Meta-Learning for Data and Processing Efficiency

Sachin Ravi

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance by
the Department of
Computer Science
Adviser: Kai Li

June 2019

© Copyright by Sachin Ravi, 2019.

All rights reserved.

Abstract

Deep learning models have shown great success in a variety of machine learning benchmarks; however, these models still lack the efficiency and flexibility of humans. Current deep learning methods involve training on a large amount of data to produce a model that can then specialize to the specific task encoded by the training data. Humans, on the other hand, are able to learn new concepts throughout our lives with comparatively little feedback. In order to bridge this gap, previous work has suggested the use of meta-learning. Rather than learning how to do a specific task, meta-learning involves learning how-to-learn and utilizing this knowledge to learn new tasks more effectively. This thesis focuses on using meta-learning to improve the data and processing efficiency of deep learning models when learning new tasks.

First, we discuss a meta-learning model for the few-shot learning problem, where the aim is to learn a new classification task having unseen classes with few labeled examples. We use a LSTM-based meta-learner model to learn both the initialization and the optimization algorithm used to train another neural network and show that our method compares favorably to nearest-neighbor approaches. The second part of the thesis deals with improving the predictive uncertainty of models in the few-shot learning setting. Using a Bayesian perspective, we propose a meta-learning method which efficiently amortizes hierarchical variational inference across tasks, learning a prior distribution over neural network weights so that a few steps of gradient descent will produce a good task-specific approximate posterior. Finally, we focus on applying meta-learning in the context of making choices that impact processing efficacy. When training a network on multiple tasks, we have a choice between interactive parallelism (training on different tasks one after another) and independent parallelism (using the network to process multiple tasks concurrently). For the simulation environment considered, we show that there is a trade-off between these two types of processing choices in deep neural networks. We then discuss a meta-learning algorithm for an

agent to learn how to train itself with regard to this trade-off in an environment with unknown serialization cost.

Acknowledgements

I would first like to extend my sincere gratitude to my adviser, Kai Li, for his continued support and guidance during the course of my Ph.D. As my research interests converged more towards machine learning, he was always very supportive and extremely helpful in finding collaborations so that I could pursue projects that overlapped with my interests. I appreciate him providing me this academic freedom to explore the topics that I found interesting, as this made my Ph.D. experience immensely more fulfilling.

I would also like to thank Hugo Larochelle for serving as a mentor and collaborator for the past few years. Under his mentorship, I started working on meta-learning for few-shot classification, which became the main subject for the first two chapters of this dissertation. I am truly grateful for having had the opportunity to work with and learn so much from him, especially as this experience was critical to finding the specific topic that motivated large parts of this dissertation.

I would like to thank Jonathan Cohen for all his advice and support in the collaboration that resulted in the work described in the third chapter of this thesis. It was such a great opportunity to learn about the neuroscience perspective from someone so knowledgeable. Additionally, I would like to thank Sebastian Musslick - I really enjoyed the discussions we had and I appreciate his patience and help in bringing me up to speed in an area I was initially very unfamiliar with. Thank you also to my committee members Jia Deng and Karthik Narasimhan for their invaluable feedback and time.

I would additionally like to thank all the collaborators I've worked with during the past few years and from whom I've learned so much: Eric Nalisnick, Mengye Ren, Eleni Triantafillou, Jake Snell, Kevin Swersky, Richard S. Zemel, Theodore L. Willke, and Alex Beatson. Furthermore, I would like to thank all the friends I met at Princeton who made my time here so enjoyable. In particular, thank you to Hiba,

Georgina, Kevin, Andrew, Amanda, Adam, Matt, Ari, Logan, Alex, Archit, Daniel, Harvey, and many others.

A large part of this dissertation work was supported by the Intel Corporation. Furthermore, I am grateful to Princeton University for providing generous fellowships.

Lastly, and most importantly, I would like to thank my parents and sister for all their support throughout the past few years. To my parents, thank you for all the sacrifices you have made, without which I would not be where I am today. Anything I achieve is all due to your unending encouragement and love.

To Amma and Appa.

Contents

Abstract iii

Acknowledgements v

List of Tables xi

List of Figures xii

1 Introduction **1**

1.1 Deep Learning: Successes and Shortcomings 1

1.2 Human Learning 3

1.3 Meta-Learning 4

1.4 Organization and Contributions of the Thesis 4

2 Optimization as a Model for Few-Shot Learning **7**

2.1 Introduction 7

2.2 Task Description 8

2.3 Model 10

2.3.1 Model Description 10

2.3.2 Parameter Sharing & Preprocessing 12

2.3.3 Training 13

2.3.4 Batch Normalization 15

2.4 Related Work 17

2.4.1 Meta-learning 17

2.4.2 Few-shot learning . . . 17
2.5 Evaluation . . . 18
2.5.1 Experiment Results . . . 19
2.5.2 Visualization of meta-learner . . . 21
2.6 Conclusion . . . 22

3 Amortized Bayesian Meta-Learning 24
3.1 Introduction . . . 24
3.2 Meta-Learning via Hierarchical Variational Inference . . . 26
3.3 Amortized Bayesian Meta-Learning . . . 28
3.3.1 Scaling Meta-Learning with Amortized Variational Inference . 28
3.3.2 Amortized Variational Inference using only Training Set . . . 30
3.3.3 Application Details . . . 31
3.4 Related Work . . . 33
3.5 Evaluation . . . 35
3.5.1 Contextual Bandits . . . 35
3.5.2 Few-shot Learning . . . 39
3.6 Conclusion . . . 43
3.7 Appendix . . . 44
3.7.1 Pseudocode . . . 44
3.7.2 Hyperparameters . . . 46

4 Navigating the Trade-off Between Multi-Task Learning and Multitasking Capability in Deep Neural Networks 47
4.1 Introduction . . . 47
4.2 Background: Trade-off Between Multi-task Learning and Multitasking Capability . . . 49
4.2.1 Definition of Tasks and Multitasking . . . 49

4.2.2 Processing Single and Multiple Tasks Based on Task Projections 50
4.2.3 Minimal Basis Set vs Tensor Product Representations 50
4.3 Meta-Learning for Optimal Agent 52
4.3.1 Optimal Bayesian Agent 53
4.4 Related Work 55
4.5 Experiments 57
4.5.1 Task Environment 57
4.5.2 Neural Network Architecture 58
4.5.3 Effect of Sharing Representations on Learning Speed & Multitasking Ability 60
4.5.4 Effect of Single-task vs Multitask Training 61
4.5.5 Meta-Learning 62
4.6 Discussion 65
4.7 Appendix 66
4.7.1 Predictive Distribution of Meta-Learner 66

5 Conclusion 68

Bibliography 71

List of Tables

2.1 Comparison of classification performance on *mini*ImageNet 21

3.1 Comparison of cumulative regret on wheel bandit task 38

3.2 Comparison of classification performance on *mini*ImageNet 40

3.3 Hyperparameters for wheel bandit task 46

3.4 Hyperparameters for few-shot learning experiments 46

List of Figures

2.1 Example of meta-learning setup 10

2.2 Computational graph for forward pass of meta-learner 14

2.3 Visualization of input and forget values for meta-learner 22

3.1 Graphical models for Bayesian meta-learning framework 26

3.2 Visualization of arm rewards according to learned prior distribution . 38

3.3 Comparison of reliability diagrams for models 41

3.4 Comparison of empirical CDF for models 42

3.5 Standard deviation of learned prior for convolutional kernels 43

4.1 Neural network architecture from [63]. 49

4.2 Network structure for minimal basis set and tensor product 49

4.3 Neural network architecture used to learn tasks for simulation data. . 58

4.4 Effect of varying representational overlap 60

4.5 Effect of single-task vs multitask training 63

4.6 Evaluation of meta-learning algorithm 64

4.7 Visualization of actual rewards and predictive distribution of rewards 67

Chapter 1

Introduction

1.1 Deep Learning: Successes and Shortcomings

The goal of machine learning is to train agents that learn how to interact with the world in an intelligent manner. One form of machine learning that has been very successful in the past decade has been *deep learning* [56, 34]. Deep learning involves the use of multi-layer neural networks that utilize hierarchical feature extraction from given data in order to achieve a certain task. Since the ImageNet Large Scale Visual Recognition Challenge in 2012, where the deep convolutional network from Krizhevsky et al. [53] was able to win the competition handily, various variants of deep learning models have achieved state-of-the-art results in several machine learning benchmarks: residual networks [39, 40], which allow for the use of thousands of hidden layers, excel at object recognition; recurrent neural networks, which can model long-range temporal dependencies, are highly adept at natural language processing tasks, such as machine translation [81, 3]; and deep networks trained using reinforcement learning techniques can learn to play challenging games at human-level proficiency [62, 80].

The success of deep learning models, however, is heavily predicated on the use of large amounts of labeled data (for supervised learning) or training episodes (for reinforcement learning). For example, ImageNet [21], the aforementioned benchmark for object recognition, consists of 1.2 million labeled images for 1000 object classes and was collected using extensive manual human effort coordinated via Amazon Mechanical Turk. Because deep networks often have millions of free parameters that need to be fit, they require training on an extensive amount of data to appropriately fit those parameters and prevent overfitting, wherein the model displays poor generalization to data outside of the training set. In addition to costly data collection, training networks on these large datasets requires a lot of time, as several training passes through the data must be made. As we look towards having general artificial agents that can perform a variety of tasks at human-level ability, it does not seem like this extensive data collection and lengthy training process is scalable. For any new task that we want our agent to perform, we require a lot of effort and time for our agent to learn how to master the novel task.

One potential solution that has been proposed is *transfer learning* [15, 6], where a network previously trained on a different task is then fine-tuned on the new task we are interested in. The fine-tuning process involves only updating some of the network weights (potentially with a small learning rate) because the hope is that the network weights attained from training on the first task should be somewhat applicable to the second. Thus, the benefit of transfer learning is that we do not require a lot of data for the second task because our re-use of the old network weights should hopefully inhibit overfitting. For example, in Donahue et al. [22], features derived from a convolutional network trained on ImageNet are then used as input to a simple classifier, which is then trained on a different object recognition benchmark with a small number of training examples. It is shown that doing transfer learning in this way is indeed effective in preventing overfitting for the smaller dataset. However, the

downside of transfer learning is that it is an ad-hoc process, requiring us to make decisions such as what network weights to update, what learning rate to use when updating those weights, and how many iterations the network should be fined-tuned for. Furthermore, it is not entirely clear when transfer learning will work well, as its success is highly dependent on the similarity between the old and new task we are considering [89].

1.2 Human Learning

Compared to current deep learning methods, humans are very flexible and data-efficient learners. We are capable of learning a new task without an extensive amount of feedback. For a concrete comparison, consider the deep reinforcement learning agent from Mnih et al. [62], which required tens of thousands of training episodes to master a single ATARI game. If we translate the number of training episodes that the agent needed in terms of human experience, it would be equivalent to a human playing the game for 40 days with no rest [23]. In comparison, the human player used as a baseline in Mnih et al. [62] required only 2 hours to master a game.

This begs the question: what is it that allows humans to learn new concepts or tasks in a much more efficient manner? Previous work has speculated that this is possible due to our ability to build up prior knowledge and apply it appropriately in new situations [37, 55]. For example, even children as young as five can quickly infer the correct meaning of a new word by canceling out spurious possibilities using their accumulated information about the world [14]. Thus, unlike artificial neural networks which typically begin training on a new task with randomly initialized weights, we do not face new problems starting from scratch, but instead put to good use relevant knowledge we have gathered across our lifetime. Starting from such an advantageous

position may be the reason why we are able to arrive at good solutions much more quickly when facing new tasks.

1.3 Meta-Learning

In order to bridge this gap in performance between humans and artificial agents, previous work has suggested the use of *meta-learning* [85]. Rather than learning how to do a specific task, meta-learning involves learning *how-to-learn*, with the idea being that this knowledge can be utilized to learn new tasks more quickly and effectively. Meta-learning has a long history [7, 75, 76, 5, 42] but has grown to prominence recently as many have advocated for it as key to achieving human-level intelligence in the future [55]. The ability to learn at two levels (learning within each task presented, while accumulating knowledge about the similarities and differences between tasks) is seen as being crucial to improving artificial intelligence and hypothesized to match the type of learning humans may be doing.

Older work in the meta-learning literature considers the meta-learning problem as either empirical risk minimization (ERM) or Bayesian inference. The ERM perspective involves directly optimizing a meta-learner to minimize a loss across training datasets with the hope that this model will generalize to a new test dataset [5, 42]. The Bayesian perspective casts meta-learning as learning a prior in a hierarchical graphical model [82, 26]. Our primary interest in this thesis is to take inspiration from these older ideas and apply them in the present, when we have vastly improved computation and more sophisticated networks, to substantially more complex tasks.

1.4 Organization and Contributions of the Thesis

In this thesis, we focus on using meta-learning to improve the data and processing efficiency of deep learning models when learning new tasks.

In Chapter 2, we discuss a meta-learning model for the few-shot learning problem, where the aim is to be able to learn a new classification task having unseen classes with few labeled examples. This problems reflects our desire to have deep learning models that can learn new tasks using a small amount of feedback. We use a LSTM-based meta-learner model to learn both the initialization and the optimization algorithm used to train another neural network in the few-shot regime and show that our method compares favorably to nearest-neighbor approaches

In Chapter 3, we consider improving the predictive uncertainty of models in the few-shot learning setting. Proper predictive uncertainty is key to deploying machine learning models in the wild, as proper human intervention can be applied when a model's prediction is known to be uncertain. This is especially useful in the meta-learning setting, as a model needs to indicate when its prior knowledge cannot be applied appropriately when given a new task that greatly differs from the task distribution the model was trained on. Using a Bayesian perspective, we propose a meta-learning method which efficiently amortizes hierarchical variational inference across tasks, learning a prior distribution over neural network weights so that a few steps of gradient descent will produce a good task-specific approximate posterior

In Chapter 4, we focus on applying meta-learning in the context of making choices that impact processing efficacy. When training a network on multiple tasks, we have a choice between interactive parallelism (training on different tasks one after another) and independent parallelism (using the network to process multiple tasks concurrently). For the simulation environment we consider, we show that there is a trade-off between these two types of processing choices in deep neural networks, where one can learn faster by using interactive parallelism but at the cost of having to do tasks serially. We then discuss a meta-learning algorithm for an agent to learn how to train itself with regard to this trade-off in an environment with unknown serialization cost.

Finally, in Chapter 5, we provide a review of the work in this thesis and mention possible interesting avenues for future work.

The main chapters of this dissertation have been published or are in preparation as the following conference articles:

[Chapter 2] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2017.

[Chapter 3] Sachin Ravi and Alex Beatson. Amortized bayesian meta-learning. In *International Conference on Learning Representations*, 2019.

[Chapter 4] Sachin Ravi, Sebastian Musslick, Maia Hamin, Theodore L. Willke, and Jonathan D. Cohen. Navigating the trade-off between multi-task learning and multitasking capability in deep neural networks. In preparation.

Material from the dissertation has been presented at the following scholarly conferences:

Amortized Bayesian Meta-Learning. Sachin Ravi and Alex Beatson. International Conference on Learning Representations (ICLR), 2019.

Amortized Bayesian Meta-Learning. Sachin Ravi and Alex Beatson. Neural Information Processing Systems (NeurIPS) Workshop on Meta-Learning, 2019.

Optimization as a Model for Few-Shot Learning. Sachin Ravi and Hugo Larochelle. International Conference on Learning Representations (ICLR), 2017.

Chapter 2

Optimization as a Model for Few-Shot Learning

2.1 Introduction

In this chapter, we study the problem of few-shot learning, where we would like to learn a set of concepts using just a few labeled examples from each class. Naive gradient-based optimization with deep networks fails in the face of few labeled examples for several reasons. Firstly, different variants of gradient-based optimization algorithms, such as momentum [67], Adagrad [24], Adadelta [91], and ADAM [48], weren't designed specifically to perform well under the constraint of a set number of updates. Specifically, when applied to non-convex optimization problems, with a reasonable choice of hyperparameters these algorithms don't have very strong guarantees of speed of convergence, beyond that they will eventually converge to a good solution after what could be many millions of iterations. Secondly, for each separate dataset considered, the network would have to start from a random initialization of its parameters, which considerably hurts its ability to converge to a good solution after a few updates. Transfer learning can be applied to alleviate this problem by

fine-tuning a pre-trained network from another task which has more labelled data; however, as mentioned in the previous chapter, it has been observed that the benefit of a pre-trained network greatly decreases as the task the network was trained on diverges from the target task. What is needed is a systematic way to learn a beneficial common initialization that would serve as a good point to start training for the set of datasets being considered. This would provide the same benefits as transfer learning, but with the guarantee that the initialization is an optimal starting point for fine-tuning.

This chapter presents a method that addresses the weakness of neutral networks trained with gradient-based optimization on the few-shot learning problem by framing the problem within a meta-learning setting. We propose a LSTM-based *meta-learner* optimizer that is trained to optimize a *learner* neural network classifier. The meta-learner captures both short-term knowledge within a task and long-term knowledge common among all the tasks. By using an objective that directly captures an optimization algorithm's ability to have good generalization performance given only a set number of updates, the meta-learner model is trained to converge a learner classifier to a good solution quickly on each task. Additionally, the formulation of our meta-learner model allows it to learn a task-common initialization for the learner classifier, which captures fundamental knowledge shared among all the tasks. We demonstrate that this meta-learning model is competitive with deep metric-learning techniques for few-shot learning.

2.2 Task Description

We first begin by detailing the meta-learning formulation we use. In the typical machine learning setting, we are interested in a dataset D and usually split D so that we optimize parameters ϕ on a training set D_{train} and evaluate its generalization

on the test set D_{test}. In meta-learning, however, we are dealing with meta-sets $\mathscr{D}$ containing multiple regular datasets, where each $D \in \mathscr{D}$ has a split of D_{train} and D_{test}.

We consider the k-shot, n-class classification task, where for each dataset D, the training set consists of k labelled examples for each of n classes, meaning that D_{train} consists of $N = n \cdot k$ examples, and D_{test} has a set number of examples for evaluation. We note that previous work [86] has used the term *episode* to describe each dataset consisting of a training and test set.

In meta-learning, we thus have different meta-sets for meta-training, meta-validation, and meta-testing ($\mathscr{D}_{meta-train}$, $\mathscr{D}_{meta-validation}$, and $\mathscr{D}_{meta-test}$, respectively). On $\mathscr{D}_{meta-train}$, we are interested in training *a learning procedure* (the meta-learner) that can take as input one of its training sets D_{train} and produce a classifier (the learner) that achieves high average classification performance on its corresponding test set D_{test}. Using $\mathscr{D}_{meta-validation}$ we can perform hyper-parameter selection of the meta-learner and evaluate its generalization performance on $\mathscr{D}_{meta-test}$.

For this formulation to correspond to the few-shot learning setting, each training set in datasets $D \in \mathscr{D}$ will contain few labeled examples (we consider $k = 1$ or $k = 5$), that must be used to generalize to good performance on the corresponding test set. An example of this formulation is given in Figure 2.1.

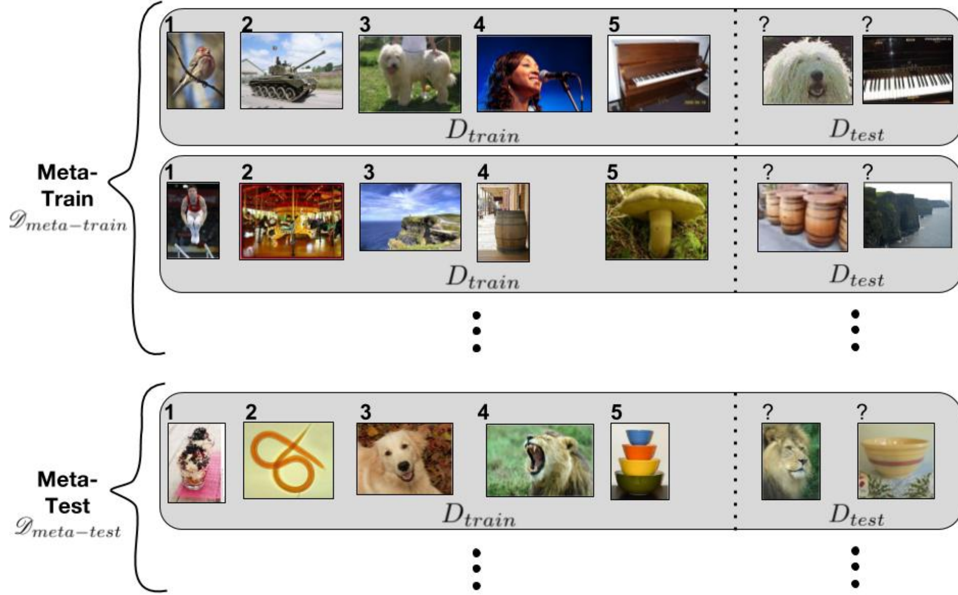


Figure 2.1: Example of meta-learning setup. The top represents the meta-training set $\mathscr{D}_{meta-train}$, where inside each gray box is a separate dataset that consists of the training set D_{train} (left side of dashed line) and the test set D_{test} (right side of dashed line). In this illustration, we are considering the 1-shot, 5-class classification task where for each dataset, we have one example from each of 5 classes (each given a label 1-5) in the training set and 2 examples for evaluation in the test set. The meta-test set $\mathscr{D}_{meta-test}$ is defined in the same way, but with a different set of datasets that cover classes not present in any of the datasets in $\mathscr{D}_{meta-train}$ (similarly, we additionally have a meta-validation set that is used to determine hyper-parameters).

2.3 Model

We now move to the description of our proposed model for meta-learning.

2.3.1 Model Description

Consider a single dataset, or episode, $D \in \mathscr{D}_{meta-train}$. Suppose we have a *learner* neural net classifier with parameters ϕ that we want to train on D_{train}. The standard optimization algorithms used to train deep neural networks are some variant of gradient descent, which uses updates of the form

$$\phi_t = \phi_{t-1} - \alpha_t \nabla_{\phi_{t-1}} \mathcal{L}_t, \tag{2.1}$$

where ϕ_{t-1} are the parameters of the learner after $t-1$ updates, α_t is the learning rate at time t, $\mathcal{L}_t$ is the loss optimized by the learner for its tth update, $\nabla_{\phi_{t-1}}\mathcal{L}_t$ is the gradient of that loss with respect to parameters ϕ_{t-1}, and ϕ_t is the updated parameters of the learner.

Our key observation that we leverage here is that this update resembles the update for the cell state in an LSTM [41]

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \tag{2.2}$$

if $f_t = 1, c_{t-1} = \phi_{t-1}, i_t = \alpha_t$, and $\tilde{c}_t = -\nabla_{\phi_{t-1}}\mathcal{L}_t$.

Thus, we propose training a *meta-learner* LSTM to learn an update rule for training a neural network. We set the cell state of the LSTM to be the parameters of the learner, or $c_t = \phi_t$, and the candidate cell state $\tilde{c}_t = \nabla_{\phi_{t-1}}\mathcal{L}_t$, given how valuable information about the gradient is for optimization. We define parametric forms for i_t and f_t so that the meta-learner can determine optimal values through the course of the updates.

Let us start with i_t, which corresponds to the learning rate for the updates. We let

$$i_t = \sigma\left(\mathbf{W}_I \cdot \left[\nabla_{\phi_{t-1}}\mathcal{L}_t, \mathcal{L}_t, \phi_{t-1}, i_{t-1}\right] + \mathbf{b}_I\right),$$

meaning that the learning rate is a function of the current parameter value ϕ_{t-1}, the current gradient $\nabla_{\phi_{t-1}}\mathcal{L}_t$, the current loss $\mathcal{L}_t$, and the previous learning rate i_{t-1}. With this information, the meta-learner should be able to finely control the learning rate so as to train the learner quickly while avoiding divergence.

As for f_t, it seems possible that the optimal choice isn't the constant 1. Intuitively, what would justify shrinking the parameters of the learner and forgetting part of its previous value would be if the learner is currently in a bad local optima and needs a large change to escape. This would correspond to a situation where the loss is high

but the gradient is close to zero. Thus, one proposal for the forget gate is to have it be a function of that information, as well as the previous value of the forget gate:

$$f_t = \sigma\left(\mathbf{W}_F \cdot \left[\nabla_{\phi_{t-1}}\mathcal{L}_t, \mathcal{L}_t, \phi_{t-1}, f_{t-1}\right] + \mathbf{b}_F\right).$$

Additionally, notice that we can also learn the initial value of the cell state c_0 for the LSTM, treating it as a parameter of the meta-learner. This corresponds to the initial weights of the classifier (that the meta-learner is training). Learning this initial value lets the meta-learner determine the optimal initial weights of the learner so that training begins from a beneficial starting point that allows optimization to proceed rapidly. Lastly, note that though the meta-learner's update rule matches the cell state update of the LSTM, the meta-learner also bears similarity to the GRU [18] hidden state update, with the exception that the forget and input gates aren't tied to sum to one.

2.3.2 Parameter Sharing & Preprocessing

Because we want our meta-learner to produce updates for deep neural networks, which consist of tens of thousands of parameters, to prevent an explosion of meta-learner parameters we need to employ some sort of parameter sharing. Thus as in Andrychowicz et al. [2], we share parameters across the coordinates of the learner gradient. This means each coordinate has its own hidden and cell state values but the LSTM parameters are the same across all coordinates. This allows us to use a compact LSTM model and additionally has the nice property that the same update rule is used for each coordinate, but one that is dependent on the respective history of each coordinate during optimization. We can easily implement parameter sharing by having the input be a batch of gradient coordinates and loss inputs $(\nabla_{\phi_{t-1,i}}\mathcal{L}_t, \mathcal{L}_t)$ for each dimension i.

Because the different coordinates of the gradients and the losses can be of very different magnitudes, we need to be careful in normalizing the values so that the meta-learner is able to use them properly during training. Thus, we also found that the preprocessing method of Andrychowicz et al. [2] worked well when applied to both the dimensions of the gradients and the losses at each time step:

$$x \rightarrow \begin{cases} \left(\frac{\log(|x|)}{p}, \operatorname{sgn}(x)\right) & \text{if } |x| \geq e^{-p} \\ (-1, e^p x) & \text{otherwise} \end{cases}$$

This preprocessing adjusts the scaling of gradients and losses, while also separating the information about their magnitude and their sign (the latter being mostly useful for gradients). We found that the suggested value of $p = 10$ in the above formula worked well in our experiments.

2.3.3 Training

The question now is how do we train the LSTM meta-learner model to be effective at few-shot learning tasks? As observed by Vinyals et al. [86], in order to perform well at this task, it is key to have training conditions match those of test time. During evaluation of the meta-learning, for each dataset (episode), $D = (D_{train}, D_{test}) \in \mathscr{D}_{meta-test}$, a good meta-learner model will, given a series of learner gradients and losses on the training set D_{train}, suggest a series of updates for the classifier that pushes it towards good performance on the test set D_{test}.

Thus to match test time conditions, when considering each dataset $D \in \mathscr{D}_{meta-train}$, the training objective we use is the loss $\mathcal{L}_{test}$ of the produced classifier on D's test set D_{test}. While iterating over the examples in D's training set D_{train}, at each time step t the LSTM meta-learner receives $(\nabla_{\phi_{t-1}} \mathcal{L}_t, \mathcal{L}_t)$ from the learner (the classifier) and proposes the new set of parameters ϕ_t. The process repeats for

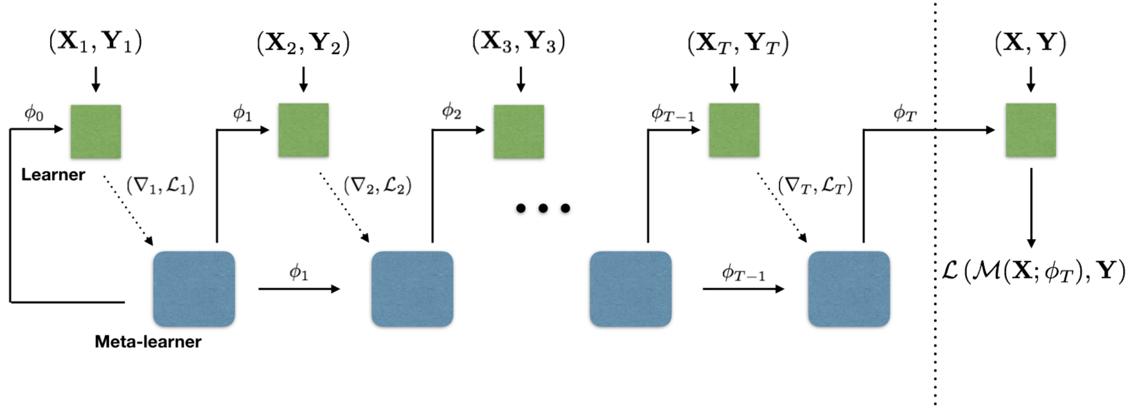


Figure 2.2: Computational graph for the forward pass of the meta-learner. The dashed line divides examples from the training set D_{train} and test set D_{test}. Each $(\mathbf{X}_i, \mathbf{Y}_i)$ is the ith batch from the training set whereas $(\mathbf{X}, \mathbf{Y})$ is all the elements from the test set. The dashed arrows indicate that we do not back-propagate through that step when training the meta-learner. We refer to the learner as $\mathcal{M}$, where $\mathcal{M}(\mathbf{X}; \phi)$ is the output of learner $\mathcal{M}$ using parameters ϕ for inputs $\mathbf{X}$. We also use ∇_t as a shorthand for $\nabla_{\phi_{t-1}} \mathcal{L}_t$.

T steps, after which the classifier and its final parameters are evaluated on the test set to produce the loss that is then used to train the meta-learner. The training algorithm is described in Algorithm 1 and the corresponding computational graph is shown in Figure 2.2.

Gradient independence assumption

Notice that our formulation would imply that the losses $\mathcal{L}_t$ and gradients $\nabla_{\phi_{t-1}} \mathcal{L}_t$ of the learner are dependent on the parameters of the meta-learner. Gradients on the meta-learner's parameters should normally take this dependency into account. However, as discussed by Andrychowicz et al. [2], this complicates the computation of the meta-learner's gradients. Thus, following Andrychowicz et al. [2], we make the simplifying assumption that these contributions to the gradients aren't important and can be ignored, which allows us to avoid taking second derivatives, a considerably expensive operation. We were still able to train the meta-learner effectively in spite of this simplifying assumption.

Initialization of Meta-Learner LSTM

When training LSTMs, it is advised to initialize the LSTM with small random weights and to set the forget gate bias to a large value so that the forget gate is initialized to be close to 1, thus enabling gradient flow [90]. In addition to the forget gate bias setting, we found that we needed to initialize the input gate bias to be small so that the input gate value (and thus the learning rate) used by the meta-learner LSTM starts out being small. With this combined initialization, the meta-learner starts close to normal gradient descent with a small learning rate, which helps initial stability of training.

2.3.4 Batch Normalization

Batch Normalization [43] is a recently proposed method to stabilize and thus speed up learning of deep neural networks by reducing internal covariate shift within the learner's hidden layers. This reduction is achieved by normalizing each layer's pre-activation, by subtracting by the mean and dividing by the standard deviation. During training, the mean and standard deviation are estimated using the current batch being trained on, whereas during evaluation a running average of both statistics calculated on the training set is used. We need to be careful with batch normalization for the learner network in the meta-learning setting, because we do not want to collect mean and standard deviation statistics during meta-testing in a way that allows information to leak between different datasets (episodes), being considered. One easy way to prevent this issue is to not collect statistics at all during the meta-testing phase, but just use our running averages from meta-training. This, however, has a bad impact on performance, because we have changed meta-training and meta-testing conditions, causing the meta-learner to learn a method of optimization that relies on batch statistics which it now does not have at meta-testing time. In order to keep the two phases as similar as possible, we found that a better strategy was to collect

Algorithm 1 Train Meta-Learner

Input: Meta-training set $\mathscr{D}_{meta-train}$, Learner $\mathcal{M}$ with parameters ϕ, Meta-Learner R with parameters θ.

1: $\phi_0 \leftarrow$ random initialization
2:
3: **for** $d = 1, \ldots$ until convergence **do**
4: $\quad D_{train}, D_{test} \leftarrow$ random dataset from $\mathscr{D}_{meta-train}$
5: $\quad \phi_0 \leftarrow c_0$ ▷ Intialize learner parameters
6:
7: $\quad$ **for** $t = 1, T$ **do**
8: $\quad\quad \mathbf{X}_t, \mathbf{Y}_t \leftarrow$ random batch from D_{train}
9: $\quad\quad \mathcal{L}_t \leftarrow \mathcal{L}(\mathcal{M}(\mathbf{X}_t; \phi_{t-1}), \mathbf{Y}_t)$ ▷ Get loss of learner on train batch
10: $\quad\quad c_t \leftarrow R((\nabla_{\phi_{t-1}} \mathcal{L}_t, \mathcal{L}_t); \theta_{d-1})$ ▷ Get output of meta-learner using Equation 2.2
11: $\quad\quad \phi_t \leftarrow c_t$ ▷ Update learner parameters
12: $\quad$ **end for**
13:
14: $\quad \mathbf{X}, \mathbf{Y} \leftarrow D_{test}$
15: $\quad \mathcal{L}_{test} \leftarrow \mathcal{L}(\mathcal{M}(\mathbf{X}; \phi_T), \mathbf{Y})$ ▷ Get loss of learner on test batch
16: $\quad$ Update θ_d using $\nabla_{\theta_{d-1}} \mathcal{L}_{test}$ ▷ Update meta-learner parameters
17:
18: **end for**

statistics for each dataset $D \in \mathscr{D}$ during $\mathscr{D}_{meta-test}$, but then erase the running statistics when we consider the next dataset. Thus, during meta-training, we use batch statistics for both the training and testing set whereas during meta-testing, we use batch statistics for the training set (and to compute our running averages) but then use the running averages during testing. This does not cause any information to leak between different datasets, but also allows the meta-learner to be trained on conditions that are matched between training and testing. Lastly, because we are doing very few training steps, we computed the running averages so that higher preference is given to the later values.

2.4 Related Work

While this work falls within the broad literature of transfer learning in general, we focus here on positioning it relative to previous work on meta-learning and few-shot learning.

2.4.1 Meta-learning

In Santoro et al. [74], a memory-augmented neural network is trained to learn how to store and retrieve memories to use for each classification task. The work of Andrychowicz et al. [2] uses an LSTM to train a neural network; however, they are interested in learning a general optimization algorithm to train neural networks for large-scale classification, whereas we are interested in the few-shot learning problem. This work also builds upon Hochreiter et al. [42] and Bosc [12], both of whom used LSTMs to train multi-layer perceptrons to learn on binary classification and time-series prediction tasks. Another related method is the work of Bertinetto et al. [10], who train a meta-learner to map a training example to the weights of a neural network that is then used to classify future examples from this class; however, unlike our method the classifier network is directly produced rather than being fine-tuned after multiple training steps. The work in this chapter also bears similarity to that of Maclaurin et al. [59], who tune the hyperparameters of gradient descent with momentum by backpropagating through the chain of gradient steps to optimize the validation performance.

2.4.2 Few-shot learning

The best performing methods for few-shot learning have been mainly metric learning methods. Deep siamese networks [51] involves training a convolutional network to embed examples so that items in the same class are close while items in different

classes are far away, according to some distance metric. Matching networks [86] refine this idea so that training and testing conditions match, by defining a differentiable nearest neighbor loss involving the cosine similarities of embeddings produced by a convolutional network.

2.5 Evaluation

In this section, we describe the results of experiments, examining the properties of our model and comparing our method's performance against different approaches. Following Vinyals et al. [86], we consider the k-shot, n-class classification setting where a meta-learner trains on many related but small training sets of k examples for each of n classes. We first split the list of all classes in the data into disjoint sets and assign them to each meta-set of meta-training, meta-validation, and meta-testing. To generate each instance of a k-shot, n-class task dataset $D = (D_{train}, D_{test}) \in \mathscr{D}$, we do the following: we first sample n classes from the list of classes corresponding to the meta-set we consider. We then sample k examples from each of those classes. These k examples together compose the training set D_{train}. Then, an additional fixed amount of the rest of the examples are sampled to yield a test set D_{test}. We generally have 15 examples per class in the test sets. When training the meta-learner, we iterate by sampling these datasets (episodes) repeatedly. For meta-validation and meta-testing, however, we produce a fixed number of these datasets to evaluate each method. We produce enough datasets to ensure that the confidence interval of the mean accuracy is small.

For the learner, we use a simple CNN containing 4 convolutional layers, each of which is a 3×3 convolution with 32 filters, followed by batch normalization, a ReLU non-linearity, and lastly a 2×2 max-pooling. The network then has a final linear layer followed by a softmax for the number of classes being considered. The loss function $\mathcal{L}$

is the average negative log-probability assigned by the learner to the correct class. For the meta-learner, we use a 2-layer LSTM, where the first layer is a normal LSTM and the second layer is our modified LSTM meta-learner. The gradients and losses are preprocessed and fed into the first layer LSTM, and the regular gradient coordinates are also used by the second layer LSTM to implement the state update rule shown in (2.1). At each time step, the learner's loss and gradient is computed on a batch consisting of the entire training set D_{train}, because we consider training sets with only a total of 5 or 25 examples. We train our LSTM with ADAM using a learning rate of 0.001 and with gradient clipping using a value of 0.25.

2.5.1 Experiment Results

The *mini*ImageNet dataset was proposed by Vinyals et al. [86] as a benchmark offering the challenges of the complexity of ImageNet images, without requiring the resources and infrastructure necessary to run on the full ImageNet dataset. Because the exact splits used in Vinyals et al. [86] were not released, we create our own version of the *mini*ImageNet dataset by selecting a random 100 classes from ImageNet and picking 600 examples of each class. We use 64, 16, and 20 classes for training, validation and testing, respectively. We consider 1-shot and 5-shot classification for 5 classes. We use 15 examples per class for evaluation in each test set. We compare against two baselines and a recent metric-learning technique, Matching Networks [86], which has achieved state-of-the-art results in few-shot learning. The results are shown in Table 2.1.

The first baseline we use is a nearest-neighbor baseline (*Baseline-nearest-neighbor*), where we first train a network to classify between all the classes jointly in the original meta-training set. At meta-test time, for each dataset D, we embed all the items in the training set using our trained network and then use nearest-neighbor matching among the embedded training examples to classify each test example.

The second baseline we use (*Baseline-finetune*) represents a coarser version of our meta-learner model. As in the first baseline, we start by training a network to classify jointly between all classes in the meta-training set. We then use the meta-validation set to search over SGD hyperparameters, where each training set is used to fine-tune the pre-trained network before evaluating on the test set. We use a fixed number of updates for fine tuning and search over the learning rate and learning rate decay used during the course of these updates.

For Matching Networks, we implemented our own version of both the basic and the fully-conditional embedding (FCE) versions. In the basic version, a convolutional network is trained to learn independent embeddings for examples in the training and test set. In the FCE version, a bidirectional-LSTM is used to learn an embedding for the training set such that each training example's embedding is also a function of all the other training examples. Additionally, an attention-LSTM is used so that a test example embedding is also a function of all the embeddings of the training set. We do not consider fine-tuning the network using the train set during meta-testing to improve performance as mentioned in Vinyals et al. [86], but do note that our meta-learner could also be fine-tuned using this data. Note that to remain consistent with Vinyals et al. [86], our baseline and matching net convolutional networks have 4 layers each with 64 filters. We also added dropout to each convolutional block in matching nets to prevent overfitting.

For our meta-learner, we train different models for the 1-shot and 5-shot tasks, that make 12 and 5 updates, respectively. We noticed that better performance for each task was attained if the meta-learner is explicitly trained to do the set number of updates during meta-training that will be used during meta-testing.

We attain results that are much better than the baselines discussed and competitive with Matching Networks. For 5-shot, we are able to do much better than Matching Networks, whereas for 1-shot, the confidence interval for our performance

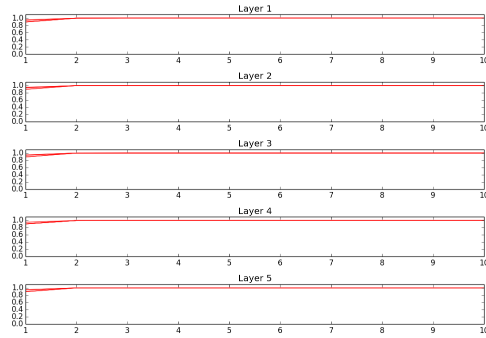
Model	5-class	
	1-shot	5-shot
Baseline-finetune	$28.86 \pm 0.54\%$	$49.79 \pm 0.79\%$
Baseline-nearest-neighbor	$41.08 \pm 0.70\%$	$51.04 \pm 0.65\%$
Matching Network	$\mathbf{43.40 \pm 0.78\%}$	$51.09 \pm 0.71\%$
Matching Network FCE	$\mathbf{43.56 \pm 0.84\%}$	$55.31 \pm 0.73\%$
Meta-Learner LSTM (OURS)	$\mathbf{43.44 \pm 0.77\%}$	$\mathbf{60.60 \pm 0.71\%}$

Table 2.1: Average classification accuracies on *mini*ImageNet with 95% confidence intervals. Marked in bold are the best results for each scenario, as well as other results with an overlapping confidence interval.

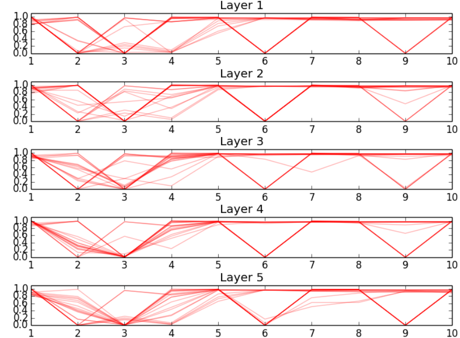
intersects the interval for Matching Networks. Again, we note that the numbers do not match the ones provided by Vinyals et al. [86] simply because we created our version of the dataset and implemented our own versions of their model. It is interesting to note that the fine-tuned baseline is worse than the nearest-neighbor baseline. Because we are not regularizing the classifier, with very few updates the fine-tuning model overfits, especially in the 1-shot case. This propensity to overfit speaks to the benefit of meta-training the initialization of the classifier end-to-end as is done in the meta-learning LSTM.

2.5.2 Visualization of meta-learner

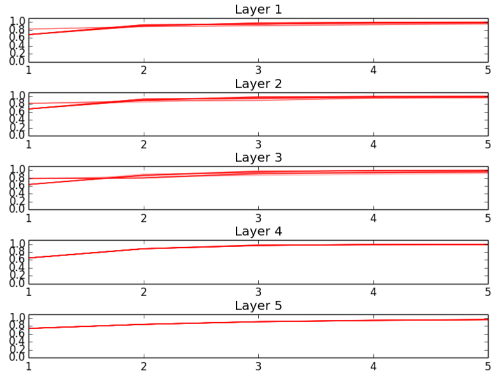
We also visualize the optimization strategy learned by the meta-learner, in Figure 2.3. We can look at the i_t and f_t gate values in Equation 2.2 at each update step, to try to get an understanding of how the meta-learner updates the learner during training. We visualize the gate values while training on different datasets D_{train}, to observe whether there are variations between training sets. We consider both 1-shot and 5-shot classification settings, where the meta-learner is making 10 and 5 updates, respectively. For the forget gate values for both tasks, the meta-learner seems to adopt a simple weight decay strategy that seems consistent across different layers. The input gate values are harder to interpret to glean the meta-learner's strategy. However, there seems to a be a lot of variability between different datasets, indicating



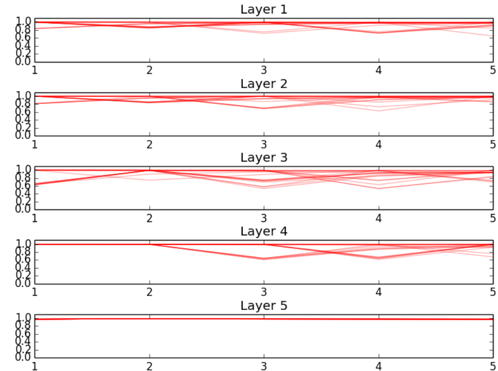
(a) Forget gate values for 1-shot task.



(b) Input gate values for 1-shot task.



(c) Forget gate values for 5-shot task.



(d) Input gate values for 5-shot task.

Figure 2.3: Visualization of the input and forget values output by the meta-learner during the course of its updates. Layers $1-4$ represent the values for a randomly selected parameter from the 4 convolutional layers and layer 5 represents the values for a random parameter from fully-connected layer. The different curves represent training steps on different datasets.

that the meta-learner isn't simply learning a fixed optimization strategy. Additionally, there seem to be differences between the two tasks, suggesting that the meta-learner has adopted different methods to deal with the different conditions of each setting.

2.6 Conclusion

We described an LSTM-based model for meta-learning, which is inspired from the parameter updates suggested by gradient descent optimization algorithms. Our LSTM meta-learner uses its state to represent the learning updates of the parameters of a classifier. It is trained to discover both a good initialization for the learner's parame-

ters, as well as a successful mechanism for updating the learner's parameters to a given small training set for some new classification task. Our experiments demonstrate that our approach outperforms natural baselines and is competitive to the state-of-the-art in metric learning for few-shot learning.

Chapter 3

Amortized Bayesian Meta-Learning

3.1 Introduction

In the previous chapter, we discussed a meta-learning method for training a model to perform well on on a distribution of meta-training tasks, where each task is a few-shot learning problem. The model was then applied to an unseen evaluation task to measure the quality of our learned model. Implicit in this setup was the assumption that the evaluation task be drawn from a task distribution somewhat similar to the one used for training. When this distributional assumption holds true, it is likely that the prior knowledge gained across training tasks will be useful for solving the evaluation task. Though the model's performance on the few-shot learning benchmark was good, it is unclear how well such a method (and other meta-learning methods in general) would perform in real-world settings, where the relationship between meta-training and meta-test tasks could be tenuous. For success in the wild, in addition to good predictive accuracy, it is also important for meta-learning models to have good predictive uncertainty - to express high confidence when a prediction is likely

to be correct but display low confidence when a prediction could be unreliable. This type of guarantee in predictive ability would allow appropriate human intervention when a prediction is known to have high uncertainty. With such an assurance, we can measure when a model struggles with a new task because the task is difficult to solve using the model's accumulated prior knowledge using the training tasks.

Bayesian methods offer a principled framework to reason about uncertainty, and approximate Bayesian methods have been used to provide deep learning models with accurate predictive uncertainty [30, 58]. By inferring a posterior distribution over neural network weights, we can produce a posterior predictive distribution that properly indicates the level of confidence on new unseen examples. Accordingly, we consider meta-learning under a Bayesian view in order to transfer the aforementioned benefits to our setting. Specifically, we extend the work of Amit and Meir [1], who considered hierarchical variational inference for meta-learning. The work primarily dealt with PAC-Bayes bounds in meta-learning and the experiments consisted of data with tens of training episodes and small networks. In this chapter, we show how the meta-learning framework defined in the previous chapter can be used to efficiently amortize variational inference for the Bayesian model of Amit and Meir [1] in order to combine the former's flexibility and scalability with the latter's uncertainty quantification.

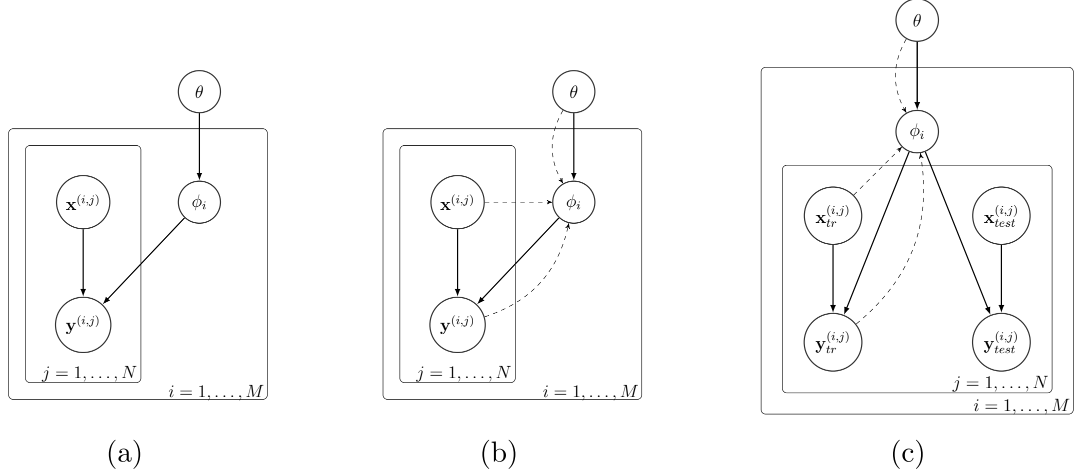


Figure 3.1: Graphical models for meta-learning framework. Dotted lines denote variational approximations. (a) Original setup in [1] where inference parameters are learned separately for each episode (b) Proposed initial amortized variational inference scheme (c) Proposed amortized variational inference scheme with train and test splits.

3.2 Meta-Learning via Hierarchical Variational Inference

We first start by reviewing the hierarchical variational bayes formulation used in Amit and Meir [1] for meta-learning. Using notation from previous chapter, assume we observe data from M episodes and assume for now that there is no train-test split, meaning that the ith episode consists of data $D^{(i)}$ containing N data items i.e. $D^{(i)} = \left\{\left(\mathbf{x}^{(i,j)}, \mathbf{y}^{(i,j)}\right)\right\}_{j=1}^{N}$. We assume a hierarchical model with global latent variable θ and episode-specific variables ϕ_i, $i = 1, \ldots M$ (see Figure 3.1).

Hierarchical variational inference can then be used to lower bound the likelihood of the data:

$$
\begin{aligned}
&\log\left[\prod_{i=1}^{M} p(D^{(i)})\right] = \log\left[\int p(\theta)\left[\prod_{i=1}^{M}\int p(D^{(i)}|\phi_i)p(\phi_i|\theta)\,d\phi_i\right]\,d\theta\right] \\
&\geq \mathbb{E}_{q(\theta;\psi)}\left[\log\left(\prod_{i=1}^{M}\int p(D^{(i)}|\phi_i)p(\phi_i|\theta)\,d\phi_i\right)\right] - \mathrm{KL}(q(\theta;\psi)\|p(\theta)) \\
&= \mathbb{E}_{q(\theta;\psi)}\left[\sum_{i=1}^{M}\log\left(\int p(D^{(i)}|\phi_i)p(\phi_i|\theta)\,d\phi_i\right)\right] - \mathrm{KL}(q(\theta;\psi)\|p(\theta)) \\
&\geq \mathbb{E}_{q(\theta;\psi)}\left[\sum_{i=1}^{M}\mathbb{E}_{q(\phi_i;\lambda_i)}\left[\log p(D^{(i)}|\phi_i)\right] - \mathrm{KL}(q(\phi_i;\lambda_i)\|p(\phi_i|\theta))\right] - \mathrm{KL}(q(\theta;\psi)\|p(\theta)) \\
&= \mathcal{L}(\psi,\lambda_1,\ldots,\lambda_M).
\end{aligned}
$$

Here, ψ and $\lambda_1,\ldots,\lambda_M$ are the variational parameters of the approximate posteriors over the global latent variables θ and the local latent variables $\phi_1,\ldots,\phi_M$, respectively.

Thus, variational inference involves solving the following optimization problem:

$$
\begin{aligned}
&\underset{\psi,\lambda_1\ldots,\lambda_M}{\arg\max}\,\mathbb{E}_{q(\theta;\psi)}\left[\sum_{i=1}^{M}\mathbb{E}_{q(\phi_i;\lambda_i)}\left[\log p(D^{(i)}|\phi_i)\right] - \mathrm{KL}(q(\phi_i;\lambda_i)\|p(\phi_i|\theta))\right] \\
&\qquad - \mathrm{KL}(q(\theta;\psi)\|p(\theta)) \qquad (3.1) \\
&\equiv \underset{\psi,\lambda_1\ldots,\lambda_M}{\arg\min}\,\mathbb{E}_{q(\theta;\psi)}\left[\sum_{i=1}^{M}-\mathbb{E}_{q(\phi_i;\lambda_i)}\left[\log p(D^{(i)}|\phi_i)\right] + \mathrm{KL}(q(\phi_i;\lambda_i)\|p(\phi_i|\theta))\right] \\
&\qquad + \mathrm{KL}(q(\theta;\psi)\|p(\theta)) \qquad (3.2)
\end{aligned}
$$

Amit and Meir [1] solve this optimization problem via mini-batch gradient descent on the objective starting from random initialization for all variational parameters. They maintain distinct variational parameters λ_i for each episode i, each of which indexes a distribution over episode-specific weights $q(\phi_i;\lambda_i)$. While they only consider problems with at most 10 or so training episodes and where each ϕ_i is small (the

weights of a 2-layer convolutional network), this approach is not scalable to problems with large numbers of episodes - such as few-shot learning, where we can generate millions of episodes by randomizing over classes and examples - and requiring deep networks.

3.3 Amortized Bayesian Meta-Learning

3.3.1 Scaling Meta-Learning with Amortized Variational Inference

Learning local variational parameters λ_i for a large number of episodes M becomes difficult as M grows due to the costs of storing and computing each λ_i. These problems are compounded when each ϕ_i is the weight of a deep neural network and each λ_i are variational parameters of the weight distribution (such as a mean and standard deviation of each weight). Instead of maintaining M different variational parameters λ_i indexing distributions over neural network weights ϕ_i, we compute λ_i on the fly with amortized variational inference (AVI), where a global learned model is used to predict λ_i from $D^{(i)}$. A popular use of AVI is training a variational autoencoder [49], where a trained encoder network produces the variational parameters for each data point. Rather than training an encoder to predict λ_i given the episode, we show that inference can be amortized by finding a good initialization, a la MAML [28]. We represent the variational parameters for each episode as the output of several steps of gradient descent from a global initialization.

Let $\mathcal{L}_{D^{(i)}}(\lambda, \theta) = -\mathbb{E}_{q(\phi_i;\lambda)}\left[\log p(D^{(i)}|\phi_i)\right] + \mathrm{KL}(q(\phi_i;\lambda)\|p(\phi_i|\theta))$ be the part of the objective corresponding to data $D^{(i)}$. Let the procedure $SGD_K(D, \lambda^{(init)}, \theta)$ represent the variational parameters produced after K steps of gradient descent on the objective $\mathcal{L}_D(\lambda, \theta)$ with respect to λ starting at the initialization $\lambda^{(0)} = \lambda^{(init)}$ and where θ is held constant i.e.:

1. $\lambda^{(0)} = \lambda^{(init)}$

2. for $k = 0, \ldots, K-1$, set
$\lambda^{(k+1)} = \lambda^{(k)} - \alpha \nabla_{\lambda^{(k)}} \mathcal{L}_D(\lambda^{(k)}, \theta)$

We represent the variational distribution for each dataset $q_\theta(\phi_i|D^{(i)})$ in terms of the local variational parameters λ_i produced after K steps of gradient descent on the loss for dataset $D^{(i)}$, starting from the global initialization θ:

$$q_\theta(\phi_i|D^{(i)}) = q(\phi_i; SGD_K(D^{(i)}, \theta, \theta)).$$

Note that θ here serves as both the global initialization of local variational parameters and the parameters of the prior $p(\phi \,|\, \theta)$. We could pick a separate prior and global initialization, but we found tying the prior and initialization did not seem to have a negative affect on performance, while significantly reducing the number of total parameters necessary. With this form of the variational distribution, this turns the optimization problem of (3.2) into

$$\begin{aligned} \arg\min_{\psi} \mathbb{E}_{q(\theta;\psi)} & \left[\sum_{i=1}^{M} -\mathbb{E}_{q_\theta(\phi_i|D^{(i)})} \left[\log p(D^{(i)}|\phi_i) \right] + \mathrm{KL}(q_\theta(\phi_i|D^{(i)}) \| p(\phi_i|\theta)) \right] \\ & + \mathrm{KL}(q(\theta;\psi) \| p(\theta)). \end{aligned} \tag{3.3}$$

Because each $q_\theta(\phi_i|D^{(i)})$ depends on ψ via θ (the initialization for the variational parameters before performing K steps of gradient descent), we can also backpropagate through the computation of q via the gradient descent process to compute updates for ψ. Though this backpropagation step requires computing the Hessian, it can be done efficiently with fast Hessian-vector products, which have been used in past work involving backpropagation through gradient updates [59, 47]. This corresponds to learning a global initialization of the variational parameters such that a few steps

of gradient descent will produce a good local variational distribution for any given dataset.

We assume a setting where $M >> N$, i.e. we have many more episodes than data points within each episode. Accordingly, we are most interested in quantifying uncertainty within a given episode and desire accurate predictive uncertainty in $q_\theta(\phi_i|D^{(i)})$. We assume that uncertainty in the global latent variables θ should be low due to the large number of episodes, and therefore use a point estimate for the global latent variables, letting $q(\theta;\psi)$ be a dirac delta function $q(\theta) = \mathbb{1}\{\theta = \theta^*\}$. This removes the need for global variational parameters ψ and simplifies our optimization problem to

$$\arg\min_\theta \left[\sum_{i=1}^{M} -\mathbb{E}_{q_\theta(\phi_i|D^{(i)})}\left[\log p(D^{(i)}|\phi_i)\right] + \mathrm{KL}(q_\theta(\phi_i|D^{(i)})\|p(\phi_i|\theta))\right] + \mathrm{KL}(q(\theta)\|p(\theta)), \tag{3.4}$$

where θ^* is the solution to the above optimization problem. Note that $\mathrm{KL}(q(\theta)\|p(\theta))$ term can be computed even when $q(\theta) = \mathbb{1}\{\theta = \theta^*\}$, as $\mathrm{KL}(q(\theta)||p(\theta)) = -\log p(\theta^*)$.

3.3.2 Amortized Variational Inference using only Training Set

As mentioned in the previous chapter, In the few-shot learning problem we must consider train and test splits for each dataset in each episode. Using notation from previous chapter, we will call the training examples in each dataset D_{train} and the test examples in each dataset D_{test} . Thus, $D^{(i)} = D^{(i)}_{train} \cup D^{(i)}_{test}$, where $D^{(i)}_{train} = \left\{\left(\mathbf{x}^{(i,j)}_{tr}, \mathbf{y}^{(i,j)}_{tr}\right)\right\}_{j=1}^{N}$ and $D^{(i)}_{test} = \left\{\left(\mathbf{x}^{(i,j)}_{test}, \mathbf{y}^{(i,j)}_{test}\right)\right\}_{j=1}^{N'}$, and the assumption is that during evaluation, we are only given $D^{(i)}_{train}$ to determine our variational distribution $q(\phi_i)$ and measure the performance of the model by evaluating the variational distribution

on corresponding $D^{(i)}_{test}$. In order to match what is done during training and evaluation, we consider a modified version of the objective of (3.4) that incorporates this training and test set split. This means that for each episode i, we only have access to data $D^{(i)}_{train}$ to compute the variational distribution, giving us the following objective:

$$\arg\min_{\theta} \left[\sum_{i=1}^{M} -\mathbb{E}_{q_\theta\left(\phi_i \middle| D^{(i)}_{train}\right)} \left[\log p(D^{(i)}|\phi_i)\right] + \text{KL}\left(q_\theta\left(\phi_i \middle| D^{(i)}_{train}\right) \middle\| p(\phi_i|\theta)\right) \right] + \text{KL}(q(\theta)\|p(\theta)), \tag{3.5}$$

where $q_\theta\left(\phi_i \middle| D^{(i)}_{train}\right) = q\left(\phi_i; SGD_K\left(D^{(i)}_{train}, \theta, \theta\right)\right)$. Note that the objective in this optimization problem still serves as a lower bound to the likelihood of all the episodic data because all that has changed is that we condition the variational distribution q on less information (using only the training set vs using the entire dataset). Conditioning on less information potentially gives us a weaker lower bound for all the training datasets, but we found empirically that the performance during evaluation was better using this type of conditioning since there is no mismatch between how the variational distribution is computed during training vs evaluation.

3.3.3 Application Details

With the objective (3.5) in mind, we now give details on how we implement the specific model. We begin with the distributional forms of the priors and posteriors. The formulation given above is flexible but we consider fully factorized Gaussian distributions for ease of implementation and experimentation. We let $\theta = \{\boldsymbol{\mu_\theta}, \boldsymbol{\sigma^2_\theta}\}$, where $\boldsymbol{\mu_\theta} \in \mathbb{R}^D$ and $\boldsymbol{\sigma^2_\theta} \in \mathbb{R}^D$ represent the mean and variance for each neural network weight, respectively. Then, $p(\phi_i|\theta)$ is

$$p(\phi_i|\theta) = \mathcal{N}(\phi_i; \boldsymbol{\mu_\theta}, \boldsymbol{\sigma^2_\theta}\mathbf{I}),$$

and $q_\theta\left(\phi_i \middle| D_{train}^{(i)}\right)$ is the following:

$$\left\{\boldsymbol{\mu}_{\boldsymbol{\lambda}}^{\boldsymbol{(K)}}, \boldsymbol{\sigma^2}_{\boldsymbol{\lambda}}^{\boldsymbol{(K)}}\right\} = SGD_K\left(D_{train}^{(i)}, \theta, \theta\right)$$
$$q_\theta\left(\phi_i \middle| D_{train}^{(i)}\right) = \mathcal{N}\left(\phi_i; \boldsymbol{\mu}_{\boldsymbol{\lambda}}^{\boldsymbol{(K)}}, \boldsymbol{\sigma^2}_{\boldsymbol{\lambda}}^{\boldsymbol{(K)}}\right).$$

We let the prior $p(\theta)$ be

$$p(\theta) = \mathcal{N}(\boldsymbol{\mu}; \mathbf{0}, \mathbf{I}) \cdot \prod_{l=1}^{D} \text{Gamma}(\tau_l; a_0, b_0),$$

where $\tau_l = \frac{1}{\boldsymbol{\sigma^2}_l}$ is the precision and a_0 and b_0 are the alpha and beta parameters for the gamma distribution. Note that with the defined distributions, the SGD process here corresponds to performing Bayes by Backprop [11] with the learned prior $p(\phi_i|\theta)$.

Optimization of (3.5) is done via mini-batch gradient descent, where we average gradients over multiple episodes at a time. The pseudo-code for training and evaluation is given in Algorithms 1 and 2 in the appendix. The KL-divergence terms are calculated analytically whereas the expectations are approximated by averaging over a number of samples from the approximate posterior, as has been done in previous work [49, 11]. The gradient computed for this approximation naively can have high variance, which can significantly harm the convergence of gradient descent [50]. Variance reduction is particularly important to the performance of our model as we perform stochastic optimization to obtain the posterior $q_\theta\left(\phi|D_{train}^{(i)}\right)$ at evaluation-time also. Previous work has explored reducing the variance of gradients involving stochastic neural networks, and we found this crucial to training the networks we use. Firstly, we use the Local Reparametrization Trick (LRT) [50] for fully-connected layers and Flipout [88] for convolutional layers to generate fully-independent (or close to fully-independent in the case of Flipout) weight samples for each example. Secondly, we can easily generate multiple weight samples in the few-shot learning setting simply

by replicating the data in each episode since we only have a few examples per class making up each episode. Both LRT and Flipout increase the operations required in the forward pass by 2 because they require two weight multiplications (or convolutions) rather than one for a normal fully-connected or convolutional layer. Replicating the data does not increase the run time too much because the total replicated data still fits on a forward pass on the GPU.

3.4 Related Work

The Bayesian inference perspective of meta-learning [82, 26] casts it as Bayesian inference in a hierarchical graphical model [52]. This approach provides a principled framework to reason about uncertainty. However, hierarchical Bayesian methods once lacked the ability to scale to complex models and large, high-dimensional datasets due to the computational costs of inference. Recent developments in variational inference [49, 11] allow efficient approximate inference with complex models and large datasets. These have been used to scale Bayesian meta-learning using a variety of approaches. Edwards and Storkey [25] infer episode-specific latent variables which can be used as auxiliary inputs for tasks such as classification. As mentioned before, Amit and Meir [1] learn a prior on the weights of a neural network and separate variational posteriors for each task.

Our method is very closely related to Finn et al. [28] and recent work proposing Bayesian variants of MAML. Grant et al. [36] provided the first Bayesian variant of MAML using the Laplace approximation. In concurrent work, Kim et al. [46] and Finn et al. [29] propose Bayesian variants of MAML with different approximate posteriors. Finn et al. [29] approximate MAP inference of the task-specific weights ϕ_i, and maintain uncertainty only in the global model θ. Our paper, however, considers tasks in which it is important to quantify uncertainty in task-specific weights - such

as contextual bandits and few-shot learning. Kim et al. [46] focus on uncertainty in task-specific weights, as we do. They use a point estimate for all layers except the final layer of a deep neural network, and use Stein Variational Gradient Descent to approximate the posterior over the weights in the final layer with an ensemble. This avoids placing Gaussian restrictions on the approximate posterior; however, the posterior's expressiveness is dependant on the number of particles in the ensemble, and memory and computation requirements scale linearly and quadratically in the size of the ensemble, respectively. The linear scaling requires one to share parameters across particles in order to scale to larger datasets.

Moreover, there has been other recent work on Bayesian methods for few-shot learning. Neural Processes achieve Gaussian Process-like uncertainty quantification with neural networks, while being easy to train via gradient descent [32, 33]. However, it has not been demonstrated whether these methods can be scaled to bigger benchmarks like *mini*ImageNet. Gordon et al. [35] adapt Bayesian decision theory to formulate the use of an amortization network to output the variational distribution over weights for each few-shot dataset. Both Kim et al. [46] and Gordon et al. [35] require one to specify the global parameters (those that are shared across all episodes and are point estimates) vs task-specific parameters (those that are specific to each episode and have a variational distribution over them). Our method, however, does not require this distinction a priori and can discover it based on the data itself. For example, in Figure 3.5, which shows the standard deviations of the learned prior, we see that many of the 1st layer convolutional kernels have standard deviations very close to 0, indicating that these weights are essentially shared because there will be a large penalty from the prior for deviating from them in any episode. Not needing to make this distinction makes it more straightforward to apply our model to new problems, like the contextual bandit task we consider.

3.5 Evaluation

We evaluate our proposed model on experiments involving contextual bandits and involving measuring uncertainty in few-shot learning benchmarks. We compare our method primarily against MAML. Unlike our model, MAML is trained by maximum likelihood estimation of the test set given a fixed number of updates on the training set, causing it to often display overconfidence in the settings we consider. For few-shot learning, we additionally compare against Probabilistic MAML [29], a Bayesian version of MAML that maintains uncertainty only in the global parameters.

3.5.1 Contextual Bandits

The first problem we consider is a contextual bandit task, specifically in the form of the wheel bandit problem introduced in Riquelme et al. [71]. The contextual bandit task involves observing a context X_t from time $t = 0, \ldots, n$ and requires the model to select, based on its internal state and X_t, one of the k available actions. Based on the context and the action selected at each time step, a reward is generated. The goal of the model is to minimize the cumulative regret, the difference between the sum of rewards of the optimal policy and the model's policy.

The wheel bandit problem is a synthetic contextual bandit problem with a scalar hyperparameter that allows us to control the amount of exploration required to be successful at the problem. The setup is the following: we consider a unit circle in $\mathbb{R}^2$ split up into 5 areas determined by the hyperparameter δ. At each time step, the agent is given a point $X = (x_1, x_2)$ inside the circle and has to determine which arm to select among $k = 5$ arms. For $\|X\| \leq \delta$ (the low-reward region), the optimal arm is $k = 1$, which gives reward $r \sim \mathcal{N}(1.2, 0.01^2)$. All other arms in this area give reward $r \sim \mathcal{N}(1, 0.01^2)$. For $\|X\| > \delta$, the optimal arm depends on which of the 4 high-reward regions X is in. Each of the 4 regions has an assigned optimal arm that

gives reward $r \sim \mathcal{N}(50, 0.01^2)$, whereas the other 3 arms will give $r \sim \mathcal{N}(1.0, 0.01^2)$ and arm $k = 1$ will always give $r \sim \mathcal{N}(1.2, 0.01^2)$. The difficulty of the problem increases with δ, as it requires increasing amount of exploration to determine where the high-reward regions are located. We refer the reader to Riquelme et al. [71] for visual examples of the problem.

Thompson Sampling [83] is a classic approach to tackling the exploration-exploitation trade-off involved in bandit problems which requires a posterior distribution over reward functions. At each time step an action is chosen by sampling a model from the posterior and acting optimally with respect to the sampled reward function. The posterior distribution over reward functions is then updated based on the observed reward for the action. When the posterior initially has high variance because of lack of data, Thompson Sampling explores more and turns to exploitation only when the posterior distribution becomes more certain about the rewards. The work of Riquelme et al. [71] compares using Thompson Sampling for different models that approximate the posterior over reward functions on a variety of contextual bandit problems, including the wheel bandit.

We use the setup described in Garnelo et al. [33] to apply meta-learning methods to the wheel bandit problem. Specifically, for meta-learning methods there is a pre-training phase in which training episodes consist of randomly generated data across δ values from wheel bandit task. Then, these methods are evaluated using Thompson sampling on problems defined by specific values of δ. We can create a random training episode for pre-training by first sampling M different wheel problems $\{\delta_i\}_{i=1}^{M}, \delta_i \sim \mathcal{U}(0, 1)$, followed by sampling tuples of the form $\{(X, a, r)\}_{j=1}^{N}$ for context X, action a, and observed reward r. As in Garnelo et al. [33], we use $M = 64$ and $N = 562$ (where the training set has 512 items and the test set has 50 items). We then evaluate the trained meta-learning models on specific instances of the wheel bandit problem (determined by setting the δ hyperparameter). Whereas the models in Riquelme et al.

[71] have no prior knowledge to start off with when being evaluated on each problem, meta-learning methods, like our model and MAML, have a chance to develop some sort of prior that they can utilize to get a head start. MAML learns a initialization of the neural network that it can then fine-tune to the given problem data, whereas our method develops a prior over the model parameters that can be utilized to develop an approximate posterior given the new data. Thus, we can straightforwardly apply Thompson sampling in our model using the approximate posterior at each time step whereas for MAML we just take a greedy action at each time step given the current model parameters.

The results of evaluating the meta-learning methods using code made available by Riquelme et al. [71] after the pre-training phase are shown in Table 3.1. We also show results from NeuralLinear, one of the best performing models from Riquelme et al. [71], to display the benefit of the pre-training phase for the meta-learning methods. We vary the number of contexts and consider $n = 80,000$ (which was used in Riquelme et al. [71]) and $n = 2,000$ (to see how the models perform under fewer time steps). We can see that as δ increases and more exploration is required to be successful at the problem, our model has a increasingly better cumulative regret when compared to MAML. Additionally, we notice that this improvement is even larger when considering smaller amount of time steps, indicating that our model converges to the optimal actions faster than MAML. Lastly, in order to highlight the difference between our method and MAML, we visualize the learned prior $p(\phi \mid \theta)$ in Figure 3.2 by showing the expectation and standard-deviation of predicted rewards for specific arms with respect to the prior. We can see that the standard deviation of the central low-reward arm is small everywhere, as there is reward little variability in this arm across δ values. For the high-reward arm in the upper-right corner, we see that the standard deviation is high at the edges of the area in which this arm can give high reward (depending on the sampled δ value). This variation is useful during

δ	**0.5**	**0.7**	**0.9**	**0.95**	**0.99**
$\mathbf{n = 80,000}$					
Uniform	100 $\pm$ 0.08	100 $\pm$ 0.09	100 $\pm$ 0.25	100 $\pm$ 0.37	100 $\pm$ 0.78
NeuralLinear	0.95 $\pm$ 0.02	1.60 $\pm$ 0.03	4.65 $\pm$ 0.18	9.56 $\pm$ 0.36	49.63 $\pm$ 2.41
MAML	**0.20** $\pm$ 0.002	0.34 $\pm$ 0.004	1.02 $\pm$ 0.01	2.10 $\pm$ 0.03	9.81 $\pm$ 0.27
Our Model	0.22 $\pm$ 0.002	**0.29** $\pm$ 0.003	**0.66** $\pm$ 0.008	**1.03** $\pm$ 0.01	**4.66** $\pm$ 0.10
$\mathbf{n = 2,000}$					
Uniform	100 $\pm$ 0.25	100 $\pm$ 0.42	100 $\pm$ 0.79	100 $\pm$ 1.15	100 $\pm$ 1.88
MAML	1.79 $\pm$ 0.04	2.10 $\pm$ 0.04	6.08 $\pm$ 0.47	16.80 $\pm$ 1.30	55.53 $\pm$ 2.18
Our Model	**1.36** $\pm$ 0.03	**1.59** $\pm$ 0.04	**3.51** $\pm$ 0.17	**7.21** $\pm$ 0.41	**35.04** $\pm$ 1.93

Table 3.1: Cumulative regret results on the wheel bandit problem with varying δ values. Results are normalized with the performance of the uniform agent (as was done in Riquelme et al. [71]) and results shown are mean and standard error for cumulative regret calculated across 50 trials

exploration as this is the region in which we would like to target our exploration to figure out what δ value we are faced with in a new problem. MAML is only able to learn the information associated with expected reward values and so is not well-suited for appropriate exploration but can only be used in a greedy manner.

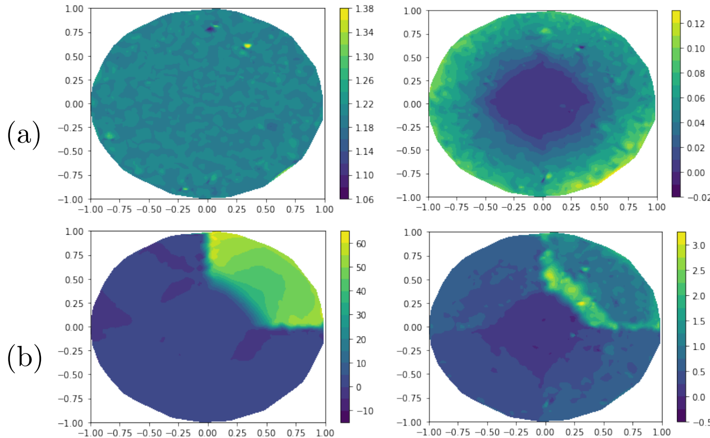


Figure 3.2: Visualization of arm rewards according to prior distribution of our model. (a) expectation and standard-deviation of low-reward arm (computed by sampling weights from the prior) evaluated on points on unit circle. (b) expectation and standard-deviation of one of the high-reward arms computed in same way as for low-reward arm.

3.5.2 Few-shot Learning

We consider two few-shot learning benchmarks: **CIFAR-100** and ***mini*ImageNet**, where both datasets consist of 100 classes and 600 images per class and where CIFAR-100 has images of size 32×32 and *mini*ImageNet has images of size 84×84. We split the 100 classes into separate sets of 64 classes for training, 16 classes for validation, and 20 classes for testing for both of the datasets (using the same split from as the previous chapter for *mini*ImageNet, while using our own for CIFAR-100 as a commonly used split does not exist). For both benchmarks, we use the convolutional architecture used in Finn et al. [28], which consists of 4 convolutional layers, each with 32 filters, and a fully-connected layer mapping to the number of classes on top. For the few-shot learning experiments, we found it necessary to downweight the inner KL term for better performance in our model.

While we focus on predictive uncertainty, we start by comparing classification accuracy of our model compared to MAML. We consider 1-shot, 5-class and 1-shot, 10-class classification on CIFAR-100 and 1-shot, 5-class classification on *mini*ImageNet, with results given in Table 3.2. For both datasets, we compare our model with our own re-implementation of MAML and Probabilistic MAML. Note that the accuracy and associated confidence interval for our implementations for *mini*ImageNet are smaller than the reference implementations because we use a bigger test set for evaluation episodes (15 vs 1 example(s) per class) and we average across more evaluation episodes (1000 vs 600), respectively, compared to Finn et al. [28]. Because we evaluate in a transductive setting [68], the evaluation performance is affected by the test set size, and we use 15 examples to be consistent with previous work. Our model achieves comparable to a little worse on classification accuracy than MAML and Probabilistic MAML on the benchmarks.

To measure the predictive uncertainty of the models, we first compute reliability diagrams [38] across many different test episodes for both models. Reliability dia-

CIFAR-100	1-shot	
	5-class	10-class
MAML (ours)	51.6 $\pm$0.74	36.2 $\pm$0.46
Prob. MAML (ours)	52.8 $\pm$0.75	36.6 $\pm$0.44
Our Model	49.5 $\pm$0.74	35.7 $\pm$0.47

*mini*ImageNet	1-shot, 5-class
MAML (ours)	47.0 $\pm$0.59
Prob. MAML (ours)	47.8 $\pm$0.61
Our Model	45.0 $\pm$0.60

Table 3.2: Few-shot classification accuracies with 95% confidence intervals on CIFAR-100 and *mini*ImageNet.

grams visually measure how well calibrated the predictions of a model are by plotting the expected accuracy as a function of the confidence of the model. A well-calibrated model will have its bars align more closely with the diagonal line, as it indicates that the probability associated with a predicted class label corresponds closely with how likely the prediction is to be correct. We also show the Expected Calibration Error (ECE) and Maximum Calibration Error (MCE) of all models, which are two quantitative ways to measure model calibration [65, 38]. ECE is a weighted average of each bin's accuracy-to-confidence difference whereas MCE is the worst-case bin's accuracy-to-confidence difference. Reliability diagrams and associated error scores are shown in Figure 3.3. We see that across different tasks and datasets, the reliability diagrams and error scores reflect the fact that our model is always better calibrated on evaluation episodes compared to MAML and Probabilitic MAML.

Another way we can measure the quality of the predictive uncertainty of a model is by measuring its confidence on out-of-distribution examples from unseen classes. This tests the model's ability to be uncertain on examples it clearly does not know how to classify. One method to visually measure this is by plotting the empirical CDF of a model's entropies on these out-of-distribution examples [58]. A model represented by a CDF curve that is towards the bottom-right is preferred, as it indicates that the probability of observing a high confidence prediction from the model is low on

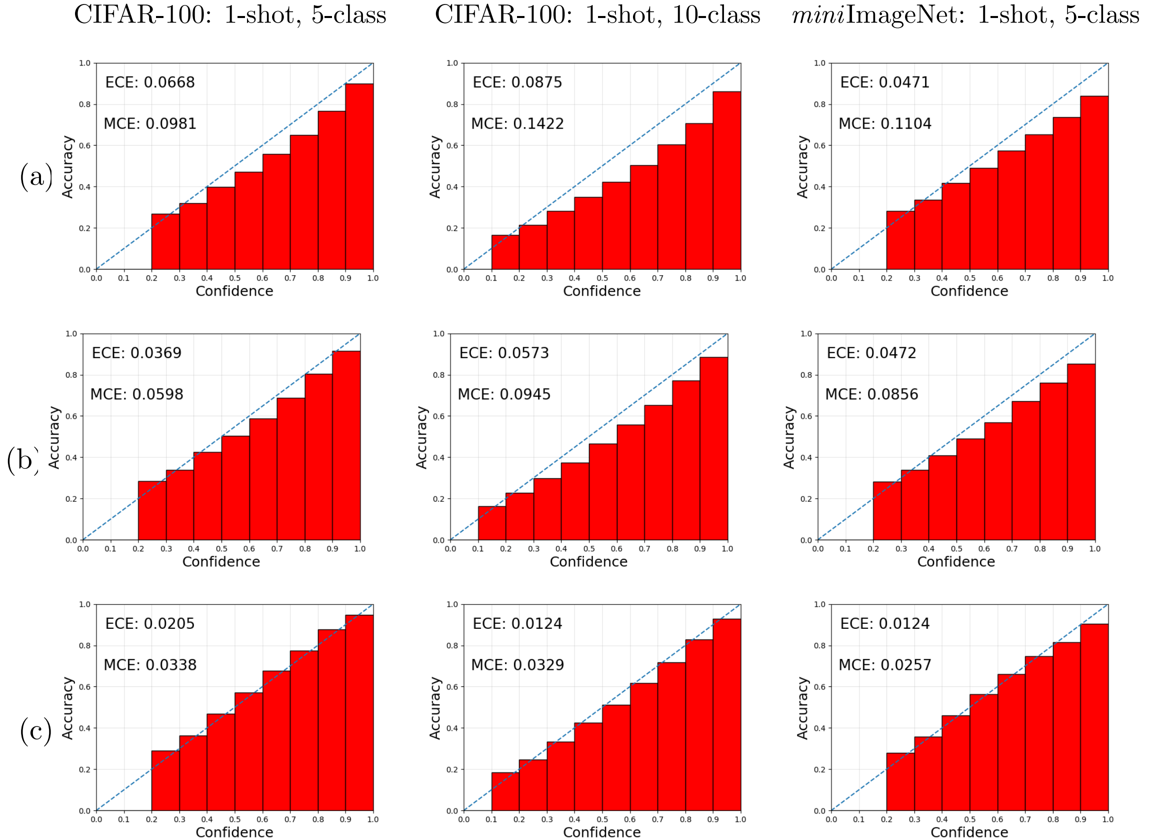


Figure 3.3: Reliability diagrams for MAML and our model on various tasks across datasets. Relibiality diagrams are computed by gathering predicted probabilities for test set examples across many episodes, where the same set of evaluation episodes are used for both models. (a) MAML (b) Probabilistic MAML (c) Our model.

an out-of-distribution example. We can plot the same type of curve in our setting by considering the model's confidence on out-of-episode examples for each test episode. Empirical CDF curves for both MAML-based models and our model are shown in Figure 3.4. We see that in general our model computes better uncertainty estimates than the comparison models, as the probability of a low entropy prediction is always smaller.

Lastly, we visualize the prior distribution $p(\phi|\theta)$ that has been learned in tasks involving deep convolutional networks. We show the standard deviations of randomly selected filters from the first convolutional layer to the last convolutional layer from our CIFAR-100 network trained on 1-shot, 5-class task in Figure 3.5. Interestingly,

the standard deviation of the prior for the filters increases as we go higher up in the network. This pattern reflects the fact that across the training episodes the prior can be very confident about the lower-level filters, as they capture general, useful lower-level features and so do not need to be modified as much on a new episode. The standard deviation for the higher-level filters is higher, reflecting that fact that these filters need to be fine-tuned to the labels present in the new episode. This variation in the standard deviation represents different learning speeds across the network on a new episode, indicating which type of weights are general and which type of weights need to be quickly modified to capture new data.

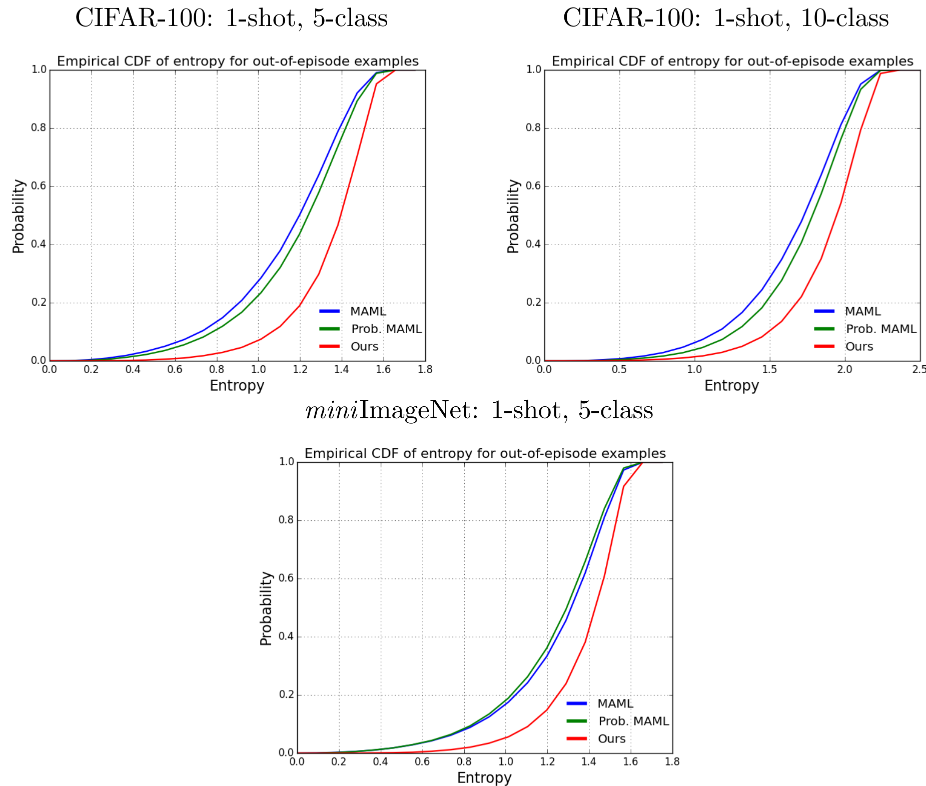


Figure 3.4: Comparison of empirical CDF of entropy of predictive distributions on out-of-episode examples on various tasks and datasets. Data for CDF comes from computing the entropy on out-of-episode examples across many episodes, where out-of-episode examples are generated by randomly sampling classes not belonging to the episode and randomly sampling examples from those classes. The same set of evaluation episodes are used for both models.

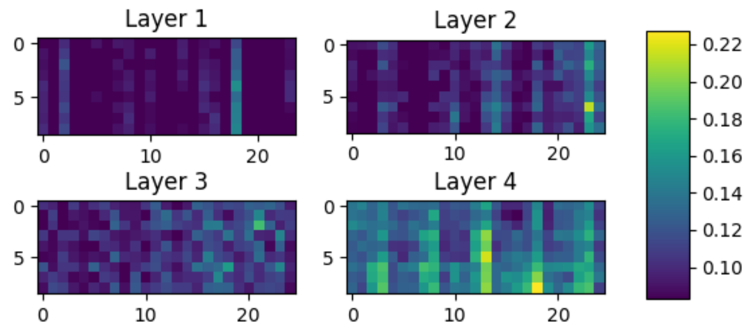


Figure 3.5: Standard deviation of prior for convolutional kernels across layers of network. The x-axis indexes different filters in each layer whereas the y-axis indexes across positions in the 3×3 kernel.

3.6 Conclusion

We described a method to efficiently use hierarchical variational inference to learn a meta-learning model that is scalable across many training episodes and large networks. The method corresponds to learning a prior distribution over the network weights so that a few steps of Bayes by Backprop will produce a good approximate posterior. Through various experiments we show that using a Bayesian interpretation allows us to reason effectively about uncertainty in contextual bandit and few-shot learning tasks. The proposed method is flexible and future work could involve considering more expressive prior (and corresponding posterior) distributions to further improve the uncertainty estimates.

3.7 Appendix

3.7.1 Pseudocode

In algorithms 1 and 2 we give the pseudocode for meta-training and meta-evaluation, respectively. Note that in practice, we do not directly parameterize variance parameters but instead parameterize the standard deviation as the output of the softplus function as was done in Blundell et al. [11] so that it is always non-negative.

Algorithm 2 Meta-training

Input: Number of update steps K, Number of total episodes M, Inner learning rate α, Outer learning rate β

1: Initialize $\theta = \left\{\boldsymbol{\mu}_\theta, \boldsymbol{\sigma^2_\theta}\right\}$

2: $p(\theta) = \mathcal{N}(\boldsymbol{\mu}; \mathbf{0}, \mathbf{I}) \cdot \prod_{l=1}^{D} \text{Gamma}(\tau_l; a_0, b_0)$

3: **for** $i = 1$ to M **do**

4: $\quad D^{(i)} = \left\{D^{(i)}_{train}, D^{(i)}_{test}\right\}$

5: $\quad \boldsymbol{\mu_\lambda^{(0)}} \leftarrow \boldsymbol{\mu}_\theta;\ \boldsymbol{\sigma^2{}_\lambda^{(0)}} \leftarrow \boldsymbol{\sigma^2_\theta}$

6: $\quad$ **for** $k = 0$ to $K - 1$ **do**

7: $\quad\quad \lambda^{(k)} \leftarrow \left\{\boldsymbol{\mu_\lambda^{(k)}}, \boldsymbol{\sigma_\lambda^{(k)}}\right\}$

8: $\quad\quad \boldsymbol{\mu_\lambda^{(k+1)}} \leftarrow \boldsymbol{\mu_\lambda^{(k)}} - \alpha \nabla_{\boldsymbol{\mu_\lambda^{(k)}}} \mathcal{L}_{D^{(i)}_{train}}\left(\lambda^{(k)}, \theta\right)$

9: $\quad\quad \boldsymbol{\sigma^2{}_\lambda^{(k+1)}} \leftarrow \boldsymbol{\sigma^2{}_\lambda^{(k)}} - \alpha \nabla_{\boldsymbol{\sigma^2{}_\lambda^{(k)}}} \mathcal{L}_{D^{(i)}_{train}}\left(\lambda^{(k)}, \theta\right)$

10: $\quad$ **end for**

11:

12: $\quad \lambda^{(K)} \leftarrow \left\{\boldsymbol{\mu_\lambda^{(K)}}, \boldsymbol{\sigma^2{}_\lambda^{(K)}}\right\}$

13: $\quad q(\theta) = \mathbb{1}\{\boldsymbol{\mu} = \boldsymbol{\mu}_\theta\} \cdot \mathbb{1}\{\boldsymbol{\sigma^2} = \boldsymbol{\sigma^2_\theta}\}$

14: $\quad \boldsymbol{\mu_\theta} \leftarrow \boldsymbol{\mu_\theta} - \beta \nabla_{\boldsymbol{\mu_\theta}} \left[\mathcal{L}_{D^{(i)}}(\lambda^{(K)}, \theta) + \frac{1}{M}\text{KL}(q(\theta) \| p(\theta))\right]$

15: $\quad \boldsymbol{\sigma^2_\theta} \leftarrow \boldsymbol{\sigma^2_\theta} - \beta \nabla_{\boldsymbol{\sigma^2_\theta}} \left[\mathcal{L}_{D^{(i)}}(\lambda^{(K)}, \theta) + \frac{1}{M}\text{KL}(q(\theta) \| p(\theta))\right]$

16: **end for**

Algorithm 3 Meta-evaluation

Input: Number of update steps K, Dataset $D = \{D_{train}, D_{test}\}$,

Parameters $\theta = \left\{\boldsymbol{\mu}_\theta, \boldsymbol{\sigma^2_\theta}\right\}$,

Inner learning rate α

1: $\boldsymbol{\mu_\lambda^{(0)}} \leftarrow \boldsymbol{\mu}_\theta$; $\boldsymbol{\sigma^2{}_\lambda^{(0)}} \leftarrow \boldsymbol{\sigma^2_\theta}$

2: **for** $k = 0$ to $K-1$ **do**

3: $\quad \lambda^{(k)} \leftarrow \left\{\boldsymbol{\mu_\lambda^{(k)}}, \boldsymbol{\sigma_\lambda^{(k)}}\right\}$

4: $\quad \boldsymbol{\mu_\lambda^{(k+1)}} \leftarrow \boldsymbol{\mu_\lambda^{(k)}} - \alpha \nabla_{\boldsymbol{\mu_\lambda^{(k)}}} \mathcal{L}_{D_{train}} \left(\lambda^{(k)}, \theta\right)$

5: $\quad \boldsymbol{\sigma^2{}_\lambda^{(k+1)}} \leftarrow \boldsymbol{\sigma^2{}_\lambda^{(k)}} - \alpha \nabla_{\boldsymbol{\sigma^2{}_\lambda^{(k)}}} \mathcal{L}_{D_{train}} \left(\lambda^{(k)}, \theta\right)$

6: **end for**

7:

8: $q_\theta\left(\phi \,|\, D_{train}\right) = \mathcal{N}\left(\phi; \boldsymbol{\mu_\lambda^{(K)}}, \boldsymbol{\sigma^2{}_\lambda^{(K)}}\right)$

9: Evaluate D_{test} using $\mathbb{E}_{q_\theta(\phi \,|\, D_{train})}\left[p(D_{test} \,|\, \phi)\right]$

3.7.2 Hyperparameters

Contextual Bandits

	$n = 2,000$	$n = 80,000$
Number of NN Layers	2	2
Hidden Units per Layer	100	100
t_s (mini-batches per training step)	100	100
t_f (frequency of training)	20	100
Optimizer	Adam	Adam
Learning rate	0.001	0.001

Table 3.3: Hyperparameters for contextual bandit experiments. These hyperparameters were used for both MAML and our model when comparing them. Hyperparameters t_f and t_s were used as defined in Riquelme et al. [71] .

Few-Shot Learning

	CIFAR-100	*mini*ImageNet
Inner Learning Rate	0.1	0.1
Outer Learning Rate	0.001	0.001
Gradient steps for q (training)	5	5
Gradient steps for q (evaluation)	10	10
Number of samples to compute expectation for updating q	5	5
Number of samples to compute expectation for outer-loss	2	2
Number of samples to compute expectation for evaluation	10	10
a_0 for hyper-prior	2	1
b_0 for hyper-prior	0.2	0.01

Table 3.4: Hyperparameters for our model for few-shot learning experiments.

Chapter 4

Navigating the Trade-off Between Multi-Task Learning and Multitasking Capability in Deep Neural Networks

4.1 Introduction

Many recent advances in machine learning can be attributed to the ability of neural networks to learn and to process complex representations by simultaneously taking into account a large number of interrelated and interacting constraints - a property often referred to as parallel distributed processing [60]. Here, we refer to this sort of parallel processing as **interactive parallelism**. However, this type of parallelism stands in contrast to the ability of a network architecture to carry out multiple processes independently at the same time. The latter can be referred to as **independent parallelism** and is heavily used, for example, in computing clusters to distribute independent units of computation appropriately so as to speed up the total required

compute time. Most applications of neural networks have exploited the benefits of interactive parallelism [9]. For instance, in the multi-task learning paradigm, learning of a task is facilitated by training a network on various related tasks one after another [16, 20, 44, 45]. This learning benefit has been hypothesized to arise due to the development of shared representation between tasks [4, 16]. However, the capacity of such networks to execute multiple tasks simultaneously[1] (what we call multitasking) has been less explored.

Recent work [63, 64] has hypothesized that the trade-off between these two types of computation is critical to certain aspects of human cognition. Specifically, though interactive parallelism allows for quicker learning and greater generalization via the use of shared representations, it poses the risk of cross-talk, thus limiting the number of tasks that can be executed at the same time (i.e multitasking). The navigation of this trade-off by the human brain may explain why we are able to multitask some tasks in daily life (such as talking while walking) but not others (for example, doing two mental arithmetic problems in our head at the same time). Musslick et al. [64] have shown that this trade-off is also faced by artificial neural networks when trained to perform simple synthetic tasks. This previous work demonstrates both computationally and analytically that the improvement in learning speed through the use of shared representation comes at the cost of limitations in conurrent multitasking.

While these studies were informative, they were limited to shallow networks and simple task environments. Moreover, an important question that arises from this work remains unanswered: how would an agent optimally trade-off the efficiency of multi-task learning against multitasking capability? In this chapter, we: (a) show that this trade-off also arises in deep convolutional networks used to learn more complex tasks; (b) demonstrate that this trade-off can be managed by using single-task vs multitask training to control whether representations are shared or not; (c) propose

[1]Here we refer to the simultaneous execution of multiple tasks in a single feed-forward pass.

and evaluate a meta-learning algorithm that can be used by a network to regulate its training and manage the trade-off between multi-task learning and multitasking in an environment with unknown serialization costs.

4.2 Background: Trade-off Between Multi-task Learning and Multitasking Capability

4.2.1 Definition of Tasks and Multitasking

Consider an environment in which there are multiple stimulus input dimensions (e.g. corresponding to different sensory modalities) and multiple output dimensions (corresponding to different response modalities). Given an input dimension I (e.g. an image) and an output dimension O (e.g. object category) of responses, a task $T : I \rightarrow O$ represents a mapping between the two (e.g. mapping set of images to set of object categories), such that the mapping is independent of any other. Thus, given N different input dimensions and K possible output dimensions, we have a total of NK possible tasks that our network can learn to perform. Finally, multitasking refers to the simultaneous execution of multiple tasks i.e. within one forward-pass from the inputs to the outputs of a network.

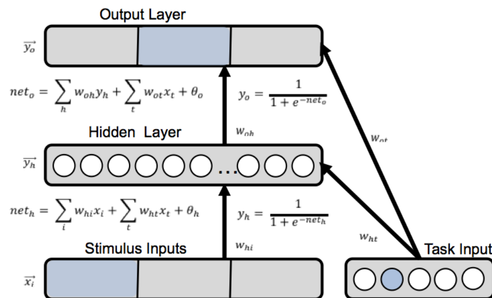


Figure 4.1: Neural network architecture from [63].

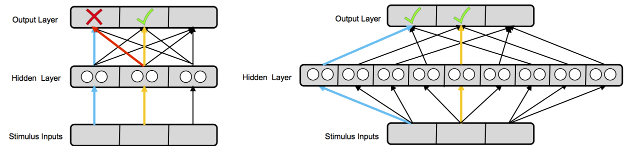


Figure 4.2: Network structure for minimal basis set (left) and tensor product (right) representations and the effects of multitasking in each.

4.2.2 Processing Single and Multiple Tasks Based on Task Projections

We focus on a network architecture that was been used predominantly in previous work [19, 13, 64] (shown in Figure 4.1). Here, in addition to the set of stimulus inputs, there is also an input dimension to indicate which task the network should perform. This task vector is projected to the hidden units and output units using learned weights. The hidden unit task projection biases the hidden representation to calculate the specific representation for each task, whereas the output unit projection biases the outputs to only allow the output that is relevant for the task. The functional role of the task layer is inspired by the notion of cognitive control in psychology and neuroscience, that is, the ability to flexibly guide information processing according to current task goals [79, 70, 19]. Assuming that the task representations used to specify different tasks are orthogonal to one another (e.g., using a one hot code for each), then multitasking can be specified by a superposition (sum) of the representations for the desired tasks on the task input layer. The weights learned for the projections from the task input units to units in the hidden layers, together with those learned within the rest of the network, co-determine what type of representation (shared or separate) the network uses.

4.2.3 Minimal Basis Set vs Tensor Product Representations

Previous work [27, 63, 64] has established that, in the extreme, there are two ways that different tasks can be represented in the hidden layer of a two-layer network. The first representational scheme is the *minimal basis set* (shown on the left in Figure 4.2), in which all tasks that rely on the same input encode the input in the same set of hidden representations. The second scheme is the *tensor product* (shown on the right in Figure 4.2), in which each task separately encodes its input in its own set of hidden

representations. Thus, the minimal basis set maximally shares representations across tasks whereas the tensor product uses separate representations for each task.

These two representational schemes pose a fundamental trade-off. The minimal basis set provides a more efficient encoding of the inputs, and allows for faster learning of the tasks because of the sharing of information across tasks. However, it prohibits executing more than one task at a time (i.e. any multitasking). This is because, with the minimal basis set, attempting to execute two tasks concurrently causes the implicit execution of other tasks because of the representational sharing between tasks. In contrast, while the tensor product network scheme is less compact, multitasking is possible since each task is encoded separately in the network, so that there is no cross-talk from any other task that is not being executed (see Figure 4.2 for an example of multitasking and its effects in both types of networks). However, learning the tensor product representation takes longer since it cannot exploit the sharing of representations across tasks.

In order to control what type of the representation the networks learns (primarily via the task projection weights), we can vary the type of task-processing we train the network on. Single-task training (what is referred to in the literature as multi-task training), which involves training on tasks one after another, induces shared representations whereas multitask training, which involves training on multiple tasks concurrently, produces separate representations. The reason this occurs is that using shared representations when multitasking causes interference and thus error in task execution. In order to minimize this error and the cross-talk that is responsible for it, the network learns task projection weights that lead to separate representations for the tasks. In single-task training, there is no such pressure, as there is no potential for interference when only executing one task at a time, and so the network can use shared representations. Thus, the crux of the trade-off is that single-task training leads us to learn the tasks faster but doesn't allow us to multitask after, whereas

explicit multitask training leads to slower learning but allows us to multitask. The above conclusions were established both theoretically and experimentally for shallow networks with one hidden-layer trained to perform simple synthetic tasks [63, 64]. Below, we report results suggesting that they generalize to deep neural networks trained on more complex tasks.

4.3 Meta-Learning for Optimal Agent

The trade-off associated with the use of shared representations begs the following question: how can an agent manage it to be successful in an environment with unknown properties. That is, how does an agent decide whether to pursue single-task or multitask training in a new environment so that it can learn the tasks most efficiently while maximizing the rewards it receives?

Suppose an agent must learn how to optimize its performance on a given set of tasks in an environment over τ trials. At trial t, for each task, the agent receives a set of inputs $\mathbf{X} = \{\mathbf{x}_k\}_{k=1}^{K}$ and is expected to produce the correct labels $\mathbf{Y} = \{\mathbf{y}_k\}_{k=1}^{K}$ corresponding to the task. Assuming that each of the tasks is classification-based, the agent's reward for the task is its accuracy on the task inputs i.e. $R_t = \frac{1}{K}\sum_{k=1}^{K} \mathbb{1}_{\hat{\mathbf{y}}_k = \mathbf{y}_k}$ where $\hat{\mathbf{y}}_k$ is the predicted label by the agent for each input $\mathbf{x}_k$. On each trial, the agent must perform all the tasks, and it can choose to do so either serially (i.e. by single-tasking) or simultaneously (i.e. by multitasking). After completion of the task and observation of the rewards, the agent also receives the correct labels $\mathbf{Y}$ for the tasks in order to train itself to improve task performance. Finally, assume that the agent's performance is measured across these trials through the entire course of learning and its goal is to maximize the sum of these rewards across all tasks.

To encode the time cost of single-tasking execution, we assume the environment has some unknown *serialization* cost c that determines the cost of performing tasks

serially i.e. one at a time. We assume that the reward for each task when done serially is $\frac{R_t}{1+c}$ where R_t is the reward as defined before. The serialization cost therefore discounts the reward in a multiplicative fashion for single-tasking. We assume that $0 \leq c \leq 1$ so that $c = 0$ indicates there is no cost enforced for serial performance whereas $c = 1$ indicates that the agent receives half the reward for all the tasks by performing them in sequence. Note that the training strategy the agent picks, not only affects the immediate rewards it receives, but also affects the future rewards, as it influences how effectively the agent learns the tasks to improve its rewards in the future. Thus, depending on the serialization cost, the agent may receive lower reward in this way for picking single-tasking but gains a benefit in learning speed that may or may not make up for it over the course of the entire learning episode. This question is at the heart of the trade-off the agent must navigate to make the optimal decision. We note that this is one simple but intuitive way to encode the cost of doing tasks in serial fashion but other mechanisms could also be designed.

4.3.1 Optimal Bayesian Agent

We assume that, on each trial, the agent has the choice between two training strategies to execute and learn the given tasks - via single-tasking or via multitasking. The method we describe involves, on each trial, the agent modeling the reward dynamics under each training strategy for each task and picking the strategy that is predicted to give the highest discounted total future reward across all tasks. To model the reward progress under each strategy, we first define the reward function for each strategy. The reward function $f_{A,i}(t)$ gives the reward for a task i under strategy A assuming strategy A has been selected t times. The reward function captures the effects of both the strategy's learning dynamics and unknown serialization cost (if it exists for the strategy). Here, $A \in \{S, M\}$ where S represents the single-tasking strategy and M represents the multitasking strategy.

We can use the reward function to get the reward for a task i at trial t' when selecting strategy A. Let $a_1, a_2, \dots, a_{t'-1}$ be the strategies picked at each trial until trial t'. Then,

$$R_{t'}^{(A,i)} = f_{A,i}\left(\sum_{t=1}^{t'-1} \mathbb{1}_{a_t=A}\right).$$

is the reward for task i at trial t' assuming we pick strategy A.

Given the reward for each task, the agent can get the total discounted future reward for a strategy A from trial t' onward assuming we repeatedly select strategy A:

$$R_{\geq t'}^{(A)} = \sum_{t=t'}^{\tau} \mu(t)\left(\sum_{i=1}^{N} R_t^{(A,i)}\right),$$

where $\mu(t)$ is the temporal discounting function, N is the total number of tasks, and τ is the total number of trials the agent has to maximize its reward on the tasks.

We now discuss how the agent maintains its estimate of each strategy's reward function for each task. The reward function is modeled as a sigmoidal function, the parameters of which are updated on each trial. Specifically, for a strategy A and task i, using parameters $\theta_{A,i} = \{w_1, b_1, w_2, b_2\}$, we model the reward function as

$$f_{A,i}(t) = \sigma(w_2 \cdot \sigma(w_1 \cdot t + b_1) + b_2).$$

We place a prior over the parameters $p(\theta_{A,i})$ and need to compute the posterior at each trial t' over the parameters $p(\theta_{A,i}|D_{t'})$ where $D_{t'}$ is the observed rewards until trial t' using strategy A on task i. Because the exact posterior is difficult to compute, we calculate the approximate posterior $q(\theta_{A,i}|D_{t'})$ using variational inference [87]. Specifically, we use Stein variational gradient descent (SVGD) [57], which is a deterministic variational inference method that approximates the posterior using a set of particles that represent samples from the approximate posterior. The benefit of using SVGD is that it allows one to select the appropriate number of particles used

so as to increase the complexity of the approximate posterior, while ensuring that the time it takes to compute this approximation is feasible.

At each trial t', the agent needs to use its estimate of the total discounted future reward for single-tasking and multitasking ($R^{(S)}_{\geq t'}$ and $R^{(M)}_{\geq t'}$, respectively) to decide which strategy to use. This can be thought of as a two-armed bandit problem, in which the agent needs to adequately explore and exploit to decide which arm, or strategy, is better. Choosing the single-tasking training regimen may give initial high reward (because of the learning speed benefit) but choosing multitasking may be the better long-term strategy because it does not suffer from any serialization cost. Thompson sampling [83, 17] is an elegant solution to the explore-exploit problem, involving sampling from the posterior over the parameters and taking decisions greedily with respect to the sample. It provides initial exploration as the posterior variance tends to be large at the start because of a lack of data and then turns to exploitation when the posterior becomes more confident due to seeing enough data. On each trial, we use Thompson sampling to pick between the training strategies by sampling from the approximate posterior over parameters for each strategy, calculating the total discounted future reward for each strategy according to the sampled parameters, and picking the strategy corresponding to the higher reward. Note that in practice we do not re-estimate the posterior in each trial (as one additional reward will not change the posterior significantly) but instead do it periodically when enough new rewards for a strategy have been observed.

4.4 Related Work

This chapter considers ideas that sit at the intersection of cognitive psychology, neuroscience and machine learning. The human ability to execute some tasks concurrently but not others is a puzzle that any explanation of human cognition must address. The

"multiple-resource" hypothesis explains this phenomenon by suggesting that multitasking limitations arise due to cross-talk caused by use of the same local resources [61, 66, 73] and that a control system serves to limit the number of active task processes so that this interference is minimized [19, 13]. Thus, the belief is that many task-processes in the brain share representations and that it is the job of cognitive control to limit their concurrent processing. This begs the questions: if shared representations impose such limitations, why would a neural system prefer to use them? Previous work in the context of artificial neural networks has highlighted the benefit of using such shared representations. Multi-task learning has shown that using shared representations between tasks leads to faster learning and better generalization [16]. This suggests a fundamental trade-off - sharing representations leads to a benefit in learning but causes limitations in multitasking ability - which we explore in this paper.

As mentioned before, we build on previous work studying the trade-off of learning speed vs multitasking ability in artificial neural networks [27, 63, 64]. Additionally, our meta-learning algorithm is similar to the one proposed in Sagiv et al. [72]. However, we explicitly use the model's estimate of future rewards under each strategy to also decide how to train the network, whereas the meta-learner in Sagiv et al. [72] was not applied to a neural network's learning dynamics. Instead, the actual learning curve for each strategy A was defined according to pre-defined synthetic function. Our algorithm is thus applied in a much more complex setting in which estimation of each strategy's future rewards directly affects how the network is trained. Furthermore, our method is fully Bayesian in the sense that we utilize uncertainty in the parameter posterior distribution to control exploration vs exploitation via Thompson sampling. Sagiv et al. [72] use logistic regression combined with the ϵ-greedy method to perform this trade-off, which requires hyper-parameters to control the degree of exploration. Lastly, we assume that the serialization penalty is unknown and model

its effects on the future reward of each strategy whereas Sagiv et al. [72] make the simplifying assumption that the penalty is known. Modeling the effects of an unknown serialization penalty on the reward makes the problem more difficult but is a necessary assumption when deploying agents that need to make such decisions in a new environment with unknown properties.

4.5 Experiments

4.5.1 Task Environment

We create a synthetic task environment using AirSim [78], an open-source simulator for autonomous vehicles built on Unreal Engine. To create our synthetic environment, we were motivated by the SnotBot project [69], which involves using drones to monitor the health of whales in the ocean by collecting the mucus exhaled from whales' lungs.

We assume a drone-agent that has two stimulus-inputs: (1) a GPS-input through which we can give the drone location-relevant information; (2) an image-input that corresponds to what the agent can see (e.g. through a camera). The agent also has two outputs: (1) A location-output that corresponds to a location in the input image; (2) An object-output that corresponds to what object the agent believes is present in different situations. Based on the definition of a task as a mapping from one input to one output, this give us the following 4 tasks that the agent can perform:

Task 1 (GPS-localization): given a GPS location, output where in the image the agent has to go to get to that location.

Task 2 (GPS-classification): given a GPS location, output what type of object the agent believes to be present in that area based on its experience.

Task 3 (Image-localization): given an image from agent's input, output the location of the object in the image.

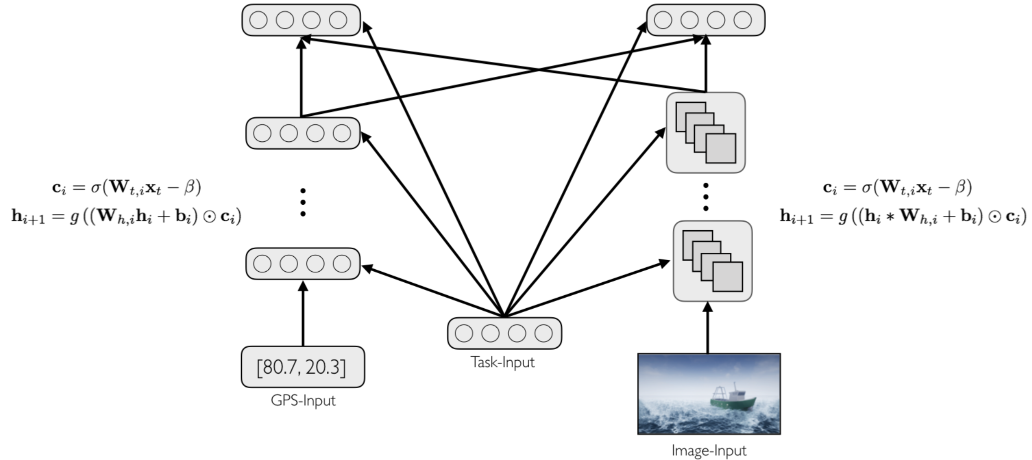


Figure 4.3: Neural network architecture used to learn tasks for simulation data.

Task 4 (Image-classification): given an image from agent's input, output what type of object is present in the image.

Using AirSim, we simulate an ocean-based environment with a set of different possible objects (such as whales, dolphins, orcas, and boats). We create training examples for the agent by randomizing the location of the agent within the environment, the type of object present in the agent's image input, and the GPS location provided to the agent. Thus, each training instance contains a set of randomized inputs and a label for each of the tasks with respect to the specific inputs. The agent can execute each task using either single-tasking (one after another) or multitasking (in which it can execute Tasks 1 and 4 together or Tasks 2 and 3 together).

4.5.2 Neural Network Architecture

The GPS-input is processed using a single-layer neural network, whereas the image-input is processed using a multi-layer convolutional neural network. The encoded inputs are then mapped via fully-connected layers to each output. We allow the task input to modify each hidden, or convolutional, layer using a learned projection of the task input specific to each layer.

More formally, the task-specific projection for the ith layer $\mathbf{c}_i$ is computed using a matrix multiplication with learned task projection matrix $\mathbf{W}_{t,i}$ and task-input $\mathbf{x}_t$, followed by a sigmoid:

$$\mathbf{c}_i = \sigma(\mathbf{W}_{t,i}\mathbf{x}_t - \beta),$$

where β is a positive constant. The subtraction by $\beta > 0$ means that task representations are by default "off". For a fully-connected layer, the task projection $\mathbf{c}_i$ modifies the hidden units for the ith layer $\mathbf{h}_i$ through multiplicative gating to compute the hidden units $\mathbf{h}_{i+1}$:

$$\mathbf{h}_{i+1} = g\left((\mathbf{W}_{h,i}\mathbf{h}_i + \mathbf{b}_i) \odot \mathbf{c}_i\right),$$

where $\mathbf{W}_{h,i}$ and $\mathbf{b}_i$ are the typical weight matrix and bias for the fully-connected layer, and g is the non-linearity. For the hidden units, we let g be the rectified linear activation function (ReLU) whereas for output units it is the identity function. Similarly, for a convolutional layer the feature maps $\mathbf{h}_{i+1}$ are computed from $\mathbf{h}_i$ as:

$$\mathbf{h}_{i+1} = g\left((\mathbf{h}_i * \mathbf{W}_{h,i} + \mathbf{b}_i) \odot \mathbf{c}_i\right),$$

where $\mathbf{W}_{h,i}$ is now the convolutional kernel. Note that we use multiplicative biasing via the task projection whereas previous work [63, 64] used additive biasing. We found multiplicative biasing to work better for settings in which the task projection matrix needs to be learned. A visual example of the network architecture is shown in Figure 4.3.

Training in this network occurs in the typical supervised way with some modifications. To train for a specific task, we feed in the stimulus-input and associated task-input, and train the network to produce the appropriate label at the output associated with the task. For outputs not associated with the task, we train the network to output some default value. In this chapter, we focus on classification-based tasks for simplicity and so the network is trained via cross-entropy loss computed

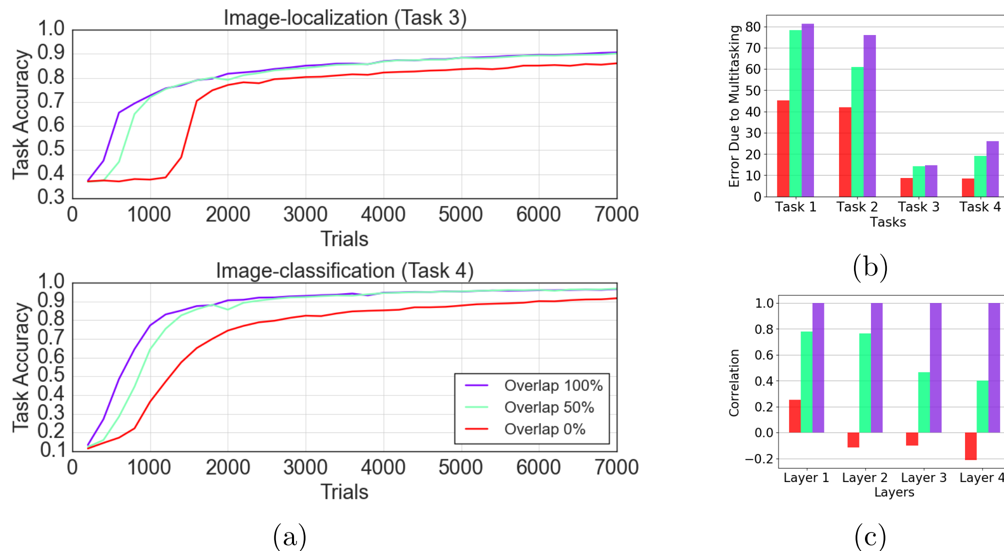


Figure 4.4: Effect of varying representational overlap. (a) Comparison of learning speed of the networks. (b) Comparison of the error in performance when multi-tasking compared to single-tasking averaged over all the data. (c) Correlation of convolutional layer representations between Tasks 3 and Tasks 4 computed using the average representation for each layer across all the data. We study learning speed and correlation for the tasks involving the convolutional network, as those tasks see the biggest benefit from sharing representations.

using the softmax over the network output logits and the true class label. To train the network on multitasking, we feed in the stimulus-input and the associated task-input (indicating which set of tasks to perform concurrently) and train the network on the sum of losses computed at the outputs associated with the set of tasks. Note that we consider the localization-based tasks as classification tasks by outputting a distribution over a set of pre-determined bounding boxes that partition the image space.

4.5.3 Effect of Sharing Representations on Learning Speed & Multitasking Ability

First, we consider the effect of the degree of shared representations on learning speed and multitasking ability. We control the level of sharing in the representations used

by the network by manipulating the task-associated weights $\mathbf{W}_{t,i}$, which implement, in effect, the control signal for each task. The more similar the control signals are for two tasks, the higher the level of sharing because more of the same hidden units are used for the two tasks. We vary $\mathbf{W}_{t,i}$ to manipulate what percent of hidden units overlap for the tasks. Thus, 100% overlap indicates that all hidden units are used by all tasks; 50% overlap indicates that 50% of the hidden units are shared between the tasks whereas the remaining 50% are split to be used independently for each task; and 0% overlap indicates that the tasks do not share any hidden units in a layer. Note that in this experiment, during training task-associated weights are frozen based on the initialization that results in the specific overlap percentage, but the weights in the remainder of the network are free to be learned. Based on previous work [63, 64], we measure the degree of sharing at a certain layer between two task representations by computing the correlation between the mean representation for the tasks, where the mean is computed by averaging the activity at the layer across all training examples for a given task.

The results of the experiment manipulating the level of overlap are shown in Figure 4.4. This shows that as the overlap is increased, sharing of representations across tasks increases (as evidenced by the increase in correlations), which is associated with an increase in the learning speed. However, this is associated with a degradation in the multitasking ability of the network, as a result of the increased interference caused by increased sharing of the representations.

4.5.4 Effect of Single-task vs Multitask Training

Having established that there is a trade-off in using shared representations in the deep neural network architecture described, we now focus on how different training regimens - single-tasking vs multitasking training - impact the representations used by the network and the network's learning speed. We compare different networks

that vary on how much multitasking they are trained to do, from 0%, in which the network is given only single-task training, to 90%, in which the network is trained most of the time to do multitasking. Note that here the task-associated weights $\mathbf{W}_{t,i}$ are initialized to be uniform across the tasks, meaning that the overlap percentage is initially high, and all the weights (including task weights) are then learned based on the training regimen encountered by the network.

The results of this experiment are shown in Figure 4.5. We see that as the network is trained to do more multitasking, the learning speed of the network decreases and the correlation of the task representations also decreases. Because the network is initialized to use a high overlap percentage, we see that a multitasking training regimen clearly forces the network to move away from this initial starting point. The effect is stronger in the later layers, possibly because these layers may contribute more directly to the interference caused when multitasking.

4.5.5 Meta-Learning

Lastly, having established that the trade-off between single-task and multitask training, we now evaluate the meta-learning algorithm to test its effectiveness in optimizing this trade-off. In order to test whether the algorithm is effective at picking the correct training regimen when interacting with an environment with unknown serialization cost, we compare it against always picking single-task or multitask training. We fix the total number of trials for evaluation to be $\tau = 5000$ and evaluate each of the methods on varying serialization costs. For the meta-learner, we average the performances over 15 different runs in order to account for the randomness involved in its sampling choices and measure its confidence interval. We fix the order in which data is presented for the tasks for all options when comparing them. Note that the meta-learner does not know the serialization cost and so has to model its effects as part of the received reward. We create two different environments to induce different

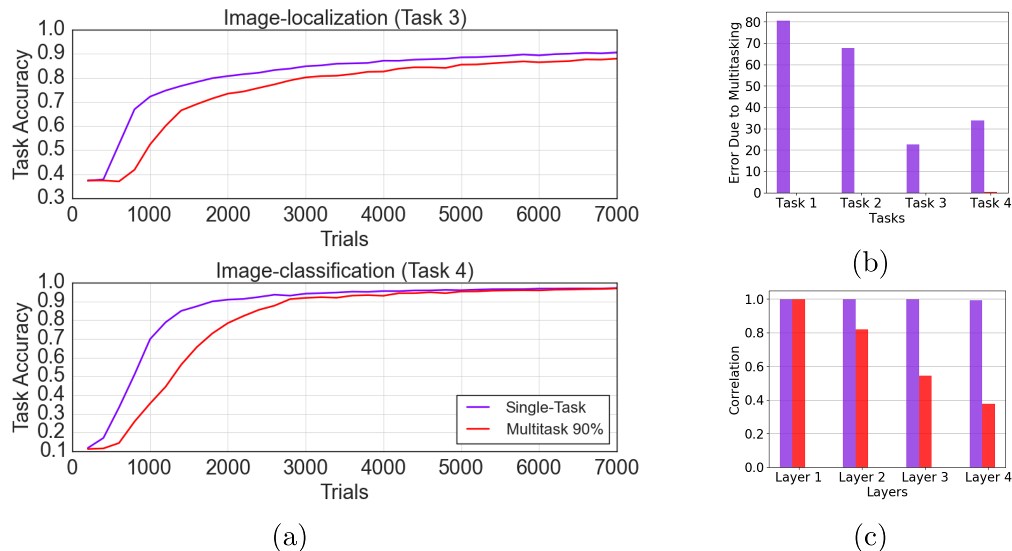


Figure 4.5: Effect of single-task vs multitask training. (a) Comparison of learning speed of the networks. (b) Comparison of the error in performance when multitasking compared to single-tasking averaged over all the data (the lack of a bar indicates no error). (c) Correlation of convolutional layer representations between Tasks 3 and Tasks 4 computed using the average representation for each layer across all the data. We study learning speed and correlation for the tasks involving the convolutional network, as those tasks see the biggest benefit from sharing representations.

trade-offs for rewards between single-tasking and multitasking. The first is the normal environment whereas in the second we add noise to the inputs. Adding noise to the inputs makes the tasks harder and seems to give bigger benefit to the minimal basis set (and single-task training). We hypothesize that this is the case because sharing information across tasks becomes more valuable when noisy information is provided for each task.

The results of this evaluation are shown in Figure 4.6. Figures 4.6a and 4.6b show that the meta-learning algorithm achieves a reward rate that closely approximates the one achieved by that the strategy that yields the greatest value for a given serialization cost. Additionally, note that in the extremes of the serialization cost, the meta-learner seems better at converging to the correct training strategy, while it achieves a lower reward when the optimal strategy is harder to assess. This difference is made even clearer when we study the percent of trials for which the meta-learner

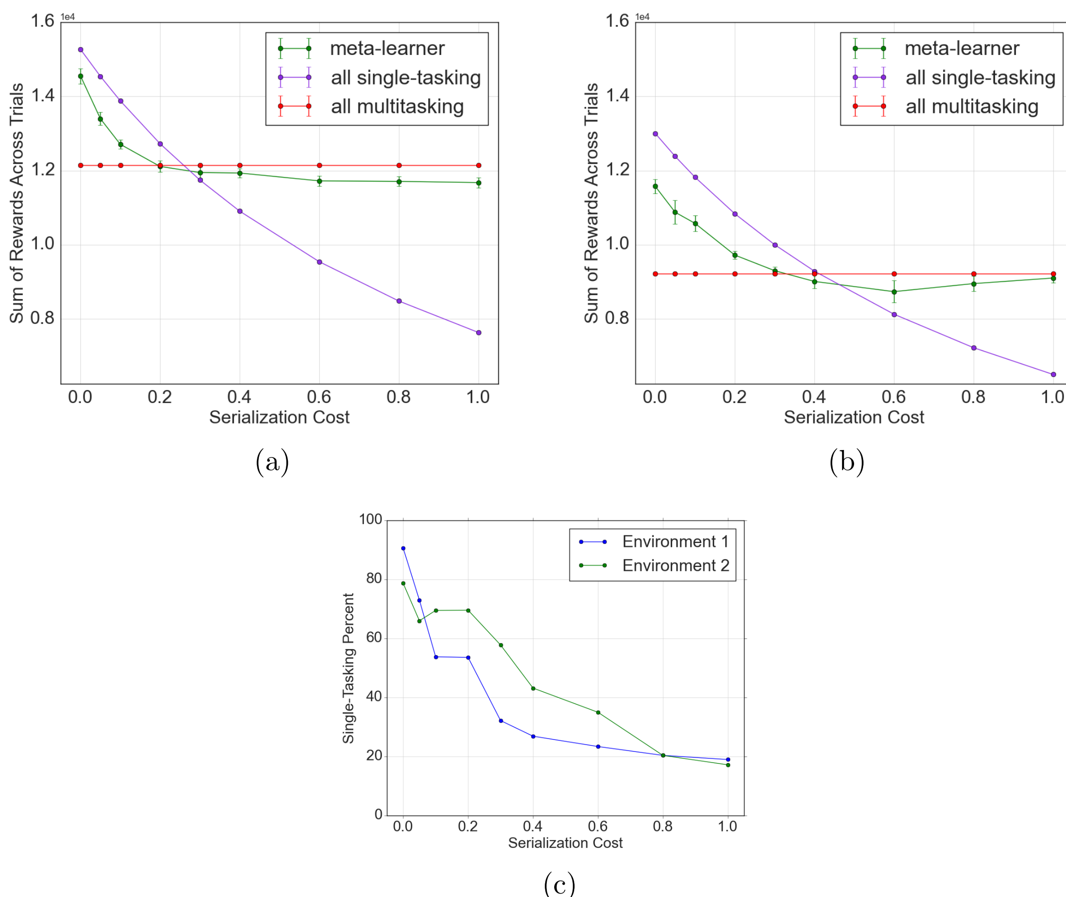


Figure 4.6: Evaluation of meta-learning algorithm. (a) Comparison of all methods on trade-off induced in original environment. (b) Comparison of all methods on trade-off induced in environment where noise is added to inputs. (c) Percent of trials for which meta-learner picks to do single-tasking in both trade-offs.

picks to do single-task training as a function of the serialization cost in Figure 4.6c. We see that the meta-learning algorithm is well-behaved, in that as the serialization cost increases, the percent of trials in which it selects to do single-tasking smoothly decreases. Additionally, at the points at which the optimal strategy is harder to determine, the meta-learner achieves reward closer to the worst strategy because it needs more time to explore and try out both strategies before settling on one.

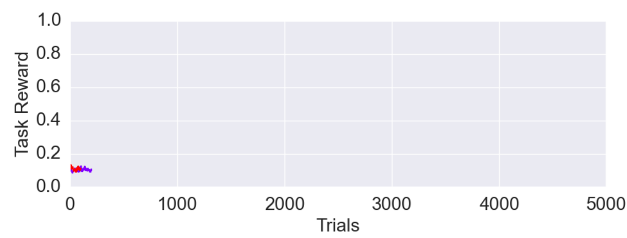
4.6 Discussion

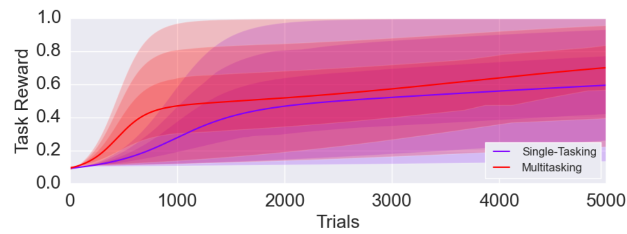
In this chapter we study the trade-off between using shared vs separate representations in deep neural networks. We experimentally show that using shared representations leads to faster learning but at the cost of degraded multitasking performance. We additionally propose and evaluate a meta-learning algorithm to decide which training strategy is best to use in an environment with unknown serialization penalty. A promising direction for future studies involves application of this meta-learner to more complex tasks. As we add more tasks, the potential for interference increases across tasks; however, as tasks become more difficult, the minimal basis set becomes more desirable, as there is even bigger benefit to sharing representations. Furthermore, in this more complicated setting, we would also like to expand our meta-learning algorithm to decide explicitly which set of tasks should be learned so that they can be executed in multitasking fashion and which set of tasks should only be executed one at a time. This requires a more complicated model, as we have to keep track of many possible strategies in order to see what will give the most reward in the future.

4.7 Appendix

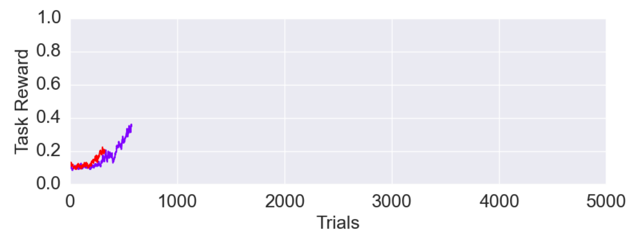
4.7.1 Predictive Distribution of Meta-Learner

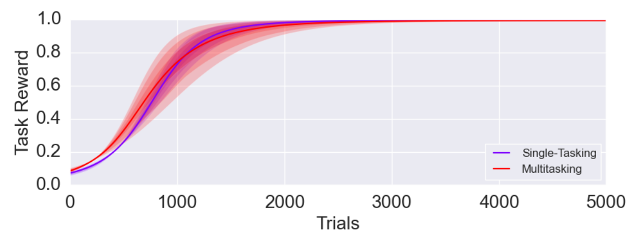
In Figure 4.7, we visualize the predictive distribution of rewards at various trials when varying amount of data has been observed. We see that the predictive distribution is initially uncertain when observing a small amount of rewards for each strategy (which is useful for exploration) and grows certain as more data is observed (which is utilized to be greedy).



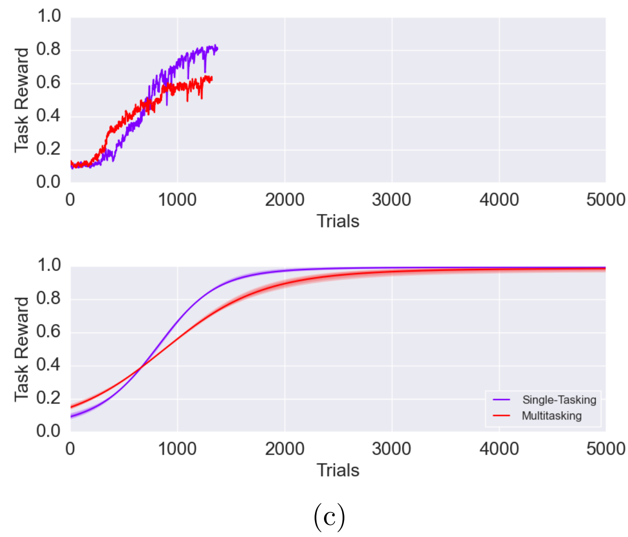


(a)





(b)



(c)

Figure 4.7: Visualization of actual rewards and predictive distribution of rewards for a specific task. Shaded areas correspond to ± 3 standard deviations around mean. For each of (a), (b), and (c), we show the actual rewards accumulated over trials for each strategy (on top) and the predictive distribution over reward data computed using samples from the posterior distribution over parameters for each strategy given the reward data (on bottom).

Chapter 5

Conclusion

In order for machine learning methods to be as flexible and efficient as humans in learning and mastering new tasks, it is critical that they have some ability to build up prior knowledge and apply it appropriately. The goal of this thesis was to study techniques that allow models to learn new tasks efficiently via intelligent application of prior information.

In this thesis, we study meta-learning, or learning-to-learn, methods as a means to improve the efficiency of neural networks with regard to data-samples and processing choices. The first two chapters consider ideas to accumulate prior knowledge (mainly in the form of an initialization of a neural network) and use it effectively so that a model is able to learn concepts in a new dataset with very few labeled examples. In the first work, the adaptation process is gradient descent on a cross-entropy loss whereas in the second it is gradient descent on a variational inference objective. The third chapter proposes a method where, rather than using explicit experience to build up a prior, we use the Thompson sampling algorithm as a sort of prior to solve a problem involving exploitation vs exploration - that of selecting how to train oneself to learn a set of tasks in an environment with unknown properties.

As machine learning methods have begun mastering specific tasks at human-level ability, the question has now arisen of how we can quicken this process of learning so that artificial agents can learn to perform a variety of tasks in a feasible way. Meta-learning methods (like the ones discussed in this thesis and many of the related work) have shown that there is a benefit to setting up a two-level problem, in which the first-level involves learning within a specific experience and the second-level involves learning useful information across different experiences. Though some progress has been made by applying ideas from the older literature on meta-learning in this era of large datasets and powerful compute, there is still a long way to proceed.

Firstly, meta-learning has been, for the most part, successfully applied to problems where the train and test tasks match highly in similarity of distribution [54, 86]. The true test of the ability to build up knowledge and apply it usefully is its effectiveness in situations that differ highly from the ones a model was trained on. Such evaluation requires a model to build on its prior information but also to successfully generalize far beyond what it initially knows. It is certainly indicative of the type of learning humans are able to perform. Benchmarks and datasets that reflect this ability would also be useful in pushing work in this area further.

Furthermore, though parts of this thesis focus on few-shot learning, it would be valuable to study models that can transition successfully from receiving a small number of examples to larger amounts of feedback. More specifically, we need machine learning methods that can improve over time on a certain task and be directly involved in the process of improvement akin to a student's interaction with a teacher. This would involve the model using the initial given training set to decide what new examples should be labeled so as to most improve the model's proficiency at the task. We would like the entire process of data-collection and re-training to be free of any hyper-parameters or manual tuning so that it can be applied straightforwardly and cyclically. Such a method would allow for the efficient use of human labor to quickly

improve model performance on a new task over time. Ideas from the active learning [77, 31], lifelong learning [84], and curriculum learning [8] literature would certainly be useful in pursuing such a problem.

Ultimately, the objective of machine learning is to design truly intelligent artificial agents. Though work in the past decade has shown a lot of promise towards this end, there are still major shortcomings that need to be overcome, some of which we have highlighted here. The hope is that ideas involving meta-learning (examples of which have been presented in this thesis) will be useful in closing the current gap between human and machine intelligence.

Bibliography

[1] Ron Amit and Ron Meir. Meta-learning by adjusting priors based on extended PAC-Bayes theory. In *Proceedings of the 35th International Conference on Machine Learning*, pages 205–214, 2018.

[2] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. *CoRR*, abs/1606.04474, 2016. URL `http://arxiv.org/abs/1606.04474`.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] Jonathan Baxter. Learning internal representations. In *Proceedings of the eighth annual conference on Computational learning theory*, pages 311–320. ACM, 1995.

[5] Samy Bengio, Yoshua Bengio, and Jocelyn Cloutier. On the search for new learning rules for ANNs. *Neural Processing Letters*, 2(4):26–30, 1995.

[6] Yoshua Bengio. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, pages 17–36, 2012.

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

[8] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[9] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[10] Luca Bertinetto, João F. Henriques, Jack Valmadre, Philip H. S. Torr, and Andrea Vedaldi. Learning feed-forward one-shot learners. *CoRR*, abs/1606.05233, 2016. URL `http://arxiv.org/abs/1606.05233`.

[11] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*, pages 1613–1622. JMLR. org, 2015.

[12] Tom Bosc. Learning to learn neural networks.

[13] Matthew M Botvinick, Todd S Braver, Deanna M Barch, Cameron S Carter, and Jonathan D Cohen. Conflict monitoring and cognitive control. *Psychological review*, 108(3):624, 2001.

[14] S CAREY. The child as word learner. *Linguistic Theory and Psychological Reality*, pages 264–293, 1978.

[15] Rich Caruana. Learning many related tasks at the same time with backpropagation. In *Advances in neural information processing systems*, pages 657–664, 1995.

[16] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[17] Olivier Chapelle and Lihong Li. An empirical evaluation of thompson sampling. In *Advances in neural information processing systems*, pages 2249–2257, 2011.

[18] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL `http://arxiv.org/abs/1406.1078`.

[19] Jonathan D Cohen, Kevin Dunbar, and James L McClelland. On the control of automatic processes: a parallel distributed processing account of the stroop effect. *Psychological review*, 97(3):332, 1990.

[20] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[21] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. 2009.

[22] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655, 2014.

[23] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl ^{2}: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[24] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1953048.2021068`.

[25] Harrison Edwards and Amos Storkey. Towards a neural statistician. In *International Conference on Learning Representations*, 2016.

[26] Li Fei-Fei and Pietro Perona. A bayesian hierarchical model for learning natural scene categories. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 524–531. IEEE, 2005.

[27] Samuel F Feng, Michael Schwemmer, Samuel J Gershman, and Jonathan D Cohen. Multitasking versus multiplexing: Toward a normative account of limitations in the simultaneous execution of control-demanding behaviors. *Cognitive, Affective, & Behavioral Neuroscience*, 14(1):129–146, 2014.

[28] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135, 2017.

[29] Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. *arXiv preprint arXiv:1806.02817*, 2018.

[30] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International conference on machine learning*, pages 1050–1059, 2016.

[31] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1183–1192. JMLR. org, 2017.

[32] Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami. Conditional neural processes. In *International Conference on Machine Learning*, pages 1690–1699, 2018.

[33] Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018.

[34] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[35] Jonathan Gordon, John Bronskill, Matthias Bauer, Sebastian Nowozin, and Richard E Turner. Decision-theoretic meta-learning: Versatile and efficient amortization of few-shot learning. *arXiv preprint arXiv:1805.09921*, 2018.

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

[37] Thomas L Griffiths, Nick Chater, Charles Kemp, Amy Perfors, and Joshua B Tenenbaum. Probabilistic models of cognition: Exploring representations and inductive biases. *Trends in cognitive sciences*, 14(8):357–364, 2010.

[38] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330, 2017.

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

[40] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[41] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[42] Sepp Hochreiter, A. Steven Younger, and Peter R. Conwell. Learning to learn using gradient descent. In *IN LECTURE NOTES ON COMP. SCI. 2130, PROC. INTL. CONF. ON ARTI NEURAL NETWORKS (ICANN-2001*, pages 87–94. Springer, 2001.

[43] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL `http://arxiv.org/abs/1502.03167`.

[44] Lukasz Kaiser, Aidan N Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit. One model to learn them all. *arXiv preprint arXiv:1706.05137*, 2017.

[45] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7482–7491, 2018.

[46] Taesup Kim, Jaesik Yoon, Ousmane Dia, Sungwoong Kim, Yoshua Bengio, and Sungjin Ahn. Bayesian model-agnostic meta-learning. *arXiv preprint arXiv:1806.03836*, 2018.

[47] Yoon Kim, Sam Wiseman, Andrew C Miller, David Sontag, and Alexander M Rush. Semi-amortized variational autoencoders. *arXiv preprint arXiv:1802.02550*, 2018.

[48] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

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

[50] Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583, 2015.

[51] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2, 2015.

[52] Daphne Koller, Nir Friedman, and Francis Bach. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[53] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[54] Brenden M Lake, Ruslan Salakhutdinov, Jason Gross, and Joshua B Tenenbaum. One shot learning of simple visual concepts. 2011.

[55] Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *CoRR*, abs/1604.00289, 2016. URL `http://arxiv.org/abs/1604.00289`.

[56] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521 (7553):436, 2015.

[57] Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. In *Advances In Neural Information Processing Systems*, pages 2378–2386, 2016.

[58] Christos Louizos and Max Welling. Multiplicative normalizing flows for variational bayesian neural networks. In *International Conference on Machine Learning*, pages 2218–2227, 2017.

[59] Dougal Maclaurin, David Duvenaud, and Ryan P Adams. Gradient-based hyperparameter optimization through reversible learning. In *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

[60] James L McClelland, David E Rumelhart, and Geoffrey E Hinton. The appeal of parallel distributed processing. *MIT Press, Cambridge MA*, pages 3–44, 1986.

[61] David E Meyer and David E Kieras. A computational theory of executive cognitive processes and multiple-task performance: Part i. basic mechanisms. *Psychological review*, 104(1):3, 1997.

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

[63] Sebastian Musslick, Biswadip Dey, Kayhan Özcimder, Md Mostofa Ali Patwary, Theodore L Willke, and Jonathan D Cohen. Controlled vs. automatic processing: A graph-theoretic approach to the analysis of serial vs. parallel processing in neural network architectures. In *CogSci*, 2016.

[64] Sebastian Musslick, Andrew Saxe, Kayhan Özcimder, Biswadip Dey, Greg Henselman, and Jonathan D Cohen. Multitasking capability versus learning efficiency in neural network architectures. In *CogSci*, 2017.

[65] Mahdi Pakdaman Naeini, Gregory F Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In *Proceedings of the... AAAI Conference on Artificial Intelligence. AAAI Conference on Artificial Intelligence*, volume 2015, page 2901. NIH Public Access, 2015.

[66] David Navon and Daniel Gopher. On the economy of the human-processing system. *Psychological review*, 86(3):214, 1979.

[67] Yurii Nesterov. A method of solving a convex programming problem with convergence rate o (1/k2). 1983.

[68] Alex Nichol and John Schulman. Reptile: a scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2018.

[69] Ocean Alliance. What is a snotbot? `https://shop.whale.org/pages/snotbot`. Accessed: 2019-01-02.

[70] MI Posner and CR Snyder. attention and cognitive control,. In *Information processing and cognition: The Loyola symposium*, pages 55–85, 1975.

[71] Carlos Riquelme, George Tucker, and Jasper Snoek. Deep bayesian bandits showdown: An empirical comparison of bayesian deep networks for thompson sampling. *arXiv preprint arXiv:1802.09127*, 2018.

[72] Yotam Sagiv, Sebastian Musslick, Yael Niv, and Jonathan D Cohen. Efficiency of learning vs. processing: Towards a normative theory of multitasking.

[73] Dario D Salvucci and Niels A Taatgen. Threaded cognition: An integrated theory of concurrent multitasking. *Psychological review*, 115(1):101, 2008.

[74] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy P. Lillicrap. One-shot learning with memory-augmented neural networks. *CoRR*, abs/1605.06065, 2016. URL `http://arxiv.org/abs/1605.06065`.

[75] Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

[76] Jürgen Schmidhuber. A neural network that embeds its own meta-levels. In *Neural Networks, 1993., IEEE International Conference on*, pages 407–412. IEEE, 1993.

[77] Burr Settles. Active learning literature survey. Technical report, University of Wisconsin-Madison Department of Computer Sciences, 2009.

[78] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.

[79] Richard M Shiffrin and Walter Schneider. Controlled and automatic human information processing: II. Perceptual learning, automatic attending and a general theory. *Psychol. Rev.*, 84(2):127, 1977.

[80] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[81] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[82] Joshua Brett Tenenbaum. *A Bayesian framework for concept learning*. PhD thesis, Massachusetts Institute of Technology, 1999.

[83] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

[84] Sebastian Thrun. Is learning the n-th thing any easier than learning the first? In *Advances in neural information processing systems*, pages 640–646, 1996.

[85] Sebastian Thrun. Lifelong learning algorithms. In *Learning to learn*, pages 181–209. Springer, 1998.

[86] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

[87] Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305, 2008.

[88] Yeming Wen, Paul Vicol, Jimmy Ba, Dustin Tran, and Roger Grosse. Flipout: Efficient pseudo-independent weight perturbations on mini-batches. *arXiv preprint arXiv:1803.04386*, 2018.

[89] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[90] Wojciech Zaremba. An empirical exploration of recurrent network architectures. 2015.

[91] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012. URL `http://arxiv.org/abs/1212.5701`.