017). In some experiments, we will also compare to a task-conditioned model, which is trained to map from both the input and the task description to the label. Like MAML, the task-conditioned model can be fine-tuned on new data using gradient descent, but is not trained for few-shot adaptation. We include more experimental details in Appendix A.7.3.

To answer the first question, we fine-tuned a model trained using MAML with many more gradient steps than used during meta-training. The results on the sinusoid domain, shown in Figure 7, show that a MAML-learned initialization trained for fast adaption in 5 steps can further improve beyond 5 gradient steps, especially on out-of-distribution tasks. In contrast, a task-conditioned model trained without MAML can easily overfit to out-of-distribution tasks. With the Omniglot dataset, as seen in Figure 9, a MAML model that was trained with 5 inner gradient steps can be fine-tuned for 100 gradient steps without leading to any drop in test accuracy. As expected, a model initialized randomly and trained from scratch quickly reaches perfect training accuracy, but overfits massively to the 20 examples.

Next, we investigate the second question, aiming to compare MAML with state-of-the-art recurrent meta-learners on tasks that are related to, but outside of the distribution of the training tasks. All three methods achieved similar performance within the distribution of training tasks for 5-way 1-shot Omniglot classification and 5-shot sinusoid

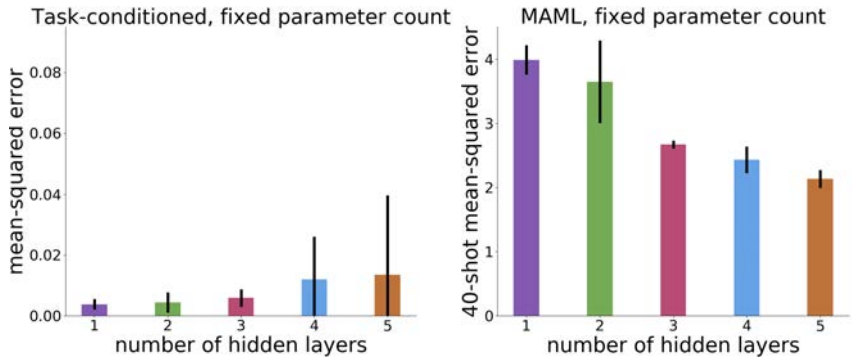


Figure 10: Comparison of depth while keeping the number of parameters constant. Task-conditioned models do not need more than one hidden layer, whereas meta-learning with MAML clearly benefits from additional depth. Error bars show standard deviation over three training runs.

regression. In the Omniglot setting, we compare each method's ability to distinguish digits that have been sheared or scaled by varying amounts. In the sinusoid regression setting, we compare on sinusoids with extrapolated amplitudes within $[5.0, 10.0]$ and phases within $[\pi, 2\pi]$. The results in Figure 8 show a clear trend that MAML recovers more generalizable learning strategies. Combined with the theoretical universality results, these experiments indicate that deep gradient-based meta-learners are not only equivalent in representational power to recurrent meta-learners, but should also be a considered as a strong contender in settings that contain domain shift between meta-training and meta-testing tasks, where their strong inductive bias for reasonable learning strategies provides substantially improved performance.

### 4.6.5 *Effect of Depth*

The proofs in Sections 4.4.1 and 4.4.2 suggest that gradient descent with deeper representations results in more expressive learning procedures. In contrast, the universal function approximation theorem only requires a single hidden layer to approximate any function. Now, we seek to empirically explore this theoretical finding, aiming to answer the question: is there a scenario for which model-agnostic meta-learning requires a deeper representation to achieve good performance, compared to the depth of the representation needed to solve the underlying tasks being learned?

To answer this question, we will study a simple regression problem, where the meta-

learning goal is to infer a polynomial function from 40 input/output datapoints. We use polynomials of degree 3 where the coefficients and bias are sampled uniformly at random within $[-1, 1]$ and the input values range within $[-3, 3]$. Similar to the conditions in the proof, we meta-train and meta-test with one gradient step, use a mean-squared error objective, use ReLU nonlinearities, and use a bias transformation variable of dimension 10. To compare the relationship between depth and expressive power, we will compare models with a fixed number of parameters, approximately $40,000$, and vary the network depth from 1 to 5 hidden layers. As a point of comparison to the models trained for meta-learning using MAML, we trained standard feedforward models to regress from the input and the 4-dimensional task description (the 3 coefficients of the polynomial and the scalar bias) to the output. These task-conditioned models act as an oracle and are meant to empirically determine the depth needed to represent these polynomials, independent of the meta-learning process. Theoretically, we would expect the task-conditioned models to require only one hidden layer, as per the universal function approximation theorem. In contrast, we would expect the MAML model to require more depth. The results, shown in Figure 10, demonstrate that the task-conditioned model does indeed not benefit from having more than one hidden layer, whereas the MAML clearly achieves better performance with more depth even though the model capacity, in terms of the number of parameters, is fixed. This empirical effect supports the theoretical finding that depth is important for effective meta-learning using MAML.

## 4.7 DISCUSSION

In this chapter, we introduced a meta-learning method based on learning easily adaptable model parameters through gradient descent. Our approach has a number of benefits. It is simple and does not introduce any learned parameters for meta-learning. It can be combined with any model representation that is amenable to gradient-based training, and any differentiable objective, including classification, regression, and reinforcement learning. Since our method merely produces a weight initialization, adaptation can be performed with any amount of data and any number of gradient steps, allowing for *consistent* learning algorithms. We further theoretically show that, for a sufficiently deep neural network, the initial representation combined with only a single gradient step can approximate any learning algorithm, achieving the property of universality discussed in Section 3.1. Our findings suggest that, from the standpoint of expressivity, there is no theoretical disadvantage to embedding gradient descent into the meta-learning process.

Empirically, we demonstrated state-of-the-art results on classification with only a few

gradient steps and one or five examples per class, and showed that our method can adapt an RL agent using policy gradients and a very modest amount of experience. In all of our experimental settings, we found that the learning strategies acquired with MAML are more successful when faced with out-of-domain tasks compared to recurrent learners. Furthermore, we show that the representations acquired with MAML are highly resilient to overfitting. These results suggest that, in addition to the desirable theoretical properties, model-agnostic meta-learning also has a number of practical benefits.

# 5

# A PROBABILISTIC MODEL-AGNOSTIC META-LEARNING ALGORITHM

While the MAML algorithm exhibits two desirable properties previously discussed, consistency and universality, it cannot effectively reason about ambiguity. The goal of this chapter is to consider the question – can we build a model-agnostic meta-learning algorithm that is both scalable and consistent, like the original MAML method, yet also has the ability to reason about uncertainty, akin to few-shot learning approaches that use Bayesian models? To do so, we build upon tools in amortized variational inference. Our approach extends MAML to model a distribution over prior model parameters, which leads to an appealing simple stochastic adaptation procedure that simply injects noise into gradient descent at meta-test time. The meta-training procedure then optimizes for this simple inference process to produce samples from an approximate model posterior.

Hence, the main contribution of this chapter is a reframing of MAML as a graphical model inference problem, where variational inference can provide us with a principled and natural mechanism for modeling uncertainty and ambiguity. Our approach enables sampling multiple potential solutions to a few-shot learning problem at meta-test time, and our experiments show that this ability can be utilized to sample multiple possible regressors for an ambiguous regression problem, as well as multiple possible classifiers for ambiguous few-shot attribute classification tasks.

## 5.1 OVERVIEW AND PRELIMINARIES

We set up a graphical model for the few-shot learning problem. In particular, we want a hierarchical Bayesian model that includes random variables for the prior distribution over function parameters, $\theta$, the distribution over parameters for a particular task, $\phi_i$, and the task training and test datapoints. This graphical model is illustrated in Figure 11 (left), where tasks are indexed over $i$ and datapoints are indexed over $j$. We will use the

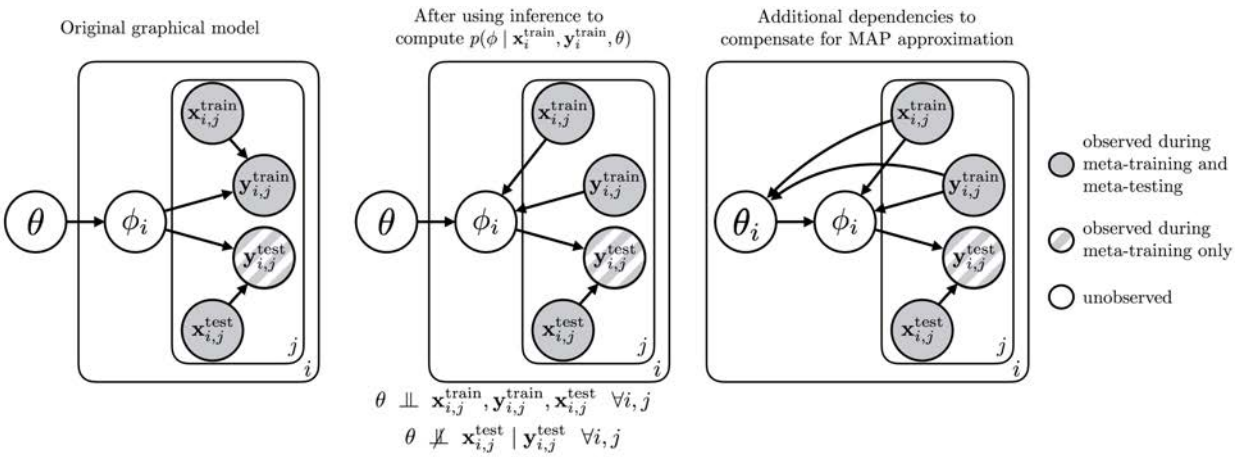


Figure 11: Graphical models corresponding to our approach. The original graphical model (left) is transformed into the center model after performing inference over $\phi_i$. We find it beneficial to introduce additional dependencies of the prior on the training data to compensate for using the MAP estimate to approximate $p(\phi_i)$, as shown on the right.

shorthand $\mathbf{x}_i^{tr}$, $\mathbf{y}_i^{tr}$, $\mathbf{x}_i^{test}$, $\mathbf{y}_i^{test}$ to denote the sets of datapoints $\{\mathbf{x}_{i,j}^{tr} | \forall j\}$, $\{\mathbf{y}_{i,j}^{tr} | \forall j\}$, $\{\mathbf{x}_{i,j}^{test} | \forall j\}$, $\{\mathbf{y}_{i,j}^{test} | \forall j\}$ and $\mathcal{D}_i^{tr}$, $\mathcal{D}_i^{test}$ to denote $\{\mathbf{x}_i^{tr}, \mathbf{y}_i^{tr}\}$ and $\{\mathbf{x}_i^{test}, \mathbf{y}_i^{test}\}$.

We will study few-shot supervised learning problems where the loss function used corresponds to the negative log likelihood of the data under the classifier or regressor: $\mathcal{L}(\theta, \mathcal{D}) = -\sum_{(\mathbf{x}_j, \mathbf{y}_j) \in \mathcal{D}} \log p(\mathbf{y}_j | \mathbf{x}_j, \theta)$.

### 5.2 Gradient-based meta-learning with variational inference

In the graphical model in Figure 11, the predictions for each task are determined by the task-specific model parameters $\phi_i$. At meta-test time, these parameters are influenced by the prior $p(\phi_i|\theta)$, as well as by the observed training data $\mathbf{x}^{tr}, \mathbf{y}^{tr}$. The test inputs $\mathbf{x}^{test}$ are also observed, but the test outputs $\mathbf{y}^{test}$, which need to be predicted, are not observed. Note that $\phi_i$ is thus independent of $\mathbf{x}^{test}$, but not of $\mathbf{x}^{tr}, \mathbf{y}^{tr}$. Therefore, posterior inference over $\phi_i$ must take into account both the evidence (training set) and the prior imposed by $p(\theta)$ and $p(\phi_i|\theta)$. Conventional MAML can be interpreted as approximating maximum a posteriori inference under a simplified model where $p(\theta)$ is a delta function, and inference is performed by running gradient descent on $\log p(\mathbf{y}^{tr}|\mathbf{x}^{tr}, \phi_i)$ for a fixed number of iterations starting from $\phi_i^0 = E[\theta]$ (Grant et al., 2018). The corresponding distribution $p(\phi_i|\theta)$ is approximately Gaussian, with a mean that depends on the step

size and number of gradient steps. When $p(\theta)$ is not deterministic, we must make a further approximation to account for the random variable $\theta$.

One way we can do this is by using structured variational inference. In structured variational inference, we approximate the distribution over the hidden variables $\theta$ and $\phi_i$ for each task with some approximate distribution $q_i(\theta, \phi_i)$. There are two reasonable choices we can make for $q_i(\theta, \phi_i)$. First, we can approximate it as a product of independent marginals, according to $q_i(\theta, \phi_i) = q_i(\theta)q_i(\phi_i)$. However, this approximation does not permit uncertainty to propagate effectively from $\theta$ to $\phi_i$. A more expressive approximation is the structured variational approximation $q_i(\theta, \phi_i) = q_i(\theta)q_i(\phi_i|\theta)$. We can further avoid storing a separate variational distribution $q_i(\phi_i|\theta)$ and $q_i(\theta)$ for each task $\mathcal{T}_i$ by employing an amortized variational inference technique (D. P. Kingma and Welling, 2013; Johnson et al., 2016; Shu et al., 2018), where we instead set $q_i(\phi_i|\theta) = q_\psi(\phi_i|\theta, \mathbf{x}_i^{tr}, \mathbf{y}_i^{tr}, \mathbf{x}_i^{test}, \mathbf{y}_i^{test})$, where $q_\psi$ is defined by some function approximator with parameters $\psi$ that takes $\mathbf{x}_i^{tr}, \mathbf{y}_i^{tr}$ as input, and the same $q_\psi$ is used for all tasks. Similarly, we can define $q_i(\theta)$ as $q_\psi(\theta|\mathbf{x}_i^{tr}, \mathbf{y}_i^{tr}, \mathbf{x}_i^{test}, \mathbf{y}_i^{test})$. We can now write down the variational lower bound on the log-likelihood as

$$\log p(\mathbf{y}_i^{test}|\mathbf{x}_i^{test}, \mathbf{x}_i^{tr}\mathbf{y}_i^{tr}) \geqslant \mathop{\mathbb{E}}_{\theta,\phi_i\sim q_\psi}\left[\log p(\mathbf{y}_i^{tr}|\mathbf{x}_i^{tr}, \phi_i) + \log p(\mathbf{y}_i^{test}|\mathbf{x}_i^{test}, \phi_i) + \log p(\phi_i|\theta) + \log p(\theta)\right] + \mathcal{H}(q_\psi(\phi_i|\theta, \mathbf{x}_i^{tr}, \mathbf{y}_i^{tr}, \mathbf{x}_i^{test}, \mathbf{y}_i^{test})) + \mathcal{H}(q_\psi(\theta|\mathbf{x}_i^{tr}, \mathbf{y}_i^{tr}, \mathbf{x}_i^{test}, \mathbf{y}_i^{test})).$$

The likelihood terms on the first line can be evaluated efficiently: given a sample $\theta, \phi_i \sim q(\theta, \phi_i|\mathbf{x}_i^{tr}, \mathbf{y}_i^{tr}, \mathbf{x}_i^{test}, \mathbf{y}_i^{test})$, the training and test likelihoods simply correspond to the loss of the network with parameters $\phi_i$. The prior $p(\theta)$ can be chosen to be Gaussian, with a learned mean and (diagonal) covariance to provide for flexibility to choose the prior parameters. This corresponds to a Bayesian version of the MAML algorithm. We will define these parameters as $\mu_\theta$ and $\sigma_\theta^2$. Lastly, $p(\phi_i|\theta)$ must be chosen. This choice is more delicate. One way to ensure a tractable likelihood is to use a Gaussian with mean $\theta$. This choice is reasonable, because it encourages $\phi_i$ to stay close to the prior parameters $\phi_i$, but we will see in the next section how a more expressive implicit conditional can be obtained using gradient descent, resulting in a procedure that more closely resembles the original MAML algorithm while still modeling the uncertainty. Lastly, we must choose a form for the inference networks $q_\psi(\phi_i|\theta, \mathbf{x}_i^{tr}, \mathbf{y}_i^{tr}, \mathbf{x}_i^{test}, \mathbf{y}_i^{test})$ and $q_\psi(\theta|\mathbf{x}_i^{tr}, \mathbf{y}_i^{tr}, \mathbf{x}_i^{test}, \mathbf{y}_i^{test})$. They must be chosen so that their entropies on the second line of the above equation are tractable. Furthermore, note that both of these distributions model very high-dimensional random variables: a deep neural network can have hundreds of thousands or millions of parameters. So while we can use an arbitrary function approximator, we would like to find a scalable solution.

One convenient solution is to allow $q_\psi$ to reuse the learned mean of the prior $\mu_\theta$. We observe that adapting the parameters with gradient descent is a good way to update them to a given training set $\mathbf{x}_i^{\text{tr}}, \mathbf{y}_i^{\text{tr}}$ and test set $\mathbf{x}_i^{\text{test}}, \mathbf{y}_i^{\text{test}}$, an design decision similar to one made by Fortunato et al. (2017). We propose an inference network of the form

$$q_\psi(\theta|\mathbf{x}_i^{\text{tr}}, \mathbf{y}_i^{\text{tr}}, \mathbf{x}_i^{\text{test}}, \mathbf{y}_i^{\text{test}}) = \mathcal{N}(\mu_\theta + \gamma_q \nabla_{\mu_\theta} \log p(\mathbf{y}_i^{\text{tr}}|\mathbf{x}_i^{\text{tr}}, \mu_\theta) + \gamma_q \nabla_{\mu_\theta} \log p(\mathbf{y}_i^{\text{test}}|\mathbf{x}_i^{\text{test}}, \mu_\theta); \mathbf{v}_q),$$

where $\mathbf{v}_q$ is a learned (diagonal) covariance, and the mean has an additional parameter beyond $\mu_\theta$, which is a "learning rate" vector $\gamma_q$ that is pointwise multiplied with the gradient. While this choice may at first seem arbitrary, there is a simple intuition: the inference network should produce a sample of $\theta$ that is close to the posterior $p(\theta|\mathbf{x}_i^{\text{tr}}, \mathbf{y}_i^{\text{tr}}, \mathbf{x}_i^{\text{test}}, \mathbf{y}_i^{\text{test}})$. A reasonable way to arrive at a value of $\theta$ close to this posterior is to adapt it to *both* the training set and test set.[1] Note that this is only done during meta-training. It remains to choose $q_\psi(\phi_i|\theta, \mathbf{x}_i^{\text{tr}}, \mathbf{y}_i^{\text{tr}}, \mathbf{x}_i^{\text{test}}, \mathbf{y}_i^{\text{test}})$, which can also be formulated as a conditional Gaussian with mean given by applying gradient descent.

Although this variational distribution is substantially more compact in terms of parameters than a separate neural network, it only provides estimates of the posterior during meta-training. At meta-test time, we must obtain the posterior $p(\phi_i|\mathbf{x}_i^{\text{tr}}, \mathbf{y}_i^{\text{tr}}, \mathbf{x}_i^{\text{test}})$, without access to $\mathbf{y}_i^{\text{test}}$. We can train a separate set of inference networks to perform this operation, potentially also using gradient descent within the inference network. However, these networks do not receive any gradient information during meta-training, and may not work well in practice. In the next section we propose an even simpler and more practical approach that uses only a single inference network during meta-training, and none during meta-testing.

### 5.3 Probabilistic model-agnostic meta-learning with hybrid inference

To formulate a simpler variational meta-learning procedure, we recall the probabilistic interpretation of MAML: as discussed by Grant et al. (2018), MAML can be interpreted as approximate inference for the posterior $p(\mathbf{y}_i^{\text{test}}|\mathbf{x}_i^{\text{tr}}, \mathbf{y}_i^{\text{tr}}, \mathbf{x}_i^{\text{test}})$ according to

$$p(\mathbf{y}_i^{\text{test}}|\mathbf{x}_i^{\text{tr}}, \mathbf{y}_i^{\text{tr}}, \mathbf{x}_i^{\text{test}}) = \int p(\mathbf{y}_i^{\text{test}}|\mathbf{x}_i^{\text{test}}, \phi_i) p(\phi_i|\mathbf{x}_i^{\text{tr}}, \mathbf{y}_i^{\text{tr}}, \theta) d\phi_i \approx p(\mathbf{y}_i^{\text{test}}|\mathbf{x}_i^{\text{test}}, \phi_i^\star), \tag{13}$$

where we use the maximum a posteriori (MAP) value $\phi_i^\star$. It can be shown that, for likelihoods that are Gaussian in $\phi_i$, gradient descent for a fixed number of iterations using

1 In practice, we can use multiple gradient steps for the mean, but we omit this for notational simplicity.

**Algorithm 4** Meta-training, differences from MAML in red

**Require:** $p(\mathcal{T})$: distribution over tasks

initialize $\Theta := \{\boldsymbol{\mu}_\theta, \boldsymbol{\sigma}_\theta^2, \mathbf{v}_q, \boldsymbol{\gamma}_p, \boldsymbol{\gamma}_q\}$
**while** not done **do**
  Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
  **for all** $\mathcal{T}_i$ **do**
    $\mathcal{D}^{\text{tr}}, \mathcal{D}^{\text{test}} = \mathcal{T}_i$
    Evaluate $\nabla_{\boldsymbol{\mu}_\theta} \mathcal{L}(\boldsymbol{\mu}_\theta, \mathcal{D}^{\text{test}})$
    Sample $\theta \sim q = \mathcal{N}(\boldsymbol{\mu}_\theta - \boldsymbol{\gamma}_q \nabla_{\boldsymbol{\mu}_\theta} \mathcal{L}(\boldsymbol{\mu}_\theta, \mathcal{D}^{\text{test}}), \mathbf{v}_q)$
    Evaluate $\nabla_\theta \mathcal{L}(\theta, \mathcal{D}^{\text{tr}})$
    Compute adapted parameters using gradient descent: $\phi_i = \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}^{\text{tr}})$
  Let $p(\theta|\mathcal{D}^{\text{tr}}) = \mathcal{N}(\boldsymbol{\mu}_\theta - \boldsymbol{\gamma}_p \nabla_{\boldsymbol{\mu}_\theta} \mathcal{L}(\boldsymbol{\mu}_\theta, \mathcal{D}^{\text{tr}}), \boldsymbol{\sigma}_\theta^2))$
  Compute $\nabla_\Theta \big( \sum_{\mathcal{T}_i} \mathcal{L}(\phi_i, \mathcal{D}^{\text{test}}) + D_{KL}(q(\theta|\mathcal{D}^{\text{test}}) \parallel p(\theta|\mathcal{D}^{\text{tr}})) \big)$
  Update $\Theta$ using Adam

**Algorithm 5** Meta-testing

**Require:** training data $\mathcal{D}_{\mathcal{T}}^{\text{tr}}$ for new task $\mathcal{T}$
**Require:** learned $\Theta$

Sample $\theta$ from the prior $p(\theta|\mathcal{D}^{\text{tr}})$
Evaluate $\nabla_\theta \mathcal{L}(\theta, \mathcal{D}^{\text{tr}})$
Compute adapted parameters with gradient descent: $\phi_i = \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}^{\text{tr}})$

$\mathbf{x}_i^{\text{tr}}, \mathbf{y}_i^{\text{tr}}$ corresponds exactly to maximum a posteriori inference under a Gaussian prior $p(\phi_i|\theta)$ (Santos, 1996). In the case of non-Gaussian likelihoods, the equivalence is only locally approximate, and the exact form of the prior $p(\phi_i|\theta)$ is intractable. However, in practice this implicit prior can actually be preferable to an explicit (and simple) Gaussian prior, since it incorporates the rich nonlinear structure of the neural network parameter manifold, and produces good performance in practice (Finn et al., 2017a; Grant et al., 2018). We can interpret this MAP approximation as inferring an approximate posterior on $\phi_i$ of the form , where $\phi_i^\star$ is obtained via gradient descent on the training set $\mathbf{x}_i^{\text{tr}}, \mathbf{y}_i^{\text{tr}}$ starting from $\theta$. Incorporating this approximate inference procedure transforms the graphical model in Figure 11 (a) into the one in Figure 11 (b), where there is now a factor over $p(\phi_i|\mathbf{x}_i^{\text{tr}}, \mathbf{y}_i^{\text{tr}}, \theta)$. While this is a crude approximation to the likelihood, it provides us with an empirically effective and simple tool that greatly simplifies the variational inference procedure described in the previous section, in the case where we aim to model a distribution over the global parameters $p(\theta)$. After using gradient descent to estimate $p(\phi_i \mid \mathbf{x}_i^{\text{tr}}, \mathbf{y}_i^{\text{tr}}, \theta)$, the graphical model is transformed into the model shown in the center of Figure 11. Note that, in this new graphical model, the global parameters $\theta$ are independent of $\mathbf{x}^{\text{tr}}$ and $\mathbf{y}^{\text{tr}}$ and are independent of $\mathbf{x}^{\text{test}}$ when $\mathbf{y}^{\text{test}}$ is not observed. Thus, we can now write down a variational lower bound for the logarithm of the *approx-*

*imate* likelihood on the second line, which is given by

$$\log p(\mathbf{y}_i^{\text{test}}|\mathbf{x}_i^{\text{test}}, \mathbf{x}_i^{\text{tr}}, \mathbf{y}_i^{\text{tr}}) \geqslant \mathbb{E}_{\theta\sim q_\psi}\left[\log p(\mathbf{y}_i^{\text{test}}|\mathbf{x}_i^{\text{test}}, \phi_i^\star) + \log p(\theta)\right] + \mathcal{H}(q_\psi(\theta|\mathbf{x}_i^{\text{test}}, \mathbf{y}_i^{\text{test}})).$$

In this bound, we essentially perform approximate inference via MAP on $\phi_i$ to obtain $p(\phi_i|\mathbf{x}_i^{\text{tr}}, \mathbf{y}_i^{\text{tr}}, \theta)$, and use the variational distribution for $\theta$ only. Note that $q_\psi(\theta|\mathbf{x}_i^{\text{test}}, \mathbf{y}_i^{\text{test}})$ is not conditioned on the training set $\mathbf{x}_i^{\text{tr}}, \mathbf{y}_i^{\text{tr}}$ since $\theta$ is independent of it in the transformed graphical model. Analogously to the previous section, the inference network is given by

$$q_\psi(\theta|\mathbf{x}_i^{\text{test}}, \mathbf{y}_i^{\text{test}}) = \mathcal{N}(\boldsymbol{\mu}_\theta + \boldsymbol{\gamma}_q \nabla \log p(\mathbf{y}_i^{\text{test}}|\mathbf{x}_i^{\text{test}}, \boldsymbol{\mu}_\theta); \mathbf{v}_q).$$

To evaluate the variational lower bound during training, we can use the following procedure: first, we evaluate the mean by starting from $\boldsymbol{\mu}_\theta$ and taking one (or more) gradient steps on $\log p(\mathbf{y}_i^{\text{test}}|\mathbf{x}_i^{\text{test}}, \theta_{\text{current}})$, where $\theta_{\text{current}}$ starts at $\boldsymbol{\mu}_\theta$. We then add noise with variance $\mathbf{v}_q$, which is made differentiable via the reparameterization trick (D. P. Kingma and Welling, 2013). We then take additional gradient steps on the training likelihood $\log p(\mathbf{y}_i^{\text{tr}}|\mathbf{x}_i^{\text{tr}}, \theta_{\text{current}})$. This accounts for the MAP inference procedure on $\phi_i$. Training of $\boldsymbol{\mu}_\theta$, $\boldsymbol{\sigma}_\theta^2$, and $\mathbf{v}_q$ is performed by backpropagating gradients through this entire procedure with respect to the variational lower bound, which includes a term for the likelihood $\log p(\mathbf{y}_i^{\text{test}}|\mathbf{x}_i^{\text{test}}, \mathbf{x}^{\text{tr}}, \mathbf{y}^{\text{tr}}, \phi_i^\star)$ and the KL-divergence between the sample $\theta \sim q_\psi$ and the prior $p(\theta)$. This meta-training procedure is detailed in Algorithm 4.

At meta-test time, the inference procedure is much simpler. The test labels are not available, so we simply sample $\theta \sim p(\theta)$ and perform MAP inference on $\phi_i$ using the training set, which corresponds to gradient steps on $\log p(\mathbf{y}_i^{\text{tr}}|\mathbf{x}_i^{\text{tr}}, \theta_{\text{current}})$, where $\theta_{\text{current}}$ starts at the sampled $\theta$. This meta-testing procedure is detailed in Algorithm 5.

### 5.4 ADDING ADDITIONAL DEPENDENCIES

In the transformed graphical model, the training data $\mathbf{x}_i^{\text{tr}}, \mathbf{y}_i^{\text{tr}}$ and the prior $\theta$ are conditionally independent. However, since we have only a crude approximation to $p(\phi_i \mid \mathbf{x}_i^{\text{tr}}, \mathbf{y}_i^{\text{tr}}, \theta)$, this independence often doesn't actually hold. We can allow the model to compensate for this approximation by additionally conditioning the learned prior $p(\theta)$ on the training data. In this case, the learned "prior" has the form $p(\theta_i|\mathbf{x}_i^{\text{tr}}, \mathbf{y}_i^{\text{tr}})$, where $\theta_i$ is now task-specific, but with global parameters $\boldsymbol{\mu}_\theta$ and $\boldsymbol{\sigma}_\theta^2$. We thus obtain the modified graphical model in Figure 11 (c). Similarly to the inference network $q_\psi$, we parameterize the learned prior as follows:

$$p(\theta_i|\mathbf{x}_i^{\text{tr}}, \mathbf{y}_i^{\text{tr}}) = \mathcal{N}(\boldsymbol{\mu}_\theta + \boldsymbol{\gamma}_p \nabla \log p(\mathbf{y}_i^{\text{tr}}|\mathbf{x}_i^{\text{tr}}, \boldsymbol{\mu}_\theta); \boldsymbol{\sigma}_\theta^2).$$

With this new form for distribution over $\theta$, the variational training objective uses the likelihood term $\log p(\theta_i|\mathbf{x}_i^{tr}, \mathbf{y}_i^{tr})$ in place of $\log p(\theta)$, but otherwise is left unchanged. At test time, we sample from $\theta \sim p(\theta|\mathbf{x}_i^{tr}, \mathbf{y}_i^{tr})$ by first taking gradient steps on $\log p(\mathbf{y}_i^{tr}|\mathbf{x}_i^{tr}, \theta_{current})$, where $\theta_{current}$ is initialized at $\boldsymbol{\mu}_\theta$, and then adding noise with variance $\boldsymbol{\sigma}_\theta^2$. Then, we proceed as before, performing MAP inference on $\phi_i$ by taking additional gradient steps on $\log p(\mathbf{y}_i^{tr}|\mathbf{x}_i^{tr}, \theta_{current})$ initialized at the sample $\theta$. In our experiments, we find that this more expressive distribution often leads to better performance.

## 5.5 RELATED WORK

Hierarchical Bayesian models are a long-standing approach for few-shot learning that naturally allow for the ability to reason about uncertainty over functions (Tenenbaum, 1999; Fei-Fei et al., 2003; Lawrence and Platt, 2004; K. Yu et al., 2005; J. Gao et al., 2008; Daumé III, 2009; Wan et al., 2012). While these approaches have been demonstrated on simple few-shot image classification datasets (Lake et al., 2015), they have yet to scale to the more complex problems, such as the experiments in this chapter. A number of works have approached the problem of few-shot learning from a meta-learning perspective (Schmidhuber, 1987; Hochreiter et al., 2001), including black-box (Santoro et al., 2016; Duan et al., 2016b; Wang and Hebert, 2016) and optimization-based approaches (Ravi and Larochelle, 2017; Finn et al., 2017a). While these approaches scale to large-scale image datasets (Vinyals et al., 2016) and visual reinforcement learning problems (Mishra et al., 2018), they typically lack the ability to reason about uncertainty.

This work is most related to methods that combine deep networks and probabilistic methods for few-shot learning (H. Edwards and Storkey, 2017; Grant et al., 2018; Lacoste et al., 2017). One approach that considers hierarchical Bayesian models for few-shot learning is the neural statistician (H. Edwards and Storkey, 2017), which uses an explicit task variable to model task distributions. Our method is fully model agnostic, and directly samples model weights for each task for any network architecture. Our experiments show that our approach improves on MAML (Chapter 4), which outperforms the model by H. Edwards and Storkey (2017). Other work that considers model uncertainty in the few-shot learning setting is the LLAMA method (Grant et al., 2018), which also builds on the MAML algorithm. LLAMA makes use of a local Laplace approximation for modeling the task parameters (post-update parameters), which introduces the need to approximate a high dimensional covariance matrix. We instead propose a method that approximately infers the pre-update parameters, which we make tractable through a choice of approximate posterior parameterized by gradient operations.

*Bayesian neural networks* (MacKay, 1992; G. E. Hinton and Van Camp, 1993; Neal, 1995; Barber and Bishop, 1998) have been studied extensively as a way to incorporate uncertainty into deep networks. Although exact inference in Bayesian neural networks is impractical, approximations based on backpropagation and sampling (Graves, 2011; Rezende et al., 2014; Hoffman et al., 2013; Blundell et al., 2015) have been effective in incorporating uncertainty into the weights of generic networks. Our approach differs from these methods in that we explicitly train a hierarchical Bayesian model over weights, where a posterior task-specific parameter distribution is inferred at meta-test time conditioned on a learned weight prior and a (few-shot) training set, while conventional Bayesian neural networks directly learn only the posterior weight distribution for a single task. Our method draws on amortized variational inference methods (D. P. Kingma and Welling, 2013; Johnson et al., 2016; Shu et al., 2018) to make this possible, but the key modification is that the model and inference networks share the same parameters. The resulting method corresponds structurally to a Bayesian version of model-agnostic meta-learning.

## 5.6 EXPERIMENTS

The goal of our experimental evaluation is to answer the following questions: (1) can our approach enable sampling from the distribution over potential functions underlying the training data?, (2) does our approach improve upon the MAML algorithm when there is ambiguity over the class of functions?, and (3) can our approach scale to deep convolutional networks? We study two illustrative toy examples and a realistic ambiguous few-shot image classification problem. For the both experimental domains, we compare MAML to our probabilistic approach. We will refer to our version of MAML as a PLATIPUS (Probabilistic LATent model for Incorporating Priors and Uncertainty in few-Shot learning), due to its unusual combination of two approximate inference methods: amortized inference and MAP. Both PLATIPUS and MAML use the same neural network architecture and the same number of inner gradient steps. We additionally provide a comparison on the MiniImagenet benchmark and specify the hyperparameters in the supplementary appendix.

ILLUSTRATIVE 5-SHOT REGRESSION. In this 1D regression problem, different tasks correspond to different underlying functions. Half of the functions are sinusoids, and half are lines, such that the task distribution is clearly multimodal. The sinusoids have amplitude and phase uniformly sampled from the range $[0.1, 5]$ and $[0, \pi]$, and the lines

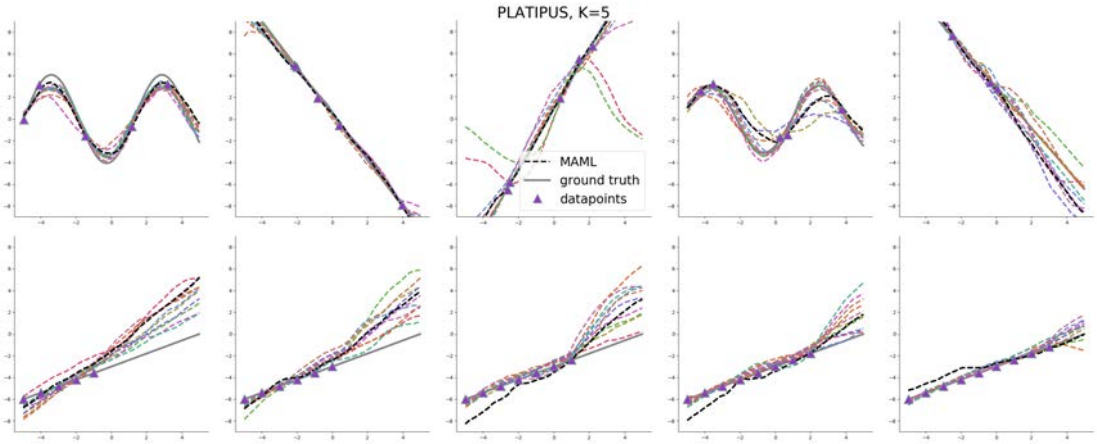


Figure 12: Samples from PLATIPUS trained for 5-shot regression, shown as colored dotted lines. The tasks consist of regressing to sinusoid and linear functions, shown in gray. MAML, shown in black, is a deterministic procedure and hence learns a single function, rather than reasoning about the distribution over potential functions. As seen on the bottom row, even though PLATIPUS is trained for 5-shot regression, it can effectively reason over its uncertainty when provided variable numbers of datapoints at test time (left vs. right).

have the slope and intercept sampled in the range $[-3, 3]$. The input domain is uniform on $[-5, 5]$, and Gaussian noise with a standard deviation of 0.3 is added to the labels. We trained both MAML and PLATIPUS for 5-shot regression. In Figure 12, we show the qualitative performance of both methods, where the ground truth underlying function is shown in gray and the datapoints in $\mathcal{D}^{tr}$ are shown as purple triangles. We show the function $f_{\phi_i}$ learned by MAML in black. For PLATIPUS, we sample 10 sets of parameters from $p(\phi_i|\theta)$ and plot the resulting functions in different colors. In the top row, we can see that PLATIPUS allows the model to effectively reason over the set of functions underlying the provided datapoints, with increased variance in parts of the function where there is more uncertainty. Further, we see that PLATIPUS is able to capture the multimodal structure, as the curves are all linear or sinusoidal.

A particularly useful application of uncertainty estimates in few-shot learning is estimating when more data would be helpful. In particular, seeing a large variance in a particular part of the input space suggests that more data would be helpful for learning the function in that part of the input space. On the bottom of Figure 12, we show the results for a single task at meta-test time with increasing numbers of training datapoints. Even though the model was only trained on training set sizes of 5 datapoints, we observe

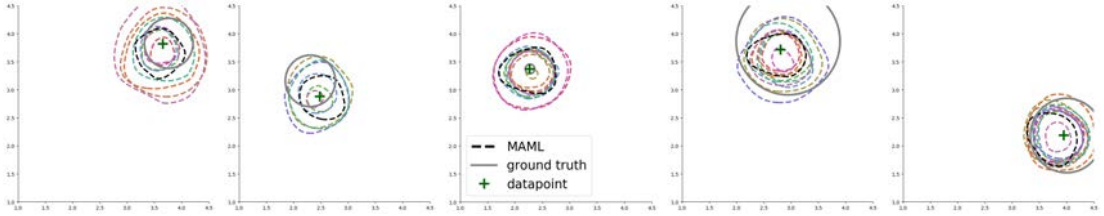


Figure 13: Samples from PLATIPUS for 1-shot classification, shown as colored dotted lines. The 2D classification tasks all involve circular decision boundaries of varying size and center, shown in gray. MAML, shown in black, is a deterministic procedure and hence learns a single function, rather than reasoning about the distribution over potential functions.

that PLATIPUS is able to effectively reduce its uncertainty as more and more datapoints are available. This suggests that the uncertainty provided by PLATIPUS can be used for approximately gauging when more data would be helpful for learning a new task.

ILLUSTRATIVE 1-SHOT 2D CLASSIFICATION. Next, we study a simple binary classification task, where there is a particularly large amount of ambiguity surrounding the underlying function: learning to learn from a single positive example. Here, the tasks consist of classifying datapoints in 2D within the range $[0, 5]$ with a circular decision boundary, where points inside the decision boundary are positive and points outside are negative. Different tasks correspond to different locations and radii of the decision boundary, sampled at uniformly at random from the ranges $[1.0, 4.0]$ and $[0.1, 2.0]$ respectively. Following Grant et al. (2017), we train both MAML and PLATIPUS with $\mathcal{D}^{\text{tr}}$ consisting of a single positive example and $\mathcal{D}^{\text{test}}$ consisting of both positive and negative examples. We plot the results using the same scheme as before, except that we plot the decision boundary (rather than the regression function) and visualize the single positive datapoint with a green plus. As seen in Figure 13, we see that PLATIPUS captures a broad distribution over possible decision boundaries, all of which are roughly circular. MAML provides a single decision boundary of average size.

AMBIGUOUS IMAGE CLASSIFICATION. The ambiguity illustrated in the previous settings is common in real world tasks where images can share multiple attributes. We study an ambiguous extension to the celebA attribute classification task. Our meta-training dataset is formed by sampling two attributes at random to form a positive class and taking the same number of random examples without either attribute to from the

| Ambiguous celebA (5-shot) | | |
|---|---|---|
| | Accuracy | Coverage (max=3) |
| MAML | 69.26 ± 2.18% | 1.00 ± 0.0 |
| MAML + noise | 54.73 ± 0.8 % | 2.60 ± 0.12 |
| **PLATIPUS (ours)** | **69.97 ± 1.32** % | **2.62 ± 0.11** |

Table 3: Our method covers significantly more tasks than MAML, with comparable accuracy. MAML + noise is a method that simply adds noise to the gradient, but does not explicitly perform variational inference. This still improves coverage, but results in a large drop in accuracy.

negative classes. To evaluate the ability to capture multiple decision boundaries while simultaneously obtaining good performance, we evaluate our method as follows: We sample from a test set of three attributes and a corresponding set of images with those attributes. Since the tasks involve classifying images that have two attributes, this task is ambiguous, and there are three possible combinations of two attributes that explain the training set. We sample models from our prior as described in Section 5.4 and assign each of the sampled models to one of the three possible tasks based on its log-likelihood. If each of the three possible tasks is assigned a nonzero number of samples, this means that the model effectively covers all three possible modes that explain the ambiguous training set. We can measure coverage and accuracy from this protocol. The coverage score indicates the average number of tasks (between 1 and 3) that receive at least one sample for each ambiguous training set, and the accuracy score is the average number of correct classifications on these tasks (according to the sampled models assigned to them). A highly random method will achieve good coverage but poor accuracy, while a deterministic method will have a coverage of 1.

Our results are summarized in Table 3 and Fig. 14. The accuracy of our method is comparable to standard, deterministic MAML. However, the deterministic algorithm only ever captures one mode for each ambiguous task, where the maximum is three. Our method on average captures between two and three modes. The qualitative analysis in Figure 14 illustrates[2] an example ambiguous training set, example images for the three possible two-attribute pairs that can correspond to this training set, and the classifications made by different sampled classifiers trained on the ambiguous training set.

2 Additional qualitative results can be found at https://sites.google.com/view/probabilistic-maml/

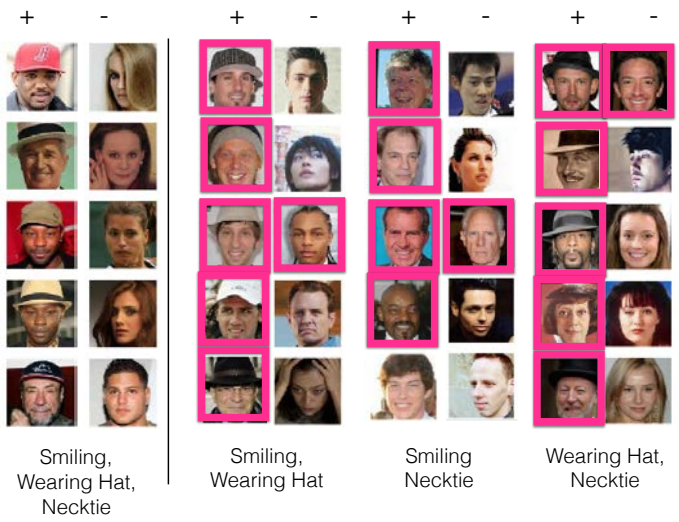


Figure 14: Sampled classifiers for an ambiguous meta-test task. In the meta-test training set (left), PLATIPUS observes five positives that share three attributes, and five negatives. A classifier that uses *any* two attributes can correctly classify the training set. On the right, we show each of the possible two-attribute tasks that this training set can correspond to, and illustrate the labels (positive indicated by red border) assigned by the best sample for that task. We see that the different samples are able to make reasonable predictions with no hats (2nd column) or pay attention to them (1st and 3rd column), and can effectively capture the three possible explanations.

Note that the different samples each pay attention to different attributes, indicating that PLATIPUS is effective at capturing the different modes of the task.

### 5.6.1 *Discussion and Future Work*

We introduced an algorithm for few-shot meta-learning that enables simple and effective sampling of models for new tasks at meta-test time. Our algorithm, PLATIPUS, adapts to new tasks by running gradient descent with injected noise. During meta-training, the model parameters are optimized with respect to a variational lower bound on the likelihood for the meta-training tasks, so as to enable this simple adaptation procedure to produce approximate samples from the model posterior when conditioned on a few-shot training set. This approach has a number of benefits. The adaptation procedure is exceedingly simple, and the method can be applied to any standard model architec-

ture. The algorithm introduces a modest number of additional parameters: besides the initial model weights, we must learn a variance on each parameter for the inference network and prior, and the number of parameters scales only linearly with the number of model weights. Our experimental results show that our method can be used to effectively sample diverse solutions to both regression and classification tasks at meta-test time, including for task families that have multi-modal task distributions.

Although our approach is simple and broadly applicable, it has a number of potential limitations that could be addressed in future work. First, the current form of the method provides a relatively impoverished estimator of posterior variance, which might be less effective at gauging uncertainty in settings where different tasks have very different degrees of ambiguity. In these cases, finding a way to make the variance dependent on the few-shot training set might produce better results, and investigating how to do this without adding a large number of additional parameters would be an interesting direction for future work. Another exciting direction for future research would be to study how our approach could be applied in settings where ambiguity and uncertainty can directly guide data acquisition, so as to devise better few-shot active learning and reinforcement learning algorithms.

# Part III

# EXTENSIONS AND APPLICATIONS

# 6

# ONLINE ADAPTIVE CONTROL

## 6.1 META-LEARNING FOR ADAPTIVE CONTROL

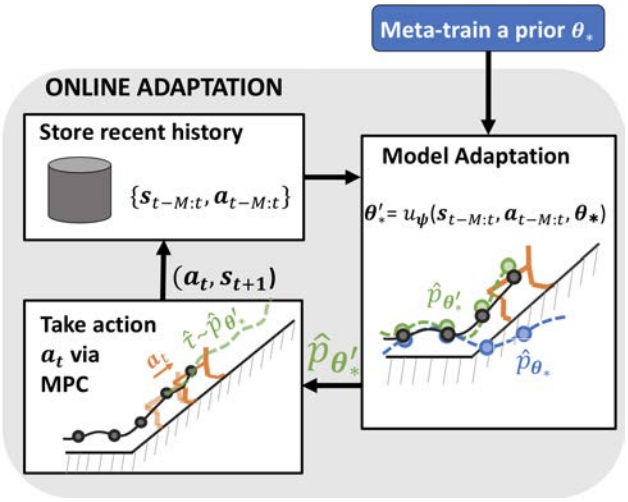


Figure 15: A meta-trained cheetah agent, using recent experience to adapt its model online to succeed in a new setting.

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

propose a meta-learning approach for learning online adaption.

To enable efficient learning for real-world applications, we specifically study the online adaptation problem in the context of model-based reinforcement learning (M. P. Deisenroth et al., 2013). In this setting, data for updating the model is readily available. But more crucially, the meta-training process is much more sample efficient than meta-training an adaptive policy with model-free RL (Duan et al., 2016b; Wang and Hebert, 2016).

Learning to adapt a model online alleviates a central challenge of model-based reinforcement learning: acquiring a global model that is accurate throughout the entire state space. If the model can adapt online, it need not be perfect everywhere a priori. This property has previously been exploited by adaptive control methods (Åström and Wittenmark, 2013; Sastry and Isidori, 1989), but scaling such methods to complex tasks and nonlinear models such as deep neural networks is exceptionally difficult, since such models typically require large amounts of data and many gradient steps to learn effectively. By training a neural network model to require only a small amount of experience to adapt, we can enable effective online adaptation in complex environments while putting less pressure on needing a perfect global model.

The primary contribution of this chapter is an approach that combines meta-learning with model-based RL to achieve fast online adaptation. As shown in Figure 15, our approach efficiently trains a global model that can use its most recent experiences to quickly adapt. We introduce two instantiations of this approach: recurrence based adaptive control (RBAC), where a recurrent model is trained to learn its own update rule (i.e., through its internal gating structure), and gradient based adaptive control (GBAC), which extends the model-agnostic meta-learning algorithm (MAML) and optimizes for initial model parameters such that a gradient descent update-rule on recent data leads to fast and effective adaptation. We evaluate our approach on stochastic and simulated continuous control tasks with complex contact dynamics. In our experiments, we show a half-cheetah robot adapting after the failure of different joints, navigating terrains with different slopes, and walking on floating platforms with varying buoyancy. We also show a quadrupedal "ant" adapting to failure of different legs, and a 7-DoF arm adapting online to random force perturbations. Our method can adapt rapidly on these tasks, and it attains substantial improvement over prior approaches, including standard model-based methods, online model-adaptive methods, and model-free methods trained with similar amounts of data. In all experiments, meta-training across multiple tasks is efficient, using only the equivalent of $1.5 - 3$ hours of real-world experience, roughly $10\times$ less than what model-free methods require to learn even a single task.

### 6.1.1 *Preliminaries: Model-Based RL*

We consider a Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma, \rho_0, T)$. Here, $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ is the state transition distribution, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a bounded reward function, $\rho_0 : \mathcal{S} \to \mathbb{R}_+$ is the initial state distribution, $\gamma$ is the discount factor, and $T$ is the horizon. A trajectory is denoted by $\tau(i, j) := (\mathbf{s}_i, \mathbf{a}_i, ..., \mathbf{s}_j, \mathbf{a}_j, \mathbf{s}_{j+1})$, and the sum of expected rewards from a trajectory is the return. RL aims to find a policy $\pi : \mathcal{S} \to \mathcal{A}$ that prescribes the optimal action to take from each state in order to maximize the expected return.

Model-based RL tries to solve this problem by learning the transition distribution $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$, otherwise referred to as the dynamics model. This can be done using a function approximator $\hat{p}_\theta(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ to approximate the dynamics, where the weights $\theta$ are optimized to maximize the log-likelihood of the distribution:

$$\max_{\theta \in \mathbb{R}^w} \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \in \mathcal{D}} \log \hat{p}_\theta(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \tag{14}$$

Here, $\mathcal{D}$ is a training dataset containing state transitions that the agent has experienced. The choice of model, its training procedure, and the method of using this model to perform action selection can all be done in many ways; we detail our approach in Sec. 6.1.2.

Learning a model in this supervised learning setup makes more efficient use of the data than the counterpart model-free methods, since we get dense training signals and we are able to use all data (even off-policy data) to make forward progress in training. A standard practice for addressing the data distribution mismatch between train time and test time is to aggregate the trajectories resulting from executing the policy, and retrain the dynamics model with the updated dataset.

### 6.1.2 *Overview*

In our approach, we first have a meta-training phase (Algorithm 6), during which we learn parameters $\theta_*$ of a model that is specifically optimized for the ability to adapt online. We refer to the mechanism that performs this model adaptation as the update rule $u_\psi$ parameterized by $\psi$. It is some function that takes in an agent's recent experience $\tau(t - M, t - 1)$ and parameters $\theta$ as inputs, and outputs new adapted parameters $\phi$. This rule can be prescribed or learned, as discussed in Sections 6.1.5 and 6.1.6. Unlike standard learning approaches that seek optimal parameters $\theta_*$ to achieve high task per-

formance, we optimize for $\theta_*$ such that application of recent experience and the update rule $u_\psi$ on this $\theta_*$ leads to high performance across various tasks.

Standard meta-learning formulations require the learned model to adapt after seeing K data points from some new "task." However, our notion of task is slightly more fluid; every time step of the trajectory can be considered a different "task." Since changes in system dynamics, terrain details, or other environmental changes can occur at any time, we consider (at each time step) the task of adapting to the M past time steps. Allowing the past M observations to guide action selection enables us to extend MDP's to partially observable MDP's (POMDP's). This is particularly important because we can consider all real-world tasks as POMDP's, since it is impossible to have state information for every environmental change that could possibly happen.

The goal in meta-learning is to learn a $\theta_\mathcal{E}$ and update rule $u_\psi$ such that $\hat{p}_{\phi_*}$, where $\phi_\mathcal{E} = u_\psi(\tau(t-M, t-1), \theta_\mathcal{E})$, is a more accurate predictor for the current dynamics in environment $\mathcal{E}$. We assume a distribution of environments $\rho(\mathcal{E})$ share some common structure, such as the same observation and action space, but display different dynamics. We denote trajectories from each environment $\mathcal{E}$ by $\tau_\mathcal{E}(i,j) \triangleq (\mathbf{s}_i^\mathcal{E}, \mathbf{a}_i^\mathcal{E}, ..., \mathbf{s}_j^\mathcal{E}, \mathbf{a}_j^\mathcal{E}, \mathbf{s}_{j+1}^\mathcal{E})$, and we formalize the meta-learning problem as:

$$\min_{\theta,\psi} \; \mathbb{E}_{\substack{\mathcal{E}\sim\rho(\mathcal{E}) \\ t\sim\mathcal{U}([T-K-1])}} \Big[\mathcal{L}(\tau_\mathcal{E}(t, t+K), \phi_\mathcal{E})\Big] \quad \text{s.t.:} \quad \phi_\mathcal{E} = u_\psi(\tau_\mathcal{E}(t-M, t-1), \theta) \tag{15}$$

In the optimization outlined above, note that past data is used to adapt $\theta$ into $\phi$, but since we want to know how good these new parameters are at adapting, the loss is evaluated on the *future* transitions. To optimize our model $\hat{p}_\theta(\mathbf{s}'|\mathbf{s}, \mathbf{a})$ of the dynamics $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$, we can consider, without loss of generality, optimizing a loss corresponding to the negative log likelihood across tasks, given by

$$\mathcal{L}(\tau_\mathcal{E}(t, t+K), \phi_\mathcal{E}) \triangleq -\frac{1}{K}\sum_{k=t}^{t+K} \log \hat{p}_{\phi_\mathcal{E}}(\mathbf{s}_{k+1}^\mathcal{E}|\mathbf{s}_k^\mathcal{E}, \mathbf{a}_k^\mathcal{E}). \tag{16}$$

### 6.1.3 *Online Model Adaptive Control*

We now discuss how to perform adaptive control at test time (Algorithm 7), given a meta-learned model $\theta_*$ . At every time step t, we consult an update rule $u_\psi$ that uses the agent's recent experience $\tau_\mathcal{E}(t-M, t-1)$ to adapt $\theta_*$ into $\phi_\mathcal{E}$. The adapted dynamics model $\hat{p}_{\phi_\mathcal{E}}$ can be viewed as a local model, since it is adapted to recently observed data and trained to perform well in this region.

**Algorithm 6** Learning a prior for online adaptation

**Require:** $\rho_{\mathcal{E}}$ distribution over tasks
**Require:** learning rate $\beta \in \mathbb{R}^+$,
**Require:** sampling frequency $n_S \in \mathbb{Z}^+$
**Require:** dataset $\mathcal{D}$

Randomly initialize $\theta$
**for** $i = 1, \ldots$ **do**
  **if** $i \bmod n_S = 0$ **then**
    Sample $\mathcal{E} \sim \rho(\mathcal{E})$
    Collect $\tau_{\mathcal{E}}$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{\tau_{\mathcal{E}}\}$
  **for** $j = 1 \ldots N$ **do**
    $\tau_{\mathcal{E}}(t-M, t-1), \tau_{\mathcal{E}}(t, t+K) \sim \mathcal{D}$
    $\phi_{\mathcal{E}} \leftarrow u(\tau_{\mathcal{E}}(t-M, t-1), \theta)$
    $\mathcal{L}_j \leftarrow \mathcal{L}(\tau_{\mathcal{E}}(t, t+K), \phi_{\mathcal{E}})$
  $\theta \leftarrow \theta - \beta \nabla_\theta \frac{1}{N} \sum_{j=1}^{N} \mathcal{L}_j$
  $\psi \leftarrow \psi - \eta \nabla_\psi \frac{1}{N} \sum_{j=1}^{N} \mathcal{L}_j$
Return $\theta_*$ and $u_\psi$

**Algorithm 7** Online adaptive control at each time step

**Require:** Parameters $\theta$ of a prior
**Require:** update rule $u$
**Require:** experience $\tau(t-M, t-1)$
**Require:** number of samples $n_A$
**Require:** horizon $H$

$\phi \leftarrow u_\psi(\tau(t-M, t-1), \theta)$
$\mathcal{A} \leftarrow \emptyset$
**for** $i = 1, \ldots, n_A$ **do**
  Sample sequence of actions $A_i$
  $\mathcal{A} \leftarrow \mathcal{A} \cup \{A^i\}$
  $\hat{\mathbf{s}}^i \leftarrow \mathbf{s}^i$
  $R^i \leftarrow 0$
**for** $A^i$ in $\mathcal{A}$ **do**
  **for** $a^i$ in $A^i$ **do**
    $R^i \leftarrow R^i + r(\hat{\mathbf{s}}^i, \mathbf{a}^i)$
    $\hat{\mathbf{s}}^i \sim \hat{p}_\phi(\cdot|\hat{\mathbf{s}}^i, \mathbf{a}^i)$
$i^* \leftarrow \arg\max_i R^i$
Return $A^{i^*}$

We then employ a computationally tractable model-based controller to perform action selection. We generate $n_A$ random candidate action sequences $\{A_1, ..., A_{n_A}\}$, where each sequence $A^i = (\mathbf{a}_0^i, ..., \mathbf{a}_{H-1}^i)$ is of length $H$. Then, we use the updated model $\hat{p}_{\theta_*}$ and the specified reward function $r(\mathbf{s}, \mathbf{a})$ to perform action selection as follows:

$$A^{i^*} = \arg\max_{A^i} \sum_{h=0}^{H} r(\hat{\mathbf{s}}_{t+h}^i, \mathbf{a}_h^i) \quad \text{s.t.: } \hat{\mathbf{s}}_t^i = \mathbf{s}_t \; ; \; \hat{\mathbf{s}}_{t+h+1}^i \sim \hat{p}_{\phi_\mathcal{E}}(\cdot|\hat{\mathbf{s}}_{t+h}^i, \mathbf{a}_h^i) \tag{17}$$

This zeroth-order optimization procedure is simple and has been shown to be effective for even non-linear and highly non-convex dynamics models (Nagabandi et al., 2017a). However, a variety of alternatives such as iterative gradient-free algorithms (Blossom, 2006) or gradient-based methods (W. Li and Todorov, 2004) can also be used. Finally, rather than executing the entire sequence of actions, we execute only the first action from the selected sequence and then repeat this planning process at the next time step. This use of MPC allows us to better deal with the imperfections of our adapted dynamics model, since we replan at each time step using updated state information. It also provides us further benefits in the setting of online adaptation, because the model $\hat{p}_{\phi_\mathcal{E}}$ will also improve at the next time step.

**Algorithm 8** Meta-Learning for Adaptive Control

**Require:** $\rho(\cdot)$ distribution over tasks, $u_\psi$, $H$, $r$, $n_A$, metaTrain(), actionSelection()

```
θ* ← metaTrain(ρ)
for each rollout do
    experience ← ∅
    for each step in rollout do
        φ* ← u_ψ( θ*, experience)
        a ← actionSelection(φ*, r, H, n_A)
        Execute a, add to experience
```

### 6.1.4 *General Algorithm of Meta-Learning for Adaptive Control*

We now combine the meta-trained prior and continuous adaptation into one cohesive framework (Algorithm 8). We first perform meta-training (Section 6.1.2) to optimize the prior model parameters $\theta_*$ and update rule $u_\psi$. Next, we perform the following operations at each time step: we use the update rule $u_\psi$ and recent experience to adapt model parameters, we select (and exe-

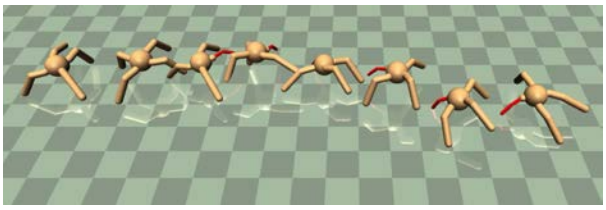

Figure 16: A meta-trained ant agent, adapting online to the unexpected crippling of its leg.

cute) actions using MPC with the updated model (Section 6.1.3), we append the observed transition to our dataset, and repeat. In the next two sections, we discuss two possible instantiations of the update rule and model prior.

### 6.1.5 *Recurrence-Based Adaptive Control (RBAC)*

One instantiation of our proposed approach of meta-learning adaptive control is to train a recurrent model to learn its own update rule for deciding how to use recent data to adapt online. This approach is an extension to previously proposed recurrence-based meta-learning methods (Santoro et al., 2016; Duan et al., 2016b). In the recurrent meta-learning scheme, an RNN ingests a dataset as well as a query point, and outputs a prediction for that query point. Following the notation in the previous section, the vectors $\theta$ and $\phi$ correspond to the hidden state of the RNN at the beginning and end of the forward pass, respectively, and the update rule $u_\psi$ is determined by the RNN weights, which determine how recent experience $\tau(t-M, t-1)$ influences the hidden state at time $t$. Typically, the prior hidden state $\theta$ is set to a constant (e.g., zero), so RBAC learns only the update rule $u_\psi$. For the dynamics model in our RBAC experiments, we use a Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) model with 2 hidden layers of 256 units each and tanh activations, and we set $M$ to be the total number of time steps in the episode so far ($M = t$).

### 6.1.6 *Gradient-Based Adaptive Control (GBAC)*

While RBAC can learn complex update rules through its recurrent weights (and is good for tasks of a sequential nature), it has limited capacity to handle out-of-distribution tasks and perturbations that differ from those seen during meta-training, as we will show in Section 6.1.8. To address this shortcoming, we also propose gradient-based adaptive control (GBAC), which builds on model-agnostic meta-learning (MAML) (Finn et al., 2017a). Unlike MAML, our method considers learning to adapt *online*. The update rule is explicitly prescribed to be gradient ascent on the likelihood of past experiences:

$$u_\psi(\tau(t-M, t-1), \theta) = \theta + \alpha \nabla_\theta \left( \frac{1}{M} \sum_{m=t-1}^{t-M} \log \hat{p}_\theta(\mathbf{s}_{m+1} | \mathbf{s}_m, \mathbf{a}_m) \right) \tag{18}$$

The only parameter of this learning rule is the rate $\alpha$, which may either be learned or selected. In our experiments, we choose $\alpha$ manually, which means that $u$ is fixed in

advance and not learned, and only the prior parameters $\theta$ are meta-learned to ensure fast adaptation. Thus, GBAC learns an initialization, $\theta^*$. Prior work has shown that this is sufficient to encode arbitrary adaptation procedures from a single gradient update if the dynamics model is sufficiently deep (Finn and Levine, 2018). Furthermore, the update rule in Eq. 18 is shown as a single update for simplicity, multiple gradient updates can be performed, which usually works better in practice. The dynamics model used in our GBAC experiments is a feedforward deep neural network with 3 hidden layers of 512 units each and ReLU activations.

The inductive bias provided by gradient descent in this update allows GBAC to extrapolate better to out-of-distribution dynamics, as we show in our results. Note that, in principle, we could combine both RBAC and GBAC using a recurrent dynamics model with a gradient-based update rule. Such a method would likely inherit the benefits of both approaches. However, in the interest of a more systematic evaluation, we study the methods separately and leave their combination for future work.

### 6.1.7 *Related Work*

Advances in learning control policies have shown success on numerous complex and high dimensional tasks (Schulman et al., 2015; T. P. Lillicrap et al., 2015; Mnih et al., 2015; Levine et al., 2016a; Silver et al., 2016). While reinforcement learning algorithms provide a framework for learning new tasks, they primarily focus on mastery of individual skills, rather than generalizing and quickly adapting to new scenarios. Furthermore, model-free approaches (Peters and Schaal, 2008) require large amounts of system interaction to learn successful control policies, which often makes them impractical for real-world systems. In contrast, model-based methods attain superior sample efficiency by first learning a model of system dynamics, and then using that model to optimize a policy (M. P. Deisenroth et al., 2013; Levine et al., 2016a; Nagabandi et al., 2017b).

**Model Learning:** Many model-based approaches aim to learn a single global model, using function approximators such as Gaussian processes (Ko and Fox, 2009; M. Deisenroth and Rasmussen, 2011) and neural networks (Lenz et al., 2015; Schaefer et al., 2007; Finn and Levine, 2017). A key challenge with these approaches is the difficulty of learning a global model that is accurate for the entire state space. Our approach alleviates the need to learn a single global model by allowing the model to be adapted automatically to different scenarios based on recent observations.

**Addressing Model Inaccuracies:** Although this use of replanning allows for compensation for slight model inaccuracies, these methods cannot generalize to parts of the

state space outside of the training distribution. In addition to continuously updating the planned actions via MPC, our approach also updates the model parameters online, enabling compensation for much more substantial changes in the dynamics. Another approach to addressing the problem of learning a good global model is to instead learn more accurate local models (Buchan et al., 2013; Levine and Abbeel, 2014). These methods result in good local performance and allow for iterative local improvements of a policy. Our approach can be viewed as learning local models online with a very small amount of recent data combined with a learned global prior.

**Online Adaptation:** An alternative approach, often referred to as online adaptation (Tanaskovic et al., 2013; Aswani et al., 2012), is to learn an approximate global model and then adapt it at test time. Dynamic evaluation algorithms (Rei, 2015; B. Krause et al., 2017; B. Krause et al., 2016; Fortunato et al., 2017), for example, learn an approximate global distribution at training time and adapt those model parameters at test time to fit the current local distribution via gradient descent. Such work in model adaptation (Levine and Koltun, 2013; Gu et al., 2016; Fu et al., 2016) has shown that a perfect global model is not necessary, and prior knowledge can be used to handle small changes. These methods, however, face a mismatch between what the model is trained for and how it is used at test time. In this work, we bridge this gap by explicitly training a model for fast and effective adaptation. As a result, our model achieves more effective adaptation compared to these prior works, as validated in our experiments.

### 6.1.8 *Experiments*

The aim of the experimental evaluation is to answer the following research questions: (1) Does our approach enable fast adaptation to varied environments and dynamics? (2) In which situations does adaptive control with GBAC or RBAC produce quantifiable improvements in performance, and in which situations is one or the other more effective? (3) How does our method perform when it encounters situations that are outside of the distribution of training environments? See the appendix for further analysis and details of our method, including learning curves, hyperparameters, and the effect of the meta-training task distribution on performance. Videos are available online[1].

Figure 17: Simulated environments in our experiments. From left to right: half-cheetah (HC) with disabled joints, HC on terrain of varying slopes, ant with crippled legs, HC running across a pier with floating blocks of varying buoyancy, and 7-DoF arm moving an object subjected to force perturbations (cyan arrow) to a specified goal (red). Adaptation is needed in these stochastic and partially observable environments.

#### 6.1.8.1 *Environments*

To answer these questions, we conducted a comparative evaluation of our online adaptation algorithm, as well as several alternative prior methods, on a variety of simulated robots using the MuJoCo physics engine (Todorov et al., 2012). Each of our environments (Fig. 17) requires different types of adaptation:

HALF-CHEETAH (HC): DISABLED JOINT. For each meta-training rollout, we randomly sample a joint to be disabled (i.e., the agent cannot apply torques to that joint). At test time, we evaluate performance in three different situations: (a) disabling a joint seen at train time, (b) disabling an unseen joint, and (c) switching between disabled joints during a rollout.

HC: SLOPED TERRAIN. During meta-training, we choose terrain of varying gentle upward and downward slopes. In this task, it is especially important to incorporate past experience into the model, since the cheetah has no means of directly observing the incline. At test time, we evaluate performance on (d) a gentle upward slope, (e) a steep hill that goes up and down, and (f) a steep upward slope.

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

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

ANT: CRIPPLED LEG. For each meta-training rollout, we randomly sample a leg on a quadrupedal robot and disable it. Disabling a leg unexpected drastically changes the dynamics. We evaluate on (h) crippling a leg from the training distribution, (i) crippling a leg from outside the training distribution, and (j) crippling a leg in the middle of a rollout.

7-DOF ARM: FORCE PERTURBATIONS. We train a 7-DoF robot arm to carry an object to a goal position while applying random perturbation forces to the object. At test time, we evaluate with (k) a constant low force to the object, (l) a force $3\times$ stronger than during training, and (m) a force that randomly changes every 50 time-steps. This allow us to evaluate the ability of our method to adapt online to perturbations that clearly lie outside the training distribution.

In all of these environments, we model the transitions probabilities as Gaussian random variables with mean parametrized by the neural network model, and fixed variance. In that case, the maximum likelihood estimation corresponds to minimizing the mean squared error.

#### 6.1.8.2 *Comparisons*

We compare both our GBAC and RBAC methods to a non-adaptive model-based method ("MB"), which employs a feedforward neural network as the dynamics model and selects actions using MPC, but does not perform adaptation (Nagabandi et al., 2017a), an adaptive control method ("MB + DE") based on dynamic-evaluation, where gradient steps are taken to adapt a model at run time (B. Krause et al., 2017), and model-free (MF) learning with TRPO (Schulman et al., 2015).

In the interest of developing efficient algorithms for potential real-world applications, we compare these methods in a low data regime, using data that translates to 1.5-3 hours of real-world experience. We hold the amount of data used constant across all methods, within an environment. We also provide the performance of a MB oracle (no adaptation) and a MF oracle (no adaptation) that are trained with unlimited data until convergence. These oracles are trained separately for each test task (rather than being required to generalize to new tasks), which drastically simplifies the problem, and the MF oracle receives the equivalent of about one day of real-world experience.

The following plots are best viewed in color, and the return is normalized such that the MB oracle achieves a return of 1, and the worst performance algorithm (TRPO with this low data) achieves 0.

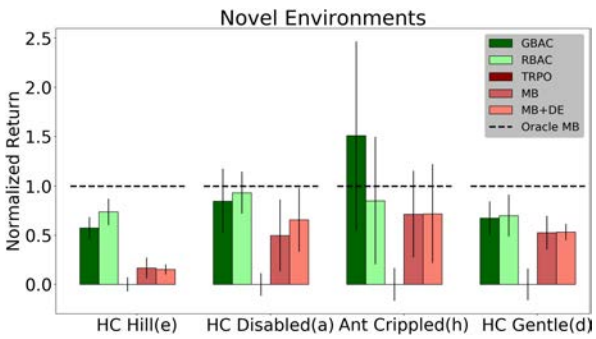


Figure 18: Results in novel static environments that require adaptation not seen during training. MF oracle policies for each task achieved 2.9, 3.4, 1.3, 3.8, respectively.

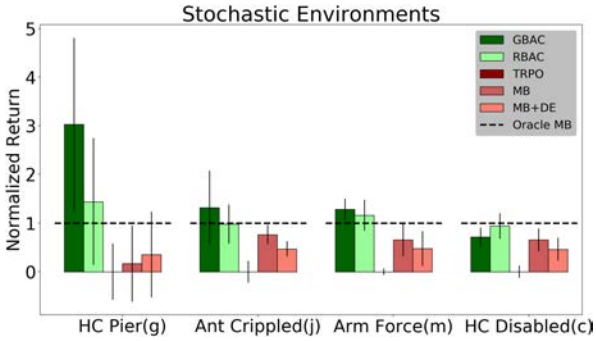


Figure 19: Results in environments where fast adaptation is crucial due to changes occuring throughout the execution of a rollout. MF oracle policies for each task achieved 5.2, 0.8, 1.4, 1.2, respectively.

#### 6.1.8.3 *Novel Environments*

We test the ability of each approach to effectively incorporate past experience to adapt in environments that were not explicitly seen during training, but that are also static throughout the duration of the test rollout. We evaluate the different approaches on the tasks $a, d, e, h$ in Fig. 18. Across these tasks, MF TRPO fails due to low quantity of training data, and MB+DE is equivalent to or slightly better than a model without dynamic evaluation (MB). Task $d$ requires minimal adaptation, so all methods achieve high rewards, except TRPO. However, when we test the performance on a more challenging steep hill ($e$), we see that RBAC outperforms the other approaches.

The strength of RBAC is in tasks where temporal progression is important (i.e., to determine terrain slope); the update rule of GBAC might not have some of this temporal insight, since it is invariant to the order in which the previous states were visited. On the other hand, GBAC attains better results on the more challenging, higher-dimensional, and less structured task of crippled ant ($h$).

#### 6.1.8.4 *Stochastic Environments*

We also compared the methods on tasks $c, g, j, m$, where stochastic and sudden changes in dynamics happen throughout the execution of a rollout (Fig. 19). GBAC outperforms the other approaches; the continuous application of the gradient-based update rule allows fast adaptation to changing dynamics. RBAC is also superior to the other compar-

isons, but it is not as good as GBAC at modeling sudden and ongoing changes in the dynamics, like in the HC pier task c. The high performance of RBAC in the HC disabled joint task is perhaps because there is not a drastic change: the disabled joint is still present, and able to be used for generating forward movement. We also see that dynamic evaluation can decrease performance when fast-adaptation is needed. Finally, the MB oracle is matched or out-performed by our approach in all tasks. This illustrates the challenge of learning a global model in stochastic environments; meta-trained models can adapt to these changes.

#### 6.1.8.5 *Extrapolation Environments*

We characterize the capacity to extrapolate to environments that are further outside of the training distribution, on tasks b, f, i, l in Fig. 20. The meta-trained dynamics model, combined with the strong signal from the gradient-based update rule, allows GBAC to perform well for big dynamics changes (i.e., ant's leg becomes both immobilized and shrunk). RBAC performs well when the underlying dynamics are smoother and temporal pattern information is very helpful for adaptation (e.g., HC walks on steep terrain).

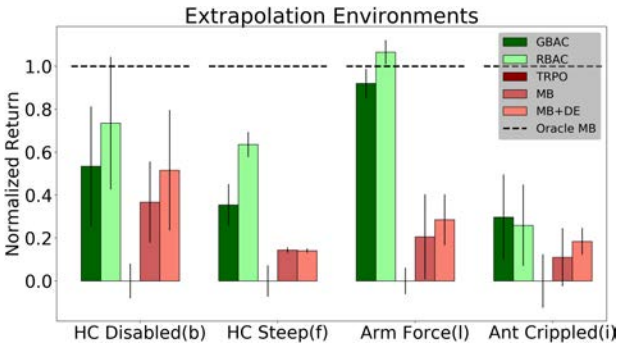


Figure 20: Comparison on tasks outside the training distribution. Our adaptive methods outperform prior work. MF oracle policies for each task achieved 2.8, 4.3, 1.1, 1.5, respectively.

#### 6.1.8.6 *Effect of Adaptation with GBAC*

Figure 21 shows the pre-update and post-update model errors, as seen during three different tasks on the 7-DoF arm with GBAC. The clear shift in distribution shows that using the past M time steps to update $\theta_*$ (pre) into $\phi_*$ (post) does indeed reduce model error when predicting the following K time steps. See appendix for further analysis on the model errors on all tasks.

### 6.1.9 *Discussion*

In this section, we presented an approach for adaptive control that combines meta-learning and model-based reinforcement. We showed that meta-learning a model for

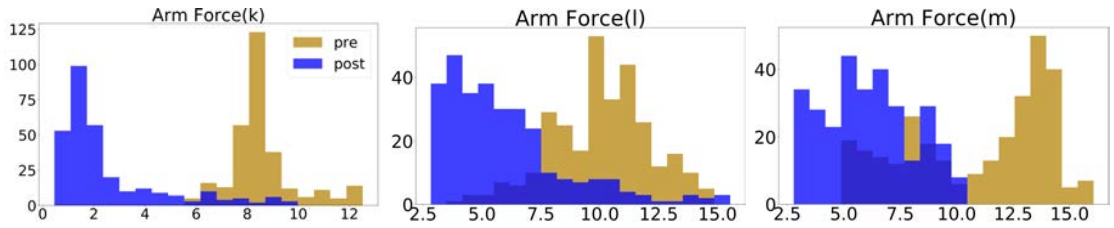


Figure 21: Histogram of normalized K-step model prediction errors (for GBAC model) in the tasks k, l, m, showing the post-update model outperforming the pre-update one.

online adaptation results in a method that is able to adapt to unseen situations or sudden and drastic changes in the environment, and is also sample efficient to train. We provided two instantiations of our approach (RBAC and GBAC), and we provided a comparison with other prior methods on a range of continuous control tasks. One question that merits future investigation is how to best combine GBAC and RBAC to reap the benefits of both of these methods without losing sample efficiency. Finally, an exciting direction for future work includes extending meta-learning for adaptive control to real-world systems. Through a combination of sample-efficient model-based learning and integration of off-policy data, our approach should be substantially more practical for real-world use than less efficient model-free meta-reinforcement learning approaches, and the capability to adapt quickly will be particularly important under complex real-world dynamics.

# 7

# FEW-SHOT IMITATION LEARNING

Demonstrations provide a descriptive medium for specifying robotic tasks. Prior work has shown that robots can acquire a range of complex skills through demonstration, such as table tennis (Mülling et al., 2013), lane following (Pomerleau, 1989), pouring water (Pastor et al., 2009), drawer opening (Rana et al., 2017), and multi-stage manipulation tasks (T. Zhang et al., 2017). However, the most effective methods for robot imitation differ significantly from how humans and animals might imitate behaviors: while robots typically need to receive demonstrations in the form of kinesthetic teaching (Pastor et al., 2011; Akgun et al., 2012) or teleoperation (Calinon et al., 2009; Rahmatizadeh et al., 2017; T. Zhang et al., 2017), humans and animals can acquire the gist of a behavior simply by *watching* someone else. In fact, we can adapt to variations in morphology, context, and task details effortlessly, compensating for whatever *domain shift* may be present and recovering a skill that we can use in new situations (Brass and Heyes, 2005). Additionally, we can do this from a very small number of demonstrations, often only one. How can we endow robots with the same ability to learn behaviors from a single human demonstration?

In this chapter, we first study the problem of one-shot imitation, where we aim to leverage data across many different skills in order to learn a new skill with minimal supervision – just a single teleoperated demonstration. Then, in Section 7.2, we aim further, learning to learn behaviors from raw third person observations of human demonstrators.

## 7.1 META-IMITATION LEARNING

We propose to combine meta-learning with imitation, enabling a robot to reuse past experience and, as a result, learn new skills from a single demonstration. Unlike prior methods that take the task identity (M. P. Deisenroth et al., 2014; Kupcsik et al., 2013; Schaul et al., 2015; Stulp et al., 2013) or a demonstration (Duan et al., 2017) as the in-

put into a contextual policy, our approach learns a parameterized policy that can be adapted to different tasks through gradient updates, effectively learning to imitation learn. As a result, the set of skills that can be learned is more flexible while using fewer overall parameters. For the first time, we demonstrate that vision-based policies can be fine-tuned end-to-end from one demonstration, using meta-learning as a pre-training procedure that uses demonstrations on a diverse range of other environments.

The primary contribution of this section is to demonstrate an approach for one-shot imitation learning from raw pixels. We evaluate our approach on two simulated planar reaching domains, on simulated pushing tasks, and on visual placing tasks on a real robot (See Figure 22). Our approach is able to learn visuomotor policies that can adapt to new task variants using only one visual demonstration, including settings where only a raw video of the demonstration is available without access to the controls applied by the demonstrator. By employing a parameter-efficient meta-learning method, our approach requires a relatively modest number of demonstrations for meta-learning and scales to raw pixel inputs. As a result, our method can successfully be applied to real robotic systems.


Figure 22: The robot learns to place a new object into a new container from a single demonstration.

### 7.1.1 *Overview*

Here, we describe how we can extend the model-agnostic meta-learning algorithm (MAML) to the imitation learning setting. The model's input, $\mathbf{o}_t$, is the agent's observation at time $t$, e.g. an image, whereas the output $\mathbf{a}_t$ is the action taken at time $t$, e.g. torques applied to the robot's joints. We will denote a demonstration trajectory as $\tau := \{\mathbf{o}_1, \mathbf{a}_1, ...\mathbf{o}_T, \mathbf{a}_T\}$ and use a mean squared error loss as a function of policy parameters $\phi$ as follows:

$$\mathcal{L}(\psi, \mathcal{D}_{\mathcal{T}_i}) = \sum_{\tau^{(j)} \sim \mathcal{D}_{\mathcal{T}_i}} \sum_t \|f_\psi(\mathbf{o}_t^{(j)}) - \mathbf{a}_t^{(j)}\|_2^2. \tag{19}$$

We will primarily consider the one-shot case, where only a single demonstration $\tau^{(j)}$ is used for the gradient update. However, we can also use multiple demonstrations to resolve ambiguity.

**Algorithm 9** Meta-Imitation Learning with MAML

**Require:** $p(\mathcal{T})$: distribution over tasks
**Require:** $\alpha$, $\beta$: step size hyperparameters
1: randomly initialize $\theta$
2: **while** not done **do**
3: Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
4: **for** each $\mathcal{T}_i$ **do**
5: Sample demonstration $\tau_i = \{\mathbf{o}_1, \mathbf{a}_1, ...\mathbf{o}_T, \mathbf{a}_T\}$ from $\mathcal{D}_{\mathcal{T}_i}$
6: Evaluate $\nabla_\theta \mathcal{L}(\theta, \{\tau_i\})$ using $\mathcal{L}$ in Equation (19)
7: Compute adapted parameters with gradient descent: $\phi_i = \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \tau_i)$
8: Sample demonstration $\tau_i' = \{\mathbf{o}_1', \mathbf{a}_1', ...\mathbf{o}_T', \mathbf{a}_T'\}$ from $\mathcal{D}_{\mathcal{T}_i} \setminus \{\tau_i\}$ for the meta-update
9: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}(\phi_i, \{\tau_i'\})$ using each $\mathcal{L}$ in Equation 19
10: **return** parameters $\theta$ that can be quickly adapted to new tasks through imitation.

For meta-learning, we assume a dataset of demonstrations with at least two demonstrations per task. This data is only used during meta-training; meta-test time assumes only one demonstration for each new task. During meta-training, each meta-optimization step entails the following: A batch of tasks is sampled and two demonstrations are sampled per task. Using one of the demonstrations, $\phi_i$ is computed for each task $\mathcal{T}_i$ using gradient descent with Equation 19. Then, the second demonstration of each task, $\tau_i'$, is used to compute the gradient of the meta-objective $\mathcal{L}(\phi_i, \tau_i')$ using the loss in Equation 19. Finally, $\theta$ is updated according to the gradient of the meta-objective. In effect, the pair of demonstrations serves as a training-validation pair. The algorithm is summarized in Algorithm 9.

The result of meta-training is a policy that can be adapted to new tasks using a single demonstration. Thus, at meta-test time, a new task $\mathcal{T}$ is sampled, one demonstration for that task is provided, and the model is updated to acquire a policy for that task. During meta-test time, a new task might involve new goals or manipulating new, previously unseen objects.

### 7.1.2 *Two-Head Architecture: Learning a Loss for Fast Adaptation*

In the standard MAML setup, outlined previously, the policy is consistent across the pre- and post-gradient update stages. However, we can make a modification such that the parameters of the final layers of the network are not shared, forming two "heads,"

as shown in Figure 23. The parameters of the pre-update head are not used for the final, post-update policy, and the parameters of the post-update head are not updated using the demonstration. But, both sets of parameters are meta-learned for effective performance after adaptation. Interestingly, this two head architecture amounts to using a different inner objective in the meta-optimization, while keeping the same outer objective. To see this, let us denote $\mathbf{y}_t^{(j)}$ as the set of post-synamptic activations of the last hidden layer, and $W$ and $b$ as the weight matrix and bias of the final layer. The inner loss function is then given by:

$$\mathcal{L}^*(\theta, \mathcal{D}_{\mathcal{T}_i}) = \sum_{\tau^{(j)} \sim \mathcal{D}_{\mathcal{T}_i}} \sum_t \|W\mathbf{y}_t^{(j)} + b - \mathbf{a}_t^{(j)}\|_2^2, \tag{20}$$

where $W$ and $b$, the weights and bias of the last layer, effectively form the parameters of the meta-learned loss function. We use the meta-learned loss function $\mathcal{L}^*$ to compute the adapted parameter $\phi_i$ of each task $\mathcal{T}_i$, via gradient descent. Then, the meta-objective becomes:

$$\min_{\theta, W, b} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}(\phi_i, \mathcal{D}_{\mathcal{T}_i}^{\text{test}}) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}(\theta - \alpha \nabla_\theta \mathcal{L}^*(\theta, \mathcal{D}_{\mathcal{T}_i}^{\text{tr}}), \mathcal{D}_{\mathcal{T}_i}^{\text{test}}). \tag{21}$$

This provides the algorithm more flexibility in how it adapts the policy parameters to the expert demonstration, which we found to increase performance in a few experiments (see Appendix B.2.3). However, the more interesting implication of using a learned loss is that we can omit the actions during 1-shot adaptation, as we discuss next.

### 7.1.3 *Learning to Imitate without Expert Actions*

Conventionally, a demonstration trajectory consists of pairs of observations and actions, as we discussed in Section 7.1. However, in many scenarios, it is more practical to simply provide a video of the task being performed, e.g. by a human or another robot. One step towards this goal, which we consider in this section, is to remove the need for the robot arm trajectory and actions at test time.[1] Though, to be clear, we will assume access to expert actions during meta-training. Without access to expert actions at test time, it is unclear what the loss function for 1-shot adaptation should be. Thus, we will meta-learn

1 We leave the problem of domain shift, i.e. between a video of a human and the robot's view, to the next section.

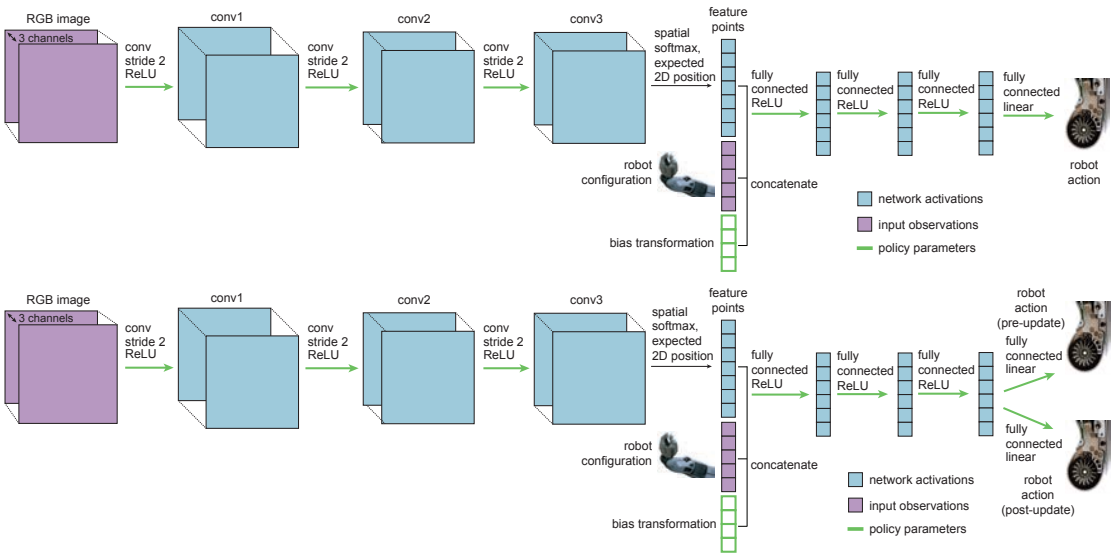


Figure 23: Diagrams of the policy architecture with a bias transformation (top and bottom) and two heads (bottom). The green arrows and boxes indicate weights that are part of the meta-learned policy parameters $\theta$.

a loss function, as discussed in the previous section. We can simply modify the loss in Equation 20 by removing the expert actions:

$$\mathcal{L}^*_{\mathcal{T}_i}(\theta) = \sum_{\tau^{(j)} \sim \mathcal{D}_{\mathcal{T}_i}} \sum_t \|W \mathbf{y}_t^{(j)} + b\|_2^2,$$

This corresponds to a learned quadratic loss function on the final layer of activations, with parameters $W$ and $b$. Though, in practice, the loss function could be more complex. With this loss function, we can learn to learn from the raw observations of a demonstration using the meta-optimization objective in Equation 21, as shown in our experiments in Sections 7.1.6.2 and 7.1.6.3.

### 7.1.4 *Model Architectures for Meta-Imitation Learning*

We use a convolutional neural network (CNN) to represent the policy, similar to prior vision-based imitation and meta-learning methods (Bojarski et al., 2016; Finn et al., 2017a). The policy observation includes both the camera image and the robot's configuration, e.g. the joint angles and end-effector pose. In this section, we overview the policy

architecture, but leave the details to be discussed in Section 5.6. The policy consists of several strided convolutions, followed by ReLU nonlinearities. The final convolutional layer is transformed into spatial feature points using a spatial soft-argmax (Levine et al., 2016a; Finn et al., 2016b) and concatenated with the robot's configuration. The result is passed through a set of fully-connected layers with ReLU nonlinearities. Because the data within a demonstration trajectory is highly correlated across time, batch normalization was not effective. Instead, we used layer normalization after each layer (J. L. Ba et al., 2016).

Although meta-imitation learning can work well with standard policy architectures such as the one described above, the optimal architecture for meta-learning does not necessarily correspond to the optimal architecture for standard supervised imitation learning. One particular modification that we found improves meta-learning performance is to concatenate a vector of parameters to a hidden layer of post-synaptic activations, which leads to what we will refer to as a *bias transformation*. This parameter vector is treated the same as other parameters in the policy during both meta-learning and test-time adaptation. Formally, let us denote the parameter vector as $\mathbf{z}$, the post-synaptic activations as $\mathbf{x}$, and the pre-synaptic activations at the next layer as $\mathbf{y}$. A standard neural network architecture sets $\mathbf{y} = W\mathbf{x} + \mathbf{b}$, for bias $\mathbf{b}$ and weight matrix $W$. The error gradient with respect to the standard bias $\frac{d\mathcal{L}}{d\mathbf{b}}$ equals the error gradient with respect to $\mathbf{y}$, $\frac{d\mathcal{L}}{d\mathbf{y}}$. Thus, a gradient update of the standard bias is directly coupled with the update to the weight matrix $W$ and parameters in earlier layers of the network. The bias transformation, which we describe next, provides more control over the updated bias by eliminating this decoupling. With a bias transformation, we set $\mathbf{y} = W_1\mathbf{x} + W_2\mathbf{z} + \mathbf{b}$, where $W = [W_1, W_2]$ and $\mathbf{b}$ are the weight matrix and bias. First, note that including $\mathbf{z}$ and $W_2$ simply corresponds to a reparameterization of the bias, $\tilde{\mathbf{b}} = W_2\mathbf{z} + \mathbf{b}$, since neither $W_2\mathbf{z}$ nor $\mathbf{b}$ depend on the input. The error gradient with respect to $\mathbf{z}$ and $W_2$ are: $\frac{d\mathcal{L}}{dW_2} = \frac{d\mathcal{L}}{d\mathbf{y}}\mathbf{z}^T$ and $\frac{d\mathcal{L}}{d\mathbf{z}} = W_2^T\frac{d\mathcal{L}}{d\mathbf{y}}$. After one gradient step, the updated transformed bias is: $\tilde{\mathbf{b}}' = (W_2 - \alpha\frac{d\mathcal{L}}{d\mathbf{y}}\mathbf{z}^T)(\mathbf{z} - \alpha W_2^T\frac{d\mathcal{L}}{d\mathbf{y}}) + \mathbf{b} - \alpha\frac{d\mathcal{L}}{d\mathbf{y}}$. Thus, a gradient update to the transformed bias can be controlled more directly by the values of $W_2$ and $\mathbf{z}$, whose values do not directly affect the gradients of other parameters in the network. To see one way in which the network might choose to control the bias, consider the setting where $\mathbf{z}$ and $\mathbf{b}$ are zero. Then, the updated bias is: $\tilde{\mathbf{b}}' = -\alpha W_2 W_2^T\frac{d\mathcal{L}}{d\mathbf{y}} - \alpha\frac{d\mathcal{L}}{d\mathbf{y}}$. In summary, the bias transformation increases the representational power of the *gradient*, without affecting the representation power of the network itself. In our experiments, we found this simple addition to the network made gradient-based meta-learning significantly more stable and effective. We include a diagram of the policy architecture with the bias transformation in Figure 23.

### 7.1.5 *Related Work*

We present a method that combines imitation learning (Schaal et al., 2003) with meta-learning (Thrun and Pratt, 1998) for one-shot learning from visual demonstrations. Efficient imitation from a small number of demonstrations has been successful in scenarios where the state of the environment, such as the poses of objects, is known (Billard et al., 2004; Schaal et al., 2005; Pastor et al., 2009; N. Ratliff et al., 2007). In this work, we focus on settings where the state of the environment is unknown, where we must instead learn from raw sensory inputs. This removes the need for pre-defined vision systems while also making the method applicable to vision-based non-prehensile manipulation in unknown, dynamic environments. Imitation learning from raw pixels has been widely studied in the context of mobile robotics (Pomerleau, 1989; Bojarski et al., 2016; Giusti et al., 2016; J. Zhang and Cho, 2017). However, learning from demonstrations has two primary challenges when applied to real-world settings. The first is the widely-studied issue of compounding errors (Ross et al., 2011; Laskey et al., 2016; J. Zhang and Cho, 2017), which we do not address in this chapter. The second is the need for a large number of demonstrations for each task. This latter limitation is a major roadblock for developing generalist robots that can learn a wide variety of tasks through imitation. Inverse reinforcement learning (Ng and Russell, 2000) can reduce the number of demonstrations needed by inferring the reward function underlying a set of demonstrations. However, this requires additional robot experience to optimize the reward (Finn et al., 2016a; Nair et al., 2017; Sermanet et al., 2017b). This experience typically comes in the form of trial-and-error learning or data for learning a model.

In this work, we drastically reduce the number of demonstrations needed for an individual task by sharing data across tasks. In particular, our goal is to learn a new task from a single demonstration of that task by using a dataset of demonstrations of many other tasks for meta-learning. Sharing information across tasks is by no means a new idea, e.g. by using task-to-task mappings (Barrett et al., 2010), gating (Mülling et al., 2013), and shared features (Gupta et al., 2017). These multi-task robotic learning methods consider the problem of generalization to new tasks from some specification of that task. A common approach, often referred to as contextual policies, is to provide the task as an input to the policy or value function, where the task is represented as a goal or demonstration (M. P. Deisenroth et al., 2014; Kupcsik et al., 2013; Stulp et al., 2013; Schaul et al., 2015; Duan et al., 2017). Another approach is to train controllers for a variety of tasks and learn a mapping from task representations to controller parameters (Pastor et al., 2009; Kober et al., 2012; Da Silva et al., 2012). In this work, we instead use meta-learning

to enable the robot to quickly learn new tasks with gradient-based policy updates. In essence, we learn policy parameters that, when finetuned on just one demonstration of a new task, can immediately learn to perform that task. This enables the robot to learn new tasks end-to-end with extreme efficiency, using only one demonstration, without requiring any additional mechanisms such as contexts or learned update functions.

### 7.1.6 *Experiments*

The goals of our experimental evaluation are to answer the following questions: (1) can our method learn to learn a policy that maps from image pixels to actions using a single demonstration of a task? (2) how does our meta-imitation learning method compare to prior one-shot imitation learning methods with varying dataset sizes? (3) can we learn to learn without expert actions? (4) how well does our approach scale to real-world robotic tasks with real images?

We evaluate our method on one-shot imitation in three experimental domains. In each setting, we compare our proposed method to a subset of the following methods:

- **random policy**: A policy that outputs random actions from a standard Normal distribution.
- **contextual policy**: A feedforward policy, which takes as input the final image of the demonstration, to indicate the goal of the task, and the current image, and outputs the current action.
- **LSTM**: A recurrent neural network which ingests the provided demonstration and the current observation, and outputs the current action, as proposed by Duan et al. (2017).
- **LSTM+attention**: A recurrent neural network using the attention architecture proposed by Duan et al. (2017). This method is only applicable to non-vision tasks.

The contextual policy, the LSTM policies, and the proposed approach are all trained using the same datasets, with the same supervision. All policies, including the proposed approach, were meta-trained via a behavioral cloning objective (mean squared error) with supervision from the expert actions, using the Adam optimizer with default hyperparameters (D. Kingma and J. Ba, 2015).

#### 7.1.6.1 *Simulated Reaching*

The first experimental domain is a family of planar reaching tasks, as illustrated in Figure 24, where the goal of a particular task is to reach a target of a particular color, amid two distractors with different colors. This simulated domain allows us to rigorously

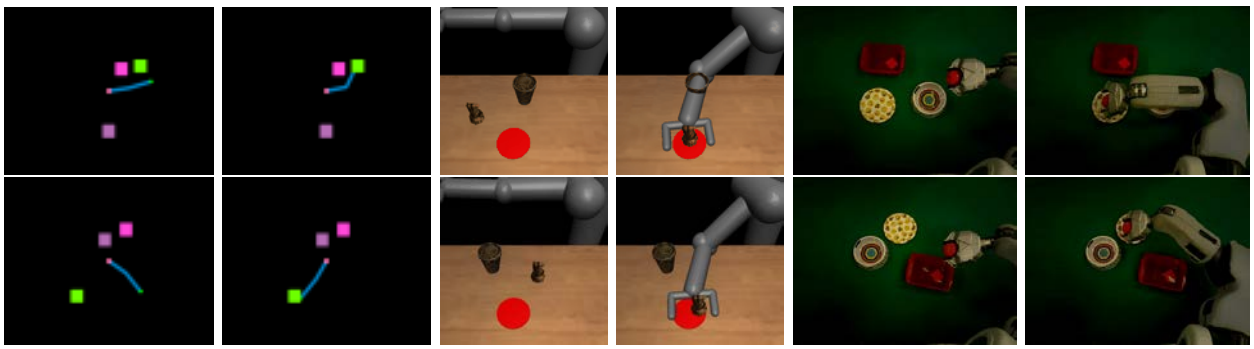

Figure 24: Example tasks from the policy's perspective. In the top row, each pair of images shows the start and final scenes of the demonstration. The bottom row shows the corresponding scenes of the learned policy roll-out. Left: Given one demonstration of reaching a target of a particular color, the policy must learn to reach for the same color in a new setting. Center: The robot pushes the target object to the goal after seeing a demonstration of pushing the same object toward the goal in a different scene. Right: We provide a demonstration of placing an object on the target, then the robot must place the object on the same target in a new setting.

evaluate our method and compare with prior approaches and baselines. We consider both vision and non-vision variants of this task, so that we can directly compare to prior methods that are not applicable to vision-based policies. See Appendix B.2.1 for more details about the experimental setup and choices of hyperparameters.

We evaluate each method on a range of meta-training dataset sizes and show the one-shot imitation success rate in Figure 25. Using vision, we find that meta-imitation learning is able to handle raw pixel inputs, while the LSTM and contextual policies struggle to learn new tasks using modestly-sized meta-learning datasets. In the non-vision case, which involves far fewer parameters, the LSTM policies fare much better, particularly when using attention, but still perform worse than MIL. Prior work demonstrated these approaches using 10,000 or more demonstrations (Duan et al., 2017). Therefore, the mediocre performance of these methods on much smaller datasets is not surprising. We also provide a comparison with and without the bias transformation discussed in Section 7.1.4. The results demonstrate that MIL with the bias transformation (bt) can perform more consistently across dataset sizes.

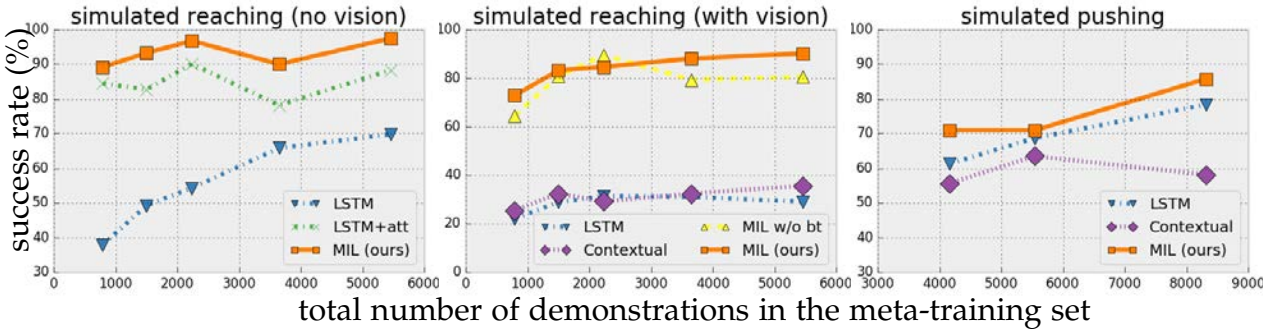


Figure 25: One-shot success rate on test tasks as a function of the meta-learning dataset size in the simulated domains. Our meta-imitation learning approach (MIL) can perform well across a range of dataset sizes, and can more effectively learn new tasks than prior approaches that feed in the goal image (contextual) or demonstration (LSTM) as input. A random policy achieves 25.7% reaching success and 0.45% pushing success. For videos of the policies, see the supplementary video[2].

#### 7.1.6.2 *Simulated Pushing*

The goal of our second set of experiments is to evaluate our approach on a challenging domain, involving 7-DoF torque control, a 3D environment, and substantially more physical and visual diversity across tasks. The experiment consists of a family of simulated table-top pushing tasks, illustrated in Figure 24, where the goal is to push a particular object with random starting position to the red target amid one distractor. We designed the pushing environment starting from the OpenAI Gym PusherEnv, using the MuJoCo physics engine (Brockman et al., 2016; Todorov et al., 2012). We modified the environment to include two objects, vision policy input, and, across tasks, a wide range of object shapes, sizes, textures, frictions, and masses. We selected 116 mesh shapes from `thingiverse.com`, 105 meshes for meta-training and 11 for evaluation. The meshes include models of chess pieces, models of animals like teddy bears and pufferfish, and other miscellaneous shapes. We randomly sampled textures from a set of over 5,000 images and used held-out textures for meta-testing. A selection of the objects and textures are shown in Figure 26. For more experimental details, hyperparameters, and ablations, see Appendix B.2.2.

The performance of one-shot pushing with held-out objects, shown in Figure 25, indicates that MIL effectively learned to learn to push new objects, with 85.8% one-shot learning success using the largest dataset size. Furthermore, MIL achieves, on average,

2 For video results and code, see `https://sites.google.com/view/one-shot-imitation`

| method | | video+state +action | video +state | video |
|---|---|---|---|---|
| LSTM | 1-shot | 78.38% | 37.61% | 34.23% |
| contextual | | n/a | 58.11% | 56.98% |
| MIL (ours) | | **85.81%** | **72.52%** | **66.44%** |
| LSTM | 5-shot | 83.11% | 39.64% | 31.98% |
| contextual | | n/a | 64.64% | 59.01% |
| MIL (ours) | | **88.75%** | **78.15%** | **70.50%** |

Table 4: One-shot and 5-shot simulating pushing success rate with varying demonstration information provided at test-time. MIL can more successfully learn from a demonstration without actions and without robot state and actions than LSTM and contextual policies.

6.5% higher success than the LSTM-based approach across dataset sizes. The contextual policy struggles, likely because the full demonstration trajectory information is informative for inferring the friction and mass of the target object.

In Table 4, we provide two additional evaluations, using the largest dataset size. The first evaluates how each approach handles input demonstrations with less information, e.g. without actions and/or the robot arm state. For this, we trained each method to be able to handle such demonstrations, as discussed in Section 7.1.3. We see that the LSTM approach has difficulty learning without the expert actions. MIL also sees a drop in performance, but one that is less dramatic. The second evaluation shows that all approaches can improve performance if five demonstrations are available, rather than one, despite all policies being trained for 1-shot learning. In this case, we averaged the predicted action over the 5 input demonstrations for the contextual and LSTM approaches, and averaged the gradient for MIL.

#### 7.1.6.3 *Real-World Placing*

The goal of our final experiment is to evaluate how well a real robot can learn to learn to interact with new, unknown objects from a single visual demonstration. Handling unseen objects is a challenge for both learning-based and non-learning-based manipulation methods, but is a necessity for robots to be capable of performing diverse tasks in unstructured real-world environments. In practice, most robot learning approaches have

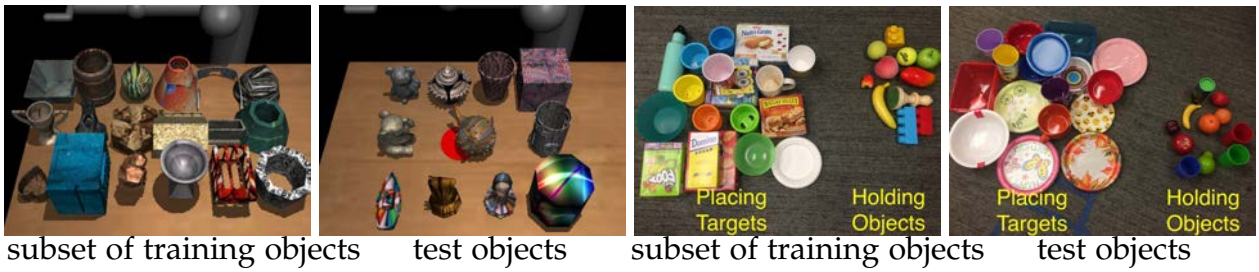


Figure 26: Training and test objects used in our simulated pushing (left) and real-world placing (right) experiments. Note that we only show a subset of the ~100 training objects used for the pushing and placing experiments, and a subset of the textures and object scales used for training and testing robot pushing.

focused on much more narrow notions of generalization, such as a varied target object location (Levine et al., 2016a) or block stacking order (Duan et al., 2017). With this goal in mind, we designed a robotic placing experiment using a 7-DoF PR2 robot arm and RGB camera, where the goal is to place a held item into a target container, such as a cup, plate, or bowl, while ignoring two distractors. We collected roughly 1300 demonstrations for meta-training using a diverse range of objects, and evaluated one-shot learning using held-out, unseen objects (see Figure 26). The policy is provided a single visual demonstration of placing the held item onto the target, but with varied positions of the target and distractors, as illustrated in Figure 24. Demonstrations were collected using human teleoperation through a motion controller and virtual reality headset (T. Zhang et al., 2017), and each demo included the camera video, the sequence of end-effector poses, and the sequence of actions – the end-effector linear and angular velocities. See Appendix B.2.3 for more explanation of data collection, evaluation, and hyperparameters.

The results, in Table 5, show that the MIL policy can learn to localize the previously-unseen target object and successfully place the held item onto the target with 90% success, using only a single visual demonstration with those objects. We found that the LSTM and contextual policies were unable to localize the correct target object, likely due to the modestly-sized meta-training dataset, and instead placed onto an arbitrary object, achieving 25% success. Using the two-head approach described in 7.1.3, we also experimented with only providing the video of the demonstration, omitting the robot end-effector trajectory and controls. MIL can also learn to handle this setting, although with less success, suggesting the need for more data and/or further research. We include

| method | test performance |
|---|---|
| LSTM | 25% |
| contextual | 25% |
| MIL | **90%** |
| MIL, video only | **68.33%** |

Table 5: One-shot success rate of placing a held item into the correct container, with a real PR2 robot, using 29 held-out test objects. Meta-training used a dataset with around 100 objects. MIL, using video only receives only the demonstration video and not the arm trajectory or actions.

videos of all placing policies in the supplementary video[3].

### 7.1.7 *Discussion and Future Work*

We proposed a method for one-shot visual imitation learning that can learn to perform tasks using visual inputs from just a single demonstration. Our approach extends gradient-based meta-learning to the imitation learning setting, and our experimental evaluation demonstrates that it substantially outperforms a prior one-shot imitation learning method based on recurrent neural networks. The use of gradient-based meta-learning makes our approach more efficient in terms of the number of demonstrations needed during meta-training, and this efficiency makes it possible for us to also evaluate the method using raw pixel inputs and on a real robotic system.

The use of meta-imitation learning can substantially improve the efficiency of robotic learning methods without sacrificing the flexibility and generality of end-to-end training, which is especially valuable for learning skills with complex sensory inputs such as images. While our experimental evaluation uses tasks with limited diversity (other than object diversity), we expect the capabilities of our method to increase substantially as it is provided with increasingly more diverse demonstrations for meta-training. Since meta-learning algorithms can incorporate demonstration data from all available tasks, they provide a natural avenue for utilizing large datasets in a robotic learning context, making it possible for robots to not only learn more skills as they are acquire more demonstrations, but to actually become *faster* and *more effective* at learning new skills

3 For video results and code, see https://sites.google.com/view/one-shot-imitation

through the process.

## 7.2 ONE-SHOT IMITATION FROM HUMANS

In the previous section, we discussed how we can apply the MAML algorithm to one-shot imitation learning problem, using robot demonstrations collected via teleoperation and a mean-squared error behavioral cloning objective for the loss $\mathcal{L}$. While this enables learning from one robot demonstration at meta-test time, it does not allow the robot to learn from a raw video of a human or handle domain shift between the demonstration medium and the robot. In this section, we aim to extend the previous approach to be able to learn from raw video demonstrations of humans.

Acquiring skills from raw camera observations presents two major challenges. First, the difference in appearance and morphology of the human demonstrator from the robot introduces a systematic domain shift, namely the correspondence problem (Nehaniv, Dautenhahn, et al., 2002; Brass and Heyes, 2005). Second, learning from raw visual inputs typically requires a substantial amount of data, with modern deep learning vision systems using hundreds of thousands to millions of images (Xiang et al., 2018; D.-K. Kim and Walter, 2017). In this section, we demonstrate that we can begin to address both of these challenges through a single approach based on meta-learning. Instead of manually specifying the correspondence between human and robot, which can be particularly complex for skills where different morphologies require different strategies, we propose a data-driven approach. Our approach can acquire new skills from only one video of a human. To enable this, it builds a rich prior over tasks during a *meta-training* phase, where both human demonstrations and teleoperated demonstrations are available for a variety of other, structurally similar tasks. In essence, the robot learns how to learn from humans using this data. After the meta-training phase, the robot can acquire new skills by combining its learned prior knowledge with one video of a human performing the new skill.

The main contribution of this section is a system for learning robotic manipulation skills from a single video of a human by leveraging large amounts of prior meta-training data, collected for different tasks. When deployed, the robot can adapt to a particular task with novel objects using just a single video of a human performing the task with those objects (e.g., see Figure 27). The video of the human need not be from the same perspective as the robot, or even be in the same room. The robot is trained using videos of humans performing tasks with various objects along with demonstrations of the robot performing the same task. Our experiments on two real robotic platforms demonstrate

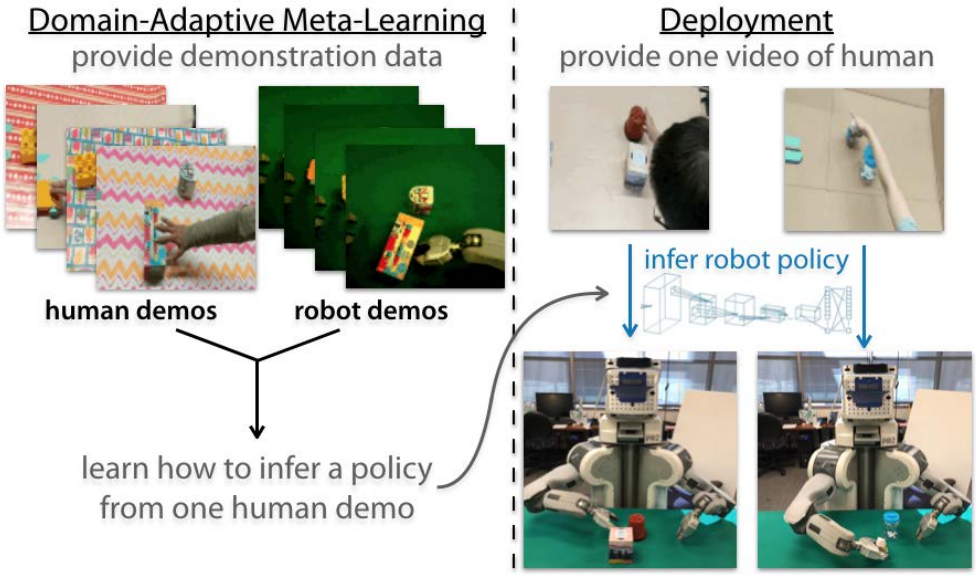


Figure 27: After meta-learning with human and robot demonstration data, the robot learns to recognize and push a new object from one video of a human.

the ability to learn directly from RGB videos of humans, and to handle novel objects, novel humans, and videos of humans in novel scenes. Videos of the results can be found on the supplementary website.[4]

### 7.2.1 *Problem Overview*

The problem of learning from human video can be viewed as an inference problem, where the goal is to infer the robot policy parameters $\phi_{\mathcal{T}_i}$ that will accomplish the task $\mathcal{T}_i$ by incorporating prior knowledge with a small amount of evidence, in the form of one human demonstration. In order to effectively learn from just one video of a human, we need a rich prior that encapsulates a visual and physical understanding of the world, what kinds of outcomes the human might want to accomplish, and which actions might allow a robot to bring about that outcome. We could choose to encode prior knowledge manually, for example by using a pre-defined vision system, a pre-determined set of human objectives, or a known dynamics model. However, this type of manual knowledge encoding is task-specific and time-consuming, and does not benefit from data. We

4 The video is available at https://sites.google.com/view/daml

will instead study how we can learn this prior automatically, using human and robot demonstration data from a variety of tasks.

Formally, we will define a demonstration from a human $\tau^h$ to be a sequence of image observations $\mathbf{o}_1, ..., \mathbf{o}_T$, and a robot demonstration $\tau^r$ to be a sequence of image observations, robot states, and robot actions: $\mathbf{o}_1, \mathbf{s}_1, \mathbf{a}_1, ..., \mathbf{o}_T, \mathbf{s}_T, \mathbf{a}_T$. The robot state includes the robot's body configuration, such as joint angles, but does not include object information, which must be inferred from the image. We do not make any assumptions about the similarities or differences between the human and robot observations; they may contain substantial domain shift, e.g. differences in the appearance of the arms, background clutter, and camera viewpoint.

Our approach consists of two phases. First, in the meta-training phase, the goal will be to acquire a prior over policies using both human and robot demonstration data, that can then be used to quickly learn to imitate new tasks with only human demonstrations. For meta-training, we will assume a distribution over tasks $p(\mathcal{T})$, a set of tasks $\{\mathcal{T}_i\}$ drawn from $p(\mathcal{T})$ and, for each task, two small datasets containing several human and robot demonstrations, respectively: $(\mathcal{D}^h_{\mathcal{T}_i}, \mathcal{D}^r_{\mathcal{T}_i})$. After the meta-training phase, the learned prior can be used in the second phase, when the method is provided with a human demonstration of a new task $\mathcal{T}$ drawn from $p(\mathcal{T})$. The robot must combine its prior with the new human demonstration to infer policy parameters $\phi_{\mathcal{T}}$ that solve the new task. We will next discuss our approach in detail.

### 7.2.2 *Domain-Adaptive Meta-Learning*

We develop a domain-adaptive meta-learning method, which will allow us to handle the setting of learning from video demonstrations of humans. While we will extend the MAML algorithm for this purpose, the key idea of our approach is applicable to other meta-learning algorithms. Like the MAML algorithm, we will learn a set of initial parameters, such that after one or a few steps of gradient descent on just one human demonstration, the model can effectively perform the new task. Thus, the data $\mathcal{D}^{tr}_{\mathcal{T}}$ will contain one human demonstration of task $\mathcal{T}$, and the data $\mathcal{D}^{val}_{\mathcal{T}}$ will contain one or more robot demonstrations of the same task.

Unfortunately, we cannot use a standard imitation learning loss for the inner adaptation objective computed using $\mathcal{D}^{tr}_{\mathcal{T}}$, since we do not have access to the human's actions. Even if we knew the human's actions, they will typically not correspond directly to the robot's actions. Instead, we propose to meta-learn an adaptation objective that does not require actions, and instead operates only on the policy activations. The intuition behind

meta-learning a loss function is that we can acquire a function that only needs to look at the available inputs (which do not include the actions), and still produce gradients that are suitable for updating the policy parameters so that it can produce effective actions after the gradient update. While this might seem like an impossible task, it is important to remember that the meta-training process still supervises the policy with true robot actions during meta-training. The role of the adaptation loss therefore may be interpreted as simply directing the policy parameter update to modify the policy to pick up on the right visual cues in the scene, so that the meta-trained action output will produce the right actions. We will discuss the particular form of $\mathcal{L}_\psi$ in the following section.

During the meta-training phase, we will learn both an initialization $\theta$ and the parameters $\psi$ of the adaptation objective $\mathcal{L}_\psi$. The parameters $\theta$ and $\psi$ are optimized for choosing actions that match the robot demonstrations in $\mathcal{D}_{\mathcal{T}}^{\text{val}}$. After meta-training, the parameters $\theta$ and $\psi$ are retained, while the data is discarded. A human demonstration $\tau^h$ is provided for a new task $\mathcal{T}$ (but not a robot demonstration). To infer the policy parameters for the new task, we use gradient descent starting from $\theta$ using the learned loss $\mathcal{L}_\psi$ and one human demonstration $\tau^h$: $\phi_{\mathcal{T}} = \theta - \alpha \nabla_\theta \mathcal{L}_\psi(\theta, \tau^h)$.

We optimize for task performance during meta-training using a behavioral cloning objective that maximizes the probability of the expert actions in $\mathcal{D}^{\text{val}}$. In particular, for a policy parameterized by $\phi$ that outputs a distribution over actions $\pi_\phi(\cdot|\mathbf{o}, \mathbf{s})$, the behavioral cloning objective is

$$\mathcal{L}_{\text{BC}}(\phi, \tau^r) = \mathcal{L}_{\text{BC}}(\phi, \{\mathbf{o}_{1:T}, \mathbf{s}_{1:T}, \mathbf{a}_{1:T}\}) = \sum_t \log \pi_\phi(\mathbf{a}_t|\mathbf{o}_t, \mathbf{s}_t)$$

Putting this together with the inner gradient descent adaptation, the meta-training objective is the following:

$$\min_{\theta,\psi} \sum_{\mathcal{T}\sim p(\mathcal{T})} \sum_{\tau^h \in \mathcal{D}_{\mathcal{T}}^h} \sum_{\tau^r \in \mathcal{D}_{\mathcal{T}}^r} \mathcal{L}_{\text{BC}}(\theta - \alpha \nabla_\theta \mathcal{L}_\psi(\theta, \tau^h), \tau^r).$$

The algorithm for optimizing this meta-objective is summarized in Algorithm 10, whereas the procedure for learning from humans at meta-test time is shown in Algorithm 11. We will next discuss the form of the learned loss function, $\mathcal{L}_\psi$, which is critical for effective learning.

### 7.2.3 *Learned Temporal Adaptation Objectives*

To learn from a video of a human, we need an adaptation objective that can effectively capture relevant information in the video, such as the intention of the human and the

**Algorithm 10** Meta-imitation learning from humans

**Require:** $\{(\mathcal{D}^h_{\mathcal{T}_i}, \mathcal{D}^r_{\mathcal{T}_i})\}$: human and robot demonstration data for a set of tasks $\{\mathcal{T}_i\}$ drawn from $p(\mathcal{T})$

**Require:** $\alpha$, $\beta$: inner and outer step size hyperparameters

1: **while** training **do**
2: Sample task $\mathcal{T} \sim p(\mathcal{T})$ ▷ or minibatch of tasks
3: Sample video of human $\tau^h \sim \mathcal{D}^h_{\mathcal{T}}$
4: Compute policy parameters $\phi_{\mathcal{T}} = \theta - \alpha \nabla_\theta \mathcal{L}_\psi(\theta, \tau^h)$
5: Sample robot demo $\tau^r \sim \mathcal{D}^r_{\mathcal{T}}$
6: Update $(\theta, \psi) \leftarrow (\theta, \psi) - \beta \nabla_{\theta,\psi} \mathcal{L}_{BC}(\phi_{\mathcal{T}}, \tau^r)$
7: **return** $\theta, \psi$

**Algorithm 11** Learning from human video after meta-learning

**Require:** meta-learned initial policy parameters $\theta$

**Require:** learned adaptation objective $\mathcal{L}_\psi$

**Require:** one video of human demo $\tau^h$ for new task $\mathcal{T}$

1: Compute policy parameters $\phi_{\mathcal{T}} = \theta - \alpha \nabla_\theta \mathcal{L}_\psi(\theta, \tau^h)$
2: **return** $\pi_\phi$

task-relevant objects. While a standard behavior cloning loss is applied to each time step independently, the learned adaptation objective must solve a harder task, since it must provide the policy with suitable gradient information *without* access to the true actions. As discussed previously, this is still possible, since the policy is trained to output good actions during meta-training. The learned loss must simply supply the gradients that are needed to modify the perceptual components of the policy to attend to the right objects in the scene, so that the action output actually performs the right task. However, determining which behavior is being demonstrated and which objects are relevant will often require examining multiple frames at the same time to determine the human's motion. To incorporate this temporal information, our learned adaptation objective therefore couples multiple time steps together, operating on policy activations from multiple time steps.

Since temporal convolutions have been shown to be effective at processing temporal and sequential data (Van Den Oord et al., 2016), we choose to adopt a convolutional network to represent the adaptation objective $\mathcal{L}_\psi$, using multiple layers of 1D convolutions over time. We choose to use temporal convolutions over a more traditional recurrent

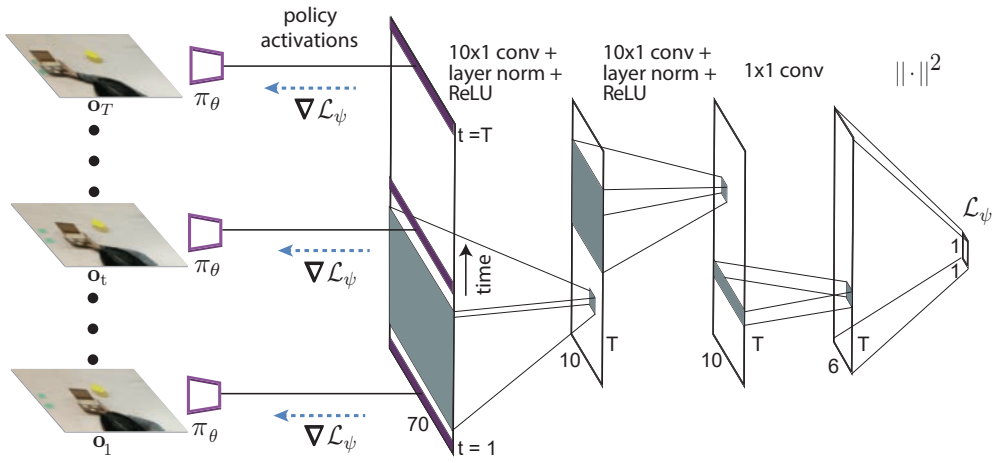


Figure 28: Visualization of the learned adaptation objective, which uses temporal convolutions to integrate temporal information in the video demonstration.

neural network like an LSTM, since they are simpler and usually more parameter efficient (Van Den Oord et al., 2016). See Figure 28 for a visualization.

Prior work introduced a two-head architecture for one-shot imitation, with one head used for the pre-update demonstration and one head used for the post-update policy (Finn et al., 2017b). The two-head architecture can be interpreted as a learned linear loss function operating on the last hidden layer of the policy network for a particular timestep. The loss and the gradient are then computed by averaging over all timesteps in the demonstration. As discussed previously, a single timestep of an observed video is often not sufficient for learning from video demonstrations without actions. Thus, this simple averaging scheme is not effective at integrating temporal information. In Section 7.2.7, we show that our learned temporal loss can enable effective learning from demonstrations without actions, substantially outperforming this single-timestep linear loss.

### 7.2.4 *Probabilistic Interpretation*

One way to interpret meta-learning with learned adaptation objectives is by casting it into the framework of probabilistic graphical models. We can accomplish this by building on a derivation proposed in prior work (Grant et al., 2018), which frames MAML as approximately inferring a posterior over policy parameters $\phi$ given the evidence $\mathcal{D}_{\mathcal{T}}^{\text{tr}} =$

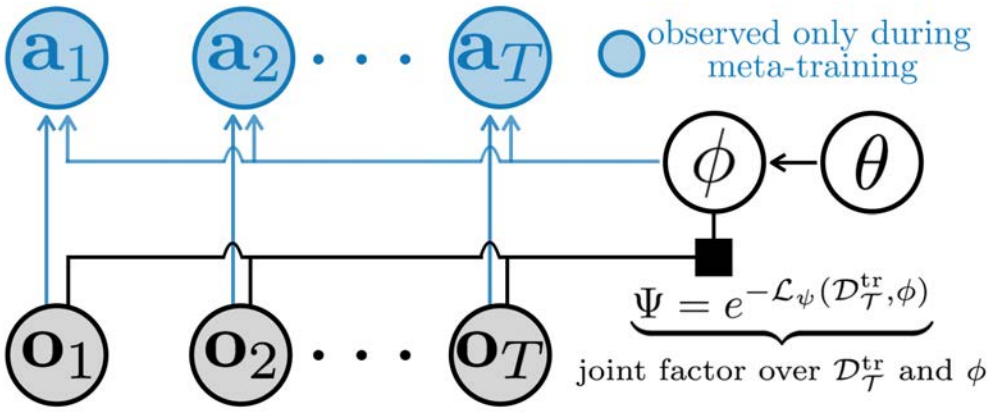


Figure 29: Graphical model underlying our approach. During meta-training, both the observations $\mathbf{o}_t$ and the actions $\mathbf{a}_t$ are observed, and our method learns $\theta$ and $\Psi$. During meta-testing, only the observations are available, from which our method combines with the learned prior $\theta$ and factor $\Psi$ to infer the task-specific policy parameters $\phi$.

$\tau_\mathcal{T}^h$ (the data for a new task $\mathcal{T}$) and a prior over the parameters, given by $\theta$. This prior work shows that a few steps of gradient descent on the likelihood $\log p(\mathcal{D}_\mathcal{T}^{\text{tr}}|\phi)$ starting from $\phi = \theta$ are approximately equivalent to maximum a posteriori (MAP) inference on $\log p(\phi|\mathcal{D}_\mathcal{T}^{\text{tr}}, \theta)$, where $\theta$ induces a Gaussian prior on the weights with mean $\theta$ and a covariance that depends on the step size and number of gradient steps.[5] The derivation is outside of the scope of this work, and we refer the reader to prior work for details (Grant et al., 2018; Santos, 1996).

In our approach, adaptation involves gradient descent on the learned loss $\mathcal{L}_\psi(\phi, \mathcal{D}_\mathcal{T}^{\text{tr}})$, rather than the likelihood $\log p(\mathcal{D}_\mathcal{T}^{\text{tr}}|\phi)$. Since we still take a fixed number of steps of gradient descent starting from $\theta$, the result in prior work still implies that we are approximately imposing the Gaussian prior $\log p(\phi|\theta)$ (Grant et al., 2018; Santos, 1996), and therefore are performing approximate MAP inference on the following joint distribution:

$$p(\phi|\mathcal{D}_\mathcal{T}^{\text{tr}}, \theta) \propto p(\phi, \mathcal{D}_\mathcal{T}^{\text{tr}}|\theta) \propto \underbrace{p(\phi|\theta)}_{\text{from GD}} \underbrace{\Psi(\phi, \mathcal{D}_\mathcal{T}^{\text{tr}})}_{\exp(-\mathcal{L}_\psi(\phi, \mathcal{D}_\mathcal{T}^{\text{tr}}))} .$$

This is a partially directed factor graph with a learned factor $\Psi$ over $\phi$ and $\mathcal{D}_\mathcal{T}^{\text{tr}}$ that has the log-energy $\mathcal{L}_\psi(\phi, \mathcal{D}_\mathcal{T}^{\text{tr}})$. Bayesian inference would require integrating out $\phi$, but MAP inference provides a tractable alternative that still produces good results in practice (Grant

5 This result is exact in the case of linear functions, and a local approximation in the case of nonlinear neural networks.

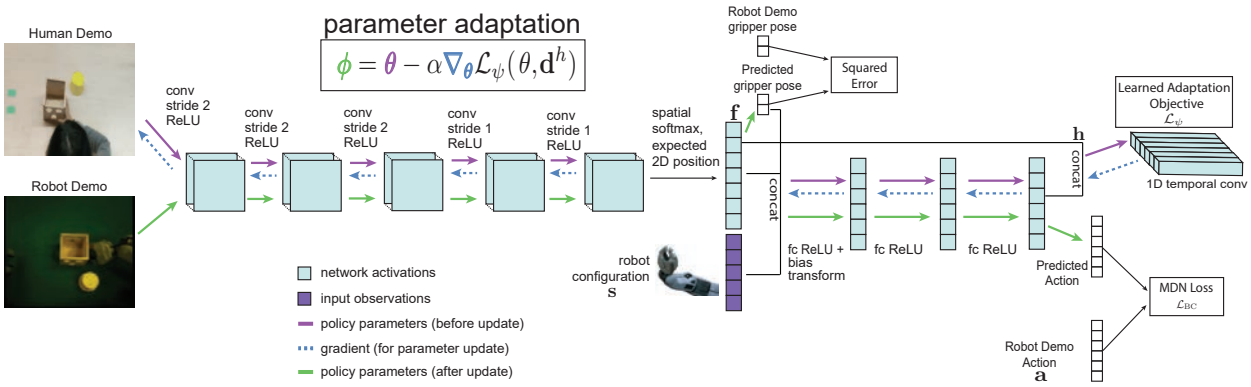


Figure 30: Illustration of the policy architecture. The policy consists of a sequence of five convolutional (conv) layers, followed by a spatial soft-argmax and fully-connected (fc) layers. The learned adaptation objective $\mathcal{L}_\psi$ is further illustrated in Figure 28. Best viewed in color.

et al., 2018). Training is performed by directly maximizing $\mathcal{L}_{BC}(\phi_\mathcal{T}, \mathcal{D}_\mathcal{T}^{tr})$, where $\phi_\mathcal{T}$ is the MAP estimate of $\phi$. Since the behavior cloning loss corresponds to the log likelihood of the actions under a Gaussian mixture policy, we directly train both the prior $\theta$ and the log-energy $\mathcal{L}_\psi$ such that MAP inference maximizes the log probability of the actions. Note that, since we use MAP inference during training, the model does not necessarily provide well-calibrated probabilities. However, the probabilistic interpretation still helps to shed light on the role of the learned adaptation objective $\mathcal{L}_\psi$, which is to induce a joint factor on the observations in $\mathcal{D}_\mathcal{T}$ and the policy parameters $\phi$. A visual illustration of the corresponding graphical model is shown in Figure 29.

### 7.2.5 *Model Architectures*

Now that we have presented our approach, we describe the form of the policy $\pi$ and the learned adaptation objective $\mathcal{L}_\psi$.

#### 7.2.5.1 *Policy Architecture*

As illustrated in Figure 30, the policy architecture is a convolutional neural network that maps from RGB images to a distribution over actions. The convolutional network begins with a few convolutional layers, which are fed into a channel-wise spatial soft-argmax that extracts 2D feature points $\mathbf{f}$ for each channel of the last convolution layer (Levine

et al., 2016a). Prior work has shown that the spatial soft-argmax is particularly effective and parameter-efficient for learning to represent the positions of objects in robotics domains (Levine et al., 2016a; Finn et al., 2016b). Following prior work (Levine et al., 2016a), we concatenate these feature points with the robot configuration, which consists of the pose of the end-effector represented by the 3D position of 3 non-axis-aligned points on the gripper. Then, we pass the concatenated feature points and robot pose into multiple fully connected layers. The distribution over actions is predicted linearly from the last hidden layer $\mathbf{h}$. We initialize the first convolutional layer from that of a network trained on ImageNet.

In our experiments, we will be using a continuous action space over the linear and angular velocity of the robot's gripper and a discrete action space over the gripper open/-close action following the setup of T. Zhang et al. (2017). Gaussian mixtures can better model multi-modal action distributions compared to Gaussian distributions and has been used in previous imitation learning works (Rahmatizadeh et al., 2018). Thus, for the continuous actions, we use a mixture density network (Bishop, 1994) to represent the output distribution over actions. For the discrete action of opening or closing the gripper, we use a sigmoid output with a cross-entropy loss.

Following prior work (T. Zhang et al., 2017), we additionally have the model predict the pose of the gripper when it contacts the target object and/or container. This is part of the outer meta-objective, and we can easily provide supervision using the robot demonstration. Note that this supervision is not needed at meta-test time when the robot is learning from a video of a human. For placing and pick-and-place tasks, the target container is located at the final end-effector pose. Thus, we use the last end-effector pose as supervision. For pushing and pick-and-place, the demonstrator manually specifies the time at which the gripper initially contacts the object and the end-effector pose at that time step is used. The model predicts this intermediate gripper pose linearly from the feature points $\mathbf{f}$, and the predicted pose is fed back into the policy. Further architecture details are included in Section 7.2.7.

#### 7.2.5.2 *Learned Adaptation Objective Architecture*

Because we may need to update both the policy's perception and control, the adaptation objective will operate on a concatenation of the predicted feature points, $\mathbf{f}$ (at the end of the perception layers), and the final hidden layer of the policy, $\mathbf{h}$ (at the end of the control layers). This allows the learned loss to more directly adapt the weights in the convolutional layers, bypassing the control layers. The updated task parameters are computed

using our temporal adaptation objective,

$$\phi = \theta - \alpha \nabla_\theta \mathcal{L}_\psi(\theta, \tau^h),$$

where we will decompose the objective into two parts: $\mathcal{L}_\psi = \mathcal{L}_{\psi_1}(\mathbf{f}_{1:T}) + \mathcal{L}_{\psi_2}(\mathbf{h}_{1:T})$ We use the same architecture for $\mathcal{L}_{\psi_1}$ and $\mathcal{L}_{\psi_2}$, which is illustrated in Figure 28. The learned objective consists of three layers of temporal convolutions, the first two with $10 \times 1$ filters and the third with $1 \times 1$ filters. The $\ell_2$ norm of the output of the convolutions is computed to produce the scalar objective value.

### 7.2.6 *Related Work*

Most imitation learning and learning from demonstration methods operate at the level of configuration-space trajectories (Schaal et al., 2003; Argall et al., 2009), which are typically collected using kinesthetic teaching (Pastor et al., 2011; Akgun et al., 2012), teleoperation (Calinon et al., 2009; Rahmatizadeh et al., 2017; T. Zhang et al., 2017), or sensors on the demonstrator (Ekvall and Kragic, 2004; Dillmann, 2004; Calinon and Billard, 2006; Kruger et al., 2010). Instead, can we allow robots to imitate just by watching the demonstrator perform the task? We focus on this problem of learning from one video demonstration of a human performing a task, in combination with human and robot demonstration data collected on other tasks. Prior work has proposed to resolve the correspondence problem by hand, for example, by manually specifying how human grasp poses correspond to robot grasps (Kjellstrom et al., 2008) or by manually defining how human activities or commands translate into robot actions (Yang et al., 2015; K. Lee et al., 2013; Nguyen et al., 2017; Ramirez-Amaro et al., 2015; Rothfuss et al., 2018). By utilizing demonstration data of how humans and robots perform each task, our approach learns the correspondence between the human and robot implicitly. Several prior approaches also explicitly represent the positions of the human's hands (J. Lee and Ryoo, 2017) or use carefully engineered pipelines for visual activity recognition (Ramirez-Amaro et al., 2015). In contrast to such approaches, which rely on precise hand detection or a prebuilt vision system, our approach is trained end-to-end, seeking to extract the aspects of the human's activity that are the most relevant for choosing actions. This places less demand on the vision system, requiring it only to implicitly deduce the task and how to accomplish it, rather than precisely tracking the human's body and nearby objects.

Other prior approaches have sought to solve the problem of learning from human demonstrations by explicitly determining the goal or reward underlying the human behavior (e.g. through inverse reinforcement learning), and then optimizing the reward

through reinforcement learning (RL). For example, Rhinehart and Kitani (2017) and Tow et al. (2017) learn a model that predicts the outcome of the human's demonstration from a particular scene. Similarly, other works have learned a reward function based on human demonstrations (Sermanet et al., 2017b; Stadie et al., 2017; Y. Liu et al., 2018; Sermanet et al., 2017a; Tai et al., 2018). Once the system has learned about the reward function or desired outcome underlying the given task, the robot runs some form of reinforcement learning to maximize the reward or to reach the desired outcome. This optimization typically requires substantial experience to be collected using the robot for each individual task. Other approaches assume a known model and perform trajectory optimization to reach the inferred goal (Muelling et al., 2017). Because all of these methods consider single tasks in isolation, they often require multiple human demonstrations of the new task (though not all, e.g. (Sermanet et al., 2017a; Muelling et al., 2017)). Our method only requires one demonstration of the new task setting and, at test time, does not require additional experience on the robot nor a known model. And, crucially, all of the data used in our approach is amortized across tasks, such that the amount of data needed for any given individual task is quite small. In contrast, these prior reward-learning methods only handle the single-task setting, where a considerable amount of data must be collected for an individual task.

Handling domain shift is a key aspect of this problem, with a shift in both the visual scene and the embodiment of the human/robot, including the degrees of freedom and the physics. Domain adaptation has received significant interest within the machine learning community, especially for varying visual domains (Aytar and Zisserman, 2011; Patel et al., 2015) and visual shift between simulation and reality (Viereck et al., 2017; Sadeghi and Levine, 2016). Many of these techniques aim to find a representation that is invariant to the domain (Fernando et al., 2013; Gong et al., 2013; Tzeng et al., 2014; Sadeghi and Levine, 2016; D. Li et al., 2018). Other approaches have sought to map datapoints from one domain to another (Shrivastava et al., 2017; Yoo et al., 2016; Bousmalis et al., 2017; You et al., 2017). The human imitation problem may involve developing invariances, for example, to the background or lighting conditions of the human and robot's environments. However, the physical correspondence between human and robot does not call for an invariant representation, nor a direct mapping between the domains. In many scenarios, a direct physical correspondence between robot and human poses might not exist. Instead, the system must implicitly recognize the goal of the human from the video and determine the appropriate action.

### 7.2.7 *Experiments*

Through our experiments, we aim to address three main questions: (1) Can our approach effectively learn a prior that enables the robot to learn to manipulate new objects after seeing just one video of a human? (2) Can our approach generalize to human demonstrations from a different perspective than the robot, on novel backgrounds, and with new human demonstrators? (3) How does the proposed approach compare to alternative approaches to meta-learning? In order to further understand our method and its applicability, we additionally evaluate it understand: (a) How important is the temporal adaptation objective? (b) Can our approach be used on more than one robot platform, and with either kinesthetic or teleoperated demonstrations for meta-training?

To answer these questions, we run our experiments primarily with a 7-DoF PR2 arm, with robot demonstrations collected via teleoperation, and RGB images collected from a consumer-grade camera (unless noted otherwise), and use a Sawyer robot with kinesthetic demonstrations to study (b). We compare the following meta-learning approaches:

- **contextual policy**: a feedforward network that takes as input the robot's observation and the final image of the human demo (to indicate the task), and outputs the predicted action.
- **DA-LSTM policy**: a recurrent network that directly ingests the human demonstration video and the current robot observation, and outputs the predicted robot action. This is a domain-adaptive version of the meta-learning algorithm proposed by Duan et al. (2017).
- **DAML, linear loss**: our approach with a linear per-timestep adaptation objective.
- **DAML, temporal loss**: our approach with the temporal adaptation objective described in Section 7.2.2.

All methods use a mixture density network (Bishop, 1994) to represent the action space, where the actions correspond to the linear and rotational velocity of the robot gripper, a 6-dimensional continuous action space. As discussed in Section 7.2.5, each network is also trained to predict the final end-effector pose using a mean-squared error objective. We train each policy using the Adam optimizer with default hyperparameters (D. Kingma and J. Ba, 2015). All methods use the same data and receive the same supervision. For measuring generalization from meta-training to meta-testing, we use held-out objects in all of our evaluations that were not seeing during meta-training, as illustrated in Figure 32, and new human demonstrators. We provide full experimental details, hyperparameters, architecture information, code for our method, and the supplementary

Figure 31: Example placing (left), pushing (middle), and pick-and-place (right) tasks, from the robot's perspective. The top row shows the human demonstrations used in Section 7.2.7.1, while the bottom shows the robot demonstration.

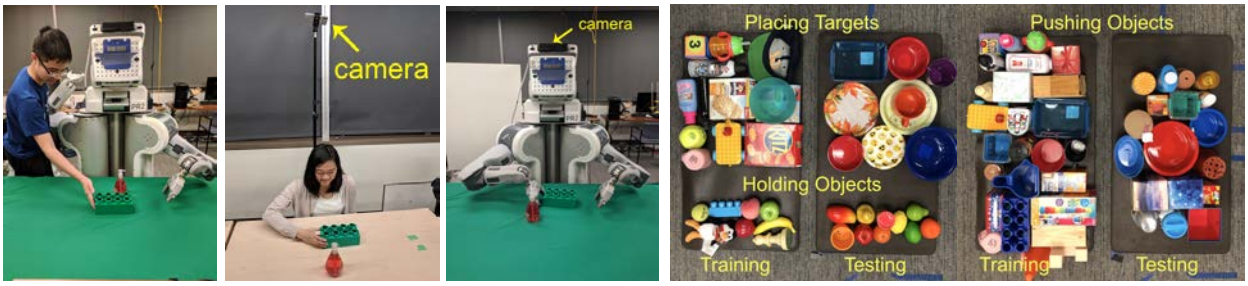


Figure 32: Left: The PR2 experimental set-up, from left to right showing the human demonstration set-up from Sections 7.2.7.1 and 7.2.7.2 respectively and the test-time set-up. Right: Subset of the objects used for training and evaluation. The robot must learn to recognize and maneuver the novel test objects using just one video of a human.

video on the project website[6].

#### 7.2.7.1 *PR2 Placing, Pushing, and Pick & Place*

We first consider three different task settings: placing a held object into a container while avoiding two distractor containers, pushing an object amid one distractor, and picking an object and placing it into a target container amid two distractor containers. The tasks are illustrated in Figure 31. In this initial experiment, we collect human demonstrations from the perspective of the robot's camera. For placing and pushing, we only use RGB images, whereas for pick-and-place, RGB-D is used. For meta-training, we collected a dataset with hundreds of objects, consisting of 1293, 640, and 1008 robot demonstrations for placing, pushing, and pick-and-place respectively, and an equal number of human demonstrations. The human and robot demonstrations for a given task need to have the

6 The project website is at sites.google.com/view/daml

same target object, but do not need to be synchronized, do not need the initial positions of objects match, and do not even need to have the same video length. We use the following metrics to define success for each task: for placing and pick & place, success if the object landed in or on any part of the correct container; for pushing, success if the correct item was pushed past or within ~5 cm of the robot's left gripper.

During evaluation, we used 15, 12, and 15 novel target objects for placing, pushing, and pick-and-place respectively, collected one human demonstration per object, and evaluated three trials of the policy inferred from the human demonstration. We report the results in Table 6. Our results show that, across the board, the robot is able to learn to interact with the novel objects using just one video of a human demo with that object, with pick-and-place being the most difficult task. We find that the DA-LSTM and contextual policies struggle, likely because they require more data to effectively infer the task. This finding is consistent with our previous findings (Section 7.1). Our results also indicate the importance of integrating temporal information when observing the human demonstration, as the linear loss performs poorly compared to using a temporal adaptation objective.

| | placing | pushing | pick and place |
|---|---|---|---|
| DA-LSTM | 33.3% | 33.3% | 5.6% |
| contextual | 36.1% | 16.7% | 16.7% |
| DAML, linear loss | 76.7% | 27.8% | 11.1% |
| DAML, temporal loss (ours) | **93.8%** | **88.9%** | **80.0%** |

Table 6: One-shot success rate of PR2 robot placing, pushing, and pick-and-place, using human demonstrations from the perspective of the robot. Evaluated using held-out objects and a novel human demonstrator.

#### 7.2.7.2 *Demonstrations with Large Domain Shift*

Now, we consider a challenging setting, where human demonstrations are collected in a different room with a different camera and camera perspective from that of the robot. As a result, the background and lighting vary substantially from the robot's environment. We use a mounted cell-phone camera to record sequences of RGB images on ten different table textures, as illustrated in the left of Figure 32. The corresponding view of the demonstrations is shown in Figure 33. We consider the pushing task, as described

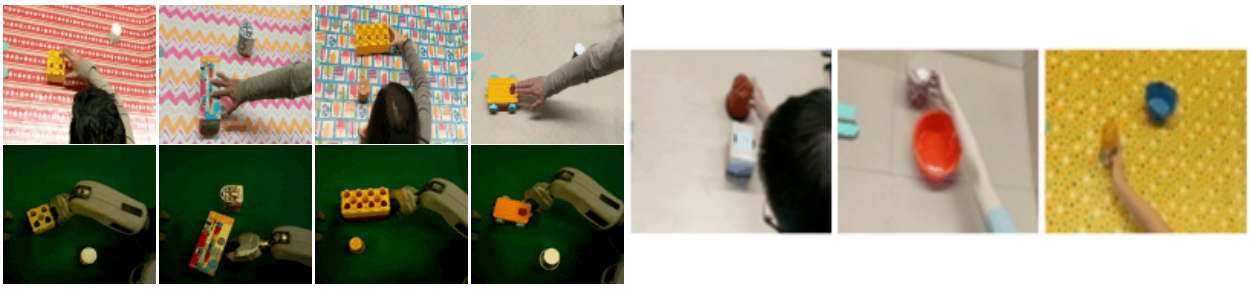

Figure 33: Left: Human and robot demonstrations used for meta-training for the experiments in Section 7.2.7.2 with large domain shift. We used ten different diverse backgrounds for collecting human demonstrations. Right: Frames from the human demos used for evaluation in Section 7.2.7.2, illustrating the background scenes. The leftmost background was in the meta-training set (seen bg), whereas the right two backgrounds are novel (novel bg1 and novel bg2). The objects and human demonstrator are novel.

in Section 7.2.7.1, reusing the same robot demonstrations and collecting an equal number of new demonstrations. We evaluate performance on novel objects, a new human demonstrator, and with one seen and two novel backgrounds, as shown in the right of Figure 33.

Like the previous experiment, we evaluated with 12 novel objects and 3 trials per object. Because we used different object pairs from the previous pushing experiment, the performance is not directly comparable to the results in Table 6. The results for this experiment are summarized in Table 7. As seen in the supplementary video, we find that the robot is able to successfully learn from the demonstrations with a different viewpoint and background. Performance degrades when using a novel background, which causes a varied shift in domain, but the robot is still able to perform the task about 70% of the time. In Table 7, we also include an analysis of the failure modes of our approach, including the number of failures caused by incorrect task identification – misidentifying the object – versus control failures – when the object was clearly correctly identified, but the robot failed to effectively push it. We see that, when the human demonstrations are on a previously seen background, the robot only fails to identify the object once out of 33 trials, whereas failures of this kind are more frequent on the novel backgrounds. Collecting data on a more diverse array of backgrounds, or using some form of background augmentation would likely reduce these types of failures. The number of control failures is similar for all backgrounds, likely indicative of the challenge of physically maneuvering a variety of previously unseen objects.

| pushing | seen bg | novel bg 1 | novel bg 2 |
|---|---|---|---|
| DAML, temporal loss (ours) | **81.8%** | **66.7%** | **72.7%** |

| Failure analysis of DAML | seen bg | novel bg 1 | novel bg 2 |
|---|---|---|---|
| # successes | 27 | 22 | 24 |
| # failures from task identification | 1 | 5 | 4 |
| # failures from control | 5 | 6 | 5 |

Table 7: Top: One-shot success rate of PR2 robot pushing, using videos of human demonstrations in a different scene and camera, with seen and novel backgrounds. Evaluated using held-out objects and a novel human. Bottom: Breakdown of the failure modes of our approach.

#### 7.2.7.3 *Sawyer Experiments*

The goal of this experiment is to evaluate the generality of our method on a different robot and a different form of robot demonstration collection. We will use a 7-DoF Sawyer robot (see Figure 34), and use kinesthetic teaching to record the robot demonstrations, which introduces additional challenges due to the presence of the human demonstrator in the recorded images. The human demonstrations are collected from the perspective of the robot. We consider the placing task described in Section 7.2.7.1. Unlike the PR2 experiments, the action space will be a single commanded pose of the end-effector and we will use mean-squared error for the outer meta-objective. Since we have thoroughly compared our method on the PR2 benchmarks, we only evaluate our proposed method in this experiment. We evaluated our method using 18 held-out objects and 3 trials per object. The result was a 77.8% placing success rate, indicating that our method can successfully be applied to the Sawyer robot and can handle kinesthetic demonstrations during meta-training.

#### 7.2.7.4 *Learned Adaptation Objective Ablation*

Finally, we study the importance of our proposed temporal adaptation objective. In order to isolate just the temporal adaptation loss, we perform this experiment in simulation, in a setting without domain shift. We use the simulated pushing task proposed by Finn et al. (2017b) in the MuJoCo physics engine (Todorov et al., 2012). To briefly summarize the experimental set-up, the imitation problem involves controlling a 7-DoF robot arm

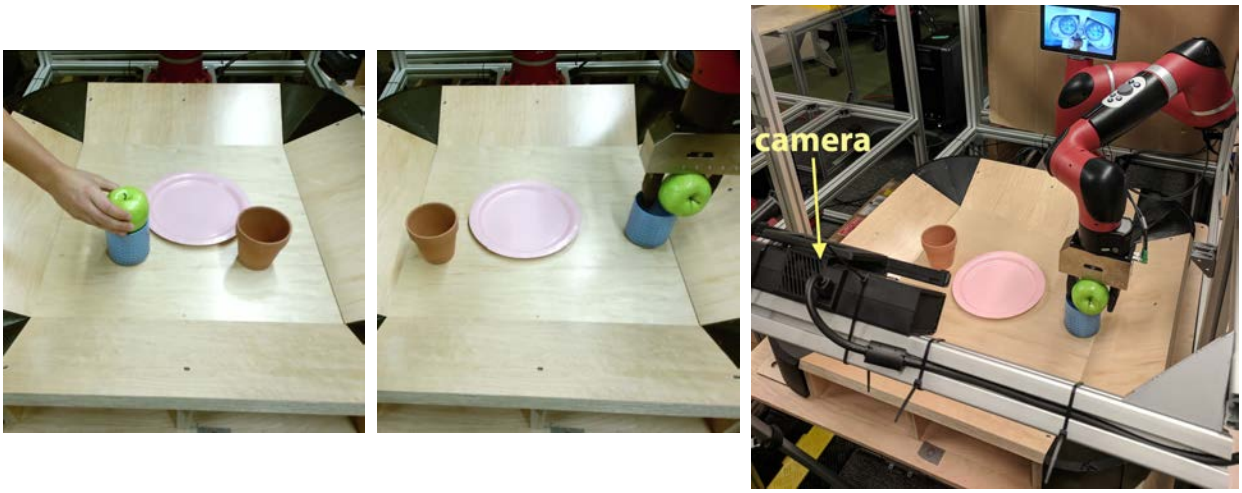


Figure 34: Sawyer robot set-up. From left the right: a human demo from the robot's perspective, the policy execution from the robot's perspective, and an photo illustrating the experimental set-up.

via torque-control to push one object to a fixed target position amid one distractor, using RGB images as input. The initial positions of the objects are randomized, as is the texture, shape, size, mass, and friction. Meta-training uses 105 object meshes, while 11 held-out meshes and multiple held-out textures are used for meta-testing. A push is considered successful if the target object lands on the target position for at least 10 timesteps within the 100-timestep episode. The results in Table 8 demonstrate a 14% absolute improvement in success by using a temporal adaptation objective, indicating the importance of integrating temporal information when learning from raw video.

### 7.2.8 *Discussion*

We presented an approach that enables a robot learning to visually recognize and manipulate a new object after observing just one video demonstration from a human user. To enable this, our method uses a meta-training phase where it acquires a rich prior over human imitation, using both human and robot demonstrations involving other objects. Our method extends a prior meta-learning approach to allow for learning cross-domain correspondences and includes a temporal adaptation loss function. Our experiments demonstrate that, after meta-learning, robots can acquire vision-based manipulation skills for a new object using from video of a human demonstrator in a substantially different

| | simulated pushing no domain shift |
|---|---|
| LSTM (Duan et al., 2017) | 34.23% |
| contextual | 56.98% |
| MIL, linear loss (Finn et al., 2017b) | 66.44% |
| MIL, temporal loss (ours) | **80.63%** |

Table 8: One-shot success rate of simulated 7-DoF pushing using video demonstrations with no domain shift

setting.

**Limitations**: While our method enables one-shot learning for manipulating new objects from one video of a human, our current experiments do not yet demonstrate the ability to learn entirely new motions in one shot. The behaviors at meta-test time are structurally similar to those observed at meta-training time, though they may involve previously unseen objects and demonstrators. We expect that more data and a higher-capacity model would likely help enable such an extension. However, we leave this to future work.

An additional challenge with our approach is the amount of demonstration data that is needed for meta-training. In our experiments, we used a few thousand demonstrations from robots and humans. However, the total amount of data *per-object* is quite low (around 10 trials), which is one or two orders of magnitude less than the number of demonstrations per-object used in recent single-task imitation learning works (Rahmatizadeh et al., 2018; T. Zhang et al., 2017). Thus, in settings where we need robots that can adapt to a diverse range of objects, our approach is substantially more practical than prior work.

**Beyond Human Imitation:** While our experiments focus on imitating humans, the proposed method is not specific to perceiving humans, and could also be used, for example, for imitating animals or a simulated robot, for simulation to real world transfer. Beyond imitation, we believe our approach is likely more broadly applicable to problems that involve inferring information from out-of-domain data, such as one-shot object recognition from product images, a problem faced by teams in the Amazon Robotics Challenge (Zeng et al., 2018).

# 8

# FEW-SHOT INTENT INFERENCE

In the previous chapter, we discussed how we can combine meta-learning with imitation for learning a policy from a few demonstrations. These methods couple the process of inferring the intent of the demonstrator with learning to perform the task. While this approach has a number of advantages, i.e. that it does not require trial-and-error experience and it yields results on challenging problems, there are a few disadvantages to the approach. First, there are settings where it is possible to infer the task from one demonstration, but where it is *not* possible to learn how to perform the task without any trial-and-error. In these settings, one-shot imitation would not be effective. Second, if the robot fails at imitating, it is difficult for a human to judge if the failure was caused by intent inference (where more human demonstrations should help), or if the intent was clear but the execution was lacking (where more robot experience should help).

Beyond practicalities, humans very clearly have an ability to mentally represent an objective and what it means to accomplish a task. This ability is a critical aspect of autonomously learning complex skills, as it is the driver of learning progress. If we aim to build agents that can autonomously learn new skills in real-world environments, where external feedback comes rarely, then we must develop agents that can build an internal understanding of its goals and a general mechanism for humans to convey these goals.

In this chapter, we present approaches for inferring intent from a few demonstrations (Section 8.1) and a few observations of success (Section 8.2), and show how we can use reinforcement learning or planning to optimize for accomplishing inferred reward function or goal. These approaches are motivated by the difficulty of conveying goals to artificial agents, as we discuss next.

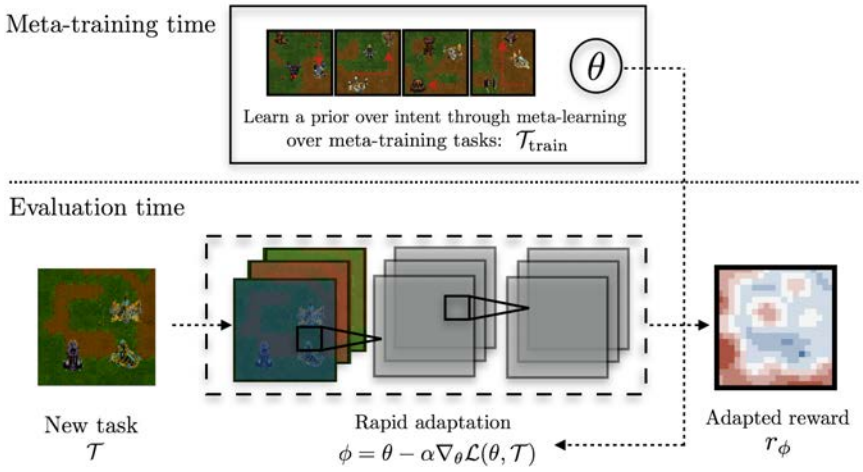


Figure 35: Diagram of our meta-inverse RL approach. Our approach attempts to remedy overfitting in few-shot IRL by learning a "prior" that constraints the set of possible reward functions to lie within a few steps of gradient descent. Standard IRL attempts to recover the reward function directly from demonstrations. In settings with only small numbers of demonstration, there is little reason to expect generalization from standard IRL, as it is analogous to training a density model with only a few examples.

## 8.1 Learning a prior over intent via meta inverse reinforcement learning

A key assumption of the reinforcement learning problem statement is the availability of a reward function that accurately describes the desired tasks. For many real world tasks, reward functions can be challenging to hand specify (e.g. encouraging a robot to be polite), while being crucial to good performance. Part of the challenge stems from the fact that most real world tasks are multifaceted and require reasoning over multiple factors in a task, while simultaneously providing appropriate reward shaping to make the task feasible with tractable exploration (Ng et al., 1999). These challenges are compounded by the inherent difficulty of specifying rewards for tasks with high-dimensional observation spaces such as images. Even for relatively simple skills such as pouring or opening a door, prior works have hand-designed mechanisms to measure a proxy for the objective.

Inverse reinforcement learning (IRL) is an approach that aims to address this problem by instead inferring the reward function from demonstrations of the task (Ng and Russell, 2000). This has the appealing benefit of taking a data-driven approach to reward

specification in place of cumbersome hand engineering. In practice, however, it can be prohibitively expensive to provide demonstrations that cover the variability common in real world tasks (e.g., collecting a dataset of demonstrations of opening every type of door). In addition, while learning a complex (reward) function from high dimensional observations might make an expressive function approximator seem like a reasonable modelling assumption, it can be difficult to unambiguously recover a good reward function with expressive function approximators from a limited set of demonstrations. Prior solutions to this problem have instead relied on low-dimensional linear models with handcrafted features that effectively encode a strong prior on the relevant features of a task. Yet, this requires engineering a set of features by hand that work well for a specific problem. In this work, we instead propose a method that explicitly optimizes for expressive features that are robust even when learning with limited demonstrations.

Our approach relies on the key observation that related tasks share common structure that we can encode by learning a "prior". To illustrate, considering a robot navigating through a home. While the exact reward function we provide to the robot may differ depending on the task, there is a structure amid the space of useful behaviours, such as navigating to a series of landmarks, and there are certain behaviors we *always* want to encourage or discourage, such as avoiding obstacles or staying a reasonable distance from humans. This notion agrees with our understanding of why humans can easily infer the intents and goals (i.e., reward functions) of even abstract agents from just one or a few demonstrations (Baker et al., 2007), as humans have access to strong priors about how other humans accomplish similar tasks accrued over many years. Hence, our objective is to learn a "prior over intent" by discovering and encoding this common structure in the reward function parameters.

More specifically, in this work we assume access to a set of tasks, along with demonstrations of the desired behaviors for those tasks, which we refer to as the *meta-training set*. From these tasks, we then learn a reward function parameterization that enables effective few-shot learning when used to initialize IRL in a novel task. Our method is summarized in Fig. 35. Our key contribution is an algorithm that enables efficient learning of new reward functions by using meta-training to build a rich "prior" for goal inference. From an empirical perspective, we present experiments that show that our method is able to recover a reward function from raw pixels that better generalizes to new environments, as well as substantially improves data efficiency when faced with a new task, in comparison to existing methods and standard baselines.

### 8.1.1 *Preliminaries and Overview*

In this section, we introduce our notation and describe the IRL and meta-learning problems.

#### 8.1.1.1 *Learning Rewards via Inverse Reinforcement Learning*

The standard Markov decision process (MDP) is defined by the tuple $(\mathcal{S}, \mathcal{A}, p_{\mathbf{s}}, R, \gamma)$ where $\mathcal{S}$ and $\mathcal{A}$ denote the set of possible states and actions respectively, $R$ is the scalar reward, $\gamma \in [0, 1]$ is the discount factor and $p_{\mathbf{s}} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ denotes the transition distribution over the next state $\mathbf{s}_{t+1}$, given the current state $\mathbf{s}_t$ and current action $\mathbf{a}_t$. Typically, the goal of "forward" RL is to maximize the expected discounted return $R(\tau) = \sum_{t=1}^{T} \gamma^{t-1} r(\mathbf{s}_t, \mathbf{a}_t)$.

In IRL, we instead assume that the reward function is unknown but that we instead have access to a set of expert demonstrations $\mathcal{D} = \{\tau_1, \ldots, \tau_K\}$, where $\tau_k = \{\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T\}$.

The goal of IRL is to recover the unknown reward function $R$ from the set of demonstrations. We build on the maximum entropy (MaxEnt) IRL framework by Ziebart et al. (2008), which models the probability of the trajectories as being distributed proportional to their exponentiated return

$$p(\tau) = \frac{1}{Z} \exp\left(R(\tau)\right), \tag{22}$$

where $Z$ is the partition function, $Z = \int_\tau \exp(R(\tau)) d\tau$. This distribution can be shown to be induced by the optimal policy in entropy regularized forward RL problem:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_{\tau \sim \pi} \left[R(\tau) - \log \pi(\tau)\right]. \tag{23}$$

This formulation allows us to pose the reward learning problem as a maximum likelihood estimation (MLE) problem in an energy-based model $R_\phi$:

$$\min_\phi \mathbb{E}_{\tau \sim \mathcal{D}} \left[\mathcal{L}_{\text{IRL}}(\tau)\right] = \min_\phi \mathbb{E}_{\tau \sim \mathcal{D}} \left[-\log p_\phi(\tau)\right]. \tag{24}$$

Learning in general energy-based models of this form is common in many applications such as structured prediction. However, in contrast to applications where learning can be supervised by millions of labels (e.g. semantic segmentation), the learning problem in Equation 24 must typically be performed with a relatively small number of example demonstrations. In this work, we seek to address this issue in IRL by providing a way to integrate information from prior tasks to constrain the optimization in Equation 24 in the regime of limited demonstrations.

### 8.1.2 *Learning to Learn Rewards*

Our goal in meta-IRL is to learn how to learn reward functions across many tasks such that the model can infer the reward function for a new task using only one or a few expert demonstrations. Intuitively, we can view this problem as aiming to learn a prior over the intentions of human demonstrators, such that when given just one or a few demonstrations of a new task, we can combine the learned prior with the new data to effectively determine the human's reward function. Such a prior is helpful in inverse reinforcement learning settings, since the space of relevant reward functions is much smaller than the space of all possible rewards definable on the raw observations.

During meta-training, we have a set of tasks $\{\mathcal{T}_i \; ; \; i = 1..N\}$. Each task $\mathcal{T}_i$ has a set of demonstrations $\mathcal{D}_\mathcal{T} = \{\tau_1, \ldots, \tau_K\}$ from an expert policy which we partition into disjoint $\mathcal{D}_\mathcal{T}^{tr}$ and $\mathcal{D}_\mathcal{T}^{test}$ sets. The demonstrations for each meta-training task are assumed to be produced by the expert according to the maximum entropy model in Section 8.1.1.1. During meta-training, these tasks will be used to encodes common structure so that our model can quickly acquire rewards for new tasks from just a few demonstrations. After meta-training, our method is presented with a new task. During this meta-test phase, the algorithm must infer the parameters of the reward function $r_\phi(\mathbf{s}_t, \mathbf{a}_t)$ for the new task from a few demonstrations. As is standard in meta-learning, we assume that the test task is from the same distribution of tasks seen during meta-training, a distribution that we denote as $p(\mathcal{T})$.

#### 8.1.2.1 *Meta Reward and Intention Learning (MandRIL)*

In order to meta-learn a reward function that can act as a prior for new tasks and new environments, we first formalize the notion of a good reward by defining a loss $\mathcal{L}(\theta, \mathcal{D}_\mathcal{T})$ on the reward function $r_\theta$ for a particular task $\mathcal{T}$. We use the MaxEnt IRL loss $\mathcal{L}_{IRL}$ discussed in Section 8.1.1, which, for a given $\mathcal{D}_\mathcal{T}$, leads to the following gradient (Ziebart et al., 2008):

$$\nabla_\theta \mathcal{L}_{IRL}(\theta, \mathcal{D}_\mathcal{T}) = \frac{\partial r_\theta}{\partial \theta} \left[ \mathbb{E}_\tau[\mu_\tau] - \mu_{\mathcal{D}_\mathcal{T}} \right]. \tag{25}$$

where $\mu_\tau$ are the state visitations under the optimal maximum entropy policy under $r_\theta$, and $\mu_\mathcal{D}$ are the mean state visitations under the demonstrated trajectories.

If our end goal were to achieve a single reward function that works as well as possible across all tasks in $\{\mathcal{T}_i \; ; \; i = 1..N\}$, then we could simply follow the *mean* gradient across all tasks. However, our objective is subtly different: instead of optimizing performance

**Algorithm 12** Meta Reward and Intention Learning (MandRIL)

**Input:** Set of meta-training tasks $\{\mathcal{T}\}^{\text{meta-train}}$
**Input:** learning hyper-parameters $\alpha, \beta$
**function** MaxEntIRL-Grad($r_\theta$, $\mathcal{T}$, $\mathcal{D}$)( ▷ )Single task update
  $\boldsymbol{\mu}_{\mathcal{D}} = \text{State-Visitations-Traj}(\mathcal{T}, \mathcal{D})$
  $\mathbb{E}_\tau[\boldsymbol{\mu}_\tau] = \text{State-Visitations-Policy}(r_\theta, \mathcal{T})$
  $\frac{\partial \mathcal{L}_{\text{IRL}}}{\partial r_\theta} = \mathbb{E}_\tau[\boldsymbol{\mu}_\tau] - \boldsymbol{\mu}_{\mathcal{D}}$ ▷ MaxEntIRL gradient (Ziebart et al., 2008)
  **Return** $\frac{\partial \mathcal{L}_{\text{IRL}}}{\partial r_\theta}$

Randomly initialize $\theta$
**while** not done **do**
  Sample batch of tasks $\mathcal{T}_i \sim \{\mathcal{T}\}^{\text{meta-train}}$
  **for all** $\mathcal{T}_i$ **do**
    Sample demos $\mathcal{D}^{\text{tr}}_{\mathcal{T}_i} = \{\tau_1, \ldots, \tau_K\} \sim \mathcal{D}_{\mathcal{T}_i}$ ▷ Inner loss computation
    $\frac{\partial \mathcal{L}_{\text{IRL}}(\theta, \mathcal{D}^{\text{tr}}_{\mathcal{T}_i})}{\partial r_\theta} = \text{MaxEntIRL-Grad}(r_\theta, \mathcal{T}_i, \mathcal{D}^{\text{tr}}_{\mathcal{T}_i})$
    Compute $\nabla_\theta \mathcal{L}_{\text{IRL}}(\theta, \mathcal{D}^{\text{tr}}_{\mathcal{T}_i})$ from $\frac{\partial \mathcal{L}_{\text{IRL}}(\theta, \mathcal{D}^{\text{tr}}_{\mathcal{T}_i})}{\partial r_\theta}$ via chain rule
    Compute updated parameters $\boldsymbol{\phi}_{\mathcal{T}_i} = \theta - \alpha \nabla_{\boldsymbol{\phi}} \mathcal{L}_{\text{IRL}}(\theta, \mathcal{D}^{\text{tr}}_{\mathcal{T}_i})$ ▷ Fast update
    Sample demos $\mathcal{D}^{\text{test}}_{\mathcal{T}_i} = \{\tau'_1, \ldots, \tau'_{K'}\} \sim \mathcal{D}_{\mathcal{T}_i}$ ▷ Outer loss computation
    $\frac{\partial \mathcal{L}_{\text{IRL}}(\boldsymbol{\phi}, \mathcal{D}^{\text{test}}_{\mathcal{T}_i})}{\partial r_\theta} = \text{MaxEntIRL-Grad}(r_{\boldsymbol{\phi}_{\mathcal{T}_i}}, \mathcal{T}_i, \mathcal{D}^{\text{test}}_{\mathcal{T}_i})$
    Compute $\nabla_\theta \mathcal{L}_{\text{IRL}}(\boldsymbol{\phi}, \mathcal{D}^{\text{test}}_{\mathcal{T}_i})$ via Eq 28 ▷ Compute meta-gradient
  Compute update to $\theta \leftarrow \theta - \beta \sum_i \nabla_\theta \mathcal{L}^{\text{test}}_{\mathcal{T}_i}$ ▷ Update initial parameters

on the meta-training tasks, we aim to learn a reward function that can be quickly and efficiently adapted to new tasks at meta-test time. In doing so, we aim to encode prior information over the task distribution in this learned reward prior.

Following MAML, we propose to implement such a learning algorithm by finding the parameters $\theta$, such that starting from $\theta$ and taking a small number of gradient steps on a few demonstrations from given task leads to a reward function for which a set of *test* demonstrations have high likelihood, with respect to the MaxEnt IRL model. In particular, we would like to find a $\theta$ such that the parameters

$$\boldsymbol{\phi}_{\mathcal{T}_i} = \theta - \alpha \nabla_\theta \mathcal{L}_{\text{IRL}}(\theta, \mathcal{D}^{\text{tr}}_{\mathcal{T}_i}) \tag{26}$$

lead to a reward function $r_{\boldsymbol{\phi}_{\mathcal{T}}}$ for task $\mathcal{T}$, such that the IRL loss (corresponding to negative

log-likelihood) for a disjoint set of test demonstrations is minimized. The corresponding optimization problem for $\theta$ can therefore be written as follows:

$$\min_{\theta} \sum_{i=1}^{N} \mathcal{L}_{\text{IRL}}(\boldsymbol{\phi}_{\mathcal{T}_i}, \mathcal{D}^{\text{test}}_{\mathcal{T}_i}) = \sum_{i=1}^{N} \mathcal{L}_{\text{IRL}}\left(\theta - \alpha \nabla_{\theta} \mathcal{L}_{\text{IRL}}(\theta, \mathcal{D}^{\text{tr}}_{\mathcal{T}_i}), \mathcal{D}^{\text{test}}_{\mathcal{T}_i}\right). \tag{27}$$

Our method acquires this prior $\theta$ over rewards in the task distribution $p(\mathcal{T})$ by optimizing this loss. This amounts to an extension of the MAML algorithm in Section 4.1 to the inverse reinforcement learning setting. This extension is quite challenging, because computing the MaxEnt IRL gradient requires repeatedly solving for the current maximum entropy policy and visitation frequencies, and the MAML objective requires computing derivatives *through* this gradient step. Next, we describe in detail how this is done. An overview of our method is also outlined in Algorithm 12.

**Meta-Training.** The computation of the meta-gradient for the objective in Equation 27 can be conceptually separated into two parts. First, we perform the update in Equation 26 by computing the *expected state visitations* $\mu$, which is the expected number of times an agent will visit each state. We denote this overall procedure as State-Visitations-Policy, and follow Ziebart et al. (2008) by first computing the maximum entropy optimal policy in Equation 23 under the current $r_\theta$, and then approximating $\mu$ using dynamic programming. Next, we compute the state visitation distribution of the expert using a procedure which we denote as State-Visitations-Traj. This can be done either empirically, by averaging the state visitation of the experts demonstrations, or by using State-Visitations-Policy if the true reward is available at meta-training time. This allows us to recover the IRL gradient according to Equation 25, which we can then apply to compute $\boldsymbol{\phi}_{\mathcal{T}}$ according to Equation 26.

Second, we need to differentiate through this update to compute the gradient of the meta-loss in Equation 27. Note that the meta-loss itself is the IRL loss evaluated with a different set of test demonstrations. We follow the same procedure as above to evaluate the gradient of $\mathcal{L}_{\text{IRL}}(\cdot, \mathcal{D}^{\text{test}}_{\mathcal{T}})$ with respect to the post-update parameters $\boldsymbol{\phi}_{\mathcal{T}}$, and then apply the chain rule to compute the meta-gradient:

$$\begin{aligned}
\nabla_{\theta} \mathcal{L}_{\text{IRL}}(\boldsymbol{\phi}_{\mathcal{T}}, \mathcal{D}^{\text{test}}_{\mathcal{T}}) &= \frac{\partial \mathcal{L}(\boldsymbol{\phi}_{\mathcal{T}}, \mathcal{D}^{\text{test}}_{\mathcal{T}})}{\partial r_{\boldsymbol{\phi}_{\mathcal{T}}}} \frac{\partial r_{\boldsymbol{\phi}_{\mathcal{T}}}}{\partial \boldsymbol{\phi}_{\mathcal{T}}} \frac{\partial}{\partial \theta} (\theta - \alpha \nabla_{\theta} \mathcal{L}(\theta, \mathcal{D}^{\text{tr}}_{\mathcal{T}})) \\
&= \frac{\partial \mathcal{L}(\boldsymbol{\phi}_{\mathcal{T}}, \mathcal{D}^{\text{test}}_{\mathcal{T}})}{\partial r_{\boldsymbol{\phi}_{\mathcal{T}}}} \frac{\partial r_{\boldsymbol{\phi}_{\mathcal{T}}}}{\partial \boldsymbol{\phi}_{\mathcal{T}}} \left( \mathbf{I} - \alpha \frac{\partial^2 \mathcal{L}_{\text{IRL}}(\theta, \mathcal{D}^{\text{tr}}_{\mathcal{T}})}{\partial \theta^2} - \alpha \frac{\partial \mathcal{L}(\theta, \mathcal{D}^{\text{tr}}_{\mathcal{T}})}{\partial \theta} \frac{\partial \mathcal{L}(\theta, \mathcal{D}^{\text{tr}}_{\mathcal{T}})}{\partial \theta}^{\top} \right),
\end{aligned} \tag{28}$$

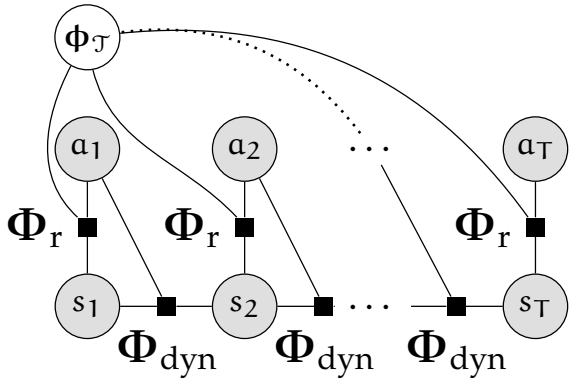


Figure 36: Our approach can be understood as approximately learning a distribution over the demonstrations $\tau$, in the factor graph $p(\tau) = \frac{1}{Z}\prod_{t=1}^{T} \mathbf{\Phi}_r(\boldsymbol{\phi}_{\mathcal{T}}, \mathbf{s}_t, \mathbf{a}_t)\mathbf{\Phi}_{dyn}(\mathbf{s}_{t+1}, \mathbf{s}_t, \mathbf{a}_t)$ (above) where we learn a prior over $\phi_{\mathcal{T}}$, which during meta-test is used for MAP inference over new expert demonstrations.

where on the second line we differentiate through the MaxEnt-IRL update (and we drop the IRL subscript on $\mathcal{L}$ for brevity). The derivation of this expression is somewhat more involved and provided in the supplementary Appendix B.5.

**Meta-Testing.** Once we have acquired the meta-trained parameters $\theta$ that encode a prior over $p(\mathcal{T})$, we can leverage this prior to enable fast, few-shot IRL of novel tasks in $\{\mathcal{T}_j \; ; \; j = 1..M\}$. For each task, we first compute the state visitations from the available set of demonstrations for that task. Next, we use these state visitations to compute the gradient, which is the same as the inner loss gradient computation of the meta-training loop in Algorithm 12. We apply this gradient to adapt the parameters $\theta$ to the new task. Even if the model was trained with only one to three inner gradient steps, we found in practice that it was beneficial to take substantially more gradient steps during meta-testing; performance continued to improve with up to 20 steps.

#### 8.1.2.2 *Interpretation as Learning a Prior over Intent*

The objective in Equation 26 optimizes for parameters that enable that reward function to adapt and generalize efficiently on a wide range of tasks. Intuitively, constraining the space of reward functions to lie within a few steps of gradient descent can be interpreted as expressing a "locality" prior over reward function parameters. This intuition can be made more concrete with the following analysis.

By viewing IRL as maximum likelihood estimation, we can take the perspective of Grant et al. (Grant et al., 2018) who showed that for a linear model, fast adaptation

via a few steps of gradient descent in MAML is performing MAP inference over $\phi$, under a Gaussian prior with the mean $\theta$ and a covariance that depends on the step size, number of steps and curvature of the loss. This is based on the connection between early stopping and regularization previously discussed by Santos (1996), which we refer the readers to for a more detailed discussion. The interpretation of MAML as imposing a Gaussian prior on the parameters is exact in the case of a likelihood that is quadratic in the parameters (such as the log-likelihood of a Gaussian in terms of its mean). For any non-quadratic likelihood, this is an approximation in a local neighborhood around $\theta$ (i.e. up to convex quadratic approximation). In the case of very complex parameterizations, such as deep function approximators, this is a coarse approximation and unlikely to be the mode of a posterior. However, we can still frame the effect of early stopping and initialization as serving as a prior in a similar way as prior work (Sjöberg and Ljung, 1995; Duvenaud et al., 2016; Grant et al., 2018). More importantly, this interpretation hints at future extensions to our approach that could benefit from employing more fully Bayesian approaches to reward and goal inference.

### 8.1.3 *Related Work*

Inverse reinforcement learning (IRL) (Ng and Russell, 2000) is the problem of inferring an expert's reward function directly from demonstrations. Prior methods for performing IRL range from margin based approaches (Abbeel and Ng, 2004; N. D. Ratliff et al., 2006) to probabilistic approaches (D. Ramachandran and Amir, 2007; Ziebart et al., 2008). Although it is possible to extend our approach to any other IRL method, in this work we base on work on the maximum entropy (MaxEnt) framework (Ziebart et al., 2008). In addition to allowing for sub-optimality in the expert demonstrations, MaxEnt-IRL can be re-framed as a familiar maximum likelihood problem in a particular factor graph (see Sec. 8.1.1.1).

In part to combat the under-specified nature of IRL, prior work has often used low-dimensional linear parameterizations with handcrafted features (Abbeel and Ng, 2004; Ziebart et al., 2008). In order to learn from high dimensional input, Wulfmeier et al. (2015) proposed applying fully convolutional networks (Shelhamer et al., 2017) to the MaxEnt IRL framework (Ziebart et al., 2008) for several navigation tasks (Wulfmeier et al., 2016a; Wulfmeier et al., 2016b). Other methods that have incorporated neural network rewards include guided cost learning (GCL) (Finn et al., 2016a), which uses importance sampling and regularization for scalability to high-dimensional spaces, and adversarial IRL (Fu et al., 2018). Several other methods have also proposed imitation learning approaches

based on adversarial frameworks that resemble IRL, but do not aim to directly recover a reward function (Ho and Ermon, 2016; Y. Li et al., 2017; Hausman et al., 2017; Kuefler and Kochenderfer, 2018). In this work, instead of improving the ability to learn reward functions on a single task, we focus on the problem of effectively learning to use prior demonstration data from other IRL tasks, allowing us to learn new tasks from a limited number demonstrations even with expressive non-linear reward functions.

Prior work has explored the problem of *multi-task* IRL, where the demonstrated behavior is assumed to have originated from multiple experts achieving different goals. Some of these approaches include those that aim to incorporate a shared prior over reward functions through extending the Bayesian IRL (D. Ramachandran and Amir, 2007) framework to the multi-task setting (Dimitrakakis and Rothkopf, 2012; Choi and K.-E. Kim, 2012). Other approaches have clustered demonstrations while simultaneously inferring reward functions for each cluster (Babeş-Vroman et al., 2011) or introduced regularization between rewards to a common "shared reward" (K. Li and Burdick, 2017). Our work is similar in that we also seek to encode prior information common to the tasks. However, a critical difference is that our method specifically aims to distill the meta-training tasks into a prior that can then be used to learn rewards for *new* tasks efficiently. The goal therefore is not to acquire good reward functions that explain the meta-training tasks, but rather to use them to learn efficiently on new tasks.

### 8.1.4 *Experiments*

We have two complementary goals in our evaluation. We wish to test our core hypothesis that leveraging prior task information improves performance. In addition, from a practical perspective, we wish to test whether this improvement enables learning rewards for new tasks with just a few demonstrations, and to compare our approach with alternative meta-learning methods.

To test our core hypothesis, we implement the naïve approach of running MaxEnt IRL for each new task, i.e. where the reward function parameters are initialized randomly. While we expect this method to overfit, particularly when only a few demonstrations are available, it is an important baseline as it is the most straight-forward application of standard IRL to a multi-task setting. Most importantly, this comparison provides valuable insight into the benefits of incorporating prior information at all.

Regarding comparisons with alternative meta-learning approaches, there is no prior work that addresses the meta-inverse reinforcement learning problem introduced in this section. To provide a point of comparison and illustrate the difficulty of the tasks, we

Figure 37: An example task: When learning a task, the agent has access to the image (left) and demonstrations (red arrows). To evaluate the agent's learning (right), the agent is tested for its ability to recover the reward for the task when the objects have been rearranged. The reward structure we wish to capture can be illustrated by considering the initial state in blue. An policy acting optimally under a correctly inferred reward should interpret the other objects as obstacles, and prefer a path on the dirt in between them.

design two approaches based on alternative meta-learning methods, and adapt them to the IRL setting:

- **Demo Conditional Model**: Our method implicitly conditions on the demonstrations through the gradient descent update. In theory, a conditional deep model with sufficient capacity could implicitly implement a similar learning rule. Thus, we consider a conditional model (often referred to as a "contextual model"), which receives the demonstration as an additional input. This tests the generalization benefit of using gradient descent as an inductive bias versus explicit conditioning.
- **Recurrent Meta-Learner**: We additionally compare to an RNN-based meta-learner (Santoro et al., 2016; Duan et al., 2017). Specifically, we implement a conditional model by feeding both images and sequences of states visited by the demonstrations to an LSTM. The comparison between this approach and ours evaluates the importance of incorporating the IRL gradient into the meta-learning process, rather than learning the adaptation process entirely from scratch.

SPRITEWORLD NAVIGATION DOMAIN In our experimental evaluation, we consider a navigation domain where we aim to recover a reward function that describes trajectories from an agent moving through the environment. Specifically, we seek to learn a convolutional neural network that directly maps image pixels to rewards. To do so, we introduce "SpriteWorld," which is a synthetically generated task, some examples of which are shown in Fig. 37. The task visuals are inspired by Starcraft and work applying learning algorithms to perform micro-management (e.g. Synnaeve et al. (2016)) although

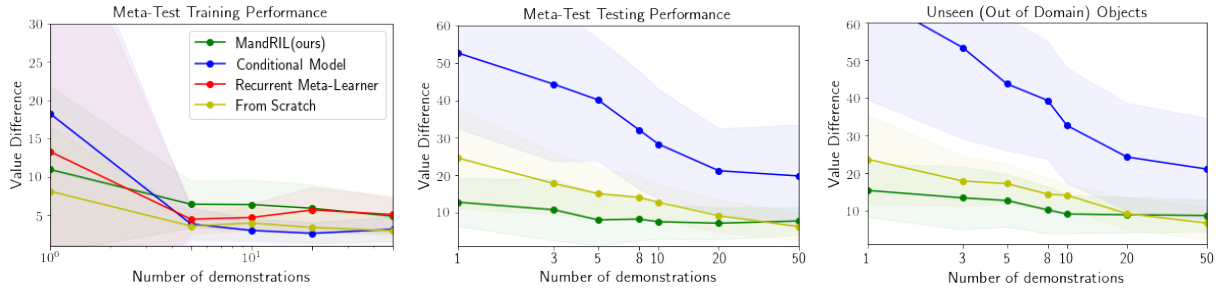


Figure 38: Meta-test performance with varying numbers of demonstrations (lower is better): held-out tasks training (left), test performance (middle), and test performance on held-out tasks with novel sprites (right). All methods are capable are overfitting to the training environment (top). However, in both test settings, MandRIL achieves comparable performance to the training environment, while the other methods overfit considerably until they receive at least 10 demonstrations. The recurrent meta-learner has a value difference larger than 60 in both test settings.

we do not use the game engine.

All tasks require learning directly from raw image inputs, and involve navigating to goal objects while exhibiting preference over terrain types (e.g. the agent prefers to traverse dirt tiles over traversing grass tiles). The goal of IRL in this setting amounts to learning the correct visual cues that indicate regions of high and low reward. Therefore, we evaluate the learned reward functions by using them in new environments that contain the same objects as the demonstration environment. In the training environment, the reward function can memorize the training demonstration without learning the right visual cues, but such a reward function will perform poorly in a test environment where object positions have changed. If the reward function picks up on the right visual cues, its performance on the training and test environments will be similar.

The underlying MDP structure of SpriteWorld is a grid, where the states are each of the grid cells, and the actions enable the agent to move to any one of its 8-connected neighbors. We generate unique tasks from this domain as follows. First, we randomly choose a set of 3 sprites from a total of 100 sprites from the original game (creating a total of 161,700 unique tasks). We randomly place these three sprites within a randomly generated terrain tiling; we designate one of the sprites to be the goal of the navigation task, i.e. the object to which the agent must navigate. The other two objects are treated as obstacles for which the agent incurs a large negative reward for not avoiding. In each task, we optimize our model on a meta-training set and evaluate the ability of the reward function to the generalize to a rearrangement of the same objects. For example,

suppose the goal of the task is to navigate to sprite A, while avoiding sprites B and C. Then, to generate an environment for evaluation, we simply resample the positions of the sprites, while the underlying task remains the same (i.e., navigate to A). This requires the model to make use of the visual patterns in the scene to generalize effectively, rather than simply memorizing positions of sprites. We evaluate on novel combinations of units seen in meta-training, as well as the ability to generalize to new unseen units. We provide further details on our setup in Appendices B.4.1 and B.4.2.

We measure performance using the expected value difference metric, which measures the sub-optimality of a policy learned under the approximate reward; this is a standard performance metric used in prior IRL work (Levine et al., 2011; Wulfmeier et al., 2015). The metric is computed by taking the difference between the value function of the optimal policy under the learned reward and the value function of the optimal policy under the true reward.

EVALUATION PROTOCOL In our evaluation, we sample new tasks from the task distribution that were unseen during meta-training. We consider two possible settings: we either sample new sprite positions and tasks, but use sprites that were present during meta-training, or we sample entirely new sprites that were unseen, which we refer to as "out of domain objects." For each task, we sample one environment (set of sprite positions) along with demonstrations for adapting the reward, and a second environment for the same task where we test the adapted reward, to ensure that it picks up on the right visual cues. We refer to the performance on the first environment as "training performance" (not to be confused with meta-training – the evaluation considers meta-testing) and to performance on the second as "testing performance". As the number of demonstrations increases, we expect all methods to perform well in terms of training performance as they can simply overfit to the training environment without acquiring the right visual cues that allow them to generalize.

EVALUATION RESULTS The results for are shown in Fig. 38, which illustrates training and test performance with in-distribution sprites, and testing performance with out-of-distribution sprites. Our approach, MandRIL, achieves roughly the same value difference on the training environment as on the test environment, and has only slightly worse error with out-of-distribution sprites. Most significantly, the method performs well even with single-digit numbers of demonstrations. By comparison, alternative meta-learning methods generally overfit considerably, attaining good training performance but very poor test performance in both conditions. Learning the reward function from scratch is in

fact the most competitive baseline – as the number of demonstrations increases, simply training the fully convolutional reward function from scratch on the new task is the only method that matches the performance of MandRIL; but with few demonstrations, MandRIL has substantially lower value difference. It is worth noting the performance of MandRIL on the out of distribution test setting: although the model is evaluated on new sprites, the method is still able to adapt via gradient descent and exceed the performance of learning from scratch and all other methods.

### 8.1.5 *Discussion*

In this section, we presented an approach that enables few-shot learning for reward functions of new tasks. We achieved this through a novel formulation of inverse reinforcement learning that learns to encode common structure across tasks. Using our approach, we showed that we can use meta-training tasks to effectively learn deep neural network reward functions from raw pixel observations for new tasks, using using a handful of demonstrations. This work paves the way for exciting future work that considers unknown dynamics, or more fully probabilistic approaches to reward and goal inference.

## 8.2 FEW-SHOT GOAL INFERENCE FOR VISUOMOTOR LEARNING AND PLANNING

We previously showed how meta-learning can be used to infer reward functions from a few demonstrations. Note, however, that a demonstration communicates both *what* the task is and *how* to perform it. For many tasks, it is possible to convey a task with much less effort by only the communicating the *what*, i.e., only communicating the goal of the task. For example, one simple approach for specifying tasks is to provide an image of the goal (Jagersand and Nelson, 1995; Deguchi and Takahashi, 1999; Watter et al., 2015; Finn et al., 2016b; Zhu et al., 2017; Srinivas et al., 2018), or more generally, provide an observation of one instance of success. There are a number of challenges with this approach, e.g. measuring the distance between the current and goal observation; but perhaps most saliently, we would like to not only encode a single instance of success, but reason about the entire space of successful behavior and generalize the high-level goal to new scenarios. To encode such goals, we can learn a reward function (Abbeel and Ng, 2004; Ziebart et al., 2008; Finn et al., 2016a; Sermanet et al., 2017b; Tung et al., 2018) or success classifier (Pinto and Gupta, 2016; Ho and Ermon, 2016; Levine et al., 2016b) that operates on the robot's observations. Unlike reward functions, which are typically learned from demonstrations (via inverse RL, as discussed previously), we

can learn success classifiers from only positive and negative examples via supervised learning. Yet, training classifiers from scratch does not solve the entire problem, as they require a considerable amount of data for acquiring an objective for a single task. If we reuse data from a range of other tasks that we might want a robot to learn, then our agents can learn goal metrics for new tasks much more efficiently.

The main contribution of this section is a framework for specifying goals to robots with minimal human supervision for each individual task. Our approach, illustrated in Figure 39, leverages our previously-discussed work in meta-learning to enable few-shot acquisition of goal classifiers, and can provide an objective for various control optimizations, including visual model-based planning and model-free reinforcement learning. In our evaluation, we find that our approach provides a means to quickly learn objectives for learning manipulation skills with deformable objects in simulation, and for performing multi-stage vision-based manipulation tasks on a real robot.

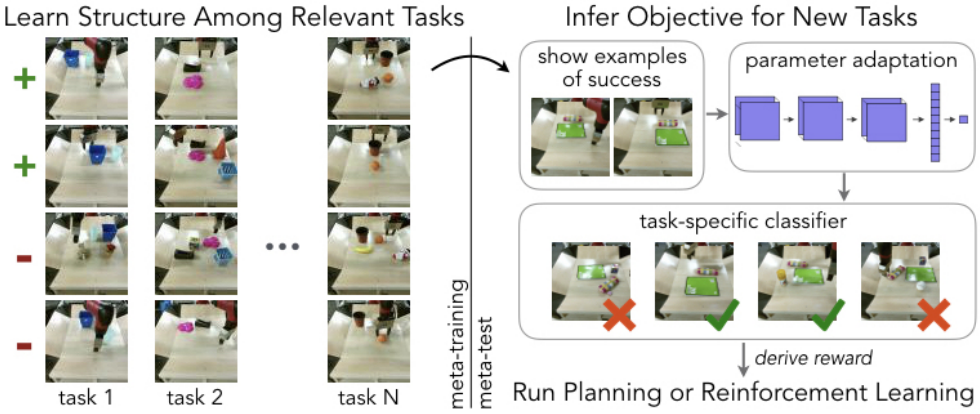


Figure 39: We propose a framework for making it practical to quickly convey the goal of a new task. To do so, we collect a small amount of positive and negative examples for a large number of other tasks (left) and use meta-learning to learn the structure within the space of these relevant tasks. We then use this learned structure to infer an objective for a new task from only a few examples of success (right). The reward or cost can be derived from the learned goal classifier for use with planning or reinforcement learning.

### 8.2.1 *Overview*

We aim to develop a framework that makes it easy to specify objectives for new tasks in a way that (1) reduces the manual engineering efforts for specifying a new task to a robot, making is fast and easy to convey goals to robots and (2) is generally-applicable to a wide range of problems and robot learning methods. Assuming that we ultimately want to convey many different tasks to a robot, we consider a multi-task problem setting where we have available a modest number of success/failure examples for a large number of tasks. We will use this data for meta-training a classifier such that, at test time, we can learn a goal classifier for a new task from only a few examples of success. By doing this, we minimize the amount of data needed for any particular task and make it possible to easily and quickly convey the goal of any new task. For full generality and minimal task-specific engineering, we need to be able to evaluate the objective using the same observation that the robot uses to solve the task, rather than external sensors or privileged information that is not available during deployment. To satisfy this requirement, we will learn success classifiers that operate directly on the robot's observation space. With these two design decisions in mind, we will next formalize the general learning problem and discuss our high-level solution that uses meta-learning.

### 8.2.2 *Problem Set-up*

Formally, we consider a goal classifier $\hat{y} = f(\mathbf{o})$, where $\mathbf{o}$ denotes the robot's observation, such as a camera image, and $\hat{y} \in [0, 1]$ indicates the predicted probability of the observation being of a successful outcome of the task. Such a classifier can be used for specifying the goal to reinforcement learning or planning algorithms, by deriving a reward function from the classifier's predictions; we discuss this in more detail in Section 8.2.4.2. Our aim is to learn a goal classifier from a few positive examples of success for a new task $\mathcal{T}_j$, as positive examples of success are intuitive for a human to provide and, in a sense, are the minimal piece of information needed to convey a task goal. Hence, we will be given a dataset $\mathcal{D}_j^+$ of $K$ positive examples of success for a task $\mathcal{T}_j$: $\mathcal{D}_j := \{(\mathbf{o}_k, 1) | k = 1...K\}_j$, and our goal is to infer a classifier for the conveyed task. How might we go about learning to infer goal classifiers for new tasks from only $K$ positive examples? To do this, we will explicitly train a model for the ability to infer goal classifiers for a wide range of previous tasks, $\{\mathcal{T}_i\}$. In particular, we assume a small dataset $\mathcal{D}_i$ for each task $\mathcal{T}_i$, where each dataset consists of both examples of success and not success: $\mathcal{D}_i := \{(\mathbf{o}_n, y_n) | n = 1...N\}_i$.

One natural question at this point is: what can the model learn from the meta-training

set that allows it to infer goals for new tasks more effectively than learning each task from scratch? If each task involves a completely distinct visual concept, without any shared structure across tasks, then it seems unlikely that the model will acquire useful knowledge from meta-training. However, practical real-world goals often share many patterns: object rearrangement tasks depend strongly on relative positioning of objects to each other, and are agnostic to the pose of the robot. Tasks that involve placing objects in containers depend on whether or not an object is inside the container, but not the container's position. By extracting such patterns from meta-training tasks, our method should be able to acquire structurally related meta-test tasks efficiently.

### 8.2.3 *Meta-learning for Few-Shot Goal Inference*

To solve the above learning problem, we propose to learn how to learn classifiers for a task from a few positive examples of that task, by using full supervision at the meta-level from both positive and negative examples. Then, at test time, we can effectively learn a classifier from only positive examples. Across all of the tasks in the set of training tasks $\{\mathcal{T}_i\}$, we will train a learner $f$ to learn a goal classifier $g_i$ from a dataset of positive examples $\mathcal{D}_i^+$ and make predictions about new observations $\mathbf{o}$:

$$g_i(\mathbf{o}) = f(\mathcal{D}_i^+, \mathbf{o}; \theta)$$

where we use $\mathcal{D}_i^+$ to denote a dataset of $K$ examples sampled uniformly from the positive examples in $\mathcal{D}_i$. To train the meta-learner parameters $\theta$, we will optimize the learned classifier for its ability to accurately classify new examples in $\mathcal{D}_i$, both positive and negative. In particular, we will optimize the following objective:

$$\min_\theta \sum_i \sum_{(\mathbf{o}_n, y_n) \in \mathcal{D}_i^{\text{test}}} \ell(y_n, g_i(\mathbf{o}_n)) = \min_\theta \sum_i \sum_{(\mathbf{o}_n, y_n) \in \mathcal{D}_i^{\text{test}}} \ell(y_n, f(\mathcal{D}_i^+, \mathbf{o}_n; \theta)) \tag{29}$$

where $\mathcal{D}_i^+$ is defined as above, $\mathcal{D}_i^{\text{test}}$ is sampled uniformly from $\mathcal{D}_i \backslash \mathcal{D}_i^+$, and the loss function $\ell$ is the standard cross entropy loss that compares the classifier's predictions to the labels. The datapoints $\mathcal{D}^{\text{test}}$ are distinct from $\mathcal{D}^+$ so that we train for good generalization. At test time, after learning $f$, we are presented with examples of success $\mathcal{D}_j^+$ for a new task $\mathcal{T}_j$. We can use this data infer a task-specific goal classifier $g_j(\cdot) = f(\mathcal{D}_j^+, \cdot; \theta)$. This classifier provides an objective, or part of an objective (as we discuss in the next section), for reinforcement learning or planning. We refer to this approach as few-shot learning of objectives (FLO). The algorithms underlying the meta-training optimization and test-time procedure are outlined in Algorithms 13 and 14.

**Algorithm 13** Few-Shot Learning of Objectives (FLO)

**Require:** for each task $\mathcal{T}_i$, a dataset of example successes and failures: $\mathcal{D}_i := \{(\mathbf{o}_n, y_n)\}_i \ \forall\, i$

randomly initialize learner parameters $\theta$
**while** not done **do**
  Sample training task $\mathcal{T}_i$ (or minibatch)
  Sample examples of success $\mathcal{D}_i^+ \sim \mathcal{D}_i$
  Sample test examples $\mathcal{D}_i^{\text{test}} \sim \mathcal{D}_i \backslash \mathcal{D}_i^+$
  Learn goal classifier from positive examples $g_i(\cdot) = f(\mathcal{D}_i^+, \cdot; \theta)$
  Update learner parameters $\theta$ according to Eq. 29 using $\mathcal{D}_i^{\text{test}}$

**Algorithm 14** FLO test-time

**Require:** examples of success $\mathcal{D}_j^+$ for new task $\mathcal{T}_j$
**Require:** learned $\theta$

Infer goal classifier: $g_j(\cdot) = f(\mathcal{D}_j^+, \cdot; \theta)$
Run RL or planning, using reward/cost derived from $g_j$

As discussed in Section 4.4, the view of meta-learning as learning the mapping $f(\mathcal{D}, \mathbf{o}; \theta) \rightarrow \hat{y}$ is general to a number of different meta-learning algorithms, including recurrent models (Santoro et al., 2016), learned optimizers (Ravi and Larochelle, 2017), and gradient-based methods (Finn et al., 2017a). Hence, this framework can be combined with combined with any of such meta-learning algorithms for few-shot classifier learning.

#### 8.2.4 *Few-Shot Goal Inference for Learning and Planning*

Having presented the general framework of few-shot goal inference, we will discuss our particular meta-learning implementation, mechanisms needed to mitigate exploitation of the learned objective by the controller, and mechanisms for specifying compound tasks by joining classifiers.

##### 8.2.4.1 *Concept Acquisition for Goal Classifiers*

While the above framework is general to any meta-learning approach, we would use a method that can efficiently learn to learn, to avoid collecting very large amounts data for meta-training. As a result, we choose to build upon model-agnostic meta-learning (MAML) (Finn et al., 2017a), which incorporates the structure of gradient descent for efficient meta-learning. In particular, MAML learns an initial parameter setting $\theta$ of the model $f_{\text{MAML}}$ such that one or a few steps of gradient descent with a few examples leads

to parameters $\phi$ that generalize well to new examples from that task. Grant et al. (2017) extended MAML for learning new concepts from only positive examples, referred to as concept acquisition through meta-learning (CAML), akin to how humans learn new concepts. We adapt CAML to the setting of acquiring binary success classifiers from positive examples. At test time, the learner uses gradient descent to adapt the meta-learned parameters $\theta$ to a dataset of positive examples $\mathcal{D}_j^+$ for task $\mathcal{T}_j$:

$$g_j(\mathbf{o}) = f(\mathcal{D}_j^+, \mathbf{o}; \theta) = f_{\text{CAML}}\big(\mathbf{o}; \phi_j\big) = f_{\text{CAML}}\big(\mathbf{o}; \theta - \alpha \nabla_\theta \sum_{(\mathbf{o}_n, y_n) \in \mathcal{D}_j^+} \ell(y_n, f_{\text{CAML}}(\mathbf{o}_n; \theta))\big)$$

where $\ell$ is the cross-entropy loss function, $\alpha$ is the step size, and $\phi_j$ denotes the parameters updated through gradient descent on task $\mathcal{T}_j$. We only write out one gradient descent step for convenience of notation; in practice, more may be used. Then, meta-training takes into account this gradient descent adaptation procedure, training for the initial parameters as follows:

$$\min_\theta \sum_i \sum_{(\mathbf{o}_n, y_n) \in \mathcal{D}_i^{\text{test}}} \ell(y_n, f_{\text{CAML}}(\mathbf{o}_n; \phi_i))$$

We optimize this objective using Adam (D. Kingma and J. Ba, 2015) on the initial parameters $\theta$. In our experiments, we consider vision-based tasks where $\mathbf{o}$ is an RGB image. We use a learner $f_{\text{CAML}}$ that is represented by a convolutional neural network with RGB image inputs. We provide details on the architecture and other implementation details in Section 8.2.6.

#### 8.2.4.2 *Deriving Rewards from Classifier Predictions*

Once we have inferred a goal classifier, we can predict the probability of an observation $\mathbf{o}$ being successful. To use this prediction as an objective for planning or reinforcement learning, we need to convert it into a reward function. One simple approach would be to treat the probability of success as the reward for that observation. We find that this works reasonably well, but that the predictions produced by the neural network are not always well-calibrated. To reduce the effect of false positives and mis-calibrated predictions, we use the classifier conservatively by thresholding the predictions so that reward is only given for confident successes. Below this threshold, we give a reward of 0 and above this threshold, we provide the predicted probability as the reward. To further avoid spurious false positives from affecting learning, we additionally only provide this nonzero reward when two consecutive frames have achieved predicted success above the threshold.

#### 8.2.4.3 *Cascading Classifiers for Compound Tasks*

To provide objectives for more complex tasks, we can join multiple classifiers. For example, if we train a classifier to recognize if a particular relative position of two objects is achieved, then we can use multiple classifiers to achieve a particular configuration of multiple objects, like a table setting. To achieve this, we provide a few positive examples $\mathcal{D}_j^+$ of each task $\mathcal{T}_j$ that we would like to achieve, and then infer the classifier for each task. To perform the sequence of tasks, we cascade the inferred classifiers – iteratively setting each objective one-by-one, and running the planner or policy for each one until the classifier indicates that the subtask has been completed. In our experiments, we illustrate how this cascading technique can be used to maneuver three objects into a desired configuration.

### 8.2.5 *Related Work*

Specifying goals is a challenge for many real-world robotics and reinforcement learning tasks. Many of the most successful demonstrations of robot learning have sidestepped this issue and instead engineered task-specific solutions for providing an objective such as hand-shaped objectives (Levine et al., 2016a; Schulman et al., 2015) and manually instrumented environments (e.g. a thermal camera, scale, or other sensors for measuring quantity poured (Schenck and Fox, 2017; Schenck and Fox, 2016; Yamaguchi et al., 2015), markers determining a pancake's orientation (Kormushev et al., 2010), and an accelerometer on a door handle (Yahya et al., 2017)). In this section, we aim to develop a more general and scalable framework for specifying a goals that does not require manual instrumentation or shaping, and instead learns to represent goals using the robot's sensors that are used to complete the task.

A number of works have proposed to specify vision-based tasks using an image or visual representation of the goal (Jagersand and Nelson, 1995; Deguchi and Takahashi, 1999; Watter et al., 2015; Finn et al., 2016b; A. Edwards et al., 2016; Zhu et al., 2017; A. D. Edwards et al., 2017; Srinivas et al., 2018). Unlike these works, we aim to acquire a classifier that can effectively recognize successful task executions that may not directly match an image of the goal. This enables us to recognize goals that are more abstract than an entire goal visual scene, such as relative positions of objects, approximate shapes of deformable objects, or disjunctions. Other works have sought to learn objectives by training a classifier (Pinto and Gupta, 2016; Ho and Ermon, 2016; Levine et al., 2016b) or reward function (Sermanet et al., 2017b; Christiano et al., 2017; Tung et al., 2018), including in the framework of inverse RL (Ng and Russell, 2000; Abbeel and Ng, 2004; Ziebart et al., 2008;

Boularias et al., 2011; Kalakrishnan et al., 2013; Finn et al., 2016a; Wulfmeier et al., 2016c; Rhinehart and Kitani, 2017). However, training a classifier from scratch per task has a number of challenges: modern vision methods require large training sets, and generally require both positive and negative examples. Providing many examples is onerous, and requiring users to provide negative examples is time-consuming and counter-intuitive to the average user. We aim to address both of these issues by considering the fact that we ultimately care about learning objectives of many different skills – we can use meta-learning to share data across tasks, such that only a modest amount of data is needed for any individual task, and learn how to learn a task objective from only a handful of positive examples.

### 8.2.6 *Experiments*

To study the generality of our approach, we evaluate the effectiveness of our learned objectives with both vision-based planning and reinforcement learning methods, with an emphasis on tasks that have visually nuanced goals that are difficult to specify manually. Videos of our results can be viewed on the supplementary website[1].

Since our goal is to provide an easy way to compute an objective for new tasks from a few observations of success for that task, we compare our approach to a few alternative and prior methods for doing so under the same assumptions as our method:

**Pixel distance**: Given one observation of success, a naive metric for evaluating the objective is to measure the $\ell_2$ distance between the current observation and the successful observation. In some scenarios with low-dimensional observation spaces, this can perform reasonably well; however, this approach cannot capture the invariances that a goal classifier can represent, nor does it scale well to visual or other high-dimensional sensory observation spaces.

**Latent space distance**: An alternative approach is to measure the distance between current and goal observations in a learned latent space (Watter et al., 2015; Finn et al., 2016b). For learning the latent space, we train an autoencoder (Lange et al., 2012; Finn et al., 2016b) on the meta-training data used for our approach. We expect this metric to be better grounded than distances in pixel space, but still cannot capture invariances across and within goals.

**Oracle**: In our simulated experiments, we can also derive a precise objective using the ground truth state of the environment, which is generally not available in the physical world. We run reinforcement learning with respect to this objective to provide an upper

1 The supplementary video is at https://sites.google.com/view/few-shot-goals

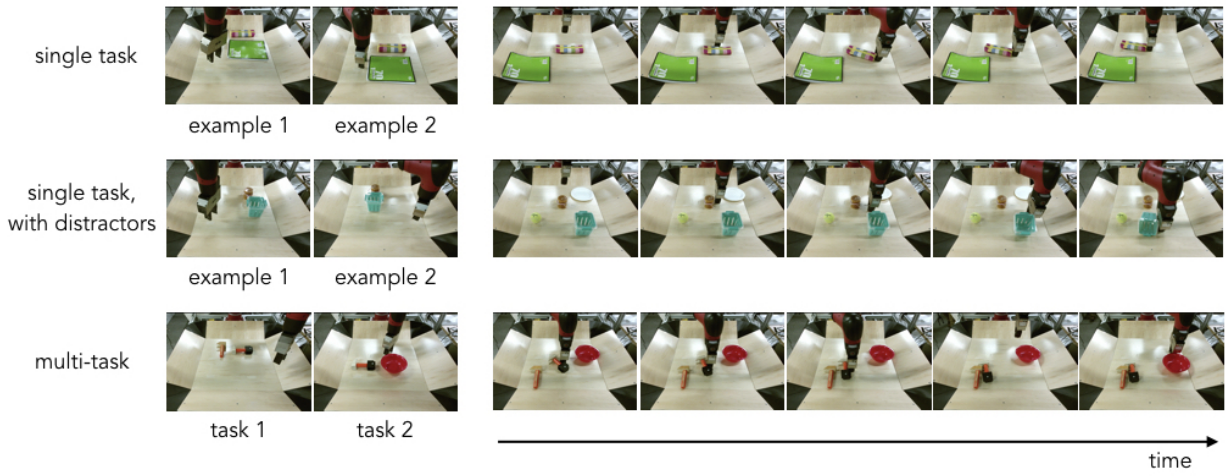


Figure 40: Object arrangement performance of our method with and without distractor objects and with two tasks. The left shows a subset of the 5 positive examples that are provided for inferring the goal classifier(s), while the right shows the robot executing the specified task(s) via visual planning.

bound on performance within the simulated evaluation.

For all tasks in our evaluation, we consider the 5-shot learning setting, where 5 examples of success are provided to method for conveying the goal of the test task. For both the pixel and latent space distance metrics above, we take the minimum distance across these five examples, allowing the optimization to try to match the closest goal observation. Our network architecture details can be found in Appendix B.6.1, and code for FLO is available in the supplementary material.

#### 8.2.6.1 *Visual Planning for Object Arrangement*

We study a visual object arrangement task, where different goals correspond to different relative arrangements of a pair of objects. For this setting, we use a Sawyer robot and use the planning method developed by Ebert et al. (2017) to optimize actions with respect to the learned objective. This planning approach learns a video prediction model from self-supervised data, and uses a sampling-based optimization with iterative replanning (MPC) to select sequences of actions at each timestep that lead to desirable futures. We evaluate our learned classifier on the predictions made by the video prediction model and derive the cost used for planning using the approach described in Section 8.2.4.2.

To collect data for meta-training the classifier, we randomly select a pair of objects

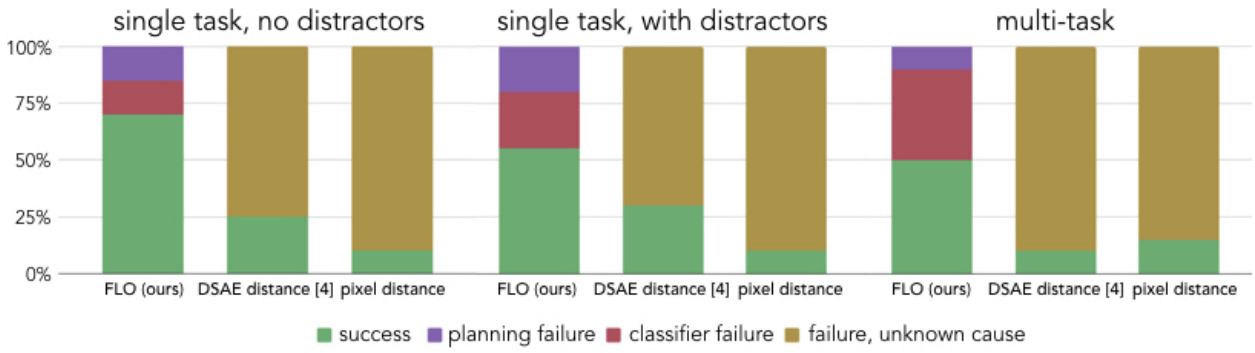


Figure 41: Quantitative performance of visual planning across different goal specification methods: ours, DSAE (Finn et al., 2016b), and pixel error. We consider three experimental settings with increasing difficulty: a single relative object positioning task, a single task with distractor objects, and a multi-task object positioning task where the robot needs to maneuver two objects into a line. Where possible, we include break down the cause of failures into errors caused by inaccurate prediction or planning and those caused by an inaccurate goal classifier.

from our set of training objects, and position them into many different relative positions, recording the image for each configuration. One task corresponds to a particular relative positioning of two objects, e.g. the first object to the left of the second, and we construct positive and negative examples for this task by labeling the aforementioned images. We randomly position the arm in each image, as it is not a determiner of task success. A good objective should ignore the position of the arm. We also include randomly-positioned distractor objects in about a third of the collected images. Some of the meta-training data is illustrated in the left of Figure 39. In total, we collect data for 92 tasks, with roughly 25 positive and 45 negative examples for each task. As some images are shared across tasks, the total number of unique images in the dataset is 4248.

We evaluate all approaches in three different experimental settings. In the first setting, the goal is to arrange two objects into a specified relative arrangement. The second setting is the same, but with distractor objects present. In the final, most challenging setting, the goal is to achieve two tasks in sequence. As described in Section 8.2.4.3, we provide positive examples for both tasks, infer the classifier for both task, perform MPC for the first task until completion, followed by MPC for the second task. To evaluate the ability to generalize to new goals and settings, we use novel, held-out objects for all of the task and distractor objects in our evaluation.

We qualitatively visualize the evaluation in Figure 40. On the left, we show a subset of

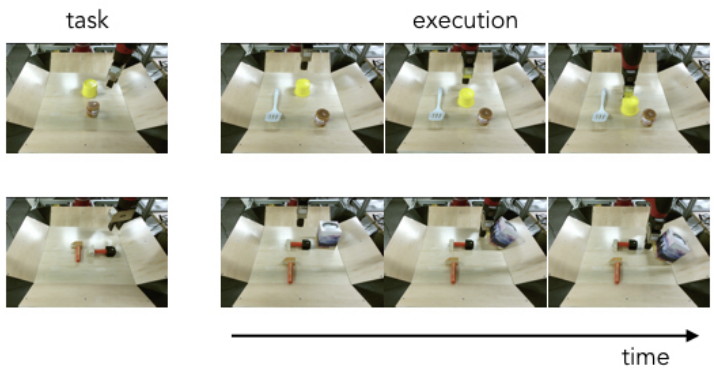


Figure 42: Visualization of the most common scenarios in which the inferred goal classifier fails. Top: the robot starts to move the object and stops when the classifier believes that the diagonal arrangement is satisfying the goal. Bottom: the classifier is thrown off by a novel distractor object. For both cases, more data or data augmentation with a wider range of diagonal relations and distractors could help mitigate the source of failure.

the five images provided to illustrate the task(s), and on the left, we show the motions performed by the robot. We see that the robot is able to execute motions which lead to a correct relative positioning of the objects. We quantitatively evaluate each method across 20 tasks, including 10 unique object pairs. The results, shown in Figure 41, indicate that prior methods for learning distance metrics struggle to infer the goal of the task, while our approach leads to substantially more successful behavior on average. Aiming to understand the results further and the fidelity of the inferred goal classifiers, we measure whether the failures are caused by the planner or the classifier. In particular, if the classifier predicted that the final image was a success that was not, or if the classifier classifies a correct plan as not successful, then the fault is on the classifier. Otherwise, the fault is the planner, e.g. if the planner finds an action plan that looks successful according to the video prediction model, but in reality is not. From this analysis, we observe that about half of the single task failures are caused by the planner / predictive model. This suggests at least 75% of the tasks are achievable by the planner with a suitable metric. It is difficult to judge the cause of failure for the latent and pixel distance metrics, since they provide soft boundaries between success and failure. However, we qualitatively observed that most of the plans that achieved low distance according to these metrics did not look successful, suggesting that most of the failures were caused by the metric. We further analyze failure common failure modes in Figure 42.


Figure 43: Rope manipulation policy roll-outs. Left: a subset of the 5 positive examples provided for inferring the goal classifier. Right: the robot executing the policy learned via RL with the learned objective.

#### 8.2.6.2 *Rope Manipulation with Reinforcement Learning*

We next study a setting where the goal is to learn to manipulate a rope of pearls into a particular shape, as specified by a few images. Here, we aim to evaluate how our objective and others can be used with a model-free reinforcement learning method, proximal policy optimization (PPO) (Schulman et al., 2017). For this setting, we use a parallel-jaw gripper as the manipulator, and our experiments are carried out in the Mujoco simulator (Todorov et al., 2012).

The tasks (i.e. goal shapes of the rope in this case) are generated through the following automated procedure: the manipulator takes random actions, some of which cause the rope to move. If any single pearl gets displaced by more than 10 cm from its original position, we freeze the rope position and call it a new task. We then displace the rope all over the table, and add small perturbations to it to generate multiple examples for this task. This encodes the fact that we only care about the shape of the rope, and not its position on the table, and small perturbations on the shape are acceptable. Negatives for the tasks are sample in two ways: actions taken by the manipulator that do not cause any of the pearls to move by more than 10 cm are labeled as negatives, and positive examples from one task are used as negatives for all other task. The number of negatives from these two sources is kept balanced. We use 240 tasks for training, with 30 positives and 30 negatives per task, leading to a total of 14400 images being used for training. We

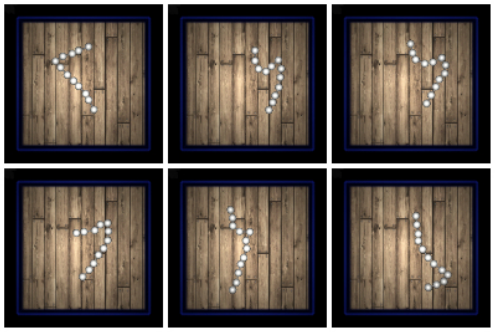

Figure 44: Illustration of six different rope manipulation tasks

| method | median distance | |
|---|---|---|
| | full state | visual features |
| pixel distance | 0.23 (0.20, 0.35) | n/a |
| AE distance | 0.60 (0.46,0.70) | 0.50 (0.42,0.60) |
| FLO (ours) | 0.37 (0.21, 0.57) | 0.38 (0.30,0.53) |
| oracle | 0.07 (0.05, 0.11) | n/a |

Table 9: Median distance across rope tasks (lower is better) comparing approaches for deriving a reward from images, using RL from full state and from visual features. Numbers in parentheses indicate the $25^{th}$ and $75^{th}$ percentiles.

use 10 tasks for testing, with five success images used per task for inferring the goal. Example tasks are illustrated in Figure 44.

When running reinforcement learning, we compare policy learning on the full, low-level state of the system to visuomotor policy learning on features derived from an autoencoder (following prior work on learning policies from autoencoder features (Lange et al., 2012; Finn et al., 2016b), see Appendix B.6.2 for details). For evaluation and the oracle, we need to devise a distance metric. We do so by translating the current and goal rope to have the same center of mass, and then measuring the average distance between corresponding pearls. This metric is used only by the oracle (as reward), and to provide a fast, automatic evaluation.

In our results, shown in Table 9, we first notice that pixel distance provides a reasonably good metric for the tasks. This is not all that surprising, since the simulated images are clean, with a fixed background. As shown in the previous experiments, pixel distance does not provide a good metric when using real images with novel objects, so we do not expect these results to transfer to real world settings. We also observe that FLO performs substantially better than using a distance metric derived from the latent space of an autoencoder, but does not reach the performance of the oracle distance metric. In most tasks where the policy performed poorly using FLO, we observed that RL was able to successfully optimize the objective, indicating that it was exploiting inaccuracies in the classifier. Integrating frameworks for mining negatives and iteratively retraining the objective is an interesting direction for future work. We show qualitative examples in

Figure 43.

### 8.2.7 *Discussion*

In this section, we proposed a framework for learning to learn task objectives from a few examples of success, motivated by the challenge of specifying goals in real world settings. We discussed how to derive a reward function from our inferred goal classifier, and how our task goal classifiers can be combined for forming more complex, compound goals. Finally, we showed how our framework for quickly inferring goals can be combined with both reinforcement learning and planning for vision-based manipulation skills such as maneuvering a rope into a given shape in simulation and rearranging multiple previously-unseen objects in the real world.

# 9

# CONCLUSION

In this thesis, we considered the problem of learning to learn, aiming to find an effective way to leverage prior experience for faster learning of new skills. To start, we formally defined the meta-learning problem and outlined a few properties to consider when developing new methods and analyzing existing ones. We then proposed a model-agnostic meta-learning algorithm, or MAML, that embeds gradient descent into the meta-learner, aiming to find an initial representation such that one or a few gradient steps leads to effective generalization. This meta-learner, by construction, will acquire consistent learning algorithms, and, with a sufficiently deep model, can scale to the full expressive power of recurrent networks. We further develop PLATIPUS, a form of MAML that can reason about ambiguity in few-shot learning.

After presenting the core MAML and PLATIPUS algorithms, we considered a number of different extensions. First, we showed how tasks can be constructed using temporal windows for enabling fast online adaptation in dynamic environments, using a meta-model-based RL framework. Then, in the context of visual imitation, we presented an approach for learning to learn from weakly-supervised and domain-shifted data, allowing a robot to learn from a video of a human. Finally, we showed how we can use meta-learning to infer goals and objectives underlying a few provided examples of behavior.

As is the case with many academic endeavors, the research presented here suggests many more questions than answers. Below, we discuss some of these open questions and future directions that we believe to be the most compelling.

**Task Construction** As discussed in Chapter 2, meta-learning assumes the ability to sample from a pre-defined distribution of tasks. Such an assumption may seem innocuous at first glance, but in practice, it requires for a great deal of human engineering. Essentially, meta-learning shifts a burden from designing algorithms to designing tasks and their corresponding objectives. Hence, a key question moving forward is, can we

also automate the process of task construction?

One promising direction, which is likely part of the solution, is to use the *environment* to create different tasks. As explored in Chapter 6, perhaps the general notion of a task is simply a temporal window, as the agent is naturally in different parts of the environment at different times. With a sufficiently large and diverse environment, temporal tasks can be constructed without much effort. Designing diverse environments in simulation, of course, does not remove the engineering burden of task design. To get diversity truly for free, one needs to consider the real world where such diversity already exists. The real world is also dynamic as many agents are constantly making decisions that affect the world and each other. Such multi-agent environments can likely also lead to interesting tasks for meta-learning without substantial hand-engineering, as initially explored by Al-Shedivat et al. (2018).

Beyond changes in and around the environment, a critical challenge is for agents to be able to construct their own objectives in those environments. While changes in objective can also be triggered by the environment or other agents, e.g. changing strategies in response to another agent's action in a game, our agents need to be able to create these objectives autonomously based on their current surroundings. Initial work by Gupta et al. (2018) suggests that meta-learning across a fixed set of random or unsupervised goals performs surprisingly well. Though, ultimately, proposing goals needs to be a dynamic process as new parts of the state space are discovered and as the agent becomes more competent, likely in the form of a curricula. Interestingly, the process of constructing such goals might look like a two-player game itself (Sukhbaatar et al., 2018), with one player setting goals for the other.

**Consistent and Universal Meta-Reinforcement Learning** We showed how model-agnostic meta-learning algorithms are both universal and consistent for few-shot supervised learning, unlike other meta-learning algorithms that exhibit one or the other property. Hence, there lies an interesting open question: is it possible for a meta-reinforcement learning algorithm to be both universal and be guaranteed to produce consistent learning algorithms? Universality holds neither for the model-free reinforcement learning variant of MAML presented in Section 4.2.2 nor the model-based variant presented in Chapter 6. To see this for the model-free variant, consider a simple counterexample to universality: if all of the K roll-outs have zero return, the policy gradient will be zero and, hence, the updated policy will not depend on the states visited and actions taken regardless of the initialization. A more expressive RL procedure would be able to change the policy based on the states and actions even with zero return. In contrast, recurrent meta-RL algorithms, e.g. those proposed by Wang and Hebert, 2016; Duan et al., 2016b, are uni-

versal, but do not produce consistent reinforcement learning procedures. Generally, the space of meta-RL algorithms has been explored much less than that of meta-supervised learning procedures, leaving significant room for future work.

**Lifelong Meta-Learning** Ultimately, we want systems that can build upon prior experience, not just to complete a range of new tasks, but to then incorporate those new tasks, preparing for the next set of tasks. In such lifelong learning settings, meta-learning has great potential for improving forward transfer to new tasks. That said, handling non-stationary task distributions presents a major challenge, since data on previous tasks may not be enough to prepare the model for new tasks. The results in Section 4.6.4 suggest that model-agnostic meta-learning algorithms are better equipped to address non-stationarity that black-box meta-learners. But, how to perform continual learning to learn remains an open question, one which we believe to be crucial for building agents that effectively learn continually.

* * *

With systems capable of building upon previous experiences rather than learning from scratch, our agents will be more prepared to handle the diversity of the real world and better equipped to improve themselves; we will be able to move away from end-to-end training from scratch. Most importantly, we will be closer to developing agents that exhibit the generality and flexibility of human intelligence, and perhaps closer to understanding the nature of general intelligence itself.

# Part IV

# APPENDICES

# A

# MODEL-AGNOSTIC META-LEARNING METHODS

## A.1 SUPPLEMENTARY PROOFS FOR 1-SHOT UNIVERSALITY

### A.1.1 *Proof of Lemma 4.4.1*

While there are likely a number of ways to prove Lemma 4.4.1 (copied below for convenience), here we provide a simple, though inefficient, proof of Lemma 4.4.1.

**Lemma 4.4.1.** *Let us assume that $\overline{e}(\mathbf{y})$ can be chosen to be any linear (but not affine) function of $\mathbf{y}$. Then, we can choose $\theta_{ft}$, $\theta_h$, $\{A_i; i > 1\}$, $\{B_i; i < N\}$ such that the function*

$$\hat{f}(\mathbf{x}^\star; \phi) = h_{post}\left(-\alpha \sum_{i=1}^{N} A_i \overline{e}(\mathbf{y}) k_i(\mathbf{x}, \mathbf{x}^\star); \theta_h\right) \tag{12}$$

*can approximate any continuous function of $(\mathbf{x}, \mathbf{y}, \mathbf{x}^\star)$ on compact subsets of $\mathbb{R}^{\dim(\mathbf{y})}$.*[1]

To prove this lemma, we will proceed by showing that we can choose $\overline{e}$, $\theta_{ft}$, and each $A_i$ and $B_i$ such that the summation contains a complete description of the values of $\mathbf{x}$, $\mathbf{x}^\star$, and $\mathbf{y}$. Then, because $h_{post}$ is a universal function approximator, $\hat{f}(\mathbf{x}^\star, \phi)$ will be able to approximate any function of $\mathbf{x}$, $\mathbf{x}^\star$, and $\mathbf{y}$.

Since $A_1 = I$ and $B_N = I$, we will essentially ignore the first and last elements of the sum by defining $B_1 := \epsilon I$ and $A_N := \epsilon I$, where $\epsilon$ is a small positive constant to ensure positive definiteness. Then, we can rewrite the summation, omitting the first and last terms:

$$\hat{f}(\mathbf{x}^\star; \phi) \approx h_{post}\left(-\alpha \sum_{i=2}^{N-1} A_i \overline{e}(\mathbf{y}) k_i(\mathbf{x}, \mathbf{x}^\star); \theta_h\right)$$

1 The assumption with regard to compact subsets of the output space is inherited from the UFA theorem.

Next, we will re-index using two indexing variables, $j$ and $l$, where $j$ will index over the discretization of $\mathbf{x}$ and $l$ over the discretization of $\mathbf{x}^\star$.

$$\hat{f}(\mathbf{x}^\star; \phi) \approx h_{\text{post}}\left(-\alpha \sum_{j=0}^{J-1}\sum_{l=0}^{L-1} A_{jl}\bar{e}(\mathbf{y})k_{jl}(\mathbf{x}, \mathbf{x}^\star); \theta_h\right)$$

Next, we will define our chosen form of $k_{jl}$ in Equation 30. We show how to acquire this form in the next section.

**Lemma A.1.1.** *We can choose $\theta_{ft}$ and each $B_{jl}$ such that*

$$k_{jl}(\mathbf{x}, \mathbf{x}^\star) := \begin{cases} 1 & \textit{if } \text{discr}(\mathbf{x}) = \mathbf{e}_j \textit{ and } \text{discr}(\mathbf{x}^\star) = \mathbf{e}_l \\ 0 & \textit{otherwise} \end{cases} \tag{30}$$

*where* $\text{discr}(\cdot)$ *denotes a function that produces a one-hot discretization of its input and* $\mathbf{e}$ *denotes the o-indexed standard basis vector.*

Now that we have defined the function $k_{jl}$, we will next define the other terms in the sum. Our goal is for the summation to contain complete information about $(\mathbf{x}, \mathbf{x}^\star, \mathbf{y})$. To do so, we will chose $\bar{e}(\mathbf{y})$ to be the linear function that outputs $J * L$ stacked copies of $\mathbf{y}$. Then, we will define $A_{jl}$ to be a matrix that selects the copy of $\mathbf{y}$ in the position corresponding to $(j, l)$, i.e. in the position $j + J * l$. This can be achieved using a diagonal $A_{jl}$ matrix with diagonal values of $1 + \epsilon$ at the positions corresponding to the $k$th vector, and $\epsilon$ elsewhere, where $k = (j + J * l)$ and $\epsilon$ is used to ensure that $A_{jl}$ is positive definite.

As a result, the post-update function is as follows:

$$\hat{f}(\mathbf{x}^\star; \phi) \approx h_{\text{post}}\left(-\alpha v(\mathbf{x}, \mathbf{x}^\star, \mathbf{y}); \theta_h\right), \text{ where } v(\mathbf{x}, \mathbf{x}^\star, \mathbf{y}) \approx \begin{bmatrix} \mathbf{o} \\ \vdots \\ \mathbf{o} \\ \mathbf{y} \\ \mathbf{o} \\ \vdots \\ \mathbf{o} \end{bmatrix},$$

where $\mathbf{y}$ is at the position $j + J * l$ within the vector $v(\mathbf{x}, \mathbf{x}^\star, \mathbf{y})$, where $j$ satisfies $\text{discr}(\mathbf{x}) = \mathbf{e}_j$ and where $l$ satisfies $\text{discr}(\mathbf{x}^\star) = \mathbf{e}_l$. Note that the vector $-\alpha v(\mathbf{x}, \mathbf{x}^\star, \mathbf{y})$ is a complete

description of $(\mathbf{x}, \mathbf{x}^\star, \mathbf{y})$ in that $\mathbf{x}$, $\mathbf{x}^\star$, and $\mathbf{y}$ can be decoded from it. Therefore, since $h_{\text{post}}$ is a universal function approximator and because its input contains all of the information of $(\mathbf{x}, \mathbf{x}^\star, \mathbf{y})$, the function $\hat{f}(\mathbf{x}^\star; \phi) \approx h_{\text{post}}(-\alpha v(\mathbf{x}, \mathbf{x}^\star, \mathbf{y}); \theta_h)$ is a universal function approximator with respect to its inputs $(\mathbf{x}, \mathbf{x}^\star, \mathbf{y})$.

### A.1.2 *Proof of Lemma A.1.1*

In this section, we show one way of proving Lemma A.1.1:

**Lemma A.1.1.** *We can choose $\theta_{ft}$ and each $B_{jl}$ such that*

$$k_{jl}(\mathbf{x}, \mathbf{x}^\star) := \begin{cases} 1 & \text{if } \operatorname{discr}(\mathbf{x}) = \mathbf{e}_j \text{ and } \operatorname{discr}(\mathbf{x}^\star) = \mathbf{e}_l \\ 0 & \text{otherwise} \end{cases} \tag{30}$$

*where* $\operatorname{discr}(\cdot)$ *denotes a function that produces a one-hot discretization of its input and* $\mathbf{e}$ *denotes the o-indexed standard basis vector.*

Recall that $k_{jl}(\mathbf{x}, \mathbf{x}^\star)$ is defined as $\tilde{\phi}(\mathbf{x}; \theta_{ft}, \theta_b)^\top B_{jl}^\top B_{jl} \tilde{\phi}(\mathbf{x}^\star; \theta_{ft}, \phi_b)$, where $\theta_b = 0$. Since the gradient with respect to $\theta_b$ can be chosen to be any linear function of the label $\mathbf{y}$ (see Section 4.4.3), we can assume without loss of generality that $\phi_b \neq 0$.

We will choose $\tilde{\phi}$ and $B_{jl}$ as follows:

$$\tilde{\phi}(\cdot; \theta_{ft}, \theta_b) := \begin{cases} \begin{bmatrix} \operatorname{discr}(\cdot) \\ \mathbf{0} \end{bmatrix} & \text{if } \theta_b = 0 \\ \begin{bmatrix} \mathbf{0} \\ \operatorname{discr}(\cdot) \end{bmatrix} & \text{otherwise} \end{cases} \qquad B_{jl} = \begin{bmatrix} E_{jj} & E_{jl} \\ E_{lj} & 0 \end{bmatrix} + \epsilon I$$

where we use $E_{ik}$ to denote the matrix with a 1 at $(i, k)$ and 0 elsewhere, and $\epsilon I$ is added to ensure the positive definiteness of $B_{jl}$ as required in the construction.

Using the above definitions, we can see that:

$$\tilde{\phi}(\mathbf{x}; \theta_{ft}, 0)^\top B_{jl}^\top \approx \begin{cases} \begin{bmatrix} \mathbf{e}_j \\ \mathbf{0} \end{bmatrix}^\top & \text{if } \operatorname{discr}(\mathbf{x}) = \mathbf{e}_j \\ \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix}^\top & \text{otherwise} \end{cases} \qquad B_{jl} \tilde{\phi}(\mathbf{x}^\star; \theta_{ft}, \phi_b) \approx \begin{cases} \begin{bmatrix} \mathbf{e}_j \\ \mathbf{0} \end{bmatrix} & \text{if } \operatorname{discr}(\mathbf{x}^\star) = \mathbf{e}_l \\ \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix} & \text{otherwise} \end{cases}$$

Thus, we have proved the lemma, showing that we can choose a $\tilde{\phi}$ and each $B_{jl}$ such that:

$$k_{jl}(\mathbf{x}, \mathbf{x}^\star) \approx \begin{cases} \begin{bmatrix} \mathbf{e}_j & \mathbf{o} \end{bmatrix} \begin{bmatrix} \mathbf{e}_j \\ \mathbf{o} \end{bmatrix} = 1 & \text{if } \operatorname{discr}(\mathbf{x}) = \mathbf{e}_j \text{ and } \operatorname{discr}(\mathbf{x}^\star) = \mathbf{e}_l \\ 0 & \text{otherwise} \end{cases}$$

### A.1.3 *Form of linear weight matrices*

The goal of this section is to show that we can choose a form of $\tilde{W}$, $\overline{W}$, and $\check{w}$ such that we can simplify the form of $\mathbf{z}^\star$ in Equation 6 into the following:

$$\overline{\mathbf{z}}^\star = -\alpha \sum_{i=1}^{N} A_i \overline{e}(\mathbf{y}) \tilde{\phi}(\mathbf{x}; \theta_{ft}, \theta_b)^T B_i^T B_i \tilde{\phi}(\mathbf{x}^\star; \theta_{ft}, \phi_b), \tag{31}$$

where $A_1 = \mathbf{I}$, $A_i = \overline{M}_{i-1} \overline{M}_{i-1}^T$ for $i > 1$, $B_i = \tilde{M}_{i+1}$ for $i < N$ and $B_N = \mathbf{I}$.

Recall that we decomposed $W_i$, $\phi$, and the error gradient into three parts, as follows:

$$W_i := \begin{bmatrix} \tilde{W}_i & 0 & 0 \\ 0 & \overline{W}_i & 0 \\ 0 & 0 & \check{w}_i \end{bmatrix} \quad \phi(\cdot; \theta_{ft}, \theta_b) := \begin{bmatrix} \tilde{\phi}(\cdot; \theta_{ft}, \theta_b) \\ \mathbf{o} \\ \theta_b \end{bmatrix} \quad \nabla_{\mathbf{z}} \ell(\mathbf{y}, \hat{f}(\mathbf{x}; \theta)) := \begin{bmatrix} \mathbf{o} \\ \overline{e}(\mathbf{y}) \\ \check{e}(\mathbf{y}) \end{bmatrix} \tag{32}$$

where the initial value of $\theta_b$ will be 0. The top components, $\tilde{W}_i$ and $\tilde{\phi}$, have equal dimensions, as do the middle components, $\overline{W}_i$ and $\mathbf{o}$. The bottom components are scalars. As a result, we can see that $\mathbf{z}$ will likewise be made up of three components, which we will denote as $\tilde{\mathbf{z}}$, $\overline{\mathbf{z}}$, and $\check{z}$, where, before the gradient update, $\tilde{\mathbf{z}} = \prod_{i=1}^{N} \tilde{W}_i \tilde{\phi}(\mathbf{x}; \theta_{ft})$, $\overline{\mathbf{z}} = \mathbf{o}$, and $\check{z} = 0$. Lastly, we construct the top component of the error gradient to be $\mathbf{o}$, whereas the middle and bottom components, $\overline{e}(\mathbf{y})$ and $\check{e}(\mathbf{y})$, can be set to be any linear (but not affine) function of $\mathbf{y}$.

Using the above definitions and noting that $\phi_{ft} = \theta_{ft} - \alpha \nabla_{\theta_{ft}} \ell = \theta_{ft}$, we can simplify the form of $\mathbf{z}^\star$ in Equation 6, such that the middle component, $\overline{\mathbf{z}}^\star$, is the following:

$$\overline{\mathbf{z}}^\star = -\alpha \sum_{i=1}^{N} \left( \prod_{j=1}^{i-1} \overline{W}_j \right) \left( \prod_{j=1}^{i-1} \overline{W}_j \right)^T \overline{e}(\mathbf{y}) \tilde{\phi}(\mathbf{x}; \theta_{ft}, \theta_b)^T \left( \prod_{j=i+1}^{N} \tilde{W}_j \right)^T \left( \prod_{j=i+1}^{N} \tilde{W}_j \right) \tilde{\phi}(\mathbf{x}^\star; \theta_{ft}, \phi_b)$$

We aim to independently control the backward information flow from the gradient $e$ and the forward information flow from $\tilde{\phi}$. Thus, choosing all $\tilde{W}_i$ and $\overline{W}_i$ to be square and full rank, we will set

$$\tilde{W}_i = \tilde{M}_i \tilde{M}_{i+1}^{-1} \qquad \overline{W}_i = \overline{M}_{i-1}^{-1} \overline{M}_i,$$

so that we have

$$\prod_{j=i+1}^{N} \tilde{W}_j = \tilde{M}_{i+1} \qquad \prod_{j=1}^{i-1} \overline{W}_j = \overline{M}_{i-1},$$

for $i \in \{1...N\}$ where $\tilde{M}_{N+1} = \mathbf{I}$ and $\overline{M}_0 = \mathbf{I}$. Then we can again simplify the form of $\overline{\mathbf{z}}^\star$:

$$\overline{\mathbf{z}}^\star = -\alpha \sum_{i=1}^{N} A_i \overline{e}(\mathbf{y}) \tilde{\phi}(\mathbf{x}; \theta_{ft}, \theta_b)^T B_i^T B_i \tilde{\phi}(\mathbf{x}^\star; \theta_{ft}, \phi_b), \tag{33}$$

where $A_1 = \mathbf{I}$, $A_i = \overline{M}_{i-1} \overline{M}_{i-1}^T$ for $i > 1$, $B_i = \tilde{M}_{i+1}$ for $i < N$ and $B_N = \mathbf{I}$.

### A.1.4 *Output function*

In this section, we will derive the post-update version of the output function $f_{out}(\cdot; \theta_{out})$. Recall that $f_{out}$ is defined as a neural network that approximates the following multiplexer function and its derivatives (as shown possible by Hornik et al., 1990):

$$f_{out}\left(\begin{bmatrix} \tilde{\mathbf{z}} \\ \overline{\mathbf{z}} \\ \check{z} \end{bmatrix}; \theta_{out}\right) = \mathbb{1}(\overline{\mathbf{z}} = \mathbf{0}) g_{pre}\left(\begin{bmatrix} \tilde{\mathbf{z}} \\ \overline{\mathbf{z}} \\ \check{z} \end{bmatrix}; \theta_g\right) + \mathbb{1}(\overline{\mathbf{z}} \neq \mathbf{0}) h_{post}(\overline{\mathbf{z}}; \theta_h). \tag{34}$$

The parameters $\{\theta_g, \theta_h\}$ are a part of $\theta_{out}$, in addition to the parameters required to estimate the indicator functions and their corresponding products. Since $\overline{\mathbf{z}} = \mathbf{0}$ and $h_{post}(\overline{\mathbf{z}}) = \mathbf{0}$ when the gradient step is taken, we can see that the error gradients with respect to the parameters in the last term in Equation 34 will be approximately zero. Furthermore, as seen in the definition of $g_{pre}$ in Section 4.4.3, the value of $g_{pre}(\mathbf{z}, \theta_g)$ is also zero, resulting in a gradient of approximately zero for the first indicator function.[2]

2 To guarantee that g and h are zero when evaluated at $\mathbf{x}$, we make the assumption that $g_{pre}$ and $h_{post}$ are neural networks with no biases and nonlinearity functions that output zero when evaluated at zero.

The post-update value of $f_{out}$ is therefore:

$$f_{out}\left(\begin{bmatrix}\tilde{\mathbf{z}}^\star\\ \overline{\mathbf{z}}^\star\\ \check{\mathbf{z}}^\star\end{bmatrix};\phi_{out}\right) \approx \mathbb{1}(\overline{\mathbf{z}}^\star=\mathbf{0})g_{pre}\left(\begin{bmatrix}\tilde{\mathbf{z}}^\star\\ \overline{\mathbf{z}}^\star\\ \check{\mathbf{z}}^\star\end{bmatrix};\phi_g\right)+\mathbb{1}(\overline{\mathbf{z}}^\star\neq\mathbf{0})h_{post}(\overline{\mathbf{z}}^\star;\theta_h) \tag{35}$$

$$= h_{post}(\overline{\mathbf{z}}^\star;\theta_h) \tag{36}$$

as long as $\overline{\mathbf{z}}^\star \neq \mathbf{0}$. In Appendix A.1.1, we can see that $\mathbf{z}^\star$ is indeed not equal to zero.

## A.2 FULL K-SHOT PROOF OF UNIVERSALITY

In this appendix, we provide a full proof of the universality of gradient-based meta-learning in the general case with $K > 1$ datapoints. This proof will share a lot of content from the proof in the 1-shot setting, but we include it for completeness.

We aim to show that a deep representation combined with one step of gradient descent can approximate any permutation invariant function of a dataset and test datapoint $(\{(\mathbf{x},\mathbf{y})_i; i \in 1...K\}, \mathbf{x}^\star)$ for $K > 1$. Note that K does not need to be small.

We will proceed by construction, showing that there exists a neural network function $\hat{f}(\cdot;\theta)$ such that $\hat{f}(\cdot;\phi)$ approximates $f_{target}(\{(\mathbf{x},\mathbf{y})_k\},\mathbf{x}^\star)$ up to arbitrary precision, where $\phi = \theta - \alpha\frac{1}{K}\sum_{k=1}^{K}\nabla_\theta \ell(\mathbf{y}_k, f(\mathbf{x}_k;\theta)))$ and $\alpha$ is the learning rate. As we discuss in Section 4.4.3, the loss function $\ell$ cannot be any loss function, but the standard cross-entropy and mean-squared error objectives are both suitable. In this proof, we will start by presenting the form of $\hat{f}$ and deriving its value after one gradient step. Then, to show universality, we will construct a setting of the weight matrices that enables independent control of the information flow coming forward from the inputs $\{\mathbf{x}_k\}$ and $\mathbf{x}^\star$, and backward from the labels $\{\mathbf{y}_k\}$.

We will start by constructing $\hat{f}$. With the same motivation as in Section 4.4.1, we will construct $\hat{f}(\cdot;\theta)$ as the following:

$$\hat{f}(\cdot;\theta) = f_{out}\left(\left(\prod_{i=1}^{N} W_i\right)\phi(\cdot;\theta_{ft},\theta_b);\theta_{out}\right).$$

$\phi(\cdot;\theta_{ft},\theta_b)$ represents an input feature extractor with parameters $\theta_{ft}$ and a scalar bias transformation variable $\theta_b$, $\prod_{i=1}^{N} W_i$ is a product of square linear weight matrices, $f_{out}(\cdot,\theta_{out})$ is a readout function at the output, and the learned parameters are $\theta := \{\theta_{ft},\theta_b,\{W_i\},\theta_{out}\}$. The input feature extractor and readout function can be represented with fully connected neural networks with one or more hidden layers, which we

know are universal function approximators, while $\prod_{i=1}^{N} W_i$ corresponds to a set of linear layers. Note that deep ReLU networks act like deep linear networks when the input and pre-synaptic activations are non-negative. We will later show that this is indeed the case within these linear layers, meaning that the neural network function $\hat{f}$ is fully generic and can be represented by deep ReLU networks, as visualized in Figure 2.

Next, we will derive the form of the model prediction after one gradient update $\hat{f}(x^\star; \phi)$. Let $z_k = \left(\prod_{i=1}^{N} W_i\right) \phi(x_k)$ and we denote its gradient with respect to the loss as $\nabla_{z_k} \ell = e(x_k, y_k)$. The gradient with respect to any of the weight matrices $W_i$ for a single datapoint $(x, y)$ is given by

$$\nabla_{W_i} \ell(y, \hat{f}(x_k, \theta)) = \left(\prod_{j=1}^{i-1} W_j\right)^T e(x, y) \phi(x; \theta_{ft}, \theta_b)^T \left(\prod_{j=i+1}^{N} W_j\right)^T.$$

Therefore, the post-update value of $\prod_{i=1}^{N} W_i' = \prod_{i=1}^{N} (W_i - \alpha \frac{1}{K} \sum_k \nabla_{W_i})$ is given by

$$\prod_{i=1}^{N} W_i - \frac{\alpha}{K} \sum_{k=1}^{K} \sum_{i=1}^{N} \left(\prod_{j=1}^{i-1} W_j\right)\left(\prod_{j=1}^{i-1} W_j\right)^T e(x_k, y_k) \phi(x_k; \theta_{ft}, \theta_b)^T \left(\prod_{j=i+1}^{N} W_j\right)^T \left(\prod_{j=i+1}^{N} W_j\right) - O(\alpha^2),$$

where we move the summation over $k$ to the left and where we will disregard the last term, assuming that $\alpha$ is comparatively small such that $\alpha^2$ and all higher order terms vanish. In general, these terms do not necessarily need to vanish, and likely would further improve the expressiveness of the gradient update, but we disregard them here for the sake of the simplicity of the derivation. Ignoring these terms, we now note that the post-update value of $z^\star$ when $x^\star$ is provided as input into $\hat{f}(\cdot; \phi)$ is given by

$$\begin{aligned} z^\star = &\prod_{i=1}^{N} W_i \phi(x^\star; \phi_{ft}, \phi_b) \\ &- \frac{\alpha}{K} \sum_{k=1}^{K} \sum_{i=1}^{N} \left(\prod_{j=1}^{i-1} W_j\right)\left(\prod_{j=1}^{i-1} W_j\right)^T e(x_k, y_k) \phi(x_k; \theta_{ft}, \theta_b)^T \left(\prod_{j=i+1}^{N} W_j\right)^T \left(\prod_{j=i+1}^{N} W_j\right) \phi(x^\star; \phi_{ft}, \phi_b), \end{aligned} \tag{37}$$

and $\hat{f}(x^\star; \phi) = f_{out}(z^\star; \phi_{out})$.

Our goal is to show that that there exists a setting of $W_i$, $f_{out}$, and $\phi$ for which the above function, $\hat{f}(x^\star, \phi)$, can approximate any function of $(\{(x, y)_k\}, x^\star)$. To show universality, we will aim independently control information flow from $\{x_k\}$, from $\{y_k\}$, and from $x^\star$ by

multiplexing forward information from $\{\mathbf{x}_k\}$ and $\mathbf{x}^\star$ and backward information from $\{\mathbf{y}_k\}$. We will achieve this by decomposing $W_i$, $\phi$, and the error gradient into three parts, as follows:

$$W_i := \begin{bmatrix} \tilde{W}_i & 0 & 0 \\ 0 & \overline{W}_i & 0 \\ 0 & 0 & \check{w}_i \end{bmatrix} \quad \phi(\cdot; \theta_{ft}, \theta_b) := \begin{bmatrix} \tilde{\phi}(\cdot; \theta_{ft}, \theta_b) \\ \mathbf{0} \\ \theta_b \end{bmatrix} \quad \nabla_{\mathbf{z}_k} \ell(\mathbf{y}_k, \hat{f}(\mathbf{x}_k; \theta)) := \begin{bmatrix} \mathbf{0} \\ \overline{e}(\mathbf{y}_k) \\ \check{e}(\mathbf{y}_k) \end{bmatrix} \tag{38}$$

where the initial value of $\theta_b$ will be 0. The top components all have equal numbers of rows, as do the middle components. As a result, we can see that $\mathbf{z}_k$ will likewise be made up of three components, which we will denote as $\tilde{\mathbf{z}}_k$, $\overline{\mathbf{z}}_k$, and $\check{z}_k$. Lastly, we construct the top component of the error gradient to be $\mathbf{0}$, whereas the middle and bottom components, $\overline{e}(\mathbf{y}_k)$ and $\check{e}(\mathbf{y}_k)$, can be set to be any linear (but not affine) function of $\mathbf{y}_k$. We discuss how to achieve this gradient in the latter part of this section when we define $f_{out}$ and in Section 4.4.3.

In Appendix A.1.3, we show that we can choose a particular form of $\tilde{W}_i$, $\overline{W}_i$, and $\check{w}_i$ that will simplify the products of $W_j$ matrices in Equation 37, such that we get the following form for $\overline{\mathbf{z}}^\star$:

$$\overline{\mathbf{z}}^\star = -\alpha \frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{N} A_i \overline{e}(\mathbf{y}_k) \tilde{\phi}(\mathbf{x}_k; \theta_{ft}, \theta_b)^\top B_i^\top B_i \tilde{\phi}(\mathbf{x}^\star; \theta_{ft}, \phi_b), \tag{39}$$

where $A_1 = I$, $B_N = I$, $A_i$ can be chosen to be any symmetric positive-definite matrix, and $B_i$ can be chosen to be any positive definite matrix. In Appendix A.4, we will further show that these definitions of the weight matrices satisfy the condition that their activations are non-negative, meaning that the model $\hat{f}$ can be represented by a generic deep network with ReLU nonlinearities.

Finally, we need to define the function $f_{out}$ at the output. When a training input $\mathbf{x}_k$ is passed in, we need $f_{out}$ to propagate information about its corresponding label $\mathbf{y}_k$ as defined in Equation 38. And, when the test input $\mathbf{x}^\star$ is passed in, we need a function defined on $\overline{\mathbf{z}}^\star$. Thus, we will define $f_{out}$ as a neural network that approximates the following multiplexer function and its derivatives (as shown possible by Hornik et al., 1990):

$$f_{out}\left(\begin{bmatrix} \tilde{\mathbf{z}} \\ \overline{\mathbf{z}} \\ \check{z} \end{bmatrix}; \theta_{out}\right) = \mathbb{1}(\overline{\mathbf{z}} = \mathbf{0}) g_{pre}\left(\begin{bmatrix} \tilde{\mathbf{z}} \\ \overline{\mathbf{z}} \\ \check{z} \end{bmatrix}; \theta_g\right) + \mathbb{1}(\overline{\mathbf{z}} \neq \mathbf{0}) h_{post}(\overline{\mathbf{z}}; \theta_h), \tag{40}$$

where $g_{\text{pre}}$ is a linear function with parameters $\theta_g$ such that $\nabla_{\mathbf{z}}\ell = e(\mathbf{y})$ satisfies Equation 38 (see Section 4.4.3) and $h_{\text{post}}(\cdot;\theta_h)$ is a neural network with one or more hidden layers. As shown in Appendix A.1.4, the post-update value of $f_{\text{out}}$ is

$$f_{\text{out}}\left(\begin{bmatrix}\tilde{\mathbf{z}}^\star\\ \overline{\mathbf{z}}^\star\\ \check{z}^\star\end{bmatrix};\phi_{\text{out}}\right) = h_{\text{post}}(\overline{\mathbf{z}}^\star;\theta_h). \tag{41}$$

Now, combining Equations 39 and 41, we can see that the post-update value is the following:

$$\hat{f}(\mathbf{x}^\star;\phi) = h_{\text{post}}\left(-\alpha\frac{1}{K}\sum_{k=1}^{K}\sum_{i=1}^{N} A_i\overline{e}(\mathbf{y}_k)\tilde{\phi}(\mathbf{x}_k;\theta_{\text{ft}},\theta_b)^T B_i^T B_i\tilde{\phi}(\mathbf{x}^\star;\theta_{\text{ft}},\phi_b);\theta_h\right) \tag{42}$$

In summary, so far, we have chosen a particular form of weight matrices, feature extractor, and output function to decouple forward and backward information flow and recover the post-update function above. Now, our goal is to show that the above function $\hat{f}(\mathbf{x}^\star;\phi)$ is a universal learning algorithm approximator, as a function of $(\{(\mathbf{x},\mathbf{y})_k\},\mathbf{x}^\star)$. For notational clarity, we will use use $k_i(\mathbf{x}_k,\mathbf{x}^\star) := \tilde{\phi}(\mathbf{x}_k;\theta_{\text{ft}},\theta_b)^T B_i^T B_i\tilde{\phi}(\mathbf{x}^\star;\theta_{\text{ft}},\phi_b)$ to denote the inner product in the above equation, noting that it can be viewed as a type of kernel with the RKHS defined by $B_i\tilde{\phi}(\mathbf{x};\theta_{\text{ft}},\theta_b)$.[3] The connection to kernels is not in fact needed for the proof, but provides for convenient notation and an interesting observation. We can now simplify the form of $\hat{f}(\mathbf{x}^\star,\phi)$ as the following equation:

$$\hat{f}(\mathbf{x}^\star;\phi) = h_{\text{post}}\left(-\alpha\frac{1}{K}\sum_{i=1}^{N}\sum_{k=1}^{K} A_i\overline{e}(\mathbf{y}_k)k_i(\mathbf{x}_k,\mathbf{x}^\star);\theta_h\right) \tag{43}$$

Intuitively, Equation 43 can be viewed as a sum of basis vectors $A_i\overline{e}(\mathbf{y}_k)$ weighted by $k_i(\mathbf{x}_k,\mathbf{x}^\star)$, which is passed into $h_{\text{post}}$ to produce the output. In Appendix A.3, we show that we can choose $\overline{e}$, $\theta_{\text{ft}}$, $\theta_h$, each $A_i$, and each $B_i$ such that Equation 43 can approximate any continuous function of $(\{(\mathbf{x},\mathbf{y})_k\},\mathbf{x}^\star)$ on compact subsets of $\mathbb{R}^{\dim(\mathbf{y})}$. As in the one-shot setting, the bias transformation variable $\theta_b$ plays a vital role in our construction, as it breaks the symmetry within $k_i(\mathbf{x},\mathbf{x}^\star)$. Without such asymmetry, it would not be possible for our constructed function to represent any function of $\mathbf{x}$ and $\mathbf{x}^\star$ after one gradient step.

In conclusion, we have shown that there exists a neural network structure for which $\hat{f}(\mathbf{x}^\star;\phi)$ is a universal approximator of $f_{\text{target}}(\{(\mathbf{x},\mathbf{y})_k\},\mathbf{x}^\star)$.

3 Due to the symmetry of kernels, this requires interpreting $\theta_b$ as part of the input, rather than a kernel hyperparameter, so that the left input is $(\mathbf{x}_k,\theta_b)$ and the right one is $(\mathbf{x}^\star,\phi_b)$.

In Section 4.4.2 and Appendix A.2, we showed that the post-update function $\hat{f}(\mathbf{x}^\star;\phi)$ takes the following form:

$$\hat{f}(\mathbf{x}^\star;\phi) = h_{\text{post}}\left(-\alpha\frac{1}{K}\sum_{i=1}^{N}\sum_{k=1}^{K} A_i\overline{e}(\mathbf{y}_k)k_i(\mathbf{x}_k,\mathbf{x}^\star);\theta_h\right)$$

In this section, we aim to show that the above form of $\hat{f}(\mathbf{x}^\star;\phi)$ can approximate any function of $\{(\mathbf{x},\mathbf{y})_k; k \in 1...K\}$ and $\mathbf{x}^\star$ that is invariant to the ordering of the training datapoints $\{(\mathbf{x},\mathbf{y})_k; k \in 1...K\}$. The proof will be very similar to the one-shot setting proof in Appendix A.1.1

Similar to Appendix A.1.1, we will ignore the first and last elements of the sum by defining $B_1$ to be $\epsilon I$ and $A_N$ to be $\epsilon I$, where $\epsilon$ is a small positive constant to ensure positive definiteness. We will then re-index the first summation over $i = 2...N-1$ to instead use two indexing variables $j$ and $l$ as follows:

$$\hat{f}(\mathbf{x}^\star;\phi) \approx h_{\text{post}}\left(-\alpha\frac{1}{K}\sum_{j=0}^{J-1}\sum_{l=0}^{L-1}\sum_{k=1}^{K} A_{jl}\overline{e}(\mathbf{y}_k)k_{jl}(\mathbf{x}_k,\mathbf{x}^\star);\theta_h\right)$$

As in Appendix A.1.1, we will define the function $k_{jl}$ to be an indicator function over the values of $\mathbf{x}_k$ and $\mathbf{x}^\star$. In particular, we will reuse Lemma A.1.1, which was proved in Appendix A.1.2 and is copied below:

**Lemma A.1.1.** *We can choose $\theta_{ft}$ and each $B_{jl}$ such that*

$$k_{jl}(\mathbf{x},\mathbf{x}^\star) := \begin{cases} 1 & \textit{if } \text{discr}(\mathbf{x}) = \mathbf{e}_j \textit{ and } \text{discr}(\mathbf{x}^\star) = \mathbf{e}_l \\ 0 & \textit{otherwise} \end{cases} \tag{30}$$

*where* $\text{discr}(\cdot)$ *denotes a function that produces a one-hot discretization of its input and* $\mathbf{e}$ *denotes the o-indexed standard basis vector.*

Likewise, we will chose $\overline{e}(\mathbf{y}_k)$ to be the linear function that outputs $J * L$ stacked copies of $\mathbf{y}_k$. Then, we will define $A_{jl}$ to be a matrix that selects the copy of $\mathbf{y}_k$ in the position corresponding to $(j,l)$, i.e. in the position $j + J * l$. This can be achieved using a diagonal $A_{jl}$ matrix with diagonal values of $1+\epsilon$ at the positions corresponding to the $n$th vector, and $\epsilon$ elsewhere, where $n = (j + J * l)$ and $\epsilon$ is used to ensure that $A_{jl}$ is positive definite.

As a result, the post-update function is as follows:

$$\hat{f}(\mathbf{x}^\star;\phi) \approx h_{\text{post}}\left(-\alpha\frac{1}{K}\sum_{k=1}^{K} v(\mathbf{x}_k,\mathbf{x}^\star,\mathbf{y}_k);\theta_h\right), \text{ where } v(\mathbf{x},\mathbf{x}^\star,\mathbf{y}) \approx \begin{bmatrix} \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbf{y}_k \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix},$$

where $\mathbf{y}_k$ is at the position $j + J * l$ within the vector $v(\mathbf{x}_k,\mathbf{x}^\star,\mathbf{y}_k)$, where $j$ satisfies $\text{discr}(\mathbf{x}_k) = \mathbf{e}_j$ and where $l$ satisfies $\text{discr}(\mathbf{x}^\star_k) = \mathbf{e}_l$.

For discrete, one-shot labels $\mathbf{y}_k$, the summation over $v$ amounts to frequency counts of the triplets $(\mathbf{x}_k,\mathbf{x}^\star,\mathbf{y}_k)$. In the setting with continuous labels, we cannot attain frequency counts, as we do not have access to a discretized version of the label. Thus, we must make the assumption that no two datapoints share the same input value $\mathbf{x}_k$. With this assumption, the summation over $v$ will contain the output values $\mathbf{y}_{k'}$ at the index corresponding to the value of $(\mathbf{x}_{k'},\mathbf{x}^\star)$. For both discrete and continuous labels, this representation is redundant in $\mathbf{x}^\star$, but nonetheless contains sufficient information to decode the test input $\mathbf{x}^\star$ and set of datapoints $\{(\mathbf{x},\mathbf{y})_k\}$ (but not the order of datapoints).

Since $h_{\text{post}}$ is a universal function approximator and because its input contains all of the information of $(\{(\mathbf{x},\mathbf{y})_k\},\mathbf{x}^\star)$, the function $\hat{f}(\mathbf{x}^\star;\phi) \approx h_{\text{post}}\left(-\alpha\frac{1}{K}\sum_{k=1}^{K} v(\mathbf{x}_k,\mathbf{x}^\star,\mathbf{y}_k);\theta_h\right)$ is a universal function approximator with respect to $\{(\mathbf{x},\mathbf{y})_k\}$ and $\mathbf{x}^\star$.

## A.4 UNIVERSALITY WITH DEEP RELU NETWORKS

In this appendix, we show that the network architecture with linear layers analyzed in the Sections 4.4.1 and 4.4.2 can be represented by a deep network with ReLU nonlinearities. We will do so by showing that the input and activations within the linear layers are all non-negative.

First, consider the input $\phi(\cdot;\theta_{ft},\theta_b)$ and $\tilde{\phi}(\cdot;\phi_{ft},\phi_b)$. The input $\tilde{\phi}(\cdot;\theta_{ft},\theta_b)$ is defined to consist of three terms. The top term, $\tilde{\tilde{\phi}}$ is defined in Appendices A.1.2 and A.3 to be a discretization (which is non-negative) both before and after the parameters are updated. The middle term is defined to be a constant $\mathbf{0}$. The bottom term, $\theta_b$, is defined to be $0$ before the gradient update and is not used afterward.

Next, consider the weight matrices, $W_i$. To determine that the activations are non-negative, it is now sufficient to show that the products $W_N$, $W_{N-1}W_N$, ..., $\prod_{i=1}^{N} W_i$ are positive semi-definite. To do so, we need to show that the products $\prod_{i=j}^{N} \tilde{W}_i$, $\prod_{i=j}^{N} \overline{W}_i$, and $\prod_{i=j}^{N} \check{w}_i$ are PSD for $j = 1, ..., N$. In Appendix A.1.2, each $B_i = \tilde{M}_{i+1}$ is defined to be positive definite; and in Appendix A.1.3, we define the products $\prod_{i=j+1}^{N} \tilde{W}_i = \tilde{M}_j + 1$. Thus, the conditions on the products of $\tilde{W}_i$ are satisfied. In Appendices A.1.1 and A.3, each $A_i$ are defined to be symmetric positive definite matrices. In Appendix A.1.3, we define $\overline{W}_i = \overline{M}_{i-1}^{-1}\overline{M}_i$ where $A_i = \overline{M}_{i-1}\overline{M}_{i-1}^{T}$. Thus, we can see that each $\overline{M}_i$ is also symmetric positive definite, and therefore, each $\overline{W}_i$ is positive definite. Finally, the purpose of the weights $\check{w}_i$ is to provide nonzero gradients to the input $\theta_b$, thus a positive value for each $\check{w}_i$ will suffice.

## A.5 PROOF OF THEOREM 4.4.1

Here we provide a proof of Theorem 4.4.1, copied below:

**Theorem 4.4.1.** *The gradient of the standard mean-squared error objective evaluated at $\hat{\mathbf{y}} = \mathbf{0}$ is a linear, invertible function of $\mathbf{y}$.*

For the standard mean-squared error objective, $\ell(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{2}\|\mathbf{y} - \hat{\mathbf{y}}\|^2$, the gradient is $\nabla_{\hat{\mathbf{y}}}\ell(\mathbf{y}, \mathbf{0}) = -\mathbf{y}$, which satisfies the requirement, as $A = -I$ is invertible.

## A.6 PROOF OF THEOREM 4.4.2

Here we provide a proof of Theorem 4.4.2, copied below:

**Theorem 4.4.2.** *The gradient of the softmax cross entropy loss with respect to the pre-softmax logits is a linear, invertible function of $\mathbf{y}$, when evaluated at $\mathbf{0}$.*

For the standard softmax cross-entropy loss function with discrete, one-hot labels $\mathbf{y}$, the gradient is $\nabla_{\hat{\mathbf{y}}}\ell(\mathbf{y}, \mathbf{0}) = \mathbf{c} - \mathbf{y}$ where $\mathbf{c}$ is a constant vector of value $c$ and where we are denoting $\hat{\mathbf{y}}$ as the pre-softmax logits. Since $\mathbf{y}$ is a one-hot representation, we can rewrite the gradient as $\nabla_{\hat{\mathbf{y}}}\ell(\mathbf{y}, \mathbf{0}) = (C - I)\mathbf{y}$, where $C$ is a constant matrix with value $c$. Since $A = C - I$ is invertible, the cross entropy loss also satisfies the above requirement. Thus, we have shown that both of the standard supervised objectives of mean-squared error and cross-entropy allow for the universality of gradient-based meta-learning.

## A.7 MAML EXPERIMENTAL DETAILS

In this section, we provide additional details of the experimental set-up and hyperparameters for the experiments in Section 4.6.

### A.7.1 *Classification*

For N-way, K-shot classification, each gradient is computed using a batch size of NK examples. For Omniglot, the 5-way convolutional and non-convolutional MAML models were each trained with 1 gradient step with step size $\alpha = 0.4$ and a meta batch-size of 32 tasks. The network was evaluated using 3 gradient steps with the same step size $\alpha = 0.4$. The 20-way convolutional MAML model was trained and evaluated with 5 gradient steps with step size $\alpha = 0.1$. During training, the meta batch-size was set to 16 tasks. For MiniImagenet, both models were trained using 5 gradient steps of size $\alpha = 0.01$, and evaluated using 10 gradient steps at test time. Following Ravi and Larochelle (2017), 15 examples per class were used for evaluating the post-update meta-gradient. We used a meta batch-size of 4 and 2 tasks for 1-shot and 5-shot training respectively. All models were trained for 60000 iterations on a single NVIDIA Pascal Titan X GPU.

### A.7.2 *Reinforcement Learning*

In all reinforcement learning experiments, the MAML policy was trained using a single gradient step with $\alpha = 0.1$. During evaluation, we found that halving the learning rate after the first gradient step produced superior performance. Thus, the step size during adaptation was set to $\alpha = 0.1$ for the first step, and $\alpha = 0.05$ for all future steps. The step sizes for the baseline methods were manually tuned for each domain. In the 2D navigation, we used a meta batch size of 20; in the locomotion problems, we used a meta batch size of 40 tasks. The MAML models were trained for up to 500 meta-iterations, and the model with the best average return during training was used for evaluation. For the ant goal velocity task, we added a positive reward bonus at each timestep to prevent the ant from ending the episode.

### A.7.3 *Inductive Bias Experiments*

For Omniglot, all meta-learning methods were trained using code provided by the authors of the respective papers, using the default model architectures and hyperparame-

ters. The model embedding architecture was the same across all methods, using 4 convolutional layers with $3 \times 3$ kernels, 64 filters, stride 2, batch normalization, and ReLU nonlinearities. The convolutional layers were followed by a single linear layer. All methods used the Adam optimizer with default hyperparameters. Other hyperparameter choices were specific to the algorithm and can be found in the respective papers. For MAML in the sinusoid domain, we used a fully-connected network with two hidden layers of size 100, ReLU nonlinearities, and a bias transformation variable of size 10 concatenated to the input. This model was trained for 70,000 meta-iterations with 5 inner gradient steps of size $\alpha = 0.001$. For SNAIL in the sinusoid domain, the model consisted of 2 blocks of the following: 4 dilated convolutions with $2 \times 1$ kernels 16 channels, and dilation size of 1,2,4, and 8 respectively, then an attention block with key/value dimensionality of 8. The final layer is a $1 \times 1$ convolution to the output. Like MAML, this model was trained to convergence for 70,000 iterations using Adam with default hyperparameters. We evaluated the MAML and SNAIL models for 1200 trials, reporting the mean and 95% confidence intervals. For computational reasons, we evaluated the MetaNet model using 600 trials, also reporting the mean and 95% confidence intervals.

Following prior work (Santoro et al., 2016), we downsampled the Omniglot images to be $28 \times 28$. When scaling or shearing the digits to produce out-of-domain data, we transformed the original $105 \times 105$ Omniglot images, and then downsampled to $28 \times 28$.

### A.7.4 *Depth Experiments*

In the depth comparison, all models were trained to convergence using 70,000 iterations. Each model was defined to have a fixed number of hidden units based on the total number of parameters (fixed at around 40,000) and the number of hidden layers. Thus, the models with 2, 3, 4, and 5 hidden layers had 200, 141, 115, and 100 units per layer respectively. For the model with 1 hidden layer, we found that using more than $20,000$ hidden units, corresponding to $40,000$ parameters, resulted in poor performance. Thus, the results reported in the paper used a model with 1 hidden layer with 250 units which performed much better. We trained each model three times and report the mean and standard deviation of the three runs. The performance of an individual run was computed using the average over

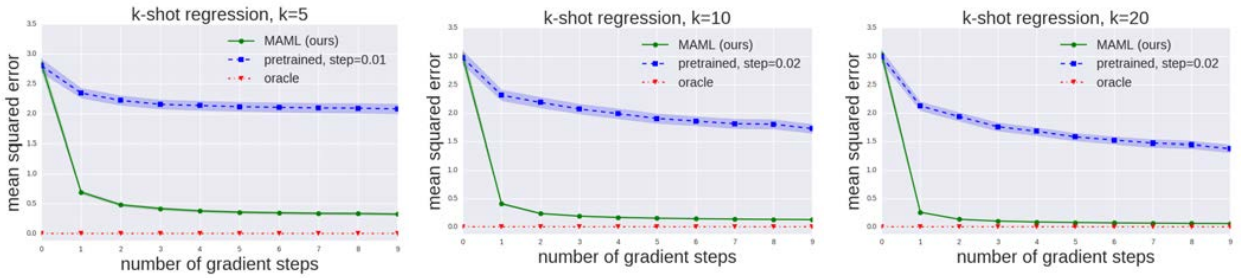


Figure 45: Quantitative sinusoid regression results showing test-time learning curves with varying numbers of K test-time samples. Each gradient step is computed using the same K examples. Note that MAML continues to improve with additional gradient steps without overfitting to the extremely small dataset during meta-testing, and achieves a loss that is substantially lower than the baseline fine-tuning approach.

## A.8 MAML ADDITIONAL SINUSOID RESULTS

In Figure 45, we show the full quantitative results of the MAML model trained on 10-shot learning and evaluated on 5-shot, 10-shot, and 20-shot. In Figure 46, we show the qualitative performance of MAML and the pretrained baseline on randomly sampled sinusoids.

## A.9 MAML ADDITIONAL COMPARISONS

In this section, we include more thorough evaluations of our approach, including additional multi-task baselines and a comparison representative of the approach of Rei (2015).

### A.9.1 *Multi-task baselines*

The pretraining baseline in the main text trained a single network on all tasks, which we referred to as "pretraining on all tasks". To evaluate the model, as with MAML, we fine-tuned this model on each test task using K examples. In the domains that we study, different tasks involve different output values for the same input. As a result, by pretraining on all tasks, the model would learn to output the average output for a particular input value. In some instances, this model may learn very little about the actual domain, and instead learn about the range of the output space.

We experimented with a multi-task method to provide a point of comparison, where

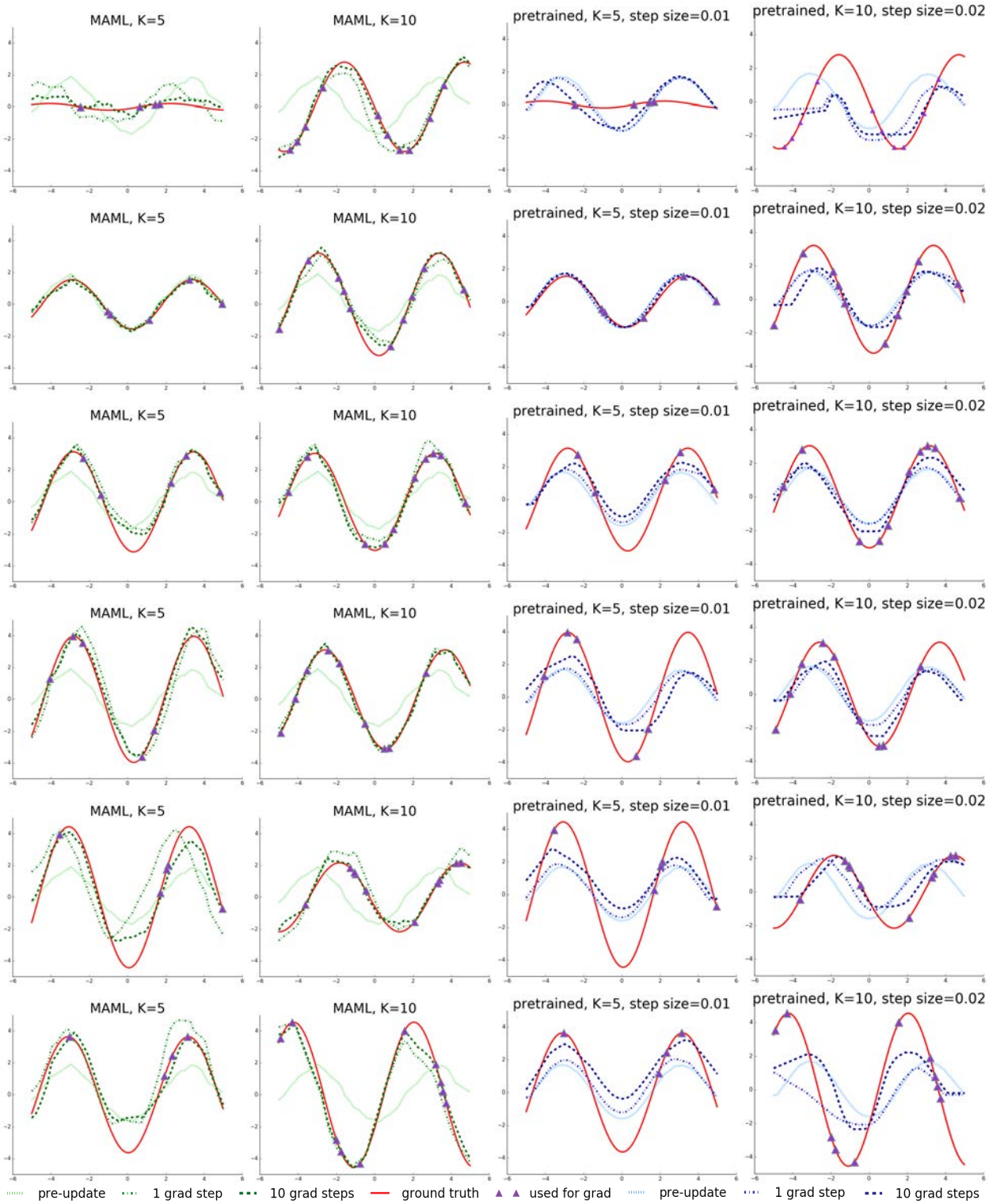


Figure 46: A random sample of qualitative results from the sinusoid regression task.

Table 10: Additional multi-task baselines on the sinusoid regression domain, showing 5-shot mean squared error. The results suggest that MAML is learning a solution more sophisticated than the mean optimal parameter vector.

| num. grad steps | 1 | 5 | 10 |
|---|---|---|---|
| multi-task, no reg | 4.19 | 3.85 | 3.69 |
| multi-task, l2 reg | 7.18 | 5.69 | 5.60 |
| multi-task, reg to mean $\theta$ | 2.91 | 2.72 | 2.71 |
| pretrain on all tasks | 2.41 | 2.23 | 2.19 |
| MAML (ours) | **0.67** | **0.38** | **0.35** |

instead of averaging in the output space, we averaged in the parameter space. To achieve averaging in parameter space, we sequentially trained 500 separate models on 500 tasks drawn from $p(\mathcal{T})$. Each model was initialized randomly and trained on a large amount of data from its assigned task. We then took the average parameter vector across models and fine-tuned on 5 datapoints with a tuned step size. All of our experiments for this method were on the sinusoid task because of computational requirements. The error of the individual regressors was low: less than 0.02 on their respective sine waves.

We tried three variants of this set-up. During training of the individual regressors, we tried using one of the following: no regularization, standard $\ell_2$ weight decay, and $\ell_2$ weight regularization to the mean parameter vector thus far of the trained regressors. The latter two variants encourage the individual models to find parsimonious solutions. When using regularization, we set the magnitude of the regularization to be as high as possible without significantly deterring performance. In our results, we refer to this approach as "multi-task". As seen in the results in Table 10, we find averaging in the parameter space (multi-task) performed worse than averaging in the output space (pretraining on all tasks). This suggests that it is difficult to find parsimonious solutions to multiple tasks when training on tasks separately, and that MAML is learning a solution that is more sophisticated than the mean optimal parameter vector.

Table 11: 5-way Omniglot Classification

| | 1-shot | 5-shot |
|---|---|---|
| context vector | $94.9 \pm 0.9\%$ | $97.7 \pm 0.3\%$ |
| MAML | $\mathbf{98.7 \pm 0.4}\%$ | $\mathbf{99.9 \pm 0.1}\%$ |

Table 12: 2D Pointmass, average return

| num. grad steps | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| context vector | −42.42 | −13.90 | −5.17 | **−3.18** |
| MAML (ours) | −40.41 | **−11.68** | **−3.33** | −3.23 |

### A.9.2 *Context vector adaptation*

Rei (2015) developed a method which learns a context vector that can be adapted online, with an application to recurrent language models. The parameters in this context vector are learned and adapted in the same way as the parameters in the MAML model. To provide a comparison to using such a context vector for meta-learning problems, we concatenated a set of free parameters $\mathbf{z}$ to the input $\mathbf{x}$, and only allowed the gradient steps to modify $\mathbf{z}$, rather than modifying the model parameters $\theta$, as in MAML. For image inputs, $\mathbf{z}$ was concatenated channel-wise with the input image. We ran this method on Omniglot and two RL domains following the same experimental protocol. We report the results in Tables 11, 12, and 13. Learning an adaptable context vector performed well on the toy pointmass problem, but sub-par on more difficult problems, likely due to a less flexible meta-optimization.

Table 13: Half-cheetah forward/backward, average return

| num. grad steps | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| context vector | −40.49 | −44.08 | −38.27 | −42.50 |
| MAML (ours) | −50.69 | **293.19** | **313.48** | **315.65** |

## A.10 AMBIGUOUS CELEBA DETAILS

To construct our ambiguous few-shot variant of CelebA, we take the entire base set of attributes holding out 10 attributes for testing. We consider every combination of 2 attributes, discarding those with insufficient numbers of examples. This leave us with a total of 387 training tasks and 43 testing attributes. We partition our meta-training set and meta-validation set to 337/50 respectively.

During meta-training, we sample 2 random attributes to construct a positive class and randomly sample examples with neither attribute as negative examples. During testing of our approach, we sample 3 attributes from the test set, and sample the 3 corresponding 2-tuples to form the test task.

The training attributes are:

5 o'clock Shadow, Arched Eyebrows, Attractive, Bags Under Eyes, Bald, Bangs, Big Lips, Big Nose, Black Hair, Blond Hair, Blurry, Brown Hair, Bushy Eyebrows, Chubby, Double Chin, Eyeglasses, Goatee, Gray Hair, Heavy Makeup, High Cheekbones, Male, Mouth Slightly Open, Mustache, Narrow Eyes, No Beard, Oval Face, Pale Skin, Pointy Nose, Receding Hairline, Rosy Cheeks

The full testing attributes are:

Sideburns, Smiling, Straight Hair, Wavy Hair, Wearing Earrings, Wearing Hat, Wearing Lipstick, Wearing Necklace, Wearing Necktie, Young.

## A.11 PLATIPUS EXPERIMENTAL DETAILS

In the illustrative experiments, we use a fully connected network with 3 ReLU layers of size 100. Following Finn et al. (2017b), we additionally use a bias transformation variable (described in Section 7.1.4), concatenated to the input, with size 20. Both methods use 5 inner gradient steps on $\mathcal{D}^{tr}$ with step size $\alpha = 0.001$ for regression and $\alpha = 0.01$ for classification. The inference network and prior for PLATIPUS both use one gradient step. For PLATIPUS, we weight the KL term in the objective by 0.1 for 1D regression and 0.01 for 2D classification.

For CelebA, we adapt the base convolutional architecture described in Section 4.6. Our approximate posterior and prior have dimensionality matching the underlying model. We tune our approach over the inner learning rate $\alpha$, a weight on the $D_{KL}$, the scale of the initialization of $\boldsymbol{\mu}_\theta, \boldsymbol{\sigma}^2_\theta, \mathbf{v}_q, \gamma_p, \gamma_q$, with early stopping on the validation set.

At meta-test time, we evaluate our approach by taking 10 samples from the prior before determining the assignments. The assignments are made based on the complete likelihood of the testing examples (including the negatives).

| MiniImagenet | 5-way, 1-shot Accuracy |
|---|---|
| MAML (Finn et al., 2017a) | $48.70 \pm 1.84\%$ |
| LLAMA (Grant et al., 2018) | $49.40 \pm 1.83\%$ |
| Reptile (Nichol and Schulman, 2018) | $\mathbf{49.97 \pm 0.32\%}$ |
| PLATIPUS (ours) | $\mathbf{50.13 \pm 1.86\%}$ |
| Meta-SGD (Z. Li et al., 2017) | $\mathbf{50.71 \pm 1.87\%}$ |
| | |
| matching nets (Vinyals et al., 2016) | $43.56 \pm 0.84\%$ |
| meta-learner LSTM (Ravi and Larochelle, 2017) | $43.44 \pm 0.77\%$ |
| SNAIL (Mishra et al., 2018)* | $45.10 \pm 0.00\%$ |
| prototypical networks (Snell et al., 2017) | $46.61 \pm 0.78\%$ |
| mAP-DLM (Snell et al., 2017) | $49.82 \pm 0.78\%$ |
| GNN (Garcia and Bruna, 2017) | $\mathbf{50.33 \pm 0.36\%}$ |
| Relation Net (Sung et al., 2017) | $\mathbf{50.44 \pm 0.82\%}$ |

Table 14: Comparison between our approach and prior MAML-based methods (top), and other prior few-shot learning techniques on the 5-way, 1-shot MiniImagenet benchmark. Our approach gives a small boost over MAML, and is comparable to other approaches. We bold the approaches that are above the highest confidence interval lower-bound. *Accuracy using comparable network architecture.

### A.12 PLATIPUS MINIIMAGENET COMPARISON

We provide an additional comparison on the MiniImagenet dataset. Since this benchmark does not contain a large amount of ambiguity, we do not aim to show state-of-the-art performance. Instead, our goal with this experiment is to compare our approach on to MAML and prior methods that build upon MAML on this standard benchmark. Since our goal is to compare algorithms, rather than achieving maximal performance, we decouple the effect of the meta-learning algorithm and the architecture used by using the standard 4-block convolutional architecture used by Vinyals et al. (2016), Ravi and Larochelle (2017), and Finn et al. (2017a) and others. We note that better performance can likely be achieved by tuning the architecture. The results, in Table 14 indicate that our method slightly outperforms MAML and achieves comparable performance to a number of other prior methods.

# B

# EXTENSIONS TO CONTROL

## B.1 ONLINE ADAPTATION: ADDITIONAL EXPERIMENTS AND EXPERIMENTAL DETAILS

### B.1.1 *Model Prediction Errors: Pre-update vs. Post-update*

In this section, we show the effect of adaptation in the case of GBAC. In particular, we show the histogram of the K step normalized error, as well as the per-timestep visualization of this error during a trajectory. Across all tasks and environments, the post-updated model $\hat{p}_{\phi_*}$ achieves lower prediction error than the pre-updated model $\hat{p}_{\theta_*}$.

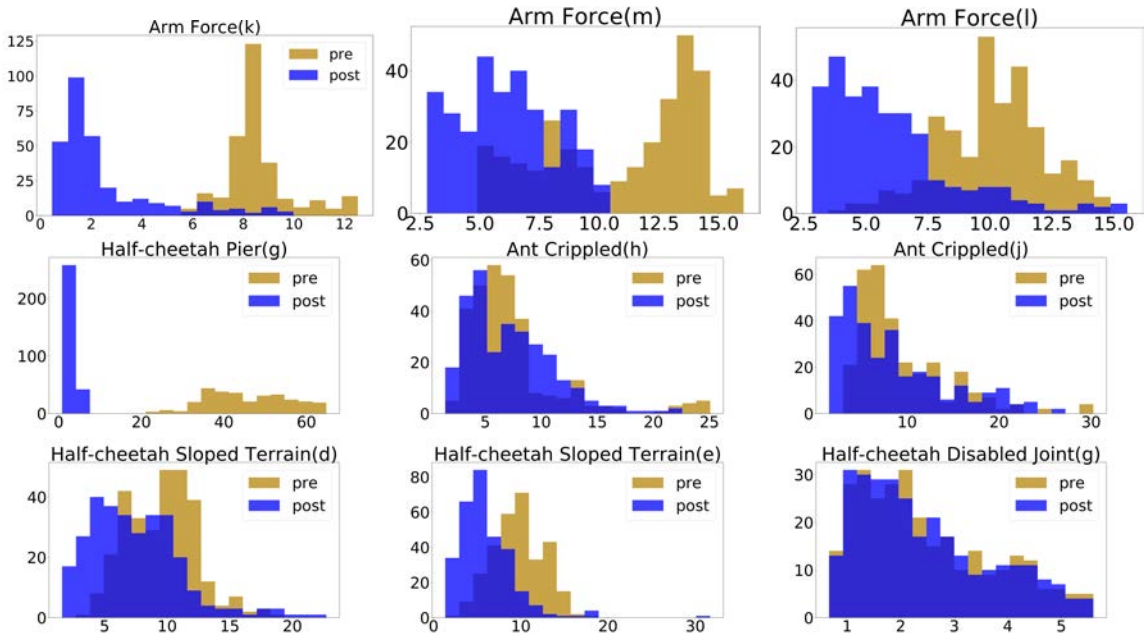


Figure 47: Histogram of the K step normalized error across different tasks. GBAC accomplishes lower model error when using the parameters given by the update rule.

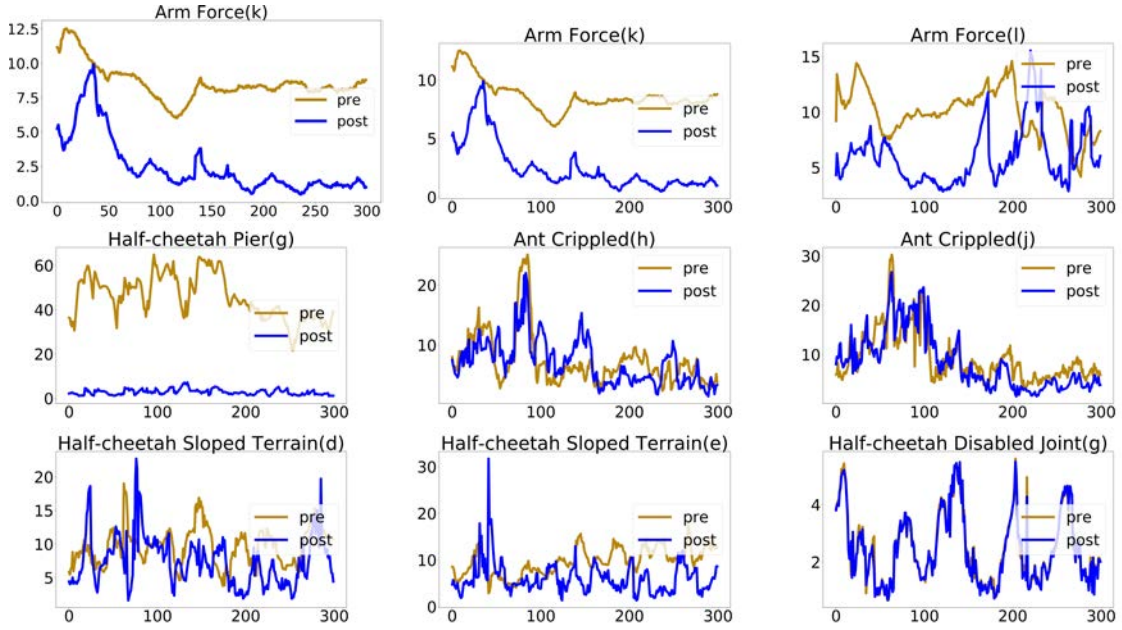


Figure 48: At each time-step we show the K step normalized error across different tasks. GBAC accomplishes lower model error using the parameters given by the update rule.

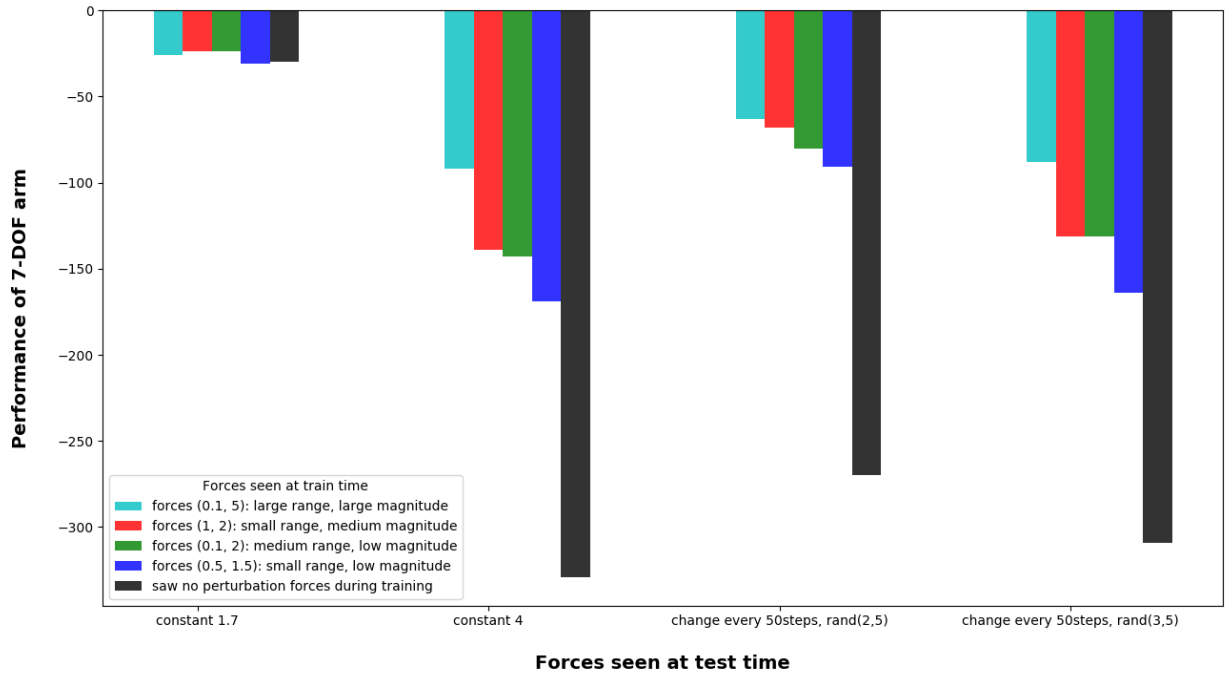


Figure 49: Effect of the meta-training distribution on test performance

### B.1.2 *Effect of Meta-Training Distribution*

To see how training distribution affects test performance, we ran an experiment that used GBAC to train models of the 7-DOF arm, where each model was trained on the same number of datapoints during meta-training, but those datapoints came from different ranges of force perturbations. We observe (in the plot below) that

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

2. A model that saw no perturbation forces during training did the worst

3. The middle 3 models show comparable performance in the "constant force = 4" case, which is an out-of-distribution task for those models. Thus, there is not actually a strong restriction on what needs to be seen during training in order for adaptation to occur at train time (though there is a general trend that more is better)

### B.1.3 *Learning Curves*

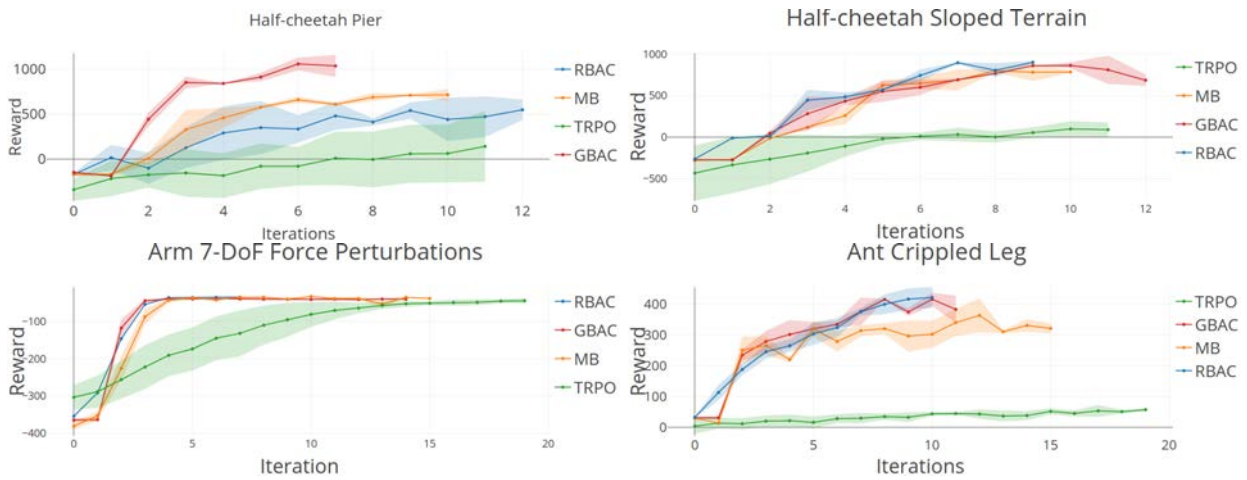


Figure 50: Learning curves of the different agents

### B.1.4 *Reward functions*

For each MuJoCo agent, the same reward function is used across its various tasks. Table 15 shows the reward functions used for each agent. We denote by $x_t$ the x-coordinate of the agent at time t, $\boldsymbol{ee}_t$ refers to the position of the end-effector of the 7-DoF arm, and $\mathbf{g}$ corresponds to the position of the desired goal.

Table 15: Reward functions

| | Reward function |
|---|---|
| **Half-cheetah** | $\frac{x_{t+1}-x_t}{0.01} - 0.05\|\mathbf{a}_t\|_2^2$ |
| **Ant** | $\frac{x_{t+1}-x_t}{0.0e} - 0.005\|\mathbf{a}_t\|_2^2 + 0.05$ |
| **7-DoF Arm** | $-\|\boldsymbol{ee}_t - \mathbf{g}\|_2^2$ |

### B.1.5 *Hyperparameters*

Below, we list the hyperparameters of our experiments. In all experiments we used a single gradient step for the update rule of GBAC. The learning rate (LR) of TRPO corresponds to the Kullback–Leibler divergence constraint. # Task/itr corresponds to the number of tasks sampled for collecting data to train the model or model, whereas # TS/itr is the total number of times steps collected (for all tasks). Finally, T refers to the horizon of the task.

Table 16: Hyperparameters for the half-cheetah tasks

| | LR | Inner LR | Epochs | K | M | Batch Size | # Tasks/itr | # TS/itr | T | $n_A$ Train | H Train | $n_A$ Test | H Test |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **GBAC** | 0.001 | 0.01 | 50 | 32 | 32 | 500 | 32 | 64000 | 1000 | 1000 | 10 | 2500 | 15 |
| **RBAC** | 0.001 | - | 50 | 32 | 32 | 500 | 32 | 64000 | 1000 | 1000 | 10 | 2500 | 15 |
| **MB** | 0.001 | - | 50 | - | - | 500 | 64 | 64000 | 1000 | 1000 | 10 | 2500 | 15 |
| **TRPO** | 0.05 | - | - | - | - | 50000 | 50 | 50000 | 1000 | - | - | - | - |

Table 17: Hyperparameters for the ant tasks

| | LR | Inner LR | Epochs | K | M | Batch Size | # Tasks/itr | # TS/itr | T | $n_A$ Train | H Train | $n_A$ Test | H Test |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **GBAC** | 0.001 | 0.001 | 50 | 10 | 16 | 500 | 32 | 24000 | 500 | 1000 | 15 | 1000 | 15 |
| **RBAC** | 0.001 | - | 50 | 32 | 16 | 500 | 32 | 32000 | 500 | 1000 | 15 | 1000 | 15 |
| **MB** | 0.001 | - | 70 | - | - | 500 | 10 | 10000 | 500 | 1000 | 15 | 1000 | 15 |
| **TRPO** | 0.05 | - | - | - | - | 50000 | 50 | 50000 | 500 | - | - | - | - |

Table 18: Hyperparameters for the 7-DoF arm tasks

| | LR | Inner LR | Epochs | K | M | Batch Size | # Tasks/itr | # TS/itr | T | $n_a$ Train | H Train | $n_a$ Test | H Test |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **GBAC** | 0.001 | 0.001 | 50 | 32 | 16 | 1500 | 32 | 24000 | 500 | 1000 | 15 | 1000 | 15 |
| **RBAC** | 0.001 | - | 50 | 32 | 16 | 1500 | 32 | 24000 | 500 | 1000 | 15 | 1000 | 15 |
| **MB** | 0.001 | - | 70 | - | - | 10000 | 10 | 10000 | 500 | 1000 | 15 | 1000 | 15 |
| **TRPO** | 0.05 | - | - | - | - | 50000 | 50 | 50000 | 500 | - | - | - | - |

## B.2 MIL EXPERIMENTAL DETAILS

In this section, we provide additional experimental details for all experiments, including information regarding data collection, evaluation, and training hyperparameters.

### B.2.1 *Simulated Reaching*

EXPERIMENTAL SETUP: In both vision and no-vision cases of this experiment, the input to the policy includes the arm joint angles and the end-effector position. In the vision variant, the $80 \times 64$ RGB image is also provided as input. In the non-vision version, the 2D positions of the objects are fed into the policy, but the index of the target object within the state vector is not known and must be inferred from the demonstration. The policy output corresponds to torques applied to the two joints of the arm. A policy rollout is considered a success if it comes within 0.05 meters of the goal within the last 10 timesteps, where the size of the arena is $0.6 \times 0.6$ meters.

To obtain the expert policies for this task, we use iLQG trajectory optimization to generate solutions for each task (using knowledge of the goal), and then collect several demonstrations per task from the resulting policy with injected Gaussian noise. At meta-test time, we evaluate the policy on 150 tasks and 10 different trials per task (1500 total trials) where each task corresponds to a held-out color. Note that the demonstration provided at meta-test time usually involves different target and distractor positions than its corresponding test trial. Thus, the one-shot learned policy must learn to localize the target using the demonstration and generalize to new positions, while meta-training must learn to handle different colors.

HYPERPARAMETERS: For all vision-based policies, we use a convolutional neural network policy with 3 convolution layers each with 40 $3 \times 3$ filters, followed by 4 fully-connected layers with hidden dimension 200. For this domain only, we simply flattened the final convolutional map rather than transforming it into spatial feature points. The recurrent policies additionally use an LSTM with 512 units that takes as input the features from the final layer. For non-vision policies, we use the same architecture without the convolutional layers, replacing the output of the convolutional layers with the state input. All methods are trained using a meta batch-size of 5 tasks. The policy trained with meta-imitation learning uses 1 meta-gradient update with step size 0.001 and bias transformation with dimension 10. We also find it helpful to clip the meta-gradient to lie in the interval $[-20, 20]$ before applying it. We use the normal single-head architecture

for MIL as shown in Figure 23.

### B.2.2 *Simulated Pushing*

EXPERIMENTAL SETUP: The policy input consists of a $125 \times 125$ RGB image and the robot joint angles, joint velocities, and end-effector pose. A push is considered a success if the center of the target object lands on the red target circle for at least 10 timesteps within a 100-timestep episode. The reported pushing success rates are computed over 74 tasks with 6 trials per task (444 total trials).

We acquired a separate demonstration policy for each task using the trust-region policy optimization (TRPO) algorithm. The expert policy inputs included the target and distractor object poses rather than vision input. To encourage the expert policies to take similar strategies, we first trained a single policy on a single task, and then initialized the parameters of all of the other policies with those from the first policy. When initializing the policy parameters, we increased the variance of the converged policy to ensure appropriate exploration.

HYPERPARAMETERS: For all methods, we use a neural network policy with 4 strided convolution layers with 16 $5 \times 5$ filters, followed by a spatial softmax and 3 fully-connected layers with hidden dimension 200. For optimization, each method use a meta-batch size of 15 tasks. MIL uses 1 inner gradient descent step with step size $\alpha = 0.01$, inner gradient clipping within the range $[-10, 10]$, and bias transformation with dimension 20. We also use an additional bias transformation that is concatenated to the image at each time step with the same size as the input image. The LSTM policy uses 512 hidden units.

Because this domain is significantly more challenging than the simulating reaching domain, we found it important to use the two-head architecture described in section 7.1.3. We include an ablation of the two-head architecture in Table 19, demonstrating the benefit of this choice.

### B.2.3 *Real-World Placing*

EXPERIMENTAL SETUP: The videos in the demo are composed of a sequence of $320 \times 240$ RGB images from the robot camera. We pre-process the demonstrations by downsampling the images by a factor of 2 and cropping them to be of size $100 \times 90$. Since the videos we collected are of variable length, we subsample the videos such that they all have fixed time horizon 30.

| method | 1-head | 2-head |
|---|---|---|
| MIL with 1-shot | 80.63% | 85.81% |
| MIL with 5-shot | 82.63% | 88.75% |

Table 19: Ablation test on 1-head and 2-head architecture for simulated pushing as shown in Figure 23, using a dataset with 9200 demonstrations for meta-learning. Using two heads leads to significantly better performance in this domain.

To collect demonstration data for one task, we randomly select one holding object and three placing containers from our training set of objects (see the third image of Figure 26), and place those three objects in front of the robot in 8 random positions. In this way, we collect 1293 demonstrations, where we use 96 of them as validation set and the rest as the training set.

During policy evaluation, we evaluate the policy with 18 tasks and 4 trials per task (72 total trials) where we use 1 placing target and 1 holding object from the test set for each task. In addition, we manually code an "open gripper" action at the end of the trajectory, which causes the robot to drop the holding object. We define success as whether or not the held object landed in or on the target container after the gripper is opened.

HYPERPARAMETERS: We use a neural network policy with 3 strided convolution layers and 2 non-strided convolutions layers with 64 $3 \times 3$ filters, followed by a spatial softmax and 3 fully-connected layers with hidden dimension 100. We initialize the first convolution layer from VGG-19 and keep it fixed during meta-training. Following prior work (T. Zhang et al., 2017), we change the objective to be a mixture of $\ell_1$ and $\ell_2$ loss, where $\ell_2$ loss is scaled down by 100, and add an auxiliary loss that regresses from the learned features at the first time step to the 2D position of the target container. We determine the position of the target container from the end-effector position at the final timestep of the demonstration; this does not require additional supervision beyond the demonstration. Additionally, we also feed the predicted 2D position of the target into the fully-connected layers of the network. MIL uses a meta-batch size of 12 tasks, 5 inner gradient descent steps with step size 0.005, inner gradient clipping within the range $[-30, 30]$, and bias transformation with dimension 20. We also use the single-head architecture for MIL just as what we do for simulated reaching. The LSTM policy uses 512 hidden units.

## B.3 DAML HYPERPARAMETER AND EXPERIMENTAL DETAILS

In this appendix, we include experimental details and hyperparameters for each of the experiments.

### B.3.1 *Data collection*

All of the human and robot demonstrations are collecting at 10 Hz, and take approximately 3-8 seconds. Then, for meta-training, we randomly sampled 40 of the frames and corresponding actions to be used as the demonstration. The robot policy is executed at 10 Hz.

### B.3.2 *Architecture Choices*

The most critical part in our network architecture is the learned adaptation objective consisting of temporal convolutions, which is simple but effective. We analyzed the difference between the temporal loss and the linear loss in Section 7.2.7, which suggests a simpler learned adaptation objective does not suffice. As for other parts of our network, any convolutional neural network with a reasonable number of parameters would work thanks to the model-agnostic property of our algorithm.

### B.3.3 *Placing, Pushing, and Pick-and-Place Experiments*

We include the details of the experiments in Section 7.2.7.1 for the placing, pushing, and pick-and-place tasks. The architecture and hyperparameters were selected by evaluating the end-effector loss and control loss on a validation set of 12, 8, 12 objects for placing, pushing, and pick-and-place respectively, sampled and held out from the training data. For placing and pushing, the inputs are RGB images with size $100 \times 90$, and for pick-and-place, we use RGB-D images with size $110 \times 90$. We train separate models for each of the three settings.

For placing, the policy architecture uses 5 convolutional layers with 64 $3 \times 3$ convolutional filters in each layer where the first three layers are with stride 2 and the last two layer is with stride 1. The first convolutional layer uses pretrained weights from VGG-19. It also uses 3 fully connected layers of size 100, and a learned adaptation objective with two layers of 32 $10 \times 1$ convolutional filters followed by one layer of $1 \times 1$ convolutions. For pushing, the policy architecture, uses the same convolutional network, 4 fully con-

nected layers of size 50, and a learned adaptation objective with two layers of 10 and 20 $10 \times 1$ convolutional filters for action and final gripper pose prediction respectively followed by one layer of $1 \times 1$ convolutions. For pick-and-place, the policy architecture uses the same convolutional network except that the first convolutional layer is not pretrained. It also uses 16 $3 \times 3$ convolutional filters that operate on the depth input and concatenates the depth stream to the RGB stream after the first convolutional layer, channel-wise, following the approach by T. Zhang et al. (2017). For pick-and-place, we use two gripper poses – one intermediate, when the gripper contacts the item to pick, and one final pose at the end of the trajectory. The architecture also uses 4 fully connected layers of size 50, and a learned adaptation objective with two layers of 10, 30, and 30, $10 \times 1$ convolutional filters for action, final gripper pose and pickup gripper pose prediction respectively followed by one layer of $1 \times 1$ convolutions. All architectures use ReLU nonlinearities and layer normalization.

All the baseline methods use the same architecture for convolutional layers as DAML for each experiment. DAML with a linear adaptation objective also uses the same fully-connected architecture as DAML with the temporal adaptation objective except that its learned adaptation objective consists of one linear layer. The LSTM uses 512 LSTM hidden units for all experiments. The contextual model uses 3 fully connected layers with size 100 for all experiments.

For placing, we use a behavioral cloning loss as a combination of $\ell_1$ and $\ell_2$ losses, where the $\ell_2$ loss is scaled down by a factor of 100, following prior work (Finn et al., 2017b). For the pushing and pick-and-place experiments, all methods use a mixture density network as mentioned in Section 7.2.5 after the last fully-connected layer with 20 modes and the negative log likelihood of the mixture density network as the behavioral cloning loss. At test time, at each time step, we sample 100 actions from the learned mixture distribution and choose the action with highest probability. For DAML with both a linear and temporal adaptation objective, we use a step size $\alpha = 0.01$ for placing and $\alpha = 0.005$ for pushing and pick-and-place with inner gradient clipping within the range $[-30, 30]$. We use 12, 10, and 4 tasks in the meta batch at each iteration for placing, pushing and pick-and-place respectively. For all methods, we use 1 human demonstration and 1 robot demonstration for each sampled task. We train the model for 50k iterations for placing and placing, and 75k iterations for pick-and-place. We use 5 inner gradient update steps and a bias transformation with dimension 20 for all experiments. Since we don't have the robot state $\mathbf{s}$ for human demonstrations, we set the state input to be 0 when computing inner gradient update and feed the robot states into the policy when we update the policy parameters with robot demonstrations.

### B.3.4 *Diverse Human Demonstration Experiments*

We include the details of the experiments in Section 7.2.7.2 using diverse human demonstrations. For meta-training, eight pushing demonstrations are taken for each of 80 total objects grouped into 40 pairs. Each demo is shot in front of a randomly selected background, among 10 backgrounds. The viewpoint for the human demos is held fixed with a phone camera mounted on a tripod. Before being fed into the model during training images are modified with noise sampled uniformly from the range $[-0.3, 0.3]$ to their lighting. This color augmentation process helps the model perform more robustly in different light conditions. We use the same input image size and policy architecture as the pushing experiment described in Section 7.2.7.1 and Appendix B.3.3.

### B.3.5 *Sawyer Robot Experiments*

We include the details of the experiments in Section 7.2.7.3 on the Sawyer robot. The primary differences between this experiment and previous placing experiments are the robot used and how demonstrations were collected. While PR2 robot demonstrations were taken using a teleoperation interface, the Sawyer arm was controlled kinesthetically by humans: the demonstrator guided the Sawyer arm to perform the goal action. Demonstrations were collected at 10 Hz. Saved at each timestep are a monocular RGB image taken by a Kinect sensor, the robot's joint angles, its joint velocities, and its gripper pose.

The architecture and hyperparameters were tuned by evaluating the gripper pose loss on a held out validation set of 20 objects. The policy architecture takes in RGB images of size $100 \times 100$. It uses the same convolutional and fully-connected layers as well as squared error behavioral cloning loss as in the model for the PR2 placing experiment, and a learned adaptation objective with three layers of 32 $20 \times 1$ convolutional filters followed by one layer of $1 \times 1$ convolutions. We use a step size $\alpha = 0.005$ with inner gradient clipping within the range $[-30, 30]$, 8 as the meta batch size, and 1 human demonstration as well as 1 robot demonstration for each sampled task. We train the model for 60k iterations.

During training, we augmented the images using random color augmentation, by adding noise uniformly sampled in $[-0.3, 0.3]$ to their hue, saturation, and value. The images used during evaluation were not modified. To control the robot during evaluation, the first image frame is used to predict the final end-effector pose of the robot. After the robot reaches the predicted gripper pose, the robot is controlled using the prediction

actions, which are continuous end-effector velocities. At this point the gripper is opened and the robot drops the held item (hopefully) into the target container.

### B.3.6 *Simulated Pushing Experiment*

Here, we include details on the experiments in Section 7.2.7.4. The pushing environment was introduced and open-sourced by Finn et al. (2017b). The expert policy for collecting demonstrations was computed using reinforcement learning. Following (Finn et al., 2017b), we compute the reported success rates over 74 tasks with 6 trials per task, totalling to 444 trials. The time horizon is $T = 100$. A trial is considered successful if the target object lands on the target position for at least 10 timesteps within the 100-timestep episode.

The inputs are RGB images of size $125 \times 125$. The policy architecture uses 3 convolutional layers with 16 $5 \times 5$ filters and stride 2 followed by 1 convolutional layer with 32 $5 \times 5$ filters with stride 1. It also has 2 fully-connected layers of size 400, and a learned adaptation objective with two layers of 64 $10 \times 1$ convolutional filters followed by one layer of $1 \times 1$ convolutions. The policy operates on a $125 \times 125$ RGB image, along with the robot joint angles, joint velocities, and end-effector pose. The behavioral cloning loss is the mean squared error between the predicted actions and the ground truth robot commands. We use step size $\alpha = 0.01$ with inner gradient clipping within the range $[-20, 20]$, and one inner gradient update step. We use 15 as the meta batch size, and 2 different robot demonstrations for each sampled task. We train our policy for 30k iterations.

## B.4 MANDRIL EXPERIMENTAL DETAILS

### B.4.1 *Hyperparameters*

The input to our reward function for all experiments is a resized $80 \times 80$ RGB image, with an output space of $20 \times 20$ in the underlying MDP state space $\mathbf{s}$. In our experiments, we parameterize the reward function for all our reward functions starting from the same base learner. The first layer is a $8 \times 8$ convolution with a stride of 2, 256 filters and symmetric padding of 4. The second layer is a $4 \times 4$ with a stride of 2, 128 filters and symmetric of 1. The third and fourth layer are $3 \times 3$ convolutions with a stride of 1, 64 filters and symmetric padding of 1. The final layer is a $1 \times 1$ convolution.

Our LSTM (Hochreiter and Schmidhuber, 1997) implementation is based on the variant used by Zaremba et al. (2014). The input to the LSTM at each time step is the location

of the agent in the, we separately embed the $(x, y)$-coordinates. This is then used to predict a grid which is then fed as an additional channel in to the base CNN architecture described above. We also experimented with conditioning the initial hidden state on image features from a separate CNN, but found that this did not improve performance.

In our demo conditional model, we preserve the spatial information of the demonstrations by feeding in the state visitation map as a image-grid, upsampled with bilinear interpolation, as an additional channel to the image. In our setup, both the demo-conditional models share the same convolutional architecture, but differ only in how they encode condition on the demonstrations.

For all our methods, we experimented with Adam (D. Kingma and J. Ba, 2015) and RMSProp (G. Hinton et al., 2012). We turned over the learning rate, the inner learning rate $\beta$ of our approach and $\ell_2$ weight decay on the initial parameters. In our LSTM learner, we experimented with different embedding sizes, as well as the dimensionality of the LSTM although we found that these hyperparameters did not significantly impact performance. A negative result we found was that bias transformation (from Section 7.1) in general did not help in our experimental setting.

### B.4.2 *Environment Details*

The sprites in our environment are extracted directly from the StarCraft files. We used in total 100 random units for meta-training. Evaluation on new objects was performed with 5 randomly selected sprites. For computational efficiency, we create a meta-training set of 1000 tasks and cache the optimal policy and state visitations under the true cost. Our set of sprites was divided into two categories: buildings and characters. Each characters had multiple poses (taken from different frames of animation, such as walking/running/flying), whereas buildings only had a single pose. During meta-training the units were randomly placed, but to avoid the possibility that the agent would not need to actively avoid obstacles, the units were placed away from the boundary of the image in both the meta-validation and meta-test set.

The terrain in each environment was randomly generated using a set of tiles, each belonging to a specific category (e.g. grass, dirt, water). For each tile, we also specified a set of possible tiles for each of the 4-neighbors. Using these constraints on the neighbors, we generated random environment terrains using a graph traversal algorithm, where successor tiles were sampled randomly from this set of possible tiles. This process resulted in randomly generated, seamless environments. The names of the units used in our experiments are as follows (names are from the original game files):

The list of buildings used is: academy, assim, barrack, beacon, cerebrat, chemlab, chrysal, cocoon, comsat, control, depot, drydock, egg, extract, factory, fcolony, forge, gateway, genelab, geyser, hatchery, hive, infest, lair, larva, mutapit, nest, nexus, nukesilo, nydustpit, overlord, physics, probe, pylon, prism, pillbox, queen, rcluster, refinery, research, robotic, sbattery, scolony, spire, starbase, stargate, starport, temple, warm, weaponpl, wessel.

The list of characters used is: acritter, arbiter, archives, archon, avenger, battlecr, brood, bugguy, carrier, civilian, defiler, dragoon, drone, dropship, firebat, gencore, ghost, guardian, hydra, intercep, jcritter, lurker, marine, missile, mutacham, mutalid, sapper, scout, scv, shuttle, snakey, spider, stank, tank, templar, trilob, ucereb, uikerr, ultra, vulture, witness, zealot, zergling.

### B.5 MANDRIL DETAILED META-OBJECTIVE DERIVATION

We define the quality of reward function $r_\theta$ parameterized by $\theta \in \mathbb{R}^k$ on task $\mathcal{T}$ with the MaxEnt IRL loss, $\mathcal{L}_{\text{IRL}}^{\mathcal{T}}(\theta)$, described in Section 8.1.2.1. The corresponding gradient is

$$\nabla_\theta \mathcal{L}_{\text{IRL}}(\theta) = \frac{\partial r_\theta}{\partial \theta}(\mathbb{E}_\tau[\mu_\tau] - \mu_{\mathcal{D}_\mathcal{T}}). \tag{44}$$

Here, $\mu_\tau \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|}$ are the state visitations under the trajectory $\tau$, and $\mu_{\mathcal{D}_\mathcal{T}} = \frac{1}{|\mathcal{D}_\mathcal{T}|}\sum_{\tau\in\mathcal{D}_\mathcal{T}} \mu_\tau$ is the mean state visitations over all trajectories in $\mathcal{D}_\mathcal{T}$. Let $\phi_\mathcal{T}$ be the updated parameters after a single gradient step. Then

$$\phi_\mathcal{T} = \theta - \alpha\nabla_\theta \mathcal{L}_\mathcal{T}^{\text{tr}}(\theta). \tag{45}$$

Let $\mathcal{L}_\mathcal{T}^{\text{test}}$ be the MaxEnt IRL loss, where the expectation over trajectories is computed with respect to a test set that is *disjoint* from the set of demonstrations used to compute $\mathcal{L}_\mathcal{T}^{\text{test}}(\theta)$ in Eq. 45. We seek to minimize

$$\sum_{\mathcal{T}\in\mathcal{T}^{\text{tr}}} \mathcal{L}_\mathcal{T}^{\text{test}}(\phi_\mathcal{T}) \tag{46}$$

over the parameters $\theta$. To do so, we first compute the gradient of Eq. 46, which we derive

here. Applying the chain rule

$$\begin{aligned}\nabla_{\theta}\mathcal{L}_{\mathcal{T}}^{\text{test}} &= \frac{\partial\,\mathcal{L}_{\mathcal{T}}^{\text{test}}}{\partial\,r_{\boldsymbol{\phi}_{\mathcal{T}}}}\frac{\partial\,r_{\boldsymbol{\phi}_{\mathcal{T}}}}{\partial\,\boldsymbol{\phi}_{\mathcal{T}}}\frac{\partial\,\boldsymbol{\phi}_{\mathcal{T}}}{\partial\,\theta} \\ &= \frac{\partial\,\mathcal{L}_{\mathcal{T}}^{\text{test}}}{\partial\,r_{\boldsymbol{\phi}_{\mathcal{T}}}}\frac{\partial\,r_{\boldsymbol{\phi}_{\mathcal{T}}}}{\partial\,\boldsymbol{\phi}_{\mathcal{T}}}\frac{\partial}{\partial\,\theta}\left(\theta - \alpha\nabla_{\theta}\mathcal{L}_{\mathcal{T}}^{\text{tr}}(\theta)\right) \\ &= \frac{\partial\,\mathcal{L}_{\mathcal{T}}^{\text{test}}}{\partial\,r_{\boldsymbol{\phi}_{\mathcal{T}}}}\frac{\partial\,r_{\boldsymbol{\phi}_{\mathcal{T}}}}{\partial\,\boldsymbol{\phi}_{\mathcal{T}}}\left(\mathbf{I} - \alpha\frac{\partial}{\partial\,\theta}\left(\frac{\partial\,r_{\theta}}{\partial\,\theta}(\mathbb{E}_{\tau}[\boldsymbol{\mu}_{\tau}] - \boldsymbol{\mu}_{\mathcal{D}_{\mathcal{T}}})\right)\right)\end{aligned}$$

where in the last line we substitute in the gradient of the MaxEnt IRL loss in Eq. 44. Note that the gradient of the last term in parentheses is a matrix, which we will denote $D \in \mathbb{R}^{k\times k}$, that is

$$D := \frac{\partial}{\partial\,\theta}\left(\frac{\partial\,r_{\theta}}{\partial\,\theta}(\mathbb{E}_{\tau}[\boldsymbol{\mu}_{\tau}] - \boldsymbol{\mu}_{\mathcal{D}_{\mathcal{T}}})\right), \tag{47}$$

with entries

$$\begin{aligned}D_{ij} &= \frac{\partial}{\partial\,\theta_i}\left(\sum_{l=1}^{|\mathcal{S}||\mathcal{A}|}\frac{\partial\,r_{\theta,l}}{\partial\,\theta_j}\left(\mathbb{E}_{\tau}[\boldsymbol{\mu}_{\tau}] - \boldsymbol{\mu}_{\mathcal{D}_{\mathcal{T}}}\right)_l\right) \\ &= \sum_{l=1}^{|\mathcal{S}||\mathcal{A}|}\frac{\partial^2\,r_{\theta,l}}{\partial\,\theta_i\partial\,\theta_j}\left(\mathbb{E}_{\tau}[\boldsymbol{\mu}_{\tau}] - \boldsymbol{\mu}_{\mathcal{D}_{\mathcal{T}}}\right)_l + \frac{\partial\,r_{\theta,l}}{\partial\,\theta_j}\frac{\partial\,r_{\theta,l}}{\partial\,\theta_i}\frac{\partial}{\partial\,r_{\theta,l}}(\mathbb{E}_{\tau}[\boldsymbol{\mu}_{\tau}])_l.\end{aligned}$$

Therefore, we can express D as

$$D = \sum_{l=1}^{|\mathcal{S}||\mathcal{A}|}\frac{\partial^2\,r_{\theta,l}}{\partial\,\theta^2}\left(\mathbb{E}_{\tau}[\boldsymbol{\mu}_{\tau}] - \boldsymbol{\mu}_{\mathcal{D}_{\mathcal{T}}}\right)_l + \frac{\partial\,r_{\theta,l}}{\partial\,\theta}\frac{\partial\,r_{\theta,l}}{\partial\,\theta}^{\top}\frac{\partial}{\partial\,r_{\theta,l}}(\mathbb{E}_{\tau}[\boldsymbol{\mu}_{\tau}])_l. \tag{48}$$

This can equivalently be expressed using tensor-vector products

$$D = \frac{\partial^2\,r_{\theta}}{\partial\,\theta^2}(\mathbb{E}_{\tau}[\boldsymbol{\mu}_{\tau}] - \boldsymbol{\mu}_{\mathcal{D}_{\mathcal{T}}}) + \frac{\partial\,r_{\theta}}{\partial\,\theta}\frac{\partial\,r_{\theta}}{\partial\,\theta}^{\top}\frac{\partial}{\partial\,r_{\theta}}\mathbb{E}_{\tau}[\boldsymbol{\mu}_{\tau}]. \tag{49}$$

This is equivalent to Eq. 28 in Section 8.1.2.1.

In order to compute D, we must take the gradient of the term $\mathbb{E}_{\tau}[\boldsymbol{\mu}_{\tau}]$. This can be done

by expanding the expectation as follows

$$\begin{aligned}
\frac{\partial}{\partial r_\theta}\mathbb{E}_\tau[\mu_\tau] &= \frac{\partial}{\partial r_\theta}\sum_\tau \left(\frac{\exp(\mu_\tau^\top r_\theta)}{\sum_{\tau'}\exp(\mu_{\tau'}^\top r_\theta)}\right)\mu_\tau \\
&= \sum_\tau \left(\left(\frac{\exp(\mu_\tau^\top r_\theta)}{\sum_{\tau'}\exp(\mu_{\tau'}^\top r_\theta)}\right)(\mu_\tau\mu_\tau^\top) - \frac{\exp(\mu_\tau^\top r_\theta)}{(\sum_{\tau'}\exp(\mu_{\tau'}^\top r_\theta))^2}\sum_{\tau'}(\mu_{\tau'}\mu_\tau^\top)\exp(\mu_{\tau'}^\top r_\theta)\right) \\
&= \sum_\tau P(\tau \mid r_\theta)(\mu_\tau\mu_\tau^\top) - \sum_\tau P(\tau|r_\theta)\sum_{\tau'} P(\tau' \mid r_\theta)(\mu_{\tau'}\mu_\tau^\top) \\
&= \mathbb{E}_\tau\left[(\mu_\tau\mu_\tau^\top) - \sum_{\tau'} P(\tau' \mid r_\theta)(\mu_{\tau'}\mu_\tau^\top)\right] \\
&= \mathbb{E}_\tau[\mu_\tau\mu_\tau^\top] - \mathbb{E}_{\tau',\tau}[\mu_{\tau'}\mu_\tau^\top] \\
&= \mathbb{E}_\tau[\mu_\tau\mu_\tau^\top] - \mathbb{E}_\tau[\mu_\tau](\mathbb{E}_\tau[\mu_\tau])^\top \\
&= \mathrm{Cov}[\mu_\tau].
\end{aligned}$$

## B.6 FLO EXPERIMENTAL DETAILS

### B.6.1 *Model Architecture*

Our model $f_{\mathrm{CAML}}$ is represented by a convolutional neural network with RGB image inputs. The network consists of three convolutional layers with 32 3×3 filters and stride 2, each followed by layer normalization and a ReLU non-linearity. A spatial soft-argmax operation extracts spatial feature points from the final convolution layer. These features are passed through one fully-connected layers with 50 units and ReLU non-linearities, followed by a linear layer to the two-dimensional softmax output. The architecture in our simulated experiments is the same, except with 16 filters in each convolution layer, feature flattening instead of a spatial soft-argmax, and three fully-connected layers instead of one. When using real images, the first convolutional layer is initialized with weights from VGG-16.

### B.6.2 *Autoencoder Comparison Details*

For the autoencoder, we use a convolutional neural network with three layers and 3x3 filters, where the layers have 64, 32, and 16 filters respectively. The stride is 2 for the first layer, and 1 for subsequent layers. We follow this up with three fully connected lay-

ers, that have 200, 100 and 50 units respectively. We train this autoencoder on the entire meta-training dataset, and our target reconstructions have dimension 32x32x3. We run reinforcement learning on top of the features from the last hidden layer of this autoencoder, and keep the autoencoder weights fixed during the policy learning process.

# BIBLIOGRAPHY


Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. (2016). "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." In: *arXiv preprint arXiv:1603.04467* (cit. on pp. 18, 21, 33).

Abbeel, P. and A. Y. Ng (2004). "Apprenticeship learning via inverse reinforcement learning." In: *International Conference on Machine Learning (ICML)*. ACM, p. 1 (cit. on pp. 112, 117, 123).

Akgun, B., M. Cakmak, J. W. Yoo, and A. L. Thomaz (2012). "Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective." In: *International Conference on Human-Robot Interaction* (cit. on pp. 73, 95).

Andrychowicz, M., M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas (2016). "Learning to learn by gradient descent by gradient descent." In: *Neural Information Processing Systems (NIPS)* (cit. on pp. 3, 11, 15, 31).

Argall, B. D., S. Chernova, M. Veloso, and B. Browning (2009). "A survey of robot learning from demonstration." In: *Robotics and Autonomous Systems* (cit. on p. 95).

Åström, K. J. and B. Wittenmark (2013). *Adaptive control*. Courier Corporation (cit. on p. 60).

Aswani, A., P. Bouffard, and C. Tomlin (2012). "Extensions of learning-based model predictive control for real-time application to a quadrotor helicopter." In: *American Control Conference (ACC), 2012* (cit. on p. 67).

Aytar, Y. and A. Zisserman (2011). "Tabula rasa: Model transfer for object category detection." In: *2011 International Conference on Computer Vision*. IEEE, pp. 2252–2259 (cit. on p. 96).

Ba, J. L., J. R. Kiros, and G. E. Hinton (2016). "Layer normalization." In: *arXiv preprint arXiv:1607.06450* (cit. on p. 78).

Ba, J., G. E. Hinton, V. Mnih, J. Z. Leibo, and C. Ionescu (2016). "Using Fast Weights to Attend to the Recent Past." In: *Neural Information Processing Systems (NIPS)* (cit. on pp. 3, 31).

Babeş-Vroman, M., V. Marivate, K. Subramanian, and M. Littman (2011). "Apprenticeship Learning About Multiple Intentions." In: *International Conference on Machine Learning (ICML)*. USA (cit. on p. 113).

Baker, C., J. B Tenenbaum, and R. R Saxe (2007). "Goal inference as inverse planning." In: (cit. on p. 106).

Barber, D. and C. M. Bishop (1998). "Ensemble learning for multi-layer networks." In: *neural information processing systems (NIPS)* (cit. on p. 52).

Barrett, S., M. E. Taylor, and P. Stone (2010). "Transfer learning for reinforcement learning on a physical robot." In: *Ninth International Conference on Autonomous Agents and Multiagent Systems-Adaptive Learning Agents Workshop (AAMAS-ALA)* (cit. on p. 79).

Bengio, S., Y. Bengio, J. Cloutier, and J. Gecsei (1992). "On the optimization of a synaptic learning rule." In: *Optimality in Artificial and Biological Neural Networks*, pp. 6–8 (cit. on pp. 2, 11, 15, 31).

Bengio, Y., S. Bengio, and J. Cloutier (1990). *Learning a synaptic learning rule*. Université de Montréal, Département d'informatique et de recherche opérationnelle (cit. on p. 31).

Billard, A., Y. Epars, S. Calinon, S. Schaal, and G. Cheng (2004). "Discovering optimal imitation strategies." In: *Robotics and autonomous systems* (cit. on p. 79).

Bishop, C. M. (1994). "Mixture density networks." In: (cit. on pp. 94, 97).

Blossom, P. (2006). *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning* (cit. on p. 64).

Blundell, C., J. Cornebise, K. Kavukcuoglu, and D. Wierstra (2015). "Weight uncertainty in neural networks." In: *arXiv preprint arXiv:1505.05424* (cit. on p. 52).

Bojarski, M., D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba (2016). "End to end learning for self-driving cars." In: *arXiv preprint arXiv:1604.07316* (cit. on pp. 77, 79).

Boularias, A., J. Kober, and J. Peters (2011). "Relative entropy inverse reinforcement learning." In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics* (cit. on p. 124).

Bousmalis, K., N. Silberman, D. Dohan, D. Erhan, and D. Krishnan (2017). "Unsupervised pixel-level domain adaptation with generative adversarial networks." In: *International Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 96).

Brass, M. and C. Heyes (2005). "Imitation: is cognitive neuroscience solving the correspondence problem?" In: *Trends in cognitive sciences* (cit. on pp. 73, 86).

Braun, D. A., A. Aertsen, D. M. Wolpert, and C. Mehring (2009). "Learning optimal adaptation strategies in unpredictable motor tasks." In: *Journal of Neuroscience* (cit. on p. 59).

Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba (2016). "OpenAI gym." In: *arXiv preprint arXiv:1606.01540* (cit. on p. 82).

Buchan, A. D., D. W. Haldane, and R. S. Fearing (2013). "Automatic identification of dynamic piecewise affine models for a running robot." In: *International Conference on Intelligent Robots and Systems (IROS)*, pp. 5600–5607 (cit. on p. 67).

Calinon, S. and A. Billard (2006). "Teaching a humanoid robot to recognize and reproduce social cues." In: *International Symposium on Robot and Human Interactive Communication (ROMAN)* (cit. on p. 95).

Calinon, S., P. Evrard, E. Gribovskaya, A. Billard, and A. Kheddar (2009). "Learning collaborative manipulation tasks by demonstration using a haptic interface." In: *International Conference on Advanced Robotics (ICAR)* (cit. on pp. 73, 95).

Caruana, R. (1993). "Multitask Learning: A Knowledge-Based Source of Inductive Bias." In: *International Conference on Machine Learning (ICML)* (cit. on p. 2).

Choi, J. and K.-E. Kim (2012). "Nonparametric Bayesian Inverse Reinforcement Learning for Multiple Reward Functions." In: *Neural Information Processing Systems (NIPS)*. NIPS. USA (cit. on p. 113).

Christiano, P. F., J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei (2017). "Deep reinforcement learning from human preferences." In: *Neural Information Processing Systems (NIPS)* (cit. on p. 123).

Clavera, I., A. Nagabandi, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn (2018). "Learning to Adapt: Meta-Learning for Model-Based Control." In: *arXiv preprint arXiv:1803.11347* (cit. on p. 4).

Collobert, R., J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa (2011). "Natural language processing (almost) from scratch." In: *Journal of Machine Learning Research* (cit. on p. 1).

Da Silva, B. C., G. Konidaris, and A. G. Barto (2012). "Learning parameterized skills." In: *International Conference of Machine Learning (ICML)* (cit. on p. 79).

Dai, A. M. and Q. V. Le (2015). "Semi-supervised sequence learning." In: *Neural Information Processing Systems (NIPS)* (cit. on p. 2).

Daumé III, H. (2009). "Bayesian multitask learning with latent hierarchies." In: *Conference on Uncertainty in Artificial Intelligence (UAI)* (cit. on pp. 3, 51).

Deguchi, K. and I. Takahashi (1999). "Image-based simultaneous control of robot and target object motions by direct-image-interpretation method." In: *International Conference on Intelligent Robots and Systems (IROS)* (cit. on pp. 117, 123).

Deisenroth, M. P., P. Englert, J. Peters, and D. Fox (2014). "Multi-task policy search for robotics." In: *International Conference on Robotics and Automation (ICRA)* (cit. on pp. 73, 79).

Deisenroth, M. P., G. Neumann, J. Peters, et al. (2013). "A survey on policy search for robotics." In: *Foundations and Trends in Robotics* (cit. on pp. 60, 66).

Deisenroth, M. and C. E. Rasmussen (2011). "PILCO: A model-based and data-efficient approach to policy search." In: *International Conference on machine learning (ICML)* (cit. on p. 66).

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009). "ImageNet: A Large-Scale Hierarchical Image Database." In: *CVPR*, pp. 248–255 (cit. on p. 2).

Dillmann, R. (2004). "Teaching and learning of robot tasks via observation of human performance." In: *Robotics and Autonomous Systems* (cit. on p. 95).

Dimitrakakis, C. and C. A. Rothkopf (2012). "Bayesian Multitask Inverse Reinforcement Learning." In: *European Conference on Recent Advances in Reinforcement Learning*. EWRL (cit. on p. 113).

Donahue, J., Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell (2014). "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition." In: *International Conference on Machine Learning (ICML)* (cit. on pp. 2, 32).

Duan, Y., M. Andrychowicz, B. Stadie, J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba (2017). "One-shot Imitation Learning." In: *Neural Information Processing Systems (NIPS)* (cit. on pp. 73, 79–81, 84, 97, 103, 114).

Duan, Y., X. Chen, R. Houthooft, J. Schulman, and P. Abbeel (2016a). "Benchmarking deep reinforcement learning for continuous control." In: *International Conference on Machine Learning (ICML)* (cit. on pp. 37, 38).

Duan, Y., J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel (2016b). "$RL^2$: Fast Reinforcement Learning via Slow Reinforcement Learning." In: *arXiv preprint arXiv:1611.02779* (cit. on pp. 3, 10, 16, 31, 36, 51, 60, 65, 132).

Duvenaud, D., D. Maclaurin, and R. Adams (2016). "Early stopping as nonparametric variational inference." In: *Artificial Intelligence and Statistics*, pp. 1070–1077 (cit. on p. 112).

Ebert, F., C. Finn, A. X. Lee, and S. Levine (2017). "Self-Supervised Visual Planning with Temporal Skip Connections." In: *Conference on Robot Learning (CoRL)* (cit. on p. 125).

Edwards, A. D., S. Sood, and C. L. Isbell Jr (2017). "Cross-domain perceptual reward functions." In: *arXiv:1705.09045* (cit. on p. 123).

Edwards, A., C. Isbell, and A. Takanishi (2016). "Perceptual reward functions." In: *IJCAI Workshop on Deep Reinforcement Learning: Frontiers and Challenges* (cit. on p. 123).

Edwards, H. and A. Storkey (2017). "Towards a Neural Statistician." In: *International Conference on Learning Representations (ICLR)* (cit. on pp. 3, 31, 36, 51).

Ekvall, S. and D. Kragic (2004). "Interactive grasp learning based on human demonstration." In: *International Conference on Robotics and Automation (ICRA)* (cit. on p. 95).

Fei-Fei, L. et al. (2003). "A Bayesian approach to unsupervised one-shot learning of object categories." In: *Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on pp. 3, 13, 51).

Fernando, B., A. Habrard, M. Sebban, and T. Tuytelaars (2013). "Unsupervised visual domain adaptation using subspace alignment." In: *International Conference on Computer Vision (ICCV)* (cit. on p. 96).

Finn, C., P. Abbeel, and S. Levine (2017a). "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks." In: *International Conference on Machine Learning (ICML)* (cit. on pp. 4, 49, 51, 65, 77, 121, 154).

Finn, C. and S. Levine (2017). "Deep visual foresight for planning robot motion." In: *International Conference on Robotics and Automation (ICRA)* (cit. on p. 66).

Finn, C. and S. Levine (2018). "Meta-Learning and Universality: Deep Representations and Gradient Descent can Approximate any Learning Algorithm." In: *International Conference on Learning Representations (ICLR)* (cit. on pp. 4, 66).

Finn, C., S. Levine, and P. Abbeel (2016a). "Guided cost learning: Deep inverse optimal control via policy optimization." In: *International Conference on Machine Learning (ICML)* (cit. on pp. 79, 112, 117, 124).

Finn, C., X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel (2016b). "Deep spatial autoencoders for visuomotor learning." In: *International Conference on Robotics and Automation (ICRA)* (cit. on pp. 78, 94, 117, 123, 124, 126, 129).

Finn, C., K. Xu, and S. Levine (2018). "Probabilistic Model-Agnostic Meta-Learning." In: *arXiv preprint arXiv:1806.02817* (cit. on p. 4).

Finn, C., T. Yu, T. Zhang, P. Abbeel, and S. Levine (2017b). "One-Shot Visual Imitation Learning via Meta-Learning." In: *Conference on Robot Learning (CoRL)* (cit. on pp. 4, 91, 101, 103, 153, 164, 166).

Fortunato, M., C. Blundell, and O. Vinyals (2017). "Bayesian recurrent neural networks." In: *arXiv preprint arXiv:1704.02798* (cit. on pp. 48, 67).

Fu, J., S. Levine, and P. Abbeel (2016). "One-shot learning of manipulation skills with online dynamics adaptation and neural network priors." In: *International Conference on Intelligent Robots and Systems (IROS)* (cit. on p. 67).

Fu, J., K. Luo, and S. Levine (2018). "Learning Robust Rewards with Adverserial Inverse Reinforcement Learning." In: *International Conference on Learning Representations (ICLR)* (cit. on p. 112).

Gao, J., W. Fan, J. Jiang, and J. Han (2008). "Knowledge transfer via multiple model local structure mapping." In: *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*. ACM (cit. on pp. 3, 51).

Garcia, V. and J. Bruna (2017). "Few-Shot Learning with Graph Neural Networks." In: *arXiv preprint arXiv:1711.04043* (cit. on p. 154).

Giusti, A., J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodrıguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, et al. (2016). "A machine learning approach to visual perception of forest trails for mobile robots." In: *IEEE Robotics and Automation Letters (RA-L)* (cit. on p. 79).

Gong, B., K. Grauman, and F. Sha (2013). "Connecting the dots with landmarks: Discriminatively learning domain-invariant features for unsupervised domain adaptation." In: *International Conference on Machine Learning (ICML)* (cit. on p. 96).

Goodfellow, I. J., J. Shlens, and C. Szegedy (2015). "Explaining and harnessing adversarial examples." In: *International Conference on Learning Representations (ICLR)* (cit. on p. 37).

Grant, E., C. Finn, S. Levine, T. Darrell, and T. Griffiths (2018). "Recasting Gradient-Based Meta-Learning as Hierarchical Bayes." In: *International Conference on Learning Representations (ICLR)* (cit. on pp. 46, 48, 49, 51, 91, 92, 111, 112, 154).

Grant, E., C. Finn, J. Peterson, J. Abbott, S. Levine, T. Griffiths, and T. Darrell (2017). "Concept acquisition via meta-learning: Few-shot learning from positive examples." In: *NIPS Workshop on Cognitively-Informed Artificial Intelligence* (cit. on pp. 54, 122).

Graves, A. (2011). "Practical variational inference for neural networks." In: *Neural Information Processing Systems (NIPS)* (cit. on p. 52).

Gu, S., T. Lillicrap, I. Sutskever, and S. Levine (2016). "Continuous Deep Q-Learning with Model-based Acceleration." In: *International Conference on Machine Learning (ICML)* (cit. on p. 67).

Gupta, A., C. Devin, Y. Liu, P. Abbeel, and S. Levine (2017). "Learning invariant feature spaces to transfer skills with reinforcement learning." In: *International Conference on Learning Representations (ICLR)* (cit. on p. 79).

Gupta, A., B. Eysenbach, C. Finn, and S. Levine (2018). "Unsupervised Meta-Learning for Reinforcement Learning." In: *arXiv preprint arXiv:1806.04640* (cit. on p. 132).

Ha, D., A. Dai, and Q. V. Le (2017). "HyperNetworks." In: *International Conference on Learning Representations (ICLR)* (cit. on pp. 3, 11, 31).

Hannun, A., C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, et al. (2014). "Deep speech: Scaling up end-to-end speech recognition." In: *arXiv preprint arXiv:1412.5567* (cit. on p. 1).

Hausman, K., Y. Chebotar, S. Schaal, G. Sukhatme, and J. J. Lim (2017). "Multi-modal imitation learning from unstructured demonstrations using generative adversarial nets." In: *Neural Information Processing Systems (NIPS)* (cit. on p. 113).

Hinton, G. E. and D. Van Camp (1993). "Keeping the neural networks simple by minimizing the description length of the weights." In: *Conference on Computational learning theory* (cit. on p. 52).

Hinton, G., N. Srivastava, and K. Swersky (2012). *Neural Networks for Machine Learning, Lecture 6a: Overview of Mini-Batch Gradient Descent* (cit. on p. 167).

Ho, J. and S. Ermon (2016). "Generative adversarial imitation learning." In: *Neural Information Processing Systems (NIPS)* (cit. on pp. 113, 117, 123).

Hochreiter, S. and J. Schmidhuber (1997). "Long Short-Term Memory." In: *Neural Computation* (cit. on pp. 65, 166).

Hochreiter, S., A. Younger, and P. Conwell (2001). "Learning to learn using gradient descent." In: *International Conference on Artificial Neural Networks (ICANN)* (cit. on pp. 3, 11, 31, 51).

Hoffman, M. D., D. M. Blei, C. Wang, and J. Paisley (2013). "Stochastic variational inference." In: *The Journal of Machine Learning Research* (cit. on p. 52).

Hornik, K., M. Stinchcombe, and H. White (1990). "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks." In: *Neural networks* (cit. on pp. 27, 139, 142).

Howard, J. and S. Ruder (2018). "Universal Language Model Fine-tuning for Text Classification." In: *Annual Meeting of the Association for Computational Linguistics (ACL)* (cit. on p. 2).

Husken, M. and C. Goerick (2000). "Fast learning for problem classes using knowledge based network initialization." In: *International Joint Conference on Neural Networks (IJCNN)* (cit. on p. 32).

Ioffe, S. and C. Szegedy (2015). "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In: *International Conference on Machine Learning (ICML)* (cit. on p. 35).

Jagersand, M. and R. Nelson (1995). "Visual space task specification, planning and control." In: *International Symposium on Computer Vision* (cit. on pp. 117, 123).

Johnson, M., D. K. Duvenaud, A. Wiltschko, R. P. Adams, and S. R. Datta (2016). "Composing graphical models with neural networks for structured representations and fast inference." In: *Neural Information Processing Systems (NIPS)* (cit. on pp. 47, 52).

Kaiser, L., O. Nachum, A. Roy, and S. Bengio (2017). "Learning to Remember Rare Events." In: *International Conference on Learning Representations (ICLR)* (cit. on p. 36).

Kalakrishnan, M., P. Pastor, L. Righetti, and S. Schaal (2013). "Learning objective functions for manipulation." In: *International Conference on Robotics and Automation (ICRA)*. IEEE (cit. on p. 124).
Kim, D.-K. and M. R. Walter (2017). "Satellite image-based localization via learned embeddings." In: *International Conference on Robotics and Automation (ICRA)* (cit. on p. 86).
Kingma, D. P. and M. Welling (2013). "Auto-encoding variational bayes." In: *arXiv preprint arXiv:1312.6114* (cit. on pp. 47, 50, 52).
Kingma, D. and J. Ba (2015). "Adam: A method for stochastic optimization." In: *International Conference on Learning Representations (ICLR)* (cit. on pp. 33, 80, 97, 122, 167).
Kirkpatrick, J., R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. (2016). "Overcoming catastrophic forgetting in neural networks." In: *arXiv preprint arXiv:1612.00796* (cit. on p. 32).
Kjellstrom, H., J. Romero, and D. Kragic (2008). "Visual recognition of grasps for human-to-robot mapping." In: *International Conference on Intelligent Robots and Systems (IROS)* (cit. on p. 95).
Ko, J. and D. Fox (2009). "GP-BayesFilters: Bayesian filtering using Gaussian process prediction and observation models." In: *Autonomous Robots* 27.1, pp. 75–90 (cit. on p. 66).
Kober, J., A. Wilhelm, E. Oztop, and J. Peters (2012). "Reinforcement learning to adjust parametrized motor primitives to new situations." In: *Autonomous Robots* (cit. on p. 79).
Koch, G., R. Zemel, and R. Salakhutdinov (2015). "Siamese neural networks for one-shot image recognition." In: *ICML Deep Learning Workshop* (cit. on pp. 3, 15, 16, 31, 36).
Kormushev, P., S. Calinon, and D. G. Caldwell (2010). "Robot motor skill coordination with EM-based reinforcement learning." In: *International Conference on Intelligent Robots and Systems (IROS)* (cit. on p. 123).
Krähenbühl, P., C. Doersch, J. Donahue, and T. Darrell (2016). "Data-dependent initializations of convolutional neural networks." In: *International Conference on Learning Representations (ICLR)* (cit. on p. 32).
Krause, B., E. Kahembwe, I. Murray, and S. Renals (2017). "Dynamic Evaluation of Neural Sequence Models." In: *CoRR* abs/1709.07432. arXiv: 1709.07432 (cit. on pp. 67, 69).
Krause, B., L. Lu, I. Murray, and S. Renals (2016). "Multiplicative LSTM for sequence modelling." In: *arXiv preprint arXiv:1609.07959* (cit. on p. 67).
Krizhevsky, A. and G. Hinton (2009). *Learning multiple layers of features from tiny images*. Tech. rep. (cit. on p. 2).

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). "Imagenet classification with deep convolutional neural networks." In: *Neural Information Processing Systems (NIPS)* (cit. on p. 1).

Kruger, V., D. L. Herzog, S. Baby, A. Ude, and D. Kragic (2010). "Learning actions from observations." In: *IEEE Robotics & Automation Magazine* (cit. on p. 95).

Kuefler, A. and M. J. Kochenderfer (2018). "Burn-In Demonstrations for Multi-Modal Imitation Learning." In: *aamas* (cit. on p. 113).

Kupcsik, A. G., M. P. Deisenroth, J. Peters, G. Neumann, et al. (2013). "Data-efficient generalization of robot skills with contextual policy search." In: *AAAI Conference on Artificial Intelligence* (cit. on pp. 73, 79).

Lacoste, A., T. Boquet, N. Rostamzadeh, B. Oreshki, W. Chung, and D. Krueger (2017). "Deep Prior." In: *arXiv preprint arXiv:1712.05016* (cit. on p. 51).

Lake, B. M., R. Salakhutdinov, J. Gross, and J. B. Tenenbaum (2011). "One shot learning of simple visual concepts." In: *Conference of the Cognitive Science Society (CogSci)* (cit. on pp. 3, 35, 36, 40).

Lake, B. M., R. Salakhutdinov, and J. B. Tenenbaum (2015). "Human-level concept learning through probabilistic program induction." In: *Science* (cit. on p. 51).

Lange, S., M. Riedmiller, and A. Voigtlander (2012). "Autonomous reinforcement learning on raw visual input data in a real world application." In: *International Joint Conference on Neural Networks (IJCNN)* (cit. on pp. 124, 129).

Laskey, M., S. Staszak, W. Y.-S. Hsieh, J. Mahler, F. T. Pokorny, A. D. Dragan, and K. Goldberg (2016). "Shiv: Reducing supervisor burden in dagger using support vectors for efficient learning from demonstrations in high dimensional state spaces." In: *International Conference on Robotics and Automation (ICRA)* (cit. on p. 79).

Lawrence, N. D. and J. C. Platt (2004). "Learning to learn with the informative vector machine." In: *International Conference on Machine Learning (ICML)*, p. 65 (cit. on pp. 3, 51).

LeCun, Y., C. Cortes, and C. Burges (1998). *The MNIST database of handwritten digits* (cit. on p. 2).

Lee, J. and M. S. Ryoo (2017). "Learning Robot Activities from First-Person Human Videos Using Convolutional Future Regression." In: *arXiv:1703.01040* (cit. on p. 95).

Lee, K., Y. Su, T.-K. Kim, and Y. Demiris (2013). "A syntactic approach to robot imitation learning using probabilistic activity grammars." In: *Robotics and Autonomous Systems* (cit. on p. 95).

Lenz, I., R. A. Knepper, and A. Saxena (2015). "DeepMPC: Learning Deep Latent Features for Model Predictive Control." In: *Robotics: Science and Systems* (cit. on p. 66).

Levine, S. and P. Abbeel (2014). "Learning neural network policies with guided policy search under unknown dynamics." In: *Neural Information Processing Systems (NIPS)*, pp. 1071–1079 (cit. on p. 67).

Levine, S., C. Finn, T. Darrell, and P. Abbeel (2016a). "End-to-end training of deep visuomotor policies." In: *Journal of Machine Learning Research (JMLR)* (cit. on pp. 1, 66, 78, 84, 93, 94, 123).

Levine, S. and V. Koltun (2013). "Guided policy search." In: *International Conference on Machine Learning (ICML)* (cit. on p. 67).

Levine, S., P. Pastor, A. Krizhevsky, and D. Quillen (2016b). "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection." In: *arXiv preprint arXiv:1603.02199* (cit. on pp. 117, 123).

Levine, S., Z. Popovic, and V. Koltun (2011). "Nonlinear inverse reinforcement learning with gaussian processes." In: *Neural Information Processing Systems (NIPS)*, pp. 19–27 (cit. on p. 116).

Li, D., Y. Yang, Y.-Z. Song, and T. M. Hospedales (2018). "Learning to Generalize: Meta-Learning for Domain Generalization." In: *AAAI Conference on Artificial Intelligence (AAAI)* (cit. on p. 96).

Li, K. and J. Malik (2017a). "Learning to Optimize." In: *International Conference on Learning Representations (ICLR)* (cit. on p. 31).

Li, K. and J. Malik (2017b). "Learning to Optimize Neural Nets." In: *arXiv preprint arXiv:1703.00441* (cit. on p. 11).

Li, K. and J. W. Burdick (2017). "Meta Inverse Reinforcement Learning via Maximum Reward Sharing for Human Motion Analysis." In: *CoRR* abs/1710.03592 (cit. on p. 113).

Li, W. and E. Todorov (2004). "Iterative linear quadratic regulator design for nonlinear biological movement systems." In: *ICINCO (1)* (cit. on p. 64).

Li, Y., J. Song, and S. Ermon (2017). "Inferring the latent structure of human decision-making from raw visual inputs." In: *arXiv preprint arXiv:1703.08840* (cit. on p. 113).

Li, Z., F. Zhou, F. Chen, and H. Li (2017). "Meta-SGD: Learning to Learn Quickly for Few Shot Learning." In: *arXiv preprint arXiv:1707.09835* (cit. on pp. 3, 154).

Lillicrap, T. P., J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra (2015). "Continuous control with deep reinforcement learning." In: *CoRR* abs/1509.02971. arXiv: 1509.02971 (cit. on p. 66).

Liu, Y., A. Gupta, P. Abbeel, and S. Levine (2018). "Imitation from observation: Learning to imitate behaviors from raw video via context translation." In: *International Conference on Robotics and Automation (ICRA)* (cit. on p. 96).

MacKay, D. J. (1992). "A practical Bayesian framework for backpropagation networks." In: *Neural computation* (cit. on p. 52).
Maclaurin, D., D. Duvenaud, and R. Adams (2015). "Gradient-based hyperparameter optimization through reversible learning." In: *International Conference on Machine Learning (ICML)* (cit. on p. 32).
Metz, L., N. Maheswaranathan, B. Cheung, and J. Sohl-Dickstein (2018). "Learning Unsupervised Learning Rules." In: *arXiv preprint arXiv:1804.00222* (cit. on p. 8).
Mishra, N., M. Rohaninejad, X. Chen, and P. Abbeel (2018). "A Simple Neural Attentive Meta-Learner." In: *International Conference on Learning Representations (ICLR)* (cit. on pp. 10, 41, 51, 154).
Mnih, V., K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller (2013). "Playing atari with deep reinforcement learning." In: *arXiv preprint arXiv:1312.5602* (cit. on p. 1).
Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. (2015). "Human-level control through deep reinforcement learning." In: *Nature* (cit. on p. 66).
Muelling, K., A. Venkatraman, J.-S. Valois, J. Downey, J. Weiss, S. Javdani, M. Hebert, A. B. Schwartz, J. L. Collinger, and J. A. Bagnell (2017). "Autonomy infused teleoperation with application to BCI manipulation." In: *Autonomous Robots* (cit. on p. 96).
Mülling, K., J. Kober, O. Kroemer, and J. Peters (2013). "Learning to select and generalize striking movements in robot table tennis." In: *The International Journal of Robotics Research (IJRR)* (cit. on pp. 73, 79).
Munkhdalai, T. and H. Yu (2017). "Meta Networks." In: *International Conference on Machine Learning (ICML)* (cit. on pp. 3, 31, 41).
Nagabandi, A., G. Kahn, R. S. Fearing, and S. Levine (2017a). "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning." In: *CoRR* abs/1708.02596 (cit. on pp. 64, 69).
Nagabandi, A., G. Yang, T. Asmar, R. Pandya, G. Kahn, S. Levine, and R. S. Fearing (2017b). "Learning Image-Conditioned Dynamics Models for Control of Underactuated Legged Millirobots." In: *arXiv preprint arXiv:1711.05253* (cit. on p. 66).
Naik, D. K. and R. Mammone (1992). "Meta-neural networks that learn by learning." In: *International Joint Conference on Neural Netowrks (IJCNN)* (cit. on pp. 2, 31).
Nair, A., P. Agarwal, D. Chen, P. Isola, P. Abbeel, and S. Levine (2017). "Combining Self-Supervised Learning and Imitation for Vision-Based Rope Manipulation." In: *International Conference on Robotics and Automation (ICRA)* (cit. on p. 79).

Neal, R. M. (1995). *Bayesian learning for neural networks*. PhD thesis, University of Toronto (cit. on p. 52).

Nehaniv, C. L., K. Dautenhahn, et al. (2002). "The correspondence problem." In: *Imitation in animals and artifacts* (cit. on p. 86).

Ng, A. Y., D. Harada, and S. Russell (1999). "Policy invariance under reward transformations: Theory and application to reward shaping." In: *ICML*. Vol. 99, pp. 278–287 (cit. on p. 105).

Ng, A. Y. and S. J. Russell (2000). "Algorithms for inverse reinforcement learning." In: *International Conference on Machine Leanring (ICML)* (cit. on pp. 79, 105, 112, 123).

Nguyen, A., D. Kanoulas, L. Muratore, D. G. Caldwell, and N. G. Tsagarakis (2017). "Translating Videos to Commands for Robotic Manipulation with Deep Recurrent Neural Networks." In: *arXiv:1710.00290* (cit. on p. 95).

Nichol, A. and J. Schulman (2018). "Reptile: a Scalable Metalearning Algorithm." In: *arXiv preprint arXiv:1803.02999* (cit. on p. 154).

Parisotto, E., J. L. Ba, and R. Salakhutdinov (2016). "Actor-mimic: Deep multitask and transfer reinforcement learning." In: *International Conference on Learning Representations (ICLR)* (cit. on p. 39).

Pastor, P., H. Hoffmann, T. Asfour, and S. Schaal (2009). "Learning and generalization of motor skills by learning from demonstration." In: *International Conference on Robotics and Automation (ICRA)* (cit. on pp. 73, 79).

Pastor, P., L. Righetti, M. Kalakrishnan, and S. Schaal (2011). "Online movement adaptation based on previous sensor experiences." In: *International Conference on Intelligent Robots and Systems (IROS)* (cit. on pp. 73, 95).

Paszke, A., S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer (2017). "Automatic differentiation in pytorch." In: (cit. on p. 21).

Patel, V. M., R. Gopalan, R. Li, and R. Chellappa (2015). "Visual domain adaptation: A survey of recent advances." In: *IEEE signal processing magazine* (cit. on p. 96).

Peters, J. and S. Schaal (2008). "Reinforcement learning of motor skills with policy gradients." In: *Neural networks* 21.4, pp. 682–697 (cit. on p. 66).

Pinto, L. and A. Gupta (2016). "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours." In: *International Conference on Robotics and Automation (ICRA)* (cit. on pp. 117, 123).

Pomerleau, D. (1989). "ALVINN: An Autonomous Land Vehicle In a Neural Network." In: *Neural Information Processing Systems (NIPS)* (cit. on pp. 73, 79).

Radford, A., K. Narasimhan, T. Salimans, and I. Sutskever (2018). "Improving Language Understanding by Generative Pre-Training." In: (cit. on p. 2).

Rahmatizadeh, R., P. Abolghasemi, A. Behal, and L. Bölöni (2018). "Learning real manipulation tasks from virtual demonstrations using LSTM." In: *AAAI Conference on Artificial Intelligence (AAAI)* (cit. on pp. 94, 103).

Rahmatizadeh, R., P. Abolghasemi, L. Bölöni, and S. Levine (2017). "Vision-Based Multi-Task Manipulation for Inexpensive Robots Using End-To-End Learning from Demonstration." In: *arXiv:1707.02920* (cit. on pp. 73, 95).

Ramachandran, D. and E. Amir (2007). "Bayesian inverse reinforcement learning." In: *Urbana* 51.61801, pp. 1–4 (cit. on pp. 112, 113).

Ramachandran, P., P. J. Liu, and Q. V. Le (2016). "Unsupervised pretraining for sequence to sequence learning." In: *arXiv preprint arXiv:1611.02683* (cit. on p. 2).

Ramirez-Amaro, K., M. Beetz, and G. Cheng (2015). "Transferring skills to humanoid robots by extracting semantic representations from observations of human activities." In: *Artificial Intelligence* (cit. on p. 95).

Rana, M. A., M. Mukadam, S. R. Ahmadzadeh, S. Chernova, and B. Boots. (2017). "Towards Robust Skill Generalization: Unifying Learning from Demonstration and Motion Planning." In: *Proceedings of the 2017 Conference on Robot Learning (CoRL)* (cit. on p. 73).

Ratliff, N. D., J. A. Bagnell, and M. A. Zinkevich (2006). "Maximum margin planning." In: *International Conference on Machine Learning (ICML)*. ACM (cit. on p. 112).

Ratliff, N., J. A. Bagnell, and S. S. Srinivasa (2007). "Imitation learning for locomotion and manipulation." In: *International Conference on Humanoid Robots* (cit. on p. 79).

Ravi, S. and H. Larochelle (2017). "Optimization as a model for few-shot learning." In: *International Conference on Learning Representations (ICLR)* (cit. on pp. 2, 3, 11, 15, 22, 31, 35, 36, 51, 121, 147, 154).

Rei, M. (2015). "Online Representation Learning in Recurrent Neural Language Models." In: *Conference on Empirical Methods in Natural Language Processing (EMNLP)* (cit. on pp. 67, 149, 152).

Rezende, D. J., S. Mohamed, I. Danihelka, K. Gregor, and D. Wierstra (2016). "One-Shot Generalization in Deep Generative Models." In: *International Conference on Machine Learning (ICML)* (cit. on p. 31).

Rezende, D. J., S. Mohamed, and D. Wierstra (2014). "Stochastic backpropagation and approximate inference in deep generative models." In: *arXiv preprint arXiv:1401.4082* (cit. on p. 52).

Rhinehart, N. and K. M. Kitani (2017). "First-Person Activity Forecasting with Online Inverse Reinforcement Learning." In: *International Conference on Computer Vision (ICCV)* (cit. on pp. 96, 124).

Ross, S., G. J. Gordon, and D. Bagnell (2011). "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning." In: *AISTATS* (cit. on p. 79).

Rothfuss, J., F. Ferreira, E. Erdal Askoy, Y. Zhou, and T. Asfour (2018). "Deep Episodic Memory: Encoding, Recalling, and Predicting Episodic Experiences for Robot Action Execution." In: *arXiv:1801.04134* (cit. on p. 95).

Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. (2015). "Imagenet large scale visual recognition challenge." In: *International Journal of Computer Vision (IJCV)* (cit. on p. 2).

Sadeghi, F. and S. Levine (2016). "(CAD)^2 RL: Real Single-Image Flight without a Single Real Image." In: *Robotics: Science and Systems (R:SS)* (cit. on p. 96).

Salimans, T. and D. Kingma (2016). "Weight normalization: A simple reparameterization to accelerate training of deep neural networks." In: *Neural Information Processing Systems (NIPS)* (cit. on p. 32).

Santoro, A., S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap (2016). "Meta-learning with memory-augmented neural networks." In: *International Conference on Machine Learning (ICML)* (cit. on pp. 3, 10, 15, 16, 31, 35, 36, 40, 51, 65, 114, 121, 148).

Santos, R. J. (1996). "Equivalence of regularization and truncated iteration for general ill-posed problems." In: *Linear algebra and its applications* (cit. on pp. 49, 92, 112).

Sastry, S. S. and A. Isidori (1989). "Adaptive control of linearizable systems." In: *IEEE Transactions on Automatic Control* (cit. on p. 60).

Saxe, A., J. McClelland, and S. Ganguli (2014). "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks." In: *International Conference on Learning Representations (ICLR)* (cit. on p. 32).

Schaal, S., A. Ijspeert, and A. Billard (2003). "Computational approaches to motor learning by imitation." In: *Philosophical Transactions of the Royal Society of London B: Biological Sciences* (cit. on pp. 79, 95).

Schaal, S., J. Peters, J. Nakanishi, and A. Ijspeert (2005). "Learning movement primitives." In: *Robotics Research* (cit. on p. 79).

Schaefer, A. M., S. Udluft, and H.-G. Zimmermann (2007). "A recurrent control neural network for data efficient reinforcement learning." In: *International Symposium on Approximate Dynamic Programming and Reinforcement Learning* (cit. on p. 66).

Schaul, T., D. Horgan, K. Gregor, and D. Silver (2015). "Universal value function approximators." In: *International Conference on Machine Learning (ICML)* (cit. on pp. 73, 79).

Schenck, C. and D. Fox (2016). "Guided policy search with delayed senor measurements." In: *arXiv:1609.03076* (cit. on p. 123).

Schenck, C. and D. Fox (2017). "Visual closed-loop control for pouring liquids." In: *International Conference on Robotics and Automation (ICRA)* (cit. on p. 123).
Schmidhuber, J. (1987). "Evolutionary principles in self-referential learning." Doctoral dissertation. Institut für Informatik, Technische Universität München (cit. on pp. 2, 15, 31, 51).
Schmidhuber, J. (1992). "Learning to control fast-weight memories: An alternative to dynamic recurrent networks." In: *Neural Computation* (cit. on p. 31).
Schulman, J., S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz (2015). "Trust Region Policy Optimization." In: *International Conference on Machine Learning (ICML)* (cit. on pp. 21, 22, 38, 66, 69, 123).
Schulman, J., F. Wolski, P. Dhariwal, A. Radford, and O. Klimov (2017). "Proximal policy optimization algorithms." In: *arXiv:1707.06347* (cit. on p. 128).
Sermanet, P., C. Lynch, J. Hsu, and S. Levine (2017a). "Time-Contrastive Networks: Self-Supervised Learning from Multi-View Observation." In: *arXiv:1704.06888* (cit. on p. 96).
Sermanet, P., K. Xu, and S. Levine (2017b). "Unsupervised Perceptual Rewards for Imitation Learning." In: *Robotics: Science and Systems (RSS)* (cit. on pp. 79, 96, 117, 123).
Sharif Razavian, A., H. Azizpour, J. Sullivan, and S. Carlsson (2014). "CNN features off-the-shelf: an astounding baseline for recognition." In: *IEEE conference on computer vision and pattern recognition workshops* (cit. on p. 2).
Al-Shedivat, M., T. Bansal, Y. Burda, I. Sutskever, I. Mordatch, and P. Abbeel (2018). "Continuous adaptation via meta-learning in nonstationary and competitive environments." In: *International Conference on Learning Representations (ICLR)* (cit. on p. 132).
Shelhamer, E., J. Long, and T. Darrell (2017). "Fully Convolutional Networks for Semantic Segmentation." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)* (cit. on p. 112).
Shrivastava, A., T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb (2017). "Learning from simulated and unsupervised images through adversarial training." In: *Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 96).
Shu, R., H. H. Bui, S. Zhao, M. J. Kochenderfer, and S. Ermon (2018). "Amortized Inference Regularization." In: *arXiv preprint arXiv:1805.08913* (cit. on pp. 47, 52).
Shyam, P., S. Gupta, and A. Dukkipati (2017). "Attentive Recurrent Comparators." In: *International Conferecence on Machine Learning (ICML)* (cit. on pp. 3, 31).
Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. (2016). "Mastering the game of Go with deep neural networks and tree search." In: *Nature* (cit. on p. 66).

Sjöberg, J. and L. Ljung (1995). "Overtraining, regularization and searching for a minimum, with application to neural networks." In: *International Journal of Control* 62.6, pp. 1391–1407 (cit. on p. 112).

Snell, J., K. Swersky, and R. S. Zemel (2017). "Prototypical Networks for Few-shot Learning." In: *Neural Information Processing Systems (NIPS)* (cit. on pp. 3, 10, 31, 154).

Srinivas, A., A. Jabri, P. Abbeel, S. Levine, and C. Finn (2018). "Universal Planning Networks." In: *International Conference on Machine Learning (ICML)* (cit. on pp. 117, 123).

Stadie, B., P. Abbeel, and I. Sutskever (2017). "Third-person imitation learning." In: *International Conference on Learning Representations (ICLR)* (cit. on p. 96).

Stulp, F., G. Raiola, A. Hoarau, S. Ivaldi, and O. Sigaud (2013). "Learning compact parameterized skills with a single regression." In: *International Confernce on Humanoid Robots* (cit. on pp. 73, 79).

Sukhbaatar, S., Z. Lin, I. Kostrikov, G. Synnaeve, A. Szlam, and R. Fergus (2018). "Intrinsic motivation and automatic curricula via asymmetric self-play." In: *International Conference on Learning Representations (ICLR)* (cit. on p. 132).

Sung, F., Y. Yang, L. Zhang, T. Xiang, P. H. S. Torr, and T. M. Hospedales (2017). "Learning to Compare: Relation Network for Few-Shot Learning." In: *CoRR* abs/1711.06025. arXiv: 1711.06025. URL: http://arxiv.org/abs/1711.06025 (cit. on p. 154).

Synnaeve, G., N. Nardelli, A. Auvolat, S. Chintala, T. Lacroix, Z. Lin, F. Richoux, and N. Usunier (2016). "Torchcraft: a library for machine learning research on real-time strategy games." In: *arXiv preprint arXiv:1611.00625* (cit. on p. 114).

Tai, L., J. Zhang, M. Liu, and W. Burgard (2018). "Socially-compliant Navigation through Raw Depth Inputs with Generative Adversarial Imitation Learning." In: *International Conference on Robotics and Automation (ICRA)* (cit. on p. 96).

Tanaskovic, M., L. Fagiano, R. Smith, P. Goulart, and M. Morari (2013). "Adaptive model predictive control for constrained linear systems." In: *Control Conference (ECC), 2013 European* (cit. on p. 67).

Tenenbaum, J. B. (1999). "A Bayesian framework for concept learning." Doctoral dissertation. Massachusetts Institute of Technology (cit. on pp. 3, 13, 51).

Thrun, S. and L. Pratt (1998). *Learning to learn*. Springer Science & Business Media (cit. on pp. 2, 31, 79).

Todorov, E., T. Erez, and Y. Tassa (2012). "Mujoco: A physics engine for model-based control." In: *International Conference on Intelligent Robots and Systems (IROS)* (cit. on pp. 39, 68, 82, 101, 128).

Tow, A., N. Sünderhauf, S. Shirazi, M. Milford, and J. Leitner (2017). "What Would You Do? Acting by Learning to Predict." In: *arXiv:1703.02658* (cit. on p. 96).

Tung, H.-Y. F., A. W. Harley, L.-K. Huang, and K. Fragkiadaki (2018). "Reward Learning from Narrated Demonstrations." In: *arXiv:1804.10692* (cit. on pp. 117, 123).

Tzeng, E., J. Hoffman, N. Zhang, K. Saenko, and T. Darrell (2014). "Deep domain confusion: Maximizing for domain invariance." In: *arXiv preprint arXiv:1412.3474* (cit. on p. 96).

Van Den Oord, A., S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu (2016). "Wavenet: A generative model for raw audio." In: *arXiv:1609.03499* (cit. on pp. 90, 91).

Viereck, U., A. Pas, K. Saenko, and R. Platt (2017). "Learning a visuomotor controller for real world robotic grasping using simulated depth images." In: *Conference on Robot Learning (CoRL)* (cit. on p. 96).

Vinyals, O., C. Blundell, T. Lillicrap, D. Wierstra, et al. (2016). "Matching networks for one shot learning." In: *Neural Information Processing Systems (NIPS)* (cit. on pp. 2, 3, 10, 16, 31, 35, 36, 51, 154).

Wan, J., Z. Zhang, J. Yan, T. Li, B. D. Rao, S. Fang, S. Kim, S. L. Risacher, A. J. Saykin, and L. Shen (2012). "Sparse Bayesian multi-task learning for predicting cognitive outcomes from neuroimaging measures in Alzheimer's disease." In: *Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on pp. 3, 51).

Wang, Y.-X. and M. Hebert (2016). "Learning to learn: Model regression networks for easy small sample learning." In: *European Conference on Computer Vision (ECCV)* (cit. on pp. 3, 10, 31, 36, 51, 60, 132).

Watter, M., J. Springenberg, J. Boedecker, and M. Riedmiller (2015). "Embed to control: A locally linear latent dynamics model for control from raw images." In: *Neural Information Processing Systems (NIPS)* (cit. on pp. 117, 123, 124).

Williams, R. J. (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning." In: *Machine learning* (cit. on pp. 22, 37).

Wu, Y., M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al. (2016). "Google's neural machine translation system: Bridging the gap between human and machine translation." In: *arXiv preprint arXiv:1609.08144* (cit. on p. 1).

Wulfmeier, M., D. Z. Wang, and I. Posner (2016a). "Watch this: Scalable cost-function learning for path planning in urban environments." In: *2016 vinyals2016matching/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2089–2095 (cit. on p. 112).

Wulfmeier, M., P. Ondruska, and I. Posner (2015). "Maximum Entropy Deep Inverse Reinforcement Learning." In: *Neural Information Processing Systems Conference, Deep Reinforcement Learning Workshop*. Vol. abs/1507.04888 (cit. on pp. 112, 116).

Wulfmeier, M., D. Rao, and I. Posner (2016b). "Incorporating Human Domain Knowledge into Large Scale Cost Function Learning." In: *CoRR* abs/1612.04318 (cit. on p. 112).

Wulfmeier, M., D. Z. Wang, and I. Posner (2016c). "Watch this: Scalable cost-function learning for path planning in urban environments." In: *International Conference on Intelligent Robots and Systems (IROS)* (cit. on p. 124).

Xiang, Y., T. Schmidt, V. Narayanan, and D. Fox (2018). "PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes." In: *Robotics Science and Systems (R:SS)* (cit. on p. 86).

Xie, A., A. Singh, S. Levine, and C. Finn (2018). "Few-Shot Goal Inference for Visuomotor Learning and Planning." In: *Under Review* (cit. on p. 4).

Xu, K., E. Ratner, A. Dragan, S. Levine, and C. Finn (2018). "Learning a Prior over Intent via Meta-Inverse Reinforcement Learning." In: *arXiv:1805.12573* (cit. on p. 4).

Yahya, A., A. Li, M. Kalakrishnan, Y. Chebotar, and S. Levine (2017). "Collective robot reinforcement learning with distributed asynchronous guided policy search." In: *International Conference on Intelligent Robots and Systems (IROS)* (cit. on p. 123).

Yamaguchi, A., C. G. Atkeson, and T. Ogasawara (2015). "Pouring skills with planning and learning modeled from human demonstrations." In: *International Journal of Humanoid Robotics* (cit. on p. 123).

Yang, Y., Y. Li, C. Fermüller, and Y. Aloimonos (2015). "Robot Learning Manipulation Action Plans by" Watching" Unconstrained Videos from the World Wide Web." In: *AAAI Conference on Artificial Intelligence (AAAI)* (cit. on p. 95).

Yi, D., Z. Lei, S. Liao, and S. Z. Li (2014). "Learning face representation from scratch." In: *arXiv preprint arXiv:1411.7923* (cit. on p. 1).

Yoo, D., N. Kim, S. Park, A. S. Paek, and I. S. Kweon (2016). "Pixel-level domain transfer." In: *European Conference on Computer Vision (ECCV)*. Springer (cit. on p. 96).

Yosinski, J., J. Clune, Y. Bengio, and H. Lipson (2014). "How transferable are features in deep neural networks?" In: *Neural Information Processing Systems (NIPS)* (cit. on p. 2).

You, Y., X. Pan, Z. Wang, and C. Lu (2017). "Virtual to Real Reinforcement Learning for Autonomous Driving." In: *arXiv:1704.03952* (cit. on p. 96).

Yu, K., V. Tresp, and A. Schwaighofer (2005). "Learning Gaussian processes from multiple tasks." In: *International Conference on Machine Learning (ICML)* (cit. on pp. 3, 51).

Yu, T., C. Finn, A. Xie, S. Dasari, T. Zhang, P. Abbeel, and S. Levine (2018). "One-Shot Imitation from Observing Humans via Domain-Adaptive Meta-Learning." In: *Robotics: Science and Systems (RSS)* (cit. on p. 4).

Zaremba, W., I. Sutskever, and O. Vinyals (2014). "Recurrent neural network regularization." In: *arXiv preprint arXiv:1409.2329* (cit. on p. 166).

Zeng, A., S. Song, K.-T. Yu, E. Donlon, F. R. Hogan, M. Bauza, D. Ma, O. Taylor, M. Liu, E. Romo, et al. (2018). "Robotic Pick-and-Place of Novel Objects in Clutter with Multi-Affordance Grasping and Cross-Domain Image Matching." In: *International Conference on Robotics and Automation (ICRA)* (cit. on p. 103).

Zhang, C., S. Bengio, M. Hardt, B. Recht, and O. Vinyals (2017). "Understanding deep learning requires rethinking generalization." In: *International Conference on Learning Representations (ICLR)* (cit. on p. 12).

Zhang, J. and K. Cho (2017). "Query-Efficient Imitation Learning for End-to-End Simulated Driving." In: *AAAI Conference on Artificial Intelligence* (cit. on p. 79).

Zhang, T., Z. McCarthy, O. Jow, D. Lee, K. Goldberg, and P. Abbeel (2017). "Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation." In: *arXiv preprint arXiv:1710.04615* (cit. on pp. 73, 84, 94, 95, 103, 162, 164).

Zhu, Y., R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi (2017). "Target-driven visual navigation in indoor scenes using deep reinforcement learning." In: *International Conference on Robotics and Automation (ICRA)* (cit. on pp. 117, 123).

Ziebart, B. D., A. L. Maas, J. A. Bagnell, and A. K. Dey (2008). "Maximum Entropy Inverse Reinforcement Learning." In: *AAAI* (cit. on pp. 107–110, 112, 117, 123).