Meta-Learning with Implicit Gradients

Aravind Rajeswaran[*,1]   **Chelsea Finn**[*,2]   **Sham Kakade**[1]   **Sergey Levine**[2]

[1] University of Washington Seattle   [2] University of California Berkeley

Abstract

A core capability of intelligent systems is the ability to quickly learn new tasks by drawing on prior experience. Gradient (or optimization) based meta-learning has recently emerged as an effective approach for few-shot learning. In this formulation, meta-parameters are learned in the outer loop, while task-specific models are learned in the inner-loop, by using only a small amount of data from the current task. A key challenge in scaling these approaches is the need to differentiate through the inner loop learning process, which can impose considerable computational and memory burdens. By drawing upon implicit differentiation, we develop the implicit MAML algorithm, which depends only on the solution to the inner level optimization and not the path taken by the inner loop optimizer. This effectively decouples the meta-gradient computation from the choice of inner loop optimizer. As a result, our approach is agnostic to the choice of inner loop optimizer and can gracefully handle many gradient steps without vanishing gradients or memory constraints. Theoretically, we prove that implicit MAML can compute accurate meta-gradients with a memory footprint no more than that which is required to compute a single inner loop gradient and at no overall increase in the total computational cost. Experimentally, we show that these benefits of implicit MAML translate into empirical gains on few-shot image recognition benchmarks.

1   Introduction

A core aspect of intelligence is the ability to quickly learn new tasks by drawing upon prior experience from related tasks. Recent work has studied how meta-learning algorithms [51, 55, 41] can acquire such a capability by learning to efficiently learn a range of tasks, thereby enabling learning of a new task with as little as a single example [50, 57, 15]. Meta-learning algorithms can be framed in terms of recurrent [25, 50, 48] or attention-based [57, 38] models that are trained via a meta-learning objective, to essentially encapsulate the learned learning procedure in the parameters of a neural network. An alternative formulation is to frame meta-learning as a bi-level optimization procedure [35, 15], where the "inner" optimization represents adaptation to a given task, and the "outer" objective is the meta-training objective. Such a formulation can be used to learn the initial parameters of a model such that optimizing from this initialization leads to fast adaptation and generalization. In this work, we focus on this class of optimization-based methods, and in particular the model-agnostic meta-learning (MAML) formulation [15]. MAML has been shown to be as expressive as black-box approaches [14], is applicable to a broad range of settings [16, 37, 1, 18], and recovers a convergent and consistent optimization procedure [13].

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

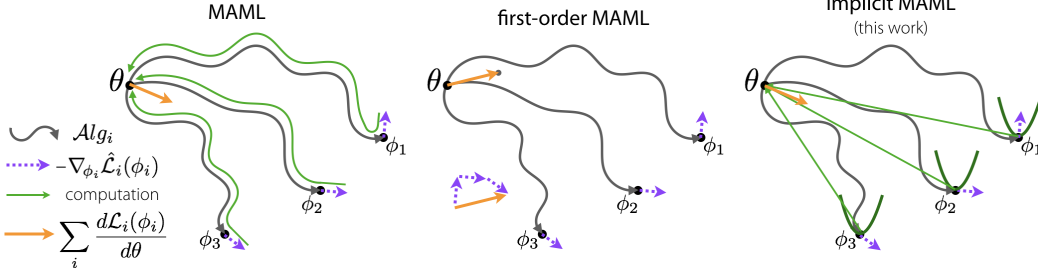[*] Equal contributions. Project page: http://sites.google.com/view/imaml

Figure 1: To compute the meta-gradient $\sum_i \frac{d\mathcal{L}_i(\phi_i)}{d\theta}$, the MAML algorithm differentiates through the optimization path, as shown in green, while first-order MAML computes the meta-gradient by approximating $\frac{d\phi_i}{d\theta}$ as I. Our implicit MAML approach derives an analytic expression for the exact meta-gradient without differentiating through the optimization path by estimating local curvature.

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

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

2 Problem Formulation and Notations

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

2.1 Review of Few-Shot Supervised Learning and MAML

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

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

$$\overbrace{\boldsymbol{\theta}_{\text{ML}}^* := \underset{\boldsymbol{\theta} \in \Theta}{\arg\min}\, F(\boldsymbol{\theta})}^{\text{outer}-\text{level}}, \text{ where } F(\boldsymbol{\theta}) = \frac{1}{M} \sum_{i=1}^M \mathcal{L}\left(\overbrace{\mathcal{A}lg(\boldsymbol{\theta}, \mathcal{D}_i^{\text{tr}})}^{\text{inner}-\text{level}}, \mathcal{D}_i^{\text{test}}\right). \quad (1)$$

2

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

In the case of MAML [15], $\mathcal{A}lg(\boldsymbol{\theta}, \mathcal{D})$ corresponds to one or multiple steps of gradient descent initialized at $\boldsymbol{\theta}$. For example, if one step of gradient descent is used, we have:

$$\boldsymbol{\phi}_i \equiv \mathcal{A}lg(\boldsymbol{\theta}, \mathcal{D}_i^{\mathrm{tr}}) = \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \mathcal{D}_i^{\mathrm{tr}}). \quad \text{(inner-level of MAML)} \tag{2}$$

Typically, α is a scalar hyperparameter, but can also be a learned vector [34]. Hence, for MAML, the meta-learned parameter $(\boldsymbol{\theta}_{\mathrm{ML}}^*)$ has a learned inductive bias that is particularly well-suited for fine-tuning on tasks from $P(\mathcal{T})$ using K samples. To solve the outer-level problem with gradient-based methods, we require a way to differentiate through $\mathcal{A}lg$. In the case of MAML, this corresponds to backpropagating through the dynamics of gradient descent.

2.2 Proximal Regularization in the Inner Level

To have sufficient learning in the inner level while also avoiding over-fitting, $\mathcal{A}lg$ needs to incorporate some form of regularization. Since MAML uses a small number of gradient steps, this corresponds to early stopping and can be interpreted as a form of regularization and Bayesian prior [20]. In cases like ill-conditioned optimization landscapes and medium-shot learning, we may want to take many gradient steps, which poses two challenges for MAML. First, we need to store and differentiate through the long optimization path of $\mathcal{A}lg$, which imposes a considerable computation and memory burden. Second, the dependence of the model-parameters $\{\boldsymbol{\phi}_i\}$ on the meta-parameters $(\boldsymbol{\theta})$ shrinks and vanishes as the number of gradient steps in $\mathcal{A}lg$ grows, making meta-learning difficult. To overcome these limitations, we consider a more explicitly regularized algorithm:

$$\mathcal{A}lg^\star(\boldsymbol{\theta}, \mathcal{D}_i^{\mathrm{tr}}) = \underset{\boldsymbol{\phi}' \in \Phi}{\operatorname{argmin}} \, \mathcal{L}(\boldsymbol{\phi}', \mathcal{D}_i^{\mathrm{tr}}) + \frac{\lambda}{2} \, ||\boldsymbol{\phi}' - \boldsymbol{\theta}||^2. \tag{3}$$

The proximal regularization term in Eq. 3 encourages $\boldsymbol{\phi}_i$ to remain close to $\boldsymbol{\theta}$, thereby retaining a strong dependence throughout. The regularization strength (λ) plays a role similar to the learning rate (α) in MAML, controlling the strength of the prior $(\boldsymbol{\theta})$ relative to the data $(\mathcal{D}_{\mathcal{T}}^{\mathrm{tr}})$. Like α, the regularization strength λ may also be learned. Furthermore, both α and λ can be scalars, vectors, or full matrices. For simplicity, we treat λ as a scalar hyperparameter. In Eq. 3, we use $\star$ to denote that the optimization problem is solved exactly. In practice, we use iterative algorithms (denoted by $\mathcal{A}lg$) for finite iterations, which return approximate minimizers. We explicitly consider the discrepancy between approximate and exact solutions in our analysis.

2.3 The Bi-Level Optimization Problem

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

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

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

$$\begin{aligned} \boldsymbol{\theta}_{\mathrm{ML}}^* &:= \underset{\boldsymbol{\theta} \in \Theta}{\operatorname{argmin}} \, F(\boldsymbol{\theta}), \text{ where } F(\boldsymbol{\theta}) = \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}_i(\mathcal{A}lg_i^\star(\boldsymbol{\theta})), \text{ and} \\ \mathcal{A}lg_i^\star(\boldsymbol{\theta}) &:= \underset{\boldsymbol{\phi}' \in \Phi}{\operatorname{argmin}} \, G_i(\boldsymbol{\phi}', \boldsymbol{\theta}), \text{ where } G_i(\boldsymbol{\phi}', \boldsymbol{\theta}) = \hat{\mathcal{L}}_i(\boldsymbol{\phi}') + \frac{\lambda}{2} \, ||\boldsymbol{\phi}' - \boldsymbol{\theta}||^2. \end{aligned} \tag{4}$$

2.4 Total and Partial Derivatives

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

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

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

3 The Implicit MAML Algorithm

Our aim is to solve the bi-level meta-learning problem in Eq. 4 using an iterative gradient based algorithm of the form $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \, d_{\boldsymbol{\theta}}F(\boldsymbol{\theta})$. Although we derive our method based on standard gradient descent for simplicity, any other optimization method, such as quasi-Newton or Newton methods, Adam [28], or gradient descent with momentum can also be used without modification. The gradient descent update be expanded using the chain rule as

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \, \frac{1}{M} \sum_{i=1}^{M} \frac{d\mathcal{A}lg_i^{\star}(\boldsymbol{\theta})}{d\boldsymbol{\theta}} \, \nabla_{\boldsymbol{\phi}}\mathcal{L}_i(\mathcal{A}lg_i^{\star}(\boldsymbol{\theta})). \tag{5}$$

Here, $\nabla_{\boldsymbol{\phi}}\mathcal{L}_i(\mathcal{A}lg_i^{\star}(\boldsymbol{\theta}))$ is simply $\nabla_{\boldsymbol{\phi}}\mathcal{L}_i(\boldsymbol{\phi})\,|_{\boldsymbol{\phi}=\mathcal{A}lg_i^{\star}(\boldsymbol{\theta})}$ which can be easily obtained in practice via automatic differentiation. For this update rule, we must compute $\frac{d\mathcal{A}lg_i^{\star}(\boldsymbol{\theta})}{d\boldsymbol{\theta}}$, where $\mathcal{A}lg_i^{\star}$ is implicitly defined as an optimization problem (Eq. 4), which presents the primary challenge. We now present an efficient algorithm (in compute and memory) to compute the meta-gradient..

3.1 Meta-Gradient Computation

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

Lemma 1. *(Implicit Jacobian) Consider* $\mathcal{A}lg_i^{\star}(\boldsymbol{\theta})$ *as defined in Eq. 4 for task* $\mathcal{T}_i$. *Let* $\boldsymbol{\phi}_i = \mathcal{A}lg_i^{\star}(\boldsymbol{\theta})$ *be the result of* $\mathcal{A}lg_i^{\star}(\boldsymbol{\theta})$. *If* $\left(\boldsymbol{I} + \frac{1}{\lambda}\nabla_{\boldsymbol{\phi}}^2\hat{\mathcal{L}}_i(\boldsymbol{\phi}_i)\right)$ *is invertible, then the derivative Jacobian is*

$$\frac{d\mathcal{A}lg_i^{\star}(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = \left(\boldsymbol{I} + \frac{1}{\lambda}\,\nabla_{\boldsymbol{\phi}}^2\hat{\mathcal{L}}_i(\boldsymbol{\phi}_i)\right)^{-1}. \tag{6}$$

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

Practical Algorithm: While Lemma 1 provides an idealized way to compute the $\mathcal{A}lg_i^{\star}$ Jacobians and thus by extension the meta-gradient, it may be difficult to directly use it in practice. Two issues are particularly relevant. First, the meta-gradients require computation of $\mathcal{A}lg_i^{\star}(\boldsymbol{\theta})$, which is the exact solution to the inner optimization problem. In practice, we may be able to obtain only approximate solutions. Second, explicitly forming and inverting the matrix in Eq. 6 for computing the Jacobian may be intractable for large deep neural networks. To address these difficulties, we consider approximations to the idealized approach that enable a practical algorithm.

First, we consider an approximate solution to the inner optimization problem, that can be obtained with iterative optimization algorithms like gradient descent.

Definition 1. *(δ–approx. algorithm) Let* $\mathcal{A}lg_i(\boldsymbol{\theta})$ *be a* δ–*accurate approximation of* $\mathcal{A}lg_i^{\star}(\boldsymbol{\theta})$, *i.e.*

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

Algorithm 1 Implicit Model-Agnostic Meta-Learning (iMAML)

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

Algorithm 2 Implicit Meta-Gradient Computation

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

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

Definition 2. *(δ'–approximate Jacobian-vector product) Let $\boldsymbol{g}_i$ be a vector such that*

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

where $\boldsymbol{\phi}_i = \mathcal{A}lg_i(\boldsymbol{\theta})$ and $\mathcal{A}lg_i$ is based on definition 1.

Note that $\boldsymbol{g}_i$ in definition 2 is an approximation of the meta-gradient for task $\mathcal{T}_i$. Observe that $\boldsymbol{g}_i$ can be obtained as an approximate solution to the optimization problem:

$$\min_{\boldsymbol{w}}\;\; \boldsymbol{w}^\top\left(\boldsymbol{I} + \frac{1}{\lambda}\,\nabla_{\boldsymbol{\phi}}^2\hat{\mathcal{L}}_i(\boldsymbol{\phi}_i)\right)\boldsymbol{w} - \boldsymbol{w}^\top\nabla_{\boldsymbol{\phi}}\mathcal{L}_i(\boldsymbol{\phi}_i) \tag{7}$$

The conjugate gradient (CG) algorithm is particularly well suited for this problem due to its excellent iteration complexity and requirement of only Hessian-vector products of the form $\nabla^2\hat{\mathcal{L}}_i(\boldsymbol{\phi}_i)\boldsymbol{v}$. Such hessian-vector products can be obtained cheaply without explicitly forming or storing the Hessian matrix (as we discuss in Appendix C). This CG based inversion has been successfully deployed in Hessian-free or Newton-CG methods for deep learning [36, 44] and trust region methods in reinforcement learning [52, 47]. Algorithm 1 presents the full practical algorithm. Note that these approximations to develop a practical algorithm introduce errors in the meta-gradient computation. We analyze the impact of these errors in Section 3.2 and show that they are controllable. See Appendix A for how iMAML generalizes prior gradient optimization based meta-learning algorithms.

3.2 Theory

In Section 3.1, we outlined a practical algorithm that makes approximations to the idealized update rule of Eq. 5. Here, we attempt to analyze the impact of these approximations, and also understand the computation and memory requirements of iMAML. We find that iMAML can match the minimax computational complexity of backpropagating through the path of the inner optimizer, but is substantially better in terms of memory usage. This work to our knowledge also provides the first non-asymptotic result that analyzes approximation error due to implicit gradients. Theorem 1 provides the computational and memory complexity for obtaining an ϵ–approximate meta-gradient. We assume $\mathcal{L}_i$ is smooth but do not require it to be convex. We assume that G_i in Eq. 4 is strongly convex, which can be made possible by appropriate choice of λ. The key to our analysis is a second order Lipshitz assumption, i.e. $\hat{\mathcal{L}}_i(\cdot)$ is ρ-Lipshitz Hessian. This assumption and setting has received considerable attention in recent optimization and deep learning literature [26, 42].

Table 1: Compute and memory for computing the meta-gradient when using a δ–accurate $\mathcal{A}lg_i$, and the corresponding approximation error. Our compute time is measured in terms of the number of $\nabla\hat{\mathcal{L}}_i$ computations. All results are in $\tilde{O}(\cdot)$ notation, which hide additional log factors; the error bound hides additional problem dependent Lipshitz and smoothness parameters (see the respective Theorem statements). $\kappa \geq 1$ is the condition number for inner objective G_i (see Equation 4), and D is the diameter of the search space. The notions of error are subtly different: we assume all methods solve the inner optimization to error level of δ (as per definition 1). For our algorithm, the error refers to the ℓ_2 error in the computation of $d_{\boldsymbol{\theta}}\mathcal{L}_i(\mathcal{A}lg_i^{\star}(\boldsymbol{\theta}))$. For the other algorithms, the error refers to the ℓ_2 error in the computation of $d_{\boldsymbol{\theta}}\mathcal{L}_i(\mathcal{A}lg_i(\boldsymbol{\theta}))$. We use Prop 3.1 of Shaban et al. [53] to provide the guarantee we use. See Appendix D for additional discussion.

Algorithm	Compute	Memory	Error
MAML (GD + full back-prop)	$\kappa \log\left(\frac{D}{\delta}\right)$	$\mathrm{Mem}(\nabla\hat{\mathcal{L}}_i) \cdot \kappa \, \log\left(\frac{D}{\delta}\right)$	0
MAML (Nesterov's AGD + full back-prop)	$\sqrt{\kappa} \log\left(\frac{D}{\delta}\right)$	$\mathrm{Mem}(\nabla\hat{\mathcal{L}}_i) \cdot \sqrt{\kappa} \, \log\left(\frac{D}{\delta}\right)$	0
Truncated back-prop [53] (GD)	$\kappa \log\left(\frac{D}{\delta}\right)$	$\mathrm{Mem}(\nabla\hat{\mathcal{L}}_i) \cdot \kappa \, \log\left(\frac{1}{\epsilon}\right)$	ϵ
Implicit MAML (this work)	$\sqrt{\kappa} \log\left(\frac{D}{\delta}\right)$	$\mathrm{Mem}(\nabla\hat{\mathcal{L}}_i)$	δ

Table 1 summarizes our complexity results and compares with MAML and truncated backpropagation [53] through the path of the inner optimizer. We use κ to denote the condition number of the inner problem induced by G_i (see Equation 4), which can be viewed as a measure of hardness of the inner optimization problem. $\mathrm{Mem}(\nabla\hat{\mathcal{L}}_i)$ is the memory taken to compute a single derivative $\nabla\hat{\mathcal{L}}_i$. Under the assumption that Hessian vector products are computed with the reverse mode of autodifferentiation, we will have that both: the compute time and memory used for computing a Hessian vector product are with a (universal) constant factor of the compute time and memory used for computing $\nabla\hat{\mathcal{L}}_i$ itself (see Appendix C). This allows us to measure the compute time in terms of the number of $\nabla\hat{\mathcal{L}}_i$ computations. We refer readers to Appendix D for additional discussion about the algorithms and their trade-offs.

Our main theorem is as follows:

Theorem 1. *(Informal Statement; Approximation error in Algorithm 2) Suppose that: $\mathcal{L}_i(\cdot)$ is B Lipshitz and L smooth function; that $G_i(\cdot, \boldsymbol{\theta})$ (in Eq. 4) is a μ-strongly convex function with condition number κ; that D is the diameter of search space for ϕ in the inner optimization problem (i.e. $\|\mathcal{A}lg_i^{\star}(\boldsymbol{\theta})\| \leq D$); and $\hat{\mathcal{L}}_i(\cdot)$ is ρ-Lipshitz Hessian.*

Let $\boldsymbol{g}_i$ be the task meta-gradient returned by Algorithm 2. For any task i and desired accuracy level ϵ, Algorithm 2 computes an approximate task-specific meta-gradient with the following guarantee:
$$\|\boldsymbol{g}_i - d_{\boldsymbol{\theta}}\mathcal{L}_i(\mathcal{A}lg_i^{\star}(\boldsymbol{\theta}))\| \leq \epsilon.$$
Furthermore, under the assumption that the Hessian vector products are computed by the reverse mode of autodifferentiation (Assumption 1), Algorithm 2 can be implemented using at most $\tilde{O}\left(\sqrt{\kappa}\log\left(\frac{poly(\kappa, D, B, L, \rho, \mu, \lambda)}{\epsilon}\right)\right)$ gradient computations of $\hat{\mathcal{L}}_i(\cdot)$ and using at most $2 \cdot \mathrm{Mem}(\nabla\hat{\mathcal{L}}_i)$ memory.

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

Corollary 1. *(iMAML finds stationary points) Suppose the conditions of Theorem 1 hold and that $F(\cdot)$ is an L_F smooth function. Then the implicit MAML algorithm (Algorithm 1), when the batch size is M (so that we are doing gradient descent), will find a point $\boldsymbol{\theta}$ such that:*
$$\|\nabla F(\boldsymbol{\theta})\| \leq \epsilon$$
in a number of calls to `Implicit-Meta-Gradient` that is at most $\frac{4ML_f(F(0) - \min_{\boldsymbol{\theta}} F(\boldsymbol{\theta}))}{\epsilon^2}$. Furthermore, the total number of gradient computations (of $\nabla\hat{\mathcal{L}}_i$) is at most $\tilde{O}\left(M\sqrt{\kappa}\frac{L_f(F(0) - \min_{\boldsymbol{\theta}} F(\theta))}{\epsilon^2} \log\left(\frac{poly(\kappa, D, B, L, \rho, \mu, \lambda)}{\epsilon}\right)\right)$, and only $\tilde{O}(\mathrm{Mem}(\nabla\hat{\mathcal{L}}_i))$ memory is required throughout.

4 Experimental Results and Discussion

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

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

To study (3), we turn to the Omniglot dataset [30] which is a popular few-shot image recognition domain. Figure 2 presents compute and memory trade-off for MAML and iMAML (on 20-way, 5-shot Omniglot). Memory for iMAML is based on Hessian-vector products and is independent of the number of GD steps in the inner loop. The memory use is also independent of the number of CG iterations, since the intermediate computations need not be stored in memory. On the other hand, memory for MAML grows linearly in grad steps, reaching the capacity of a 12 GB GPU in approximately 16 steps. First-order MAML (FOMAML) does not back-propagate through the optimization process, and thus the computational cost is only that of performing gradient descent, which is needed for all the algorithms. The computational cost for iMAML is also similar to FOMAML along with a constant overhead for CG that depends on the number of CG steps. Note however, that FOMAML does not compute an accurate meta-gradient, since it ignores the Jacobian. Compared to FOMAML, the compute cost of MAML grows at a faster rate. FOMAML requires only gradient computations, while backpropagating through GD (as done in MAML) requires a Hessian-vector products at each iteration, which are more expensive.

Finally, we study empirical performance of iMAML on the Omniglot and Mini-ImageNet domains. Following the few-shot learning protocol in prior work [57], we run the iMAML algorithm on the
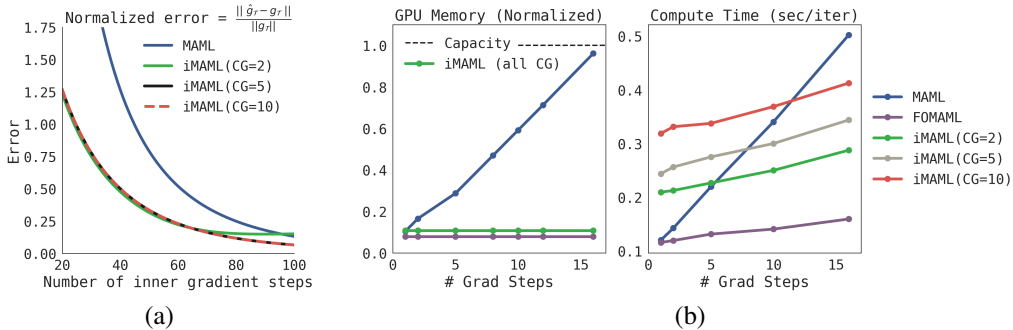

Figure 2: Accuracy, Computation, and Memory tradeoffs of iMAML, MAML, and FOMAML. (a) Meta-gradient accuracy level in synthetic example. Computed gradients are compared against the exact meta-gradient per Def 3. (b) Computation and memory trade-offs with 4 layer CNN on 20-way-5-shot Omniglot task. We implemented iMAML in PyTorch, and for an apples-to-apples comparison, we use a PyTorch implementation of MAML from: https://github.com/dragen1860/MAML-Pytorch

Table 2: Omniglot results. MAML results are taken from the original work of Finn et al. [15], and first-order MAML and Reptile results are from Nichol et al. [43]. iMAML with gradient descent (GD) uses 16 and 25 steps for 5-way and 20-way tasks respectively. iMAML with Hessian-free uses 5 CG steps to compute the search direction and performs line-search to pick step size. Both versions of iMAML use $\lambda = 2.0$ for regularization, and 5 CG steps to compute the task meta-gradient.

Algorithm	5-way 1-shot	5-way 5-shot	20-way 1-shot	20-way 5-shot
MAML [15]	$98.7 \pm 0.4\%$	$\mathbf{99.9 \pm 0.1\%}$	$95.8 \pm 0.3\%$	$98.9 \pm 0.2\%$
first-order MAML [15]	$98.3 \pm 0.5\%$	$99.2 \pm 0.2\%$	$89.4 \pm 0.5\%$	$97.9 \pm 0.1\%$
Reptile [43]	$97.68 \pm 0.04\%$	$99.48 \pm 0.06\%$	$89.43 \pm 0.14\%$	$97.12 \pm 0.32\%$
iMAML, GD (ours)	$99.16 \pm 0.35\%$	$99.67 \pm 0.12\%$	$94.46 \pm 0.42\%$	$98.69 \pm 0.1\%$
iMAML, Hessian-Free (ours)	$\mathbf{99.50 \pm 0.26\%}$	$99.74 \pm 0.11\%$	$\mathbf{96.18 \pm 0.36\%}$	$\mathbf{99.14 \pm 0.1\%}$

dataset for different numbers of class labels and shots (in the N-way, K-shot setting), and compare two variants of iMAML with published results of the most closely related algorithms: MAML, FOMAML, and Reptile. While these methods are not state-of-the-art on this benchmark, they provide an apples-to-apples comparison for studying the use of implicit gradients in optimization-based meta-learning. For a fair comparison, we use the identical convolutional architecture as these prior works. Note however that architecture tuning can lead to better results for all algorithms [27].

The first variant of iMAML we consider involves solving the inner level problem (the regularized objective function in Eq. 4) using gradient descent. The meta-gradient is computed using conjugate gradient, and the meta-parameters are updated using Adam. This presents the most straightforward comparison with MAML, which would follow a similar procedure, but backpropagate through the path of optimization as opposed to invoking implicit differentiation. The second variant of iMAML uses a second order method for the inner level problem. In particular, we consider the Hessian-free or Newton-CG [44, 36] method. This method makes a local quadratic approximation to the objective function (in our case, $G(\phi', \theta)$ and approximately computes the Newton search direction using CG. Since CG requires only Hessian-vector products, this way of approximating the Newton search direction is scalable to large deep neural networks. The step size can be computed using regularization, damping, trust-region, or linesearch. We use a linesearch on the training loss in our experiments to also illustrate how our method can handle non-differentiable inner optimization loops. We refer the readers to Nocedal & Wright [44] and Martens [36] for a more detailed exposition of this optimization algorithm. Similar approaches have also gained prominence in reinforcement learning [52, 47].

Tables 2 and 3 present the results on Omniglot and Mini-ImageNet, respectively. On the Omniglot domain, we find that the GD version of iMAML is competitive with the full MAML algorithm, and substatially better than its approximations (i.e., first-order MAML and Reptile), especially for the harder 20-way tasks. We also find that iMAML with Hessian-free optimization performs substantially better than the other methods, suggesting that powerful optimizers in the inner loop can offer benifits to meta-learning. In the Mini-ImageNet

Table 3: Mini-ImageNet 5-way-1-shot accuracy

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

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

5 Related Work

Our work considers the general meta-learning problem [51, 55, 41], including few-shot learning [30, 57]. Meta-learning approaches can generally be categorized into metric-learning approaches that learn an embedding space where non-parametric nearest neighbors works well [29, 57, 54, 45, 3], black-box approaches that train a recurrent or recursive neural network to take datapoints as input

and produce weight updates [25, 5, 33, 48] or predictions for new inputs [50, 12, 58, 40, 38], and optimization-based approaches that use bi-level optimization to embed learning procedures, such as gradient descent, into the meta-optimization problem [15, 13, 8, 60, 34, 17, 59, 23]. Hybrid approaches have also been considered to combine the benefits of different approaches [49, 56]. We build upon optimization-based approaches, particularly the MAML algorithm [15], which meta-learns an initial set of parameters such that gradient-based fine-tuning leads to good generalization. Prior work has considered a number of inner loops, ranging from a very general setting where all parameters are adapted using gradient descent [15], to more structured and specialized settings, such as ridge regression [8], Bayesian linear regression [23], and simulated annealing [2]. The main difference between our work and these approaches is that we show how to analytically derive the gradient of the outer objective without differentiating through the inner learning procedure.

Mathematically, we view optimization-based meta-learning as a bi-level optimization problem. Such problems have been studied in the context of few-shot meta-learning (as discussed previously), gradient-based hyperparameter optimization [35, 46, 19, 11, 10], and a range of other settings [4, 31]. Some prior works have derived implicit gradients for related problems [46, 11, 4] while others propose innovations to aid back-propagation through the optimization path for specific algorithms [35, 19, 24], or approximations like truncation [53]. While the broad idea of implicit differentiation is well known, it has not been empirically demonstrated in the past for learning more than a few parameters (e.g., hyperparameters), or highly structured settings such as quadratic programs [4]. In contrast, our method meta-trains deep neural networks with thousands of parameters. Closest to our setting is the recent work of Lee et al. [32], which uses implicit differentiation for quadratic programs in a final SVM layer. In contrast, our formulation allows for adapting the full network for generic objectives (beyond hinge-loss), thereby allowing for wider applications.

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

6 Conclusion

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

Broader classes of inner loop procedures. While we studied different gradient-based optimization methods in the inner loop, iMAML can in principle be used with a variety of inner loop algorithms, including dynamic programming methods such as Q-learning, two-player adversarial games such as GANs, energy-based models [39], and actor-critic RL methods, and higher-order model-based trajectory optimization methods. This significantly expands the kinds of problems that optimization-based meta-learning can be applied to.

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

Acknowledgements

References

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

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

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

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

[5] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[33] Ke Li and Jitendra Malik. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

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

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

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

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

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

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

[40] Tsendsuren Munkhdalai and Hong Yu. Meta networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2554–2563. JMLR. org, 2017.

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

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

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

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

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

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

[47] Aravind Rajeswaran, Kendall Lowrey, Emanuel Todorov, and Sham Kakade. Towards Generalization and Simplicity in Continuous Control. In *NIPS*, 2017.

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

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

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

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

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

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

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

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

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

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

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

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

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

A   Relationship between iMAML and Prior Algorithms

The presented iMAML algorithm has close connections, as well as notable differences, to a number of related algorithms like MAML [15], first-order MAML, and Reptile [43]. Conventionally, these algorithms do not consider any explicit regularization in the inner-level and instead rely on early stopping, through only a few gradient descent steps. In our problem setting described in Eq. 4, we consider an explicitly regularized inner-level problem (refer to discussion in Section 2.2). We describe the connections between the algorithms in this explicitly regularized setting below.

MAML. The MAML algorithm first invokes an iterative algorithm to solve the inner optimization problem (see definition 1). Subsequently, it backpropagates through the path of the optimization algorithm to update the meta-parameters as:

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \eta \, \frac{1}{M} \sum_{i=1}^{M} \boldsymbol{d_\theta} \mathcal{L}_i(\mathcal{A}lg_i(\boldsymbol{\theta}^k)).$$

Since $\mathcal{A}lg_i(\boldsymbol{\theta})$ approximates $\mathcal{A}lg_i^\star(\boldsymbol{\theta})$, it can be viewed that both MAML and iMAML intend to perform the same idealized update in Eq. 5. However, they perform the meta-gradient computation very differently. MAML backpropagates through the path of an iterative algorithm, while iMAML computes the meta-gradient through the implicit Jacobian approach outlined in Section 3.1 (see Figure 1 for a visual depiction). As a result, iMAML can be vastly more efficient in memory while having lesser or comparable computational requirements. It also allows for higher order optimization methods and non-differentiable components.

First-order MAML ignores the effect of meta-parameters $\boldsymbol{\theta}$ on task parameters $\{\phi_i\}$ in the meta-gradient computation and updates the meta-parameters as:

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \eta \, \frac{1}{M} \sum_{i=1}^{M} \nabla_\phi \mathcal{L}_i(\boldsymbol{\phi}_i) \, |_{\phi_i = \mathcal{A}lg_i(\boldsymbol{\theta}^k)}$$

Note that iMAML strictly generalizes this, since first-order MAML is simply iMAML when the conjugate gradient procedure is not invoked (or corresponds to 0 steps of CG). Thus, iMAML allows for an easy way to interpolate from first-order MAML to the full MAML algorithm.

Reptile [43], similar to first-order MAML, ignores the dependence of task-parameters on meta-parameters. However, instead of following the gradients at $\phi_i = \mathcal{A}lg_i(\boldsymbol{\theta}^k)$, Reptile uses the task-parameters as targets and slowly moves meta-parameters towards them:

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \eta \, \frac{1}{M} \sum_{i=1}^{M} (\boldsymbol{\theta}^k - \boldsymbol{\phi}_i).$$

From the proximal point equation in the proof of Lemma 1, we have $\boldsymbol{\phi}_i = \boldsymbol{\theta}^k - \frac{1}{\lambda} \nabla_\phi \mathcal{L}_i(\boldsymbol{\phi}_i)$, using which we see that the Reptile equation becomes: $\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \frac{\eta}{\lambda M} \sum_{i=1}^{M} \nabla_\phi \mathcal{L}_i(\boldsymbol{\phi}_i)$. Thus, Reptile and first-order MAML are identical in our problem formulation up to the choice of learning rate. Making the regularization explicit allows us to illustrate this equivalence.

B   Optimization Preliminaries

Let $f : \mathbb{R}^d \to \mathbb{R}$. A function f is B Lipschitz (or B-bounded gradient norm) if for all $x \in \mathbb{R}^d$

$$||\nabla f(x)|| \leq B \,.$$

Similarly, we say that a matrix valued function $M : \mathbb{R}^d \times \mathbb{R}^{d'} \to \mathbb{R}$ is ρ-Lipschitz if

$$||M(x) - M(x')|| \leq \rho ||x - x'|| \,,$$

where $|| \cdot ||$ denotes the spectral norm.

We say that f is L-smooth if for all $x, x' \in \mathbb{R}^d$

$$||\nabla f(x) - \nabla f(x')|| \leq L ||x - x'||$$

13

and that f is μ-strongly convex if f is convex and if for all $x, x' \in \mathbb{R}^d$,

$$||\nabla f(x) - \nabla f(x')|| \geq \mu ||x - x'||.$$

We will make use of the following black-box complexity of first-order gradient methods for minimizing strongly convex and smooth functions.

Lemma 2. *(δ-approximate solver; see [9]) Suppose f is a function that is L-smooth and μ strongly convex. Define $\kappa := L/\mu$, and let $x^\star = \operatorname{argmin} f(x)$. Nesterov's accelerated gradient descent can be used to find a point x such that:*

$$\|x - x^\star\| \leq \delta$$

using a number of gradient computations of f that is bounded as follows:

$$\# \text{ gradient computations of } f(\cdot) \leq 2\sqrt{\kappa} \log\left(2\kappa \frac{\|x^\star\|}{\delta}\right).$$

C   Review: Time and Space Complexity of Hessian-Vector Products

We briefly discuss the time and space complexity of Hessian-vector product computation using the reverse mode of automatic differentiation. The reverse mode of automatic differentiation [6, 22] is the widely used method for automatic differentiation in modern software packages like TensorFlow and PyTorch [7]. Recall that for a differentiable function $f(x)$, the reverse mode of automatic differentiation computes $\nabla f(x)$ in time that is no more than a factor of 5 of the time it takes to compute $f(x)$ itself (see [22] for review). As our algorithm makes use of Hessian vector products, we will make use of the following assumption as to how Hessian vector products will be computed when executing Algorithm 2.

Assumption 1. *(Complexity of Hessian-vector product) We assume that the time to compute the Hessian-vector product $\nabla^2_\phi \hat{\mathcal{L}}_i(\phi)v$ is no more than a (universal) constant over the time used to compute $\nabla \hat{\mathcal{L}}_i(\phi)$ (typically, this constant is 5). Furthermore, we assume that the memory used to compute the Hessian-vector product $\nabla^2_\phi \hat{\mathcal{L}}_i(\phi)v$ is no more than twice the memory used when computing $\nabla \hat{\mathcal{L}}_i(\phi)$. This assumption is valid if the reverse mode of automatic differentiation is used to compute Hessian vector products (see [21]).*

A few remarks about this assumption are in order. With regards to computation, first observe that the gradient of the scalar function $\nabla_\phi \hat{\mathcal{L}}_i(\phi)^\top v$ is the desired Hessian vector product $\nabla^2_\phi \hat{\mathcal{L}}_i(\phi)v$. Thus computing the Hessian vector product using the reverse mode is within a constant factor of computing the function itself, which is simply the cost of computing $\nabla \hat{\mathcal{L}}_i(\phi)^\top v$. The issue of memory is more subtle (see [21]), which we now discuss. The memory used to compute the gradient of a scalar cost function $f(x)$ using the reverse mode of auto-differentiation is proportional to the size of the computation graph; precisely, the memory required to compute the gradient is equal to the total space required to store all the intermediate variables used when computing $f(x)$. In practice, this is often much larger than the memory required to compute $f(x)$ itself, due to that all intermediate variables need not be simultaneously stored in memory when computing $f(x)$. However, for the special case of computing the gradient of the function $f(\phi) = \nabla_\phi \hat{\mathcal{L}}_i(\phi)^\top v$, the factor of 2 in the memory bound is a consequence of the following reason: first, using the reverse mode to compute $f(\phi)$ means we already have stored the computation graph of $\hat{\mathcal{L}}_i(\phi)$ itself. Furthermore, the size of the computation graph for computing $f(\phi) = \nabla_\phi \hat{\mathcal{L}}_i(\phi)^\top v$ is essentially the same size as the computation graph of $\hat{\mathcal{L}}_i(\phi)$. This leads to the factor of 2 memory bound; see Griewank [21] for further discussion.

D   Additional Discussion About Compute and Memory Complexity

Our main complexity results are summarized in Table 1. For these results, we consider two notions of error that are subtly different, which we explicitly define below. Let g_i be the computed meta-gradient for task $\mathcal{T}_i$. Then, the errors we consider are:

Definition 3. *Exact-solve error (our notion of error): Our goal is to accurately compute the gradient of $F(\theta)$ as defined in Equation 4, where $Alg_i^\star(\theta)$ is an exact algorithm. Specifically, we seek to compute a g_i such that:*

$$\|g_i - d_\theta \mathcal{L}_i(Alg_i^\star(\theta))\| \leq \epsilon$$

where ϵ is the error in the gradient computation.

Definition 4. *Approx-solve error: Here we suppose that Alg_i computes a δ–accurate solution to the inner optimization problem over G_i in Eq. 4, i.e. that Alg_i satisfies $\|Alg_i(\theta) - Alg_i^\star(\theta)\| \leq \delta$, as per definition 1. Then the objective is to compute a g such that:*

$$\|g - d_\theta \mathcal{L}_i(Alg_i(\theta))\| \leq \epsilon$$

where ϵ is the error in the gradient computation of $d_\theta \mathcal{L}_i(Alg_i(\theta))$. Subtly, note that the gradient is with respect to the δ-approximate algorithm, as opposed to using $Alg_i^\star$.

For the complexity results, we assume that MAML invokes Alg_i to get a δ-approximate solution for inner problem (recall definition 1). The exact-solve error for MAML is not known in the literature; in particular, even as $\delta \to 0$ it is not evident if the approx-solve solution tends to the exact-solve solution, unless further regularity conditions are imposed. The approx-solve error for MAML is 0, ignoring finite-precision and numerical issues, since it backpropagates through the path. Truncated backprop [53] also invokes Alg_i to obtain a δ-approximate solution but instead performs a truncated or partial back-propagation so that it uses a smaller number of iterations when computing the gradient through the path of $Alg_i(\theta)$. Exact-solve error for truncated backprop is also not known, but a small approx-solve error can be obtained with less memory than full back-prop. We use Prop 3.1 of Shaban et al. [53] to provide a guarantee that leads to an ϵ–accurate approximation of the full-backprop (i.e. MAML) gradient. It is not evident how accurate the truncated procedure is when an accelerated method is used instead. Finally, our iMAML algorithm also invokes an approximate solver Alg_i rather than $Alg_i^\star$. However, importantly, we guarantee a small exact-solve error even though we do not require access to $Alg_i^\star$. Furthermore, the iMAML algorithm also requires substantially less memory. Up to small constant factors, it only utilizes the memory required for computing a single gradient of $\hat{\mathcal{L}}_i(\cdot)$.

E   Proofs

Lemma 1, restated. Consider $Alg_i^\star(\theta)$ as defined in Eq. 4 for task $\mathcal{T}_i$. Let $\phi_i = Alg_i^\star(\theta)$ be the result of $Alg_i^\star(\theta)$. If $\left(I + \frac{1}{\lambda}\nabla_\phi^2 \hat{\mathcal{L}}_i(\phi_i)\right)$ is invertible, then the derivative Jacobian is

$$\frac{d Alg_i^\star(\theta)}{d\theta} = \left(I + \frac{1}{\lambda}\nabla_\phi^2 \hat{\mathcal{L}}_i(\phi_i)\right)^{-1}.$$

Proof. We drop the task i subscripts in the proof for convenience. Since $\phi = Alg^\star(\theta)$ is the minimizer of $G(\phi', \theta)$ in Eq. 4, the stationary point conditions imply that

$$\nabla_{\phi'} G(\phi', \theta)\mid_{\phi'=\phi} = 0 \implies \nabla\hat{\mathcal{L}}(\phi) + \lambda(\phi - \theta) = 0 \implies \phi = \theta - \frac{1}{\lambda}\nabla\hat{\mathcal{L}}(\phi),$$

which is an implicit equation that often arises in proximal point methods. When the derivative exists, we can differentiate the above equation to obtain:

$$\frac{d\phi}{d\theta} = I - \frac{1}{\lambda}\nabla^2\hat{\mathcal{L}}(\phi)\frac{d\phi}{d\theta} \implies \left(I + \frac{1}{\lambda}\nabla^2\hat{\mathcal{L}}(\phi)\right)\frac{d\phi}{d\theta} = I.$$

which completes the proof. $\square$

Recall that:

$$G_i(\phi', \theta) := \hat{\mathcal{L}}_i(\phi') + \frac{\lambda}{2}\|\phi' - \theta\|^2.$$

Assumption 2. *(Regularity conditions) Suppose the following holds for all tasks i:*

1. $\mathcal{L}_i(\cdot)$ *is B Lipshitz and L smooth.*

2. *For all $\boldsymbol{\theta}$, $G_i(\cdot, \boldsymbol{\theta})$ is both a β-smooth function and a μ-strongly convex function. Define:*

$$\kappa := \frac{\beta}{\mu}.$$

3. *$\hat{\mathcal{L}}_i(\cdot)$ is ρ-Lipshitz Hessian, i.e. $\nabla^2 \hat{\mathcal{L}}_i(\cdot)$ is ρ-Lipshitz.*

4. *For all $\boldsymbol{\theta}$, suppose the arg-minimizer of $G_i(\cdot, \boldsymbol{\theta})$ is unique and bounded in a ball of radius D, i.e. for all $\boldsymbol{\theta}$,*

$$\|\mathcal{A}lg_i^\star(\boldsymbol{\theta})\| \leq D.$$

Lemma 3. *(Implicit Gradient Accuracy) Suppose Assumption 2 holds. Fix a task i. Suppose that ϕ_i satisfies:*

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

and that $\boldsymbol{g}_i$ satisfies:

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

Assuming that $\delta < \mu/(2\rho)$, we have that:

$$\|\boldsymbol{g}_i - \boldsymbol{d}_{\boldsymbol{\theta}}\mathcal{L}_i(\mathcal{A}lg_i^\star(\boldsymbol{\theta}))\| \leq \left(2\frac{\lambda\rho}{\mu^2}B + \frac{\lambda L}{\mu} \right)\delta + \delta'$$

Proof. First, observe that:

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

For notational convenience, we drop the i subscripts within the proof. We have:

$$\|\boldsymbol{d}_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{A}lg^\star(\boldsymbol{\theta})) - \boldsymbol{g}\|$$
$$\leq \quad \|\boldsymbol{d}_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{A}lg^\star(\boldsymbol{\theta})) - \left(\boldsymbol{I} + \frac{1}{\lambda}\nabla^2 \hat{\mathcal{L}}(\phi) \right)^{-1} \nabla_\phi \mathcal{L}(\phi)\| + \delta'$$
$$\leq \quad \|\boldsymbol{d}_{\boldsymbol{\theta}}\mathcal{L}(\mathcal{A}lg^\star(\boldsymbol{\theta})) - \left(\boldsymbol{I} + \frac{1}{\lambda}\nabla^2 \hat{\mathcal{L}}(\phi) \right)^{-1} \nabla_\phi \mathcal{L}(\mathcal{A}lg^\star(\boldsymbol{\theta}))\| +$$
$$\| \left(\boldsymbol{I} + \frac{1}{\lambda}\nabla^2 \hat{\mathcal{L}}(\phi) \right)^{-1} (\nabla_\phi \mathcal{L}(\mathcal{A}lg^\star(\boldsymbol{\theta})) - \nabla_\phi \mathcal{L}(\phi)) \| + \delta'$$

where the first inequality uses the triangle inequality.

We now bound each of these terms. For the second term,

$$\| \left(\boldsymbol{I} + \frac{1}{\lambda}\nabla^2 \hat{\mathcal{L}}(\phi) \right)^{-1} (\nabla_\phi \mathcal{L}(\mathcal{A}lg^\star(\boldsymbol{\theta})) - \nabla_\phi \mathcal{L}(\phi)) \|$$
$$\leq \quad \| \left(\boldsymbol{I} + \frac{1}{\lambda}\nabla^2 \hat{\mathcal{L}}(\phi) \right)^{-1} \|\|\nabla_\phi \mathcal{L}(\mathcal{A}lg^\star(\boldsymbol{\theta})) - \nabla_\phi \mathcal{L}(\phi)\|$$
$$\leq \quad \lambda L\| \left(\lambda\boldsymbol{I} + \nabla^2 \hat{\mathcal{L}}(\phi) \right)^{-1} \|\|\mathcal{A}lg^\star(\boldsymbol{\theta}) - \phi\|$$
$$= \quad \lambda L\|\nabla_\phi^2 G(\phi, \boldsymbol{\theta})^{-1}\|\|\mathcal{A}lg^\star(\boldsymbol{\theta}) - \phi\|$$
$$\leq \quad \frac{\lambda L}{\mu}\delta$$

where we the second inequality uses that $\nabla_\phi \mathcal{L}$ is L-smooth and the final inequality uses that G is μ strongly convex.

16

For the first term, we have:

$$\|\boldsymbol{d_\theta}\mathcal{L}(Alg^\star(\boldsymbol{\theta})) - \left(\boldsymbol{I} + \frac{1}{\lambda}\,\nabla^2\hat{\mathcal{L}}(\boldsymbol{\phi})\right)^{-1}\nabla_{\boldsymbol{\phi}}\mathcal{L}(Alg^\star(\boldsymbol{\theta}))\|$$

$$= \quad \|\left(\left(\boldsymbol{I} + \frac{1}{\lambda}\,\nabla^2\hat{\mathcal{L}}(Alg^\star(\boldsymbol{\theta}))\right)^{-1} - \left(\boldsymbol{I} + \frac{1}{\lambda}\,\nabla^2\hat{\mathcal{L}}(\boldsymbol{\phi})\right)^{-1}\right)\nabla_{\boldsymbol{\phi}}\mathcal{L}(Alg^\star(\boldsymbol{\theta}))\|$$

$$\leq \quad \lambda\|\left(\lambda\boldsymbol{I} + \nabla^2\hat{\mathcal{L}}(Alg^\star(\boldsymbol{\theta}))\right)^{-1} - \left(\lambda\boldsymbol{I} + \nabla^2\hat{\mathcal{L}}(\boldsymbol{\phi})\right)^{-1}\|B,$$

using that $\nabla_{\boldsymbol{\phi}}\mathcal{L}$ is B Lipshitz. Now let

$$\Delta := \nabla^2\hat{\mathcal{L}}(Alg^\star(\boldsymbol{\theta})) - \nabla^2\hat{\mathcal{L}}(\boldsymbol{\phi}), \quad M := \nabla^2_{\boldsymbol{\phi}}G(\boldsymbol{\phi},\boldsymbol{\theta}) = \lambda\boldsymbol{I} + \nabla^2\hat{\mathcal{L}}(\boldsymbol{\phi})$$

Due to that $\nabla^2\hat{\mathcal{L}}(\cdot)$ is Lipshitz Hessian, $\|\Delta\| \leq \rho\delta$. Also, by our assumption on δ, we have that:

$$\|M^{-1}\Delta\| \leq \|\Delta\|/\mu \leq \rho\delta/\mu \leq 1/2,$$

which implies that $\|\left(\boldsymbol{I} + M^{-1}\Delta\right)^{-1}\| \leq 2$. Hence,

$$\|\left(\lambda\boldsymbol{I} + \nabla^2\hat{\mathcal{L}}(Alg^\star(\boldsymbol{\theta}))\right)^{-1} - \left(\lambda\boldsymbol{I} + \nabla^2\hat{\mathcal{L}}(\boldsymbol{\phi})\right)^{-1}\|$$

$$= \quad \|\left(M + \Delta\right)^{-1} - M^{-1}\|$$

$$\leq \quad \|M^{-1}\|\|\left(\boldsymbol{I} + M^{-1}\Delta\right)^{-1} - \boldsymbol{I}\|$$

$$= \quad \|M^{-1}\|\|\left(\boldsymbol{I} + M^{-1}\Delta\right)^{-1}\left(\boldsymbol{I} - \left(\boldsymbol{I} + M^{-1}\Delta\right)\right)\|$$

$$\leq \quad \|M^{-1}\|\|\left(\boldsymbol{I} + M^{-1}\Delta\right)^{-1}\|\|M^{-1}\Delta\|$$

$$\leq \quad \frac{1}{\mu}\cdot 2\cdot\frac{\rho\delta}{\mu} = 2\frac{\rho}{\mu^2}\delta.$$

The proof is completed by substitution. $\qquad\square$

Theorem 2. *(Approximate Implicit Gradient Computation) Suppose Assumption 2 holds. Fix a task i. Let*

$$B_1 \quad := \quad 2\frac{\lambda\rho}{\mu^2}B + \frac{\lambda L}{\mu}$$

Suppose Nesterov's accelerated gradient descent algorithm is used to compute ϕ (as desired in Algorithm 2), using a number of iterations that is:

$$2\sqrt{\kappa}\,\log\left(8\kappa D\left(\frac{B_1}{\epsilon} + \frac{\rho}{\mu}\right)\right)$$

and suppose Nesterov's accelerated gradient descent algorithm (or the conjugate gradient algorithm [1]) is used to compute $\boldsymbol{g}_i$ using a number of iterations that is:

$$2\sqrt{\kappa}\,\log\left(4\kappa\frac{(\lambda/\mu)B}{\epsilon}\right).$$

We have that:

$$\|\boldsymbol{g}_i - \boldsymbol{d_\theta}\mathcal{L}_i(Alg^\star_i(\boldsymbol{\theta}))\| \leq \epsilon.$$

Proof. The result will follow from the guarantees in Lemma 2. Specifically, let us set $\delta = \min\{\epsilon/(2B_1), \mu/(2\rho)\}$ and $\delta' = \epsilon/2$. To ensure the bound of δ, by Lemma 3, it suffices to use a number of iterations that is bounded by:

$$2\log\left(2\kappa\frac{\|D\|}{\delta}\right) \leq 2\sqrt{\kappa}\,\log\left(8\kappa D\left(\frac{B_1}{\epsilon} + \frac{\rho}{\mu}\right)\right)$$

---

[1]The conjugate gradient descent algorithm also suffices and give a slightly improved iteration complexity in terms of log factors.

To ensure the bound of δ', the algorithm will be solving the sub-problem in Equation 7. First observe that in the context of in Lemma 2, note that $\|x^\star\| = \left\| \left(I + \frac{1}{\lambda} \nabla^2 \hat{\mathcal{L}}_i(\phi) \right)^{-1} \nabla \mathcal{L}_i(\phi) \right\| \leq (\lambda/\mu)B$, and so it suffices to use a number of iterations that is bounded by:

$$2 \log \left(2\kappa \frac{\|x^\star\|}{\delta} \right) \leq 2 \log \left(4\kappa \frac{(\lambda/\mu)B}{\epsilon} \right),$$

which completes the proof. $\qquad\qquad\square$

F   Experiment Details

Here, we provide additional details of the experimental set-up for the experiments in Section 4. All training runs were conducted on a single NVIDIA (Titan Xp) GPU.

F.1   Synthetic Experiments

For the synthetic experiments, we consider a linear regression problem. We consider parametric models of the form $h_\phi(\mathbf{x}) = \phi^T \mathbf{x}$, where $\mathbf{x}$ can either be the raw inputs or features (e.g. Fourier features) of the input. For task $\mathcal{T}_i$, we can equivalently write a quadratic objective that represents the task loss as:

$$\hat{\mathcal{L}}_i(\phi) = \frac{1}{2} \mathbb{E}_{(\mathbf{x},\mathbf{y}) \sim \mathcal{D}_i^{\mathrm{tr}}} \left[\|h_\phi(\mathbf{x}) - \mathbf{y}\|^2 \right] = \frac{1}{2} \phi^T A_i \phi + \phi^T b_i,$$

where $A_i = \mathbb{E}_{(\mathbf{x},\mathbf{y}) \sim \mathcal{D}_i^{\mathrm{tr}}} \left[\mathbf{x}\mathbf{x}^T \right]$ and $b_i = \mathbb{E}_{(\mathbf{x},\mathbf{y}) \sim \mathcal{D}_i^{\mathrm{tr}}} \left[\mathbf{x}^T \mathbf{y} \right]$. Thus, the inner level objective and corresponding minimizer can be written as:

$$G_i(\phi', \boldsymbol{\theta}) = \frac{1}{2} \phi'^T A_i \phi' + \phi'^T b_i + \frac{\lambda}{2} (\phi' - \boldsymbol{\theta})^T (\phi' - \boldsymbol{\theta})$$

$$\mathcal{A}lg_i^\star(\boldsymbol{\theta}) = (A_i + \lambda \boldsymbol{I})^{-1} (\lambda \boldsymbol{\theta} - b_i)$$

Thus, the exact meta-gradient can be written as

$$\boldsymbol{d_\theta} \mathcal{L}_i(\mathcal{A}lg_i^\star(\boldsymbol{\theta})) = \lambda (A_i + \lambda \boldsymbol{I})^{-1} \nabla_\phi \mathcal{L}_i(\boldsymbol{\theta}) \big|_{\phi = \mathcal{A}lg_i^\star(\boldsymbol{\theta})} .$$

We compare this gradient with the gradients computed by the iMAML and MAML algorithms. We considered the case of $\mathbf{x} \in \mathbb{R}^{50}$, $\mathbf{y} \in \mathbb{R}$, $\lambda = 5.0$, and $\kappa = 50$, for the presented results.

F.2   Omniglot and Mini-ImageNet experiments

We follow the standard training and evaluation protocol as in prior works [50, 57, 15].

Omniglot Experiments   The GD version of iMAML uses 16 gradient steps for 5-way 1-shot and 5-way 5-shot settings, and 25 gradient steps for 20-way 1-shot and 20-way 5-shot settings. A regularization strength of $\lambda = 2.0$ was used for both. 5 steps of conjugate gradient was used to compute the meta-gradient for each task in the mini-batch, and the meta-gradients were averaged before taking a step with the default parameters of Adam in the outer loop.

The Hessian-free version of MAML proceeds by using Hessian-free or Newton-CG method for solving the inner optimization problem (with respect to ϕ) with objective $G_i(\phi, \boldsymbol{\theta})$. This method proceeds by constructing a local quadratic approximation to the objective and approximately computing the Newton direction with conjugate gradient. 5 CG steps are used for this process in our experiments. This allows us to compute the search direction, following which a step size has to be picked. We pick the step size through line-search. This procedure of computing the approximate Newton direction and linesearch is repeated 3 times in our experiments to solve the inner optimization problem well.

Mini-ImageNet   For the GD version of iMAML, 10 GD steps were used with regularization strength of $\lambda = 0.5$. Again, 5 CG steps are used to compute the meta-gradient. Similarly, in the Hessian-Free variant, we again use 5 CG steps to compute the search direction followed by line search. This process is repeated 3 times to solve the inner level optimization. Again, to compute the meta-gradient, 5 steps of CG are used.