When MAML Can Adapt Fast and How to Assist When It Cannot

Sébastien M. R. Arnold
seb.arnold@usc.edu
University of Southern California

Shariq Iqbal
shariqiq@usc.edu
University of Southern California

Fei Sha
fsha@google.com
Google

Abstract

Model-Agnostic Meta-Learning (MAML) and its variants have achieved success in meta-learning tasks on many datasets and settings. Nonetheless, we have just started to understand and analyze how they are able to adapt fast to new tasks. In this work, we contribute by conducting a series of empirical and theoretical studies, and discover several interesting, previously unknown properties of the algorithm. First, we find MAML adapts better with a deep architecture even if the tasks need only a shallow one. Secondly, linear layers can be added to the output layers of a shallower model to increase the depth without altering the modelling capacity, leading to improved performance in adaptation. Alternatively, an external and separate neural network meta-optimizer can also be used to transform the gradient updates of a smaller model so as to obtain improved performances in adaptation. Drawing from these evidences, we theorize that for a deep neural network to meta-learn well, the upper layers must transform the gradients of the bottom layers as if the upper layers were an external meta-optimizer, operating on a smaller network that is composed of the bottom layers.

1 Introduction

Meta-learning or *learning to learn* has been an appealing idea for addressing several important challenges in machine learning (Schmidhuber, 1987; Bengio et al., 1991; Vanschoren, 2019; Finn et al., 2017). In particular, learning from prior tasks but being able to adapt quickly to new tasks improves learning efficiency with fewer samples, *i.e.*, few-shot learning (Vinyals et al., 2016). A promising set of techniques, Model-Agnostic Meta-Learning or MAML (Finn et al., 2017) and its variants – often referred as Gradient-based Meta-Learning (GBML) – have attracted a lot of interest (Nichol et al., 2018; Lee and Choi, 2018; Grant et al., 2018; Flennerhag et al., 2019).

In GBML, the learning model is "meta"-trained on a set of *meta-training* tasks and is expected to perform well on *meta-testing* (*i.e.*, post-training-adaptation) tasks. In the phase of meta-training, the model parameters are optimized so that when applied to meta-testing tasks, a few gradient-based parameter updates lead to a significant reduction in the learning losses, a desideratum referred as "fast adaptation". To this end, MAML optimizes what is called MAML loss (§2).

In this paper, we take an unexplored direction to understand how MAML and its alike work: we investigate what types of model can meta-learn. Our work answers a few questions inspired by existing work.

First, most research work in the literature focuses on deep learning models — presumably one can posit that a sufficiently large deep learning model should be able to learn the right inductive bias to meta-learn as neural networks are universal approximators. While the argument is patently valid, our research work aims to refine it: what sense do we mean with *sufficiently large*? Is there a regime where the model is not sufficiently large such that it cannot meta-learn?

Second, the recently proposed ANIL algorithm suggests that for deep learning models, there is *almost no need* to use the MAML loss to optimize the bottom layers of the neural network Raghu et al. (2020). This observation is closely related to multitask learning (Baxter, 2000; Caruana, 1997) but does not explain what *the special roles of the models' heads* are in ensuring the bottom layers are updated as effectively as the original MAML.

Third, preconditioning methods introduce additional parameters to control the gradients during the meta-

Proceedings of the 24[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2021, San Diego, California, USA. PMLR: Volume 130.

testing to improve fast adaptation Li et al. (2017); Park and Oliva (2019); Lee and Choi (2018); Flennerhag et al. (2019). They assume those additional parameters, after being meta-trained, generalize to new tasks. However, those works do not explain why the original model can adapt fast *without* those parameters. Moreover, if given a model that is not "sufficiently large" to meta-learn, how effective would those methods be? For example, imagine those methods were given the bottom layers of a deep neural network model, could they update those layers to match the performance of the original bigger neural network?

To answer these questions, we need a way to measure how large a model is and a metric to measure how effective meta-learning is. For the former, we use the *depth*, ie, the number of layers in deep models as it is one of the most frequently cited quantity to characterize the size of a model. For the latter, we use the performance metrics (error rates or accuracies) on meta-testing tasks, a common practice in existing literature. We concentrate on few-shot learning tasks and leave other application scenarios of MAML and its alike to future study.

We use a theoretical analysis (of mathematically tractable models) to gain insights and to generate hypotheses around how meta-learning is enabled in deep learning models. We then use empirical studies to validate those hypotheses and inspire a new algorithm, dubbed META-KFO, for meta-learning.

We summarize the key findings from our research. We conclude that the *depth of a deep model is important to meta-learning*. Even if a task is solvable with a shallow or linear model, a deeper model with at least one hidden layer is required for meta-learning. The reason is that the meta-learner needs to use the upper layers of a deep model to control how the bottom layers' parameters are to be updated.

This control can be achieved in three ways. The first, which is the default and implicit strategy in existing work, is to use a sufficiently large deep neural network. The second one is to add linear layers to the output of a shallower network to increase the depth. This has the advantage that the adapted model is smaller than the first approach as the linear layers can be absorbed into the shallower network. The third one is to use especially designed preconditioning methods that directly control the gradients that update the shallower network. Those methods, including previous work (Li et al., 2017; Park and Oliva, 2019; Lee and Choi, 2018) and the proposed META-KFO algorithm (§5.3), improve meta-learnability of shallower networks that otherwise do not meta-learn well. Moreover, the proposed algorithm and its empirical behavior yields new insights: we surmise that in a deep neural network, *the upper layers have the equivalent effect of transforming the gradients of the bottom layers as if the upper layers were an external meta-optimizer, operating on a smaller network that is composed only of the bottom layers.*

While it is plainly correct to state that the upperlayers of deep models affect the bottom layers' parameter updates (as in any gradient-based learning), our work is the first to refine this argument by pointing out this influence is crucial for enabling meta-learning. This is established through a mix of theoretical analysis (§4) and empirical studies (§5) carefully designed to reveal how increasing the depth enables meta-learning. The proposed META-KFO algorithm is motivated by our findings and also contributes to the work on meta-learning by enabling shallower models to meta-learn and attaining state-of-the-art performance on several benchmarks.

2 Background and Notation

In MAML and its many variants, we have a model whose parameters are denoted by θ. We would like to optimize θ such that the resulting model can adapt to new and unseen tasks fast. We are given a set of meta-training tasks, indexed by τ. To each such task, we associate a loss $\ell_\tau(\theta)$. Distinctively, MAML minimizes the expected task loss after an *adaptation* phase, consisting of a few steps of gradient descent from the model's current parameters. Since we do not have access to how the target tasks are distributed, we use the expected loss over the meta-training tasks,

$$\mathcal{L}^{\text{MAML}}(\theta) = \mathbb{E}_{\tau\sim p(\tau)}[\ell_\tau\left(\theta - \alpha\nabla\ell_\tau(\theta)\right)] \qquad (1)$$

where the expectation is taken with respect to the distribution of the training tasks. $p(\tau)$ is a short-hand for the distribution of the task: $p(\tau) = p(\theta_\tau)p(\boldsymbol{x}, y; \theta_\tau)$, for a set of conditional regression tasks where the data $(\boldsymbol{x}, y)$ follows a distribution parameterized by θ_τ. α is the learning rate for the adaptation phase. The right-hand-side of eq. (1) uses only one step gradient descent such that the aim is to adapt *fast*: in one step, we would like to reduce the loss as much as possible. In practice, a few more steps are often used in the meta-training phase. We use

$$\theta^{\text{MAML}} = \arg\min_\theta \mathcal{L}^{\text{MAML}}(\theta) \qquad (2)$$

to denote the minimizer of this loss, *i.e.*, the MAML solution. Note that it is most likely different from each ℓ_τ's minimizer. If we use gradient descent during meta-training, the parameter is updated as follows:

$$(\text{META-TRAINING}) \quad \theta \leftarrow \theta - \beta\frac{\partial\mathcal{L}^{\text{MAML}}(\theta)}{\partial\theta} \qquad (3)$$

where the step size β is called meta-update learning rate. During the meta-testing, the MAML solution is used as an initialization for solving new tasks with regular (stochastic) gradient descent:

$$(\text{META-TESTING}) \quad \theta \leftarrow \theta - \alpha \frac{\partial \ell_{\tau'}(\theta)}{\partial \theta} \tag{4}$$

where τ' denote a new task, and α is the adaptation learning rate.

3 Overview of Our Approach

To understand the relationship between the depth and the meta-learnability, we start by creating a *failure* scenario by identifying a base model and task setup where MAML fails to meta-learn. Then we employ a *strategy of increasing the depth* of the base model such that it becomes meta-learnable. Finally, we elucidate what the increased depth achieves and how it is related to existing methods of improving meta-learnability.

To achieve these 3 desiderata, however, is challenging with deep learning models. In essence, when the depth is increased, the improvement in performance metrics could be caused by several entangled factors.

To see this, consider a base model $\mathcal{M}_1$ and a bigger model $\mathcal{M}_2$ which has more layers and thus at least as powerful, if not strictly more. Suppose the adaptation performances (say, classification accuracy) are such $\mathcal{M}_1^{\text{MAML}} \leq \mathcal{M}_2^{\text{MAML}}$. With respect to these models' Bayes optimal performances, what we would like to have first is the following relation:

$$\mathcal{M}_1^{\text{MAML}} \leq \mathcal{M}_2^{\text{MAML}} \leq \mathcal{M}_1^{\text{BAYES}} \leq \mathcal{M}_2^{\text{BAYES}} \tag{5}$$

where we can identify that the increase in performance metrics is solely due to the improved meta-learning when the depth is increased[1].

It is hard to guarantee $\mathcal{M}_2^{\text{MAML}} \leq \mathcal{M}_1^{\text{BAYES}}$ on real-world data as we do not know their true underlying distributions. However, in theoretical analysis, this can be achieved by analyzing problem settings where the (base) model $\mathcal{M}_1^{\text{BAYES}}$ (and thus $\mathcal{M}_2^{\text{BAYES}}$ also) achieves 100% accuracy. §4 follows this design thinking by applying (correctly specified) models of linear regression and logistic regression to data. The design also enables us to recognize "failure mode" of MAML when the base model $\mathcal{M}_1^{\text{MAML}}$ is significantly worse than $\mathcal{M}_1^{\text{BAYES}}$, say, at a chance level for classification.

[1]Consider the alternative relation $\mathcal{M}_1^{\text{MAML}} \leq \mathcal{M}_1^{\text{BAYES}} \leq \mathcal{M}_2^{\text{MAML}} \leq \mathcal{M}_2^{\text{BAYES}}$. Then the observed increase in performance has several possible explanations: the increased depth makes meta-learning more effective, improves the model's power in solving the tasks, or results in a combination of the both. Our design needs to rule the latter out.

Secondly, to ensure $\mathcal{M}_2$ does not increase the power of $\mathcal{M}_1$ in solving the tasks, our design of theoretical analysis in §4 and empirical studies in §5.1 and §5.2 increases the depth by adding linear layers to the outputs of the base models. We refer this as "LinNet" strategy for adaptation. While linear layers are often cited for implicit regularization to improve generalization of models, (Gidel et al., 2019; Saxe et al., 2019) in our settings, there is no overfitting. So those linear layers are indeed the only explanation for why meta-learning has been improved (not increased model representational capacity). We refer the reader to the Suppl. Material for details, including how collapsing deep models into shallow ones ruins meta-learnability.

To hypothesize how the depth facilitates meta-learning, our design goes beyond the standard argument that the upper layers of a deep neural net or added linear nets influence the bottom layers' gradients. We specifically design an algorithm called META-KFO (§5.3) where a separate neural network learns to *explicitly* transform the gradients and enable meta-adaptation on shallower models that otherwise adapt poorly.

The most important feature of this algorithm is to keep the base model's modeling capacity unchanged. This allows us to directly compare to deep models. Our empirical observations support the hypothesis that in a deep neural network, the upper layers transform the gradients of the bottom layers *as if the bottom layers alone were being meta-trained.*

4 Theoretical Analysis

We conduct theoretical analysis on mathematically tractable models and task setups, following the design outlined in the previous section. We start with creating a failure mode of MAML by employing a 1-D regression as a base model that is not meta-learnable but nonetheless can solve the meta-testing tasks. We then increase the size of the base model to make it meta-learnable by overparameterzing it (*i.e.*, adding a "linear layer"). We describe the setup in §4.1 and empirical observations of the base and the overparameterized models in §4.2. We analyze them in §4.3 and §4.4 and contrast the difference in parameter updates for both meta-training and meta-testing. The insights are discussed in §4.5, which motivates our empirical studies in §5.

4.1 Setup

We consider the task of one-dimensional linear regression. Let the task parameter $\theta_\tau \sim N(0, 1)$ be a normal distributed scalar and likewise, the covariate

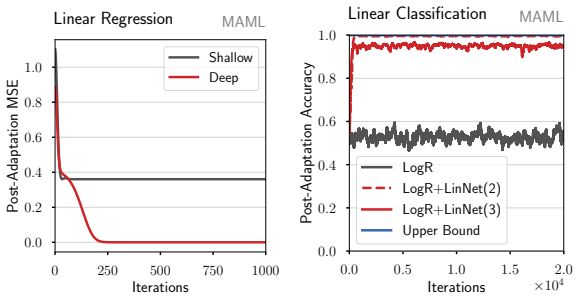

Figure 1: SHALLOW models for regression (Left) and classification (Right) fail but overparameterized DEEP models are able to meta-learn.

$x \sim N(0,1)$. The observed outcome is $y \sim N(\theta_\tau x, 1)$. We investigate two models for their meta-learning performance:

$$\text{SHALLOW:} \quad \hat{y} = cx \tag{6}$$

$$\text{DEEP:} \quad \hat{y} = abx \tag{7}$$

Note that the "deep" model is overparameterized and can be seen as two-layer neural nets with weights being a and b respectively. For each task, we use the least-square loss

$$\ell_\tau(c) = \mathbb{E}_{p(x,y|\theta_\tau)}(y - cx)^2 \tag{8}$$

$$\ell_\tau(ab) = \mathbb{E}_{p(x,y|\theta_\tau)}(y - abx)^2 \tag{9}$$

Note that the data of these tasks are generated according to the models used for meta-learning.

4.2 SHALLOW fails; DEEP meta-learns

Fig. 1(left) contrasts the two models' surprising differences in performance on meta-learning. While the DEEP (the red curve) quickly reduce the MSE on meta-testing tasks, the black curve demonstrates the poor performance of the MAML algorithm on the SHALLOW model.

The results are unexpected as both models are fully capably of solving the problem given enough data — in particular, the SHALLOW model has only one parameter c to learn.

In Fig. 1(right), we show a similar study of meta-learning linear classifiers where the data is generated according to the models. The base model is $\hat{y} = \text{BERNOULLI}(\sigma(c^{\mathrm{T}}x))$ and its overparameterized version $\hat{y} = \text{BERNOULLI}(\sigma(a^{\mathrm{T}}Bx))$ where a, B and c are matrices or vectors and x is a vector. (For details, see the the Suppl. Material). The base model attains an accuracy at the chance level while the overparameterized one reaches near-perfect classification accuracy. As in the 1-D regression, the overparameterization (equivalent to adding two or three linear layers) enables meta-learning and fast adaptation. *What roles could those additional parameters have played*?

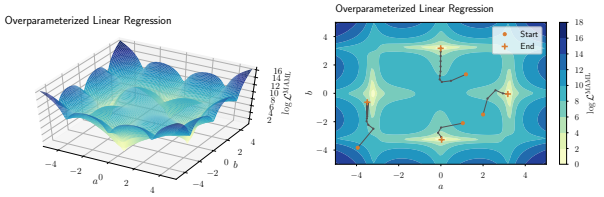

Figure 2: Meta-learning of a 1D linear regression model (§4.1). **(Left)** MAML loss of DEEP, showing multiple (local) minima with deep valleys. **(Right)** 4 meta-training trajectories (of parameters) converging to each of the 4 solutions.

4.3 Analysis of the SHALLOW model

The MAML Solution It is easy to see that the MAML solution is the origin $c^{\text{MAML}} = 0$ given the symmetries in both x and θ_τ. We state the following results (see the Suppl. Material for proofs):

$$\ell_\tau(c - \alpha\nabla\ell_\tau) = (1-\alpha)^2(c-\theta_\tau)^2 + \text{CONST} \tag{10}$$

$$\mathcal{L}^{\text{MAML}}_{\text{SHALLOW}} = 2(1-\alpha)^2c^2 + \text{CONST} \tag{11}$$

where the MAML loss is a convex function with the minimizer at $c^{\text{MAML}} = 0$, in accordance with our intuition. The gradient of the τth task is given by

$$\frac{\partial \ell_\tau(c - \alpha\nabla\ell_\tau)}{\partial c} \propto (1-\alpha)^2(c-\theta_\tau) \tag{12}$$

Note the the gradient is proportional to the deviation from the "ground-truth" parameter θ_τ. The parameter updates during meta-training and adaptation are given by (cf. §2))

$$\text{(META-TRAINING)} \quad c \leftarrow c - \beta(1-\alpha)^2(c-\theta_\tau) \tag{13}$$

$$\text{(META-TESTING)} \quad c \leftarrow c - \alpha(c-\theta_\tau) \tag{14}$$

No One Step Adaptation Suppose we would like to adapt from a task whose parameter is θ', from the MAML solution $c^{\text{MAML}} = 0$, we get

$$c \leftarrow c^{\text{MAML}} - \alpha(c^{\text{MAML}} - \theta') = \alpha\theta' \tag{15}$$

Thus, unless α happens to be 1, the optimal solution cannot be achieved in one step of adaptation. However, when $\alpha = 1$, the gradient of $\mathcal{L}^{\text{MAML}}_{\text{SHALLOW}}$ is zero (cf. eq (12)), thus meta-learning cannot occur.

4.4 Analysis of the DEEP model

The MAML solution Unfortunately, for the DEEP model, both the gradients and the losses are very complicated. With the details in the Suppl. Material, we state the following

- The origin of the parameter space ($a = 0, b = 0$) is a stationary point and the Hessian is $H = -4\alpha \boldsymbol{I}$. Thus, the origin is a local *maximum*, thus *not* a MAML solution.
- The following 4 pairs of ($a^{\text{MAML}} = \pm 1/\sqrt{\alpha}, b^{\text{MAML}} = 0$) and ($a^{\text{MAML}} = 0, b^{\text{MAML}} = \pm 1/\sqrt{\alpha}$) are locally *minimum*, with the Hessian given by $\text{DIAG}(8\alpha, 6\alpha^3)$, are thus MAML solutions.

We visualize $\mathcal{L}^{\text{MAML}}_{\text{DEEP}}$ in Fig. 2, where we can see clearly the 4 local minimum (as well as the deep valleys, "ravines") and how trajectories of parameter updates converge to them.

The gradients involve high-order polynomials of a and b – they are given in the the Suppl. Material. To gain insights, in the below, we hold a fixed and examine the gradient with respect to b during meta-training. This is reminiscent of ANIL Raghu et al. (2020). The resulting form of the gradients is greatly simplified yet remains insightful:

$$\frac{\partial \ell_\tau(a(b - \alpha \nabla_b \ell_\tau))}{\partial b} \propto a(1 - \alpha a^2)^2(ab - \theta_\tau) \tag{16}$$

Note that the symbol ∇_b indicate that only b is meta-learned with a fixed. This leads to the following:

$$\text{(META-TRAINING) } b \leftarrow b - \beta a(1 - \alpha a^2)^2(ab - \theta_\tau) \tag{17}$$

$$\text{(META-TESTING) } b \leftarrow b - \alpha a(ab - \theta_\tau) \tag{18}$$

$$a \leftarrow a - \alpha b(ab - \theta_\tau) \tag{19}$$

One Step Fast Adaptation The DEEP model has qualitatively very different adaptation behavior from the SHALLOW model. As before, at the MAML solution ($a^{\text{MAML}} = 1/\sqrt{\alpha}, b^{\text{MAML}} = 0$), we perform an adaptation on a new task with the ground-truth θ'. Holding $a^{\text{MAML}} = 1/\sqrt{\alpha}$ fixed, the update to b is

$$b^{\text{NEW}} \leftarrow b^{\text{MAML}} - \alpha a^{\text{MAML}}(a^{\text{MAML}} b^{\text{MAML}} - \theta') = \sqrt{\alpha}\theta' \tag{20}$$

Note that ($a^{\text{MAML}} = 1/\sqrt{\alpha}, b^{\text{NEW}} = \sqrt{\alpha}\theta'$) is **precisely** the optimum solution to the task as $a^{\text{MAML}} b^{\text{NEW}} = \theta'$. In other words, we need only one parameter update to arrive at the optimum solution! In fact, this **fast** adaptation does **not** depend on what α is and does not even depend on whether we adapt from the right b^{MAML} — for any random b^{RANDOM}, the update in eq. (20) immediately brings b^{RANDOM} to $b^{\text{NEW}} = \sqrt{\alpha}\theta'$!

4.5 Insights from SHALLOW versus DEEP

We examine the gradient updates of the two models. First, for adaptation during meta-testing, both eq. (18) and eq. (14) share the same element being proportional to the error signal: $(ab - \theta_\tau)$ for DEEP and $(c - \theta_\tau)$ for SHALLOW. However the additional factor a in the DEEP model enables **one-step fast adaptation** that is not possible to attain by the SHALLOW MODEL, as shown in the previous section.

Turning to the meta-training, we also notice the different scaling factors to the error signals. Contrasting eq. (17) to eq. (13), the effective step size for the former depends on $a(1 - \alpha a^2)^2$ and cannot be absorbed into the meta-learning rate β if a is also updated. Namely, in the meta-training phase, the step size for updating model parameters is **dynamically adjusted**, while the step size for the SHALLOW model is **fixed**.

While fully characterizing how a's dynamics change parameter updates is left for future work, we concentrate our analysis in the neighborhood of the solutions, say, ($a^{\text{MAML}} = 1/\sqrt{\alpha}, b^{\text{MAML}} = 0$) (the other 3 are symmetric to this one). Note that the farther a is away from a^{MAML}, the bigger the step size (in magnitude) is to amplify the error signal $(ab - \theta_\tau)$. This has the effect to move b more quickly toward the solution θ_τ / a for the τ-th task, or toward the MAML solution $b^{\text{MAML}} = 0$ (as the expectation with respect to the task distribution is 0).

Furthermore, when at the MAML solution ($a^{\text{MAML}} = 1/\sqrt{\alpha}, b^{\text{MAML}} = 0$), the update eq. (17) is stationary for *any* task. Imagine a new task τ' with ground-truth parameter θ' is randomly sampled for meta-training. Even if $a^{\text{MAML}} b^{\text{MAML}} \neq \theta'$, the gradient-based update will not change b^{MAML} from 0. On the other hand, for the SHALLOW model, even if the model is at the solution $c^{\text{MAML}} = 0$, the randomly sampled task will update the solution to $c \leftarrow -\alpha\theta'$, drifting away from the MAML solution. In other words, the MAML solution is more stable in DEEP than in SHALLOW, when (as a common practice) stochastic gradient descent is used.

We leave more comprehensive characterization of the local and global dynamics of the parameter updates to future work. In this work, our focus is: *how these observations shed light on more complicated models used in practice*?

The first insight is that even with the same modeling capacity, depth plays an important role in enabling meta-learning, even if the depth is the form of an additional scalar parameter or additional linear layers.

The second insight comes from extending the 1-D linear regression to multi-dimensional regression where the base model is $\hat{y} = Cx$ and its overparameterized version as $\hat{y} = a^{\text{T}} Bx$. It is not hard to see the forms of the gradients suggest that the additional parameters affect not only scaling factors (*i.e.*, magnitude) but also ***transforming gradient directions***

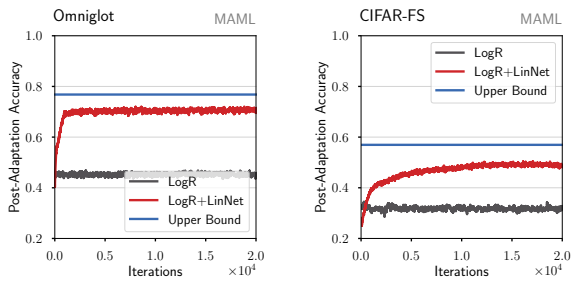

Figure 3: Meta-training logistic regression models with MAML on Omniglot and CIFAR-FS led to poor performances (Results on mini-ImageNet are in the the Suppl. Material). Adding linear nets improves meta-learning significantly, *without* changing the model's capacity.

through those additional parameters. While hard to analyze mathematically, our empirical studies below provide strong evidences.

5 How to Be (More) Meta-Learnable

The analysis on highly idealized models in the previous section needs to be empirically verified on real-world datasets. We first validate the findings in §4.2 by showing that adding linear layers ("LinNet") as a general strategy of increasing the depth of the models improves meta-learning in both shallow models (§5.1) and deep models (§5.2). To further clarify the roles of the upper layers in deep models for meta-learning, we propose a new algorithm for meta-learning in §5.3, and conduct additional empirical studies. While this algorithm META-KFO is primarily used in this work for investigating how MAML works, it also attains state-of-the-art performances. We perform our study on common benchmarks datasets of meta-learning (for few-shot classification).

Datasets and settings In the following study, we use the standard 5-ways and 5-shots setting on the Omniglot (Lake et al., 2015), CIFAR-FS (Bertinetto et al., 2019), and mini-ImageNet (Vinyals et al., 2016) datasets. We denote by CNN(X) the convolutional network with X convolutional layer; for example, CNN(4) corresponds to the baseline network also used in Finn et al. (2017) and Raghu et al. (2020) among many others. To ensure fair comparison, we independently reimplemented each algorithmic variant and found the best hyper-parameter values for each architecture-algorithm pair via grid-search. For additional details, see the Suppl. Material.

5.1 Linear layers improve shallow linear models

Fig. 3 displays the results of meta-learning using MAML on two standard benchmark datasets with logistic/softmax regression models. The light blue horizontal lines denote the best performance if the models are trained by sufficient data from the meta-testing tasks. The black lines are the meta-learning performance, which only slightly improve upon chance levels (20% on Ominglot and CIFAR-FS).

However, as in Fig. 1, when linear layers are added to these linear models, the meta-learning performances are significantly improved.

Table 1: Accuracy Improves by Adding Linear Layers

Method	MAML				MAML w/ LinNet			
CNN Layers	2	3	4	6	2	3	4	6
Omniglot	66.8	93.5	98.5	97.6	88.1	95.5	98.1	97.6
CIFAR-FS	62.2	68.9	70.9	71.3	66.1	71.1	74.4	71.9
mini-ImageNet	52.6	54.0	64.1	64.6	60.5	60.2	64.9	64.1

Table 2: Accuracy Improves on ANIL Trained CNN(2)

Dataset	w/o LinNet	w/ LinNet
Omniglot	91.00	93.02
CIFAR-FS	66.10	67.55
mini-ImageNet	56.42	56.64

5.2 Linear layers improve deep nonlinear models

Table 1 lists the positive results of adding two linear layers to different CNN architectures. When the number of CNN layers are less than 6, the addition improves meta-learning performances. At CNN(6), there are degradations in performance by the original MAML on Omniglot and CIFAR-FS such that LinNet does not improve further. On mini-ImageNet, while MAML improves, LinNet decreases though its performance at CNN(4) is still the best.

Table 2 generalizes the positive findings in Table 1 to ANIL (Raghu et al., 2020). Thus, we believe LinNet is a broadly applicable strategy for improving meta-learning.

5.3 Meta-Optimizer for fast adaptation

Main idea It is straightforward to see that the added linear layers (LinNets) function similarly to the upper layers of deep learning models. The parameter updates for the bottom layers before such layers are modulated by the parameters in the upper layers or the LinNets. However, in what ways does this modulation help meta-learning?

Related to this question is meta-learning via learn-

ing to optimize, *i.e.*, transforming the gradients of the models Li et al. (2017); Park and Oliva (2019); Lee and Choi (2018); Flennerhag et al. (2019). Those types of preconditioning techniques could also be used to make a (smaller) model (more) meta-learnable. Thus, are the parameter updates in deep models equivalent to transformed gradient updates by such techniques? Note that there is a subtle difference: in some of these techniques (such as T-Nets and WarpGrad), the loss function used to compute the gradients with respect to the bottom layers *prior to* transformation actually contains the the transformation parameters themselves, cf. eq.(25) for an example. This type of "inline" transformations de facto increase the model capacity by injecting more parameters.

Our goal, however, is different and aims to disentangle the increase in model capacity from the ability to transform gradients. The empirical observation of this approach will enable us to answer the aforementioned question more clearly.

In the following we give a brief account of various approaches for learning to optimize and our proposed META-KFO algorithm. The details are in the the Suppl. Material. META-KFO is able to merely transforming the gradients of a smaller model without increasing its modeling capacity but still results in better meta-learnability. Furthermore, the improvement diminishes when the smaller model gets bigger. We surmise why sufficiently large deep models can meta-learn: *the upper layers have the equivalent effect of transforming the gradients of the bottom layers as if the upper layers were an external meta-optimizer, operating on a smaller network that is composed only of the bottom layers.*

META-KFO and other meta-optimizer/preconditioning methods A meta-optimizer is a parameterized function U_ξ defining the model's parameter updates. For example, a linear meta-optimizer might be defined as:

$$U_\xi(g) = Ag + b, \tag{21}$$

where $\xi = (A, b)$ is the set of parameters of the linear transformation. The objective is to jointly learn the model and optimizer's parameters ξ to accelerate optimization. Motivated by the analysis of meta-learning in deep nets, we propose to use such an optimizer to transform the gradient updates:

$$\mathcal{L}_{\text{MO}}^{\text{MAML}}(\theta) = \mathbb{E}_{\tau\sim p(\tau)}[\ell_\tau\left(\theta - \alpha U_\xi(\nabla\ell_\tau(\theta))\right)] \tag{22}$$

that takes a similar role of the upper-layers in deep nets in minimizing the MAML loss:

$$\theta \leftarrow \theta - \beta\frac{\partial\mathcal{L}_{\text{MO}}^{\text{MAML}}(\theta)}{\partial\theta}, \qquad \xi \leftarrow \xi - \beta\frac{\partial\mathcal{L}_{\text{MO}}^{\text{MAML}}(\theta)}{\partial\xi} \tag{23}$$

Table 3: Meta-Optimizers Outperform MAML on CNN(2)

Dataset	MAML	MAML w/			
		MSGD	MC	T-Nets	META-KFO
Omniglot	66.6	74.07	94.63	92.27	**96.62**
CIFAR-FS	62.2	62.82	68.37	66.42	**69.64**
mini-ImageNet	52.6	**59.90**	58.95	58.47	59.08

where β is the meta-update learning rate. In this notation, Meta-Curvature Park and Oliva (2019) implements

$$\text{(MC)} \quad U_\xi\left(\nabla\ell\left(\theta\right)\right) = M\nabla\ell\left(\theta\right) \tag{24}$$

where M is a matrix (block-diagnonal tensor factorized). When M is diagonal, this becomes the Meta-SGD Li et al. (2017). Furthermore, when M is identity, this become MAML. For T-Nets (to be used with MAML-loss), the model parameters are expanded with affine transformations,

$$\text{(T-NETS)} \quad \ell_\tau\left(\mathcal{A}(\theta) - \alpha\nabla\ell_\tau(\mathcal{A}(\theta))\right)) \tag{25}$$

where the transformation $\mathcal{A}(\cdot)$ contains two components $(\mathcal{W}, \mathcal{T})$. $\mathcal{T}$ is shared by all the tasks and $\mathcal{W}$ is task-specific. Since $\mathcal{A}$ is linear, it can be absorbed into the original model after adaptation. For WarpGrad (Flennerhag et al., 2019), the transformation $\mathcal{A}$ is defined with nonlinear layers, thus strictly increasing the size of the original model (thus, is not considered in this work).

Our method takes the form

$$\text{(META-KFO)} \quad U_\xi\left(\nabla\ell\left(\theta\right)\right) = f(\nabla\ell\left(\theta\right); \phi) \tag{26}$$

where $f(\cdot)$ is a nonlinear function parameterized by a set of parameters ϕ that is *independent* of the model's parameters θ. This approach generalizes MC (eq. (24)), as it is more adaptive since the gradient $\nabla\ell\left(\theta\right)$ is used as the inputs.

For models with a large number of parameters, the transformation U (ie, $\mathcal{A}$, M, and $f(\cdot)$) could contain a lot of parameters and incur high computational cost. For details, please refer to the cited references, and the the Suppl. Material on the details of META-KFO. Essentially, $f(\cdot)$ is parameterized with a neural network where the gradients $\nabla\ell\left(\theta\right)$ are manipluated with Kronecker products.

Results Table 3 contrasts different approaches for improving meta-learning by MAML on CNN(2) *without* increasing the size of the model after adaptation. All methods improve the original MAML while META-KFO improves the most on Omniglot and CIFAR-FS. On mini-ImageNet, all methods improve about the

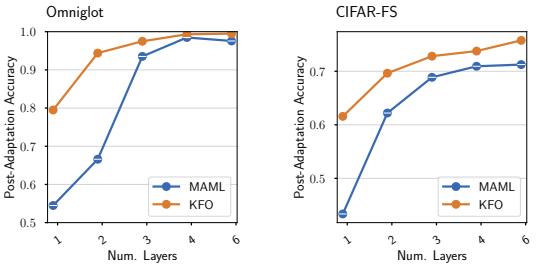

Figure 4: The effect of the number of convolutional layers on adaptation performance. First, as the model size increases, the performances of both methods improve. Besides better meta-learning, the improvement can also be caused by the model's increased capacity to learn the target tasks. Secondly, the "net gain" from the META-KFO has the diminishing trend as the size increases. In other words, the benefits of directly transforming gradients with an external meta-optimizer reduce as the upper layers of the larger models have more capacity to meta-learn to control their own bottom layers. Results on mini-ImageNet are available in the Suppl. Material.

same amount. In the Suppl. Material, we also contrast the meta-optimizers when ANIL is used to meta-learn. Again, all methods improve the baseline ANIL and META-KFO improves the most significant.

Fig. 4 examines the improvement of META-KFO over MAML with respect to the network size. As expected, META-KFO improves the most when the model is small and the improvement reduces when the model is sufficiently large. In other words, when the model is deep enough to meta-learn by itself using its top-layers to control the gradients of bottom layers, there is less advantage of using an external meta-optimizer to learn the bottom layers.

We view this as a strong evidence to support the theory that for deep neural networks that can meta-learn well, the upper layers have the equivalent effect of transforming the gradients of the bottom layers as if the upper layers were an external meta-optimizer, operating on a smaller network that is composed of the bottom layers. *They are the "external meta-optimizers that work from the inside."*.

6 Related Work

Understanding how MAML and its alike work continues to draw research interests (Finn and Levine, 2018; Fallah et al., 2019; Raghu et al., 2020; Saunshi et al., 2020). Many such studies have left open questions to be carefully analyzed, and hypotheses to be tested.

Finn and Levine (2018) showed that, when combined with deep architectures, GBML is able to approximate arbitrary meta-learning schemes. That work would have assumed the model is meta-learnable to begin with, relying on the argument that deep models are universal approximators. Fallah et al. (2019) provided convergence guarantees for MAML. Other analyses have attempted to explain the generalization ability of GBML (Guiroy et al., 2019; Nichol et al., 2018), the bias induced by restricting the number of adaptation steps (Wu et al., 2018), or the effect of higher-order terms in the meta-gradient estimation (Foerster et al., 2018; Rothfuss et al., 2019). Those work do not directly investigate what elements in deep models make them meta-learn well.

Raghu et al. (2020) suggested that the bottom layers of a neural network are for learning representations while the upper layers are responsible for the inductive bias to adapt fast. This observation echoes the success of some other approaches for meta-learning, such as ProtoNet (Snell et al., 2017) and MetaOptNet (Lee et al., 2019). But that work does not explain what is in the "magic" of the top layers to enable meta-learning.

We also investigate how adaptation could be provided by a meta-optimizer. Meta-SGD meta-learns per-parameter learning rates (Li et al., 2017) while Alpha MAML adapts those learning rates during adaptation via gradient-descent (Behl et al., 2019). Meta-Curvature learns a block-diagonal pre-conditioning matrix to compute fast-adaptation updates (Park and Oliva, 2019) and T-Nets extends that by decomposing all weight matrices of the model in two separate components (Lee and Choi, 2018). WarpGrad further extends T-Nets by allowing both components to be non-linear functions (Flennerhag et al., 2019).

The most salient difference of our work from existing ones is our focus on studying *what makes deep models meta-learnable*. Not only do we conclude being sufficiently deep is essential for meta-learning to succeed but we also theorize that the upper layers in the deep models essentially function as "embedded meta-optimizers". With extensive empirical studies, we complement the theoretical work in Saunshi et al. (2020) which suggests that deep models might attain a lower loss than shallow ones.

7 Conclusion

Are deep architectures necessary for meta-learning, even if the tasks can be solved with shallow (linear) networks? Our analysis suggests so. How does the depth benefit meta-learning? Our studies theorize that the upper layers of deep models function as external meta-optimizers that transform the gradients of a smaller network composed of only the bottom layers. We believe those observations will inspire new algorithms in future.

Acknowledgements

Fei Sha is on leave from University of Southern California. We appreciate the feedback from the reviewers. This work is partially supported by NSF Awards IIS-1513966/ 1632803/1833137, CCF-1139148, DARPA Award#: FA8750-18-2-0117, FA8750-19-1-0504, DARPA-D3M - Award UCB-00009528, Google Research Awards, gifts from Facebook and Netflix, and ARO# W911NF-12-1-0241 and W911NF-15-1-0484.

References

Arnold, S. M. R., Mahajan, P., Datta, D., Bunner, I., and Zarkias, K. S. (2020). learn2learn: A library for meta-learning research.

Baxter, J. (2000). A model of inductive bias learning. *J. Artif. Intell. Res.*, 12:149–198.

Behl, H. S., Baydin, A. G., and Torr, P. H. S. (2019). Alpha MAML: Adaptive Model-Agnostic Meta-Learning. *arXiv preprint arXiv:1905.07435*.

Bengio, Y., Bengio, S., and Cloutier, J. (1991). Learning a synaptic learning rule. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume ii, pages 969 vol.2–.

Bertinetto, L., Henriques, J. F., Torr, P., and Vedaldi, A. (2019). Meta-learning with differentiable closed-form solvers. In *International Conference on Learning Representations*.

Caruana, R. (1997). Multitask learning. *Mach. Learn.*, 28(1):41–75.

Fallah, A., Mokhtari, A., and Ozdaglar, A. (2019). On the convergence theory of Gradient-Based Model-Agnostic Meta-Learning algorithms. *arXiv preprint arXiv:1908.10400*.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-Agnostic Meta-Learning for fast adaptation of deep networks. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135, International Convention Centre, Sydney, Australia. PMLR.

Finn, C. and Levine, S. (2018). Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. In *International Conference on Learning Representations*.

Flennerhag, S., Rusu, A. A., Pascanu, R., Yin, H., and Hadsell, R. (2019). Meta-Learning with warped gradient descent. *arXiv preprint arXiv:1909.00025*.

Foerster, J., Farquhar, G., Al-Shedivat, M., Rocktäschel, T., Xing, E., and Whiteson, S. (2018). Dice: The infinitely differentiable monte carlo estimator. In *International Conference on Machine Learning*, pages 1524–1533.

Gidel, G., Bach, F., and Lacoste-Julien, S. (2019). Implicit regularization of discrete gradient dynamics in linear neural networks.

Grant, E., Finn, C., Levine, S., Darrell, T., and Griffiths, T. (2018). Recasting gradient-based meta-learning as hierarchical bayes. In *International Conference on Learning Representations*.

Guiroy, S., Verma, V., and Pal, C. (2019). Towards understanding generalization in Gradient-Based Meta-Learning. *arXiv preprint arXiv:1907.07287*.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., Del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science Engineering*, 9(3):90–95.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338.

Lee, K., Maji, S., Ravichandran, A., and Soatto, S. (2019). Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10657–10665.

Lee, Y. and Choi, S. (2018). Gradient-based meta-learning with learned layerwise metric and subspace. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2927–2936, Stockholmsmässan, Stockholm Sweden. PMLR.

Li, Z., Zhou, F., Chen, F., and Li, H. (2017). Meta-SGD: Learning to learn quickly for Few-Shot learning. *arXiv preprint arXiv:1707.09835*.

Nichol, A., Achiam, J., and Schulman, J. (2018). On First-Order Meta-Learning algorithms. *arXiv preprint arXiv:1803.02999*.

Park, E. and Oliva, J. B. (2019). Meta-Curvature. *arXiv preprint arXiv:1902.03356*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). PyTorch: An imperative style, High-Performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alche Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8026–8037. Curran Associates, Inc.

Raghu, A., Raghu, M., Bengio, S., and Vinyals, O. (2020). Rapid learning or feature reuse? towards

understanding the effectiveness of {maml}. In *International Conference on Learning Representations*.

Rothfuss, J., Lee, D., Clavera, I., Asfour, T., and Abbeel, P. (2019). ProMP: Proximal meta-policy search. In *International Conference on Learning Representations*.

Saunshi, N., Zhang, Y., Khodak, M., and Arora, S. (2020). A sample complexity separation between Non-Convex and convex Meta-Learning.

Saxe, A. M., McClelland, J. L., and Ganguli, S. (2019). A mathematical theory of semantic development in deep neural networks. *Proc. Natl. Acad. Sci. U. S. A.*, 116(23):11537–11546.

Schmidhuber, J. (1987). *Evolutionary Principles in Self-Referential Learning*. PhD thesis.

Snell, J., Swersky, K., and Zemel, R. (2017). Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pages 4077–4087.

Van Rossum, G. and Drake, Jr, F. L. (1995). *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam.

Vanschoren, J. (2019). Meta-Learning. In Hutter, F., Kotthoff, L., and Vanschoren, J., editors, *Automated Machine Learning: Methods, Systems, Challenges*, pages 35–61. Springer International Publishing, Cham.

Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., and Wierstra, D. (2016). Matching networks for one shot learning. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 3630–3638. Curran Associates, Inc.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020). SciPy 1.0: fundamental algorithms for scientific computing in python. *Nat. Methods*, 17(3):261–272.

Wu, Y., Ren, M., Liao, R., and Grosse., R. (2018). Understanding short-horizon bias in stochastic meta-optimization. In *International Conference on Learning Representations*.

Appendix

A Theoretical Analysis of Linear Models for Meta-Learning

This section provides more details to §4 in the main text.

A.1 Analytic Solution of 1D Linear Regression

We use

$$\theta_\alpha = \theta - \alpha \nabla \ell_\tau(\theta) \tag{27}$$

as a short-hand. The gradient of the MAML loss $\mathcal{L}^{\text{MAML}}$ is given by

$$\frac{\partial \mathcal{L}^{\text{MAML}}}{\partial \theta} = \mathbb{E}_\tau(\boldsymbol{I} - \alpha H_\tau(\theta)) \left.\frac{\partial \ell_\tau}{\partial \theta}\right|_{\theta - \alpha \nabla \ell_\tau} = \mathbb{E}_\tau(I - \alpha H_\tau(\theta)) \nabla \ell_\tau(\theta_\alpha) \tag{28}$$

We setup the problem as follows. Let the task parameter $\theta_\tau \sim N(0,1)$ be a normal distributed scalar. Let the covariate $x \sim N(0,1)$ be a normal distributed scalar and the outcome be $y \sim N(\theta_\tau x, 1)$. We investigate two models for their meta-learning performance:

$$\text{SHALLOW:} \quad \hat{y} = cx, \quad \text{DEEP:} \quad \hat{y} = abx \tag{29}$$

Note that the "deep" model is overparameterized. For each task, we use the least-square loss

$$\ell_\tau = \frac{1}{2}\mathbb{E}_{p(x,y|\theta_\tau)}(y - cx)^2, \quad \text{or} \quad \ell_\tau = \frac{1}{2}\mathbb{E}_{p(x,y|\theta_\tau)}(y - abx)^2 \tag{30}$$

A.1.1 Shallow Model

For the shallow model, the gradient is

$$\nabla \ell_\tau = \mathbb{E}_{p(x,y|\theta_\tau)}(cx - y)x = c - \theta_\tau \tag{31}$$

Thus, the loss for the τ-th task is given by

$$\ell_\tau(c - \alpha \nabla \ell_\tau) = \mathbb{E}_{p(x,y|\theta_\tau)}(y - [(1-\alpha)c + \alpha\theta_\tau]x)^2 \tag{32}$$

$$= [(1-\alpha)c + \alpha\theta_\tau]^2 - 2[(1-\alpha)c + \alpha\theta_\tau]\theta_\tau \tag{33}$$

Simplifying and completing the square on the θ_τ^2 term with constants, we arrive at

$$\ell_\tau(c - \alpha \nabla \ell_\tau) = (1-\alpha)^2(c - \theta_\tau)^2 + \text{CONST} \tag{34}$$

Taking the expectation of the loss with respect to $p(\tau)$, we have

$$\mathcal{L}^{\text{MAML}}_{\text{SHALLOW}} = 2(1-\alpha)^2 c^2 + \text{CONST} \tag{35}$$

A.1.2 Deep Model

The gradients of the linear regression loss w.r.t a, b are given by:

$$\frac{\partial l_\tau}{\partial a} = ab^2 - b\theta_\tau \tag{36}$$

$$\frac{\partial l_\tau}{\partial b} = a^2 b - a\theta_\tau \tag{37}$$

The post-adaptation parameters are obtained by taking one step of gradient descent with learning rate α:

$$a \leftarrow a - \alpha b\,(ab - \theta_\tau) \tag{38}$$
$$b \leftarrow b - \alpha a\,(ab - \theta_\tau) \tag{39}$$

This gives us the expression for the MAML loss:

$$\begin{aligned}
\mathcal{L}^{\text{MAML}}_{\text{DEEP}} &= \frac{1}{2}\mathop{\mathbb{E}}_{\tau}\mathop{\mathbb{E}}_{x,y}\left[y - (a - \alpha b(ab - \theta_\tau))\cdot(b - \alpha a(ab - \theta_\tau))x\right]^2 && (40)\\
&= \frac{1}{2}\mathop{\mathbb{E}}_{\tau}\mathop{\mathbb{E}}_{x,y}\left[y - f(a,b,\theta_\tau)x\right]^2 && (41)\\
&= \frac{1}{2}\mathop{\mathbb{E}}_{\tau}\left[\theta_\tau^2 - 2\theta_\tau + f^2(a,b,\theta_\tau)\right] && (42)\\
&= \frac{1}{2} + \frac{1}{2}\mathop{\mathbb{E}}_{\tau} f^2(a,b,\theta_\tau) - \frac{1}{2}\mathop{\mathbb{E}}_{\tau}\theta_\tau f(a,b,\theta_\tau) && (43)
\end{aligned}$$

where $f(a,b,\theta_\tau) = (a - \alpha b(ab - \theta_\tau))\cdot(b - \alpha a(ab - \theta_\tau))$.
We now take a closer look at the term $\mathbb{E}\, f^2$:

$$\begin{aligned}
f^2(a,b,\theta_\tau) &= \left[ab - \alpha a^2(ab - \theta_\tau) - \alpha b^2(ab - \theta_\tau) + \alpha^2 ab(ab - \theta_\tau)^2\right]^2 && (44)\\
&= \left[ab - \alpha(a^2 + b^2)(ab - \theta_\tau) + \alpha^2(ab - \theta_\tau)^2\right]^2 && (45)\\
&= \left[ab - \alpha(a^2 + b^2)ab + \alpha^2 a^3 b^3 + (\alpha(a^2 + b^2) - 2\alpha^2 a^2 b^2)\theta_\tau + \alpha^2 ab\theta_\tau^2\right]^2 && (46)\\
&= \left[p_1 + p_2\theta_\tau + p_3\theta_\tau^2\right]^2 && (47)\\
& && (48)
\end{aligned}$$

where we let:

$$p_1 = ab - \alpha(a^2 + b^2)ab + \alpha^2 a^3 b^3, \qquad p_2 = \alpha(a^2 + b^2) - 2\alpha^2 a^2 b^2, \qquad p_3 = \alpha^2 ab \tag{49}$$

We then obtain

$$\mathbb{E}\, f^2(a,b,\theta_\tau) = p_1^2 + p_2^2 + 3p_3^2 + 2p_1 p_3 \tag{50}$$

and where the last equality is obtained be remembering the expectation properties of θ_τ:

$$\mathbb{E}_{p(\tau)}\,\theta_\tau^2 = 1, \qquad \mathbb{E}_{p(\tau)}\,\theta_\tau^3 = 0, \qquad \mathbb{E}_{p(\tau)}\,\theta_\tau^4 = 3. \tag{51}$$

Furthermore, we obtain

$$\mathbb{E}\,\theta_\tau f(a,b,\theta_\tau) = p_2 \tag{52}$$

This gives us a final expression for the MAML loss:

$$\mathcal{L}^{\text{MAML}}_{\text{DEEP}} = \frac{1}{2}(1 + p_1^2 + p_2^2 + 3p_3^2 + 2p_1 p_3 - 2p_2). \tag{53}$$

We use the Matlab Symbolic Toolbox to help us simplify the rest of the calculation. The code is given below

```
%% We use sa, sb, salpha as the "symbolic version" of a, b, and alpha
syms sa sb salpha

pp1 = sa*sb*(1- salpha*(sa^2+sb^2)+ salpha^2*sa^2*sb^2);
pp2 = salpha*(sa^2+sb^2) - 2 *salpha^2 *sa^2*sb^2;
pp3 = salpha^2*sa*sb;

L = pp1*pp1 + pp2*pp2 + 3*pp3*pp3 + 2*pp1*pp3 -2 *pp2;
diffa = diff(L, sa);
diffb = diff(L, sb);

H = [diff(diffa, sa) diff(diffa, sb);diff(diffb, sa) diff(diffb, sb)];
```

Stationary Points The gradients are complicated polynomials in a and b. Instead, we give the specific cases:

$$\left.\frac{\partial \mathcal{L}_{\text{DEEP}}^{\text{MAML}}}{\partial a}\right|_{b=0} = 4\alpha a(\alpha a^2 - 1), \quad \left.\frac{\partial \mathcal{L}_{\text{DEEP}}^{\text{MAML}}}{\partial b}\right|_{b=0} = 0 \tag{54}$$

We immediately can derive that there are at least 5 stationary points:

- $(a = 0, b = 0)$, with Hessian

$$H = \begin{bmatrix} -4\alpha & 0 \\ 0 & -4\alpha \end{bmatrix} \tag{55}$$

- $(a = \pm\frac{1}{\sqrt{\alpha}}, 0)$, with Hessian

$$H = \begin{bmatrix} 8\alpha & 0 \\ 0 & 6\alpha^3 \end{bmatrix} \tag{56}$$

- $(a = 0, b = \pm\frac{1}{\sqrt{\alpha}})$, with Hessian

$$H = \begin{bmatrix} 6\alpha^3 & 0 \\ 0 & 8\alpha \end{bmatrix} \tag{57}$$

B Supplementary Experiments

This section provides additional experimental evidence to complement the evidence presented in the main text.

B.1 Binary Logistic Regression

To resonate more with our experiments of using (multinominal) logistic regression models on Omniglot, CIFAR and MNIST datasets, we also analyze binary logistic regression models on synthetic data.

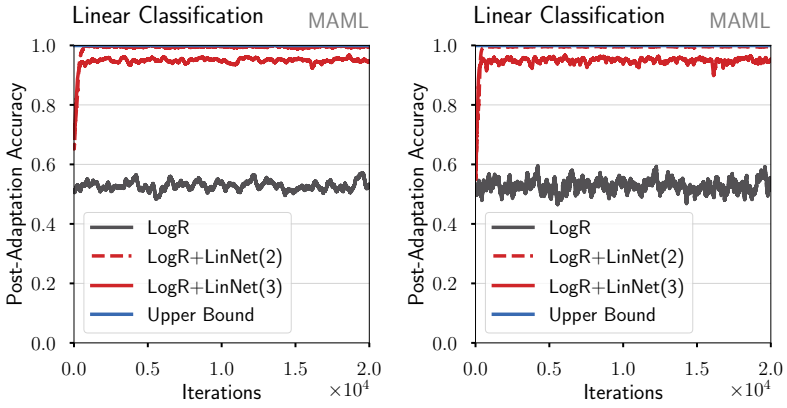

Figure B.1: Meta-learning of linear models on synthetic data. For linear separable data, MAML fails on logistic regression but succeed on logistic regression augmented with 2 or 3 linear layers. **(Left)** Meta-train accuracies. **(Right)** Meta-test accuracies.

B.1.1 MAML on Logistic Regression and Logistic Regression with Linear Layers

We randomly sample a set of 2-dimensional task parameters $\theta_\tau \in \mathbb{R}^2$ from a standard multivariate spherical Gaussian and use each of them to define a linear decision boundary of a binary classification task. We sample inputs from a 2-dimensional multivariate spherical Gaussian and the binary outputs for each task are sampled from $y \sim \sigma(\theta_\tau^{\mathrm{T}} \boldsymbol{x})$, where $\sigma()$ is the sigmoid function.

By construction, a logistic regression (LR) is sufficient to achieve very high accuracy on any task. But can MAML learn a linear classifier from a randomly sampled subset of training tasks that is able to adapt quickly

to the test tasks? It is intuitive to see the minimizer for the MAML loss $\mathcal{L}^{\text{MAML}}$ is the origin due to the rotation invariances of both the task parameters and the inputs. The origin thus provides the best initialization to adapt to new tasks by not favoring any particular task.

Fig. B.1(Left) reports the 1-step post-adaptation accuracy on the test tasks for the meta-learned logistic regression model. Equally surprising as the previous results, the model fails to perform better than chance. However, adding linear layers "rescues" the meta-learning; LR with 2 or 3 linear layers attains nearly perfect classification on any task.

B.2 Logistic Regression Failure Modes

As argued in §5.1, shallow logistic regression models fail to meta-learn on Omniglot and CIFAR-FS. Fig. B.2 displays this same failure mode on the mini-ImageNet dataset. To simplify computations, we downscaled the original mini-ImageNet images (84x84 pixels, with RGB channels) to the same size as CIFAR-FS. (32x32 pixels, with RGB channels)

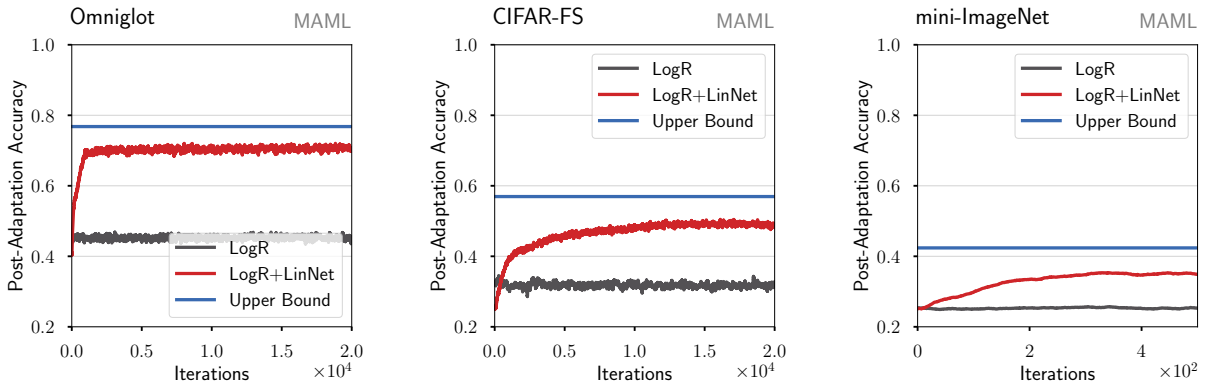

Figure B.2: Meta-training logistic regression models with MAML on Omniglot, CIFAR-FS, and mini-ImageNet led to poor performances. Adding linear nets improves meta-learning significantly, *without* changing the model's capacity.

B.3 Linear Layers Cannot Be Collapsed

Our previous results have shown that when MAML cannot meta-train well, adding multiple linear layers helps to meta-learn, cf. Fig. B.1 and Fig. 1 of the main text.

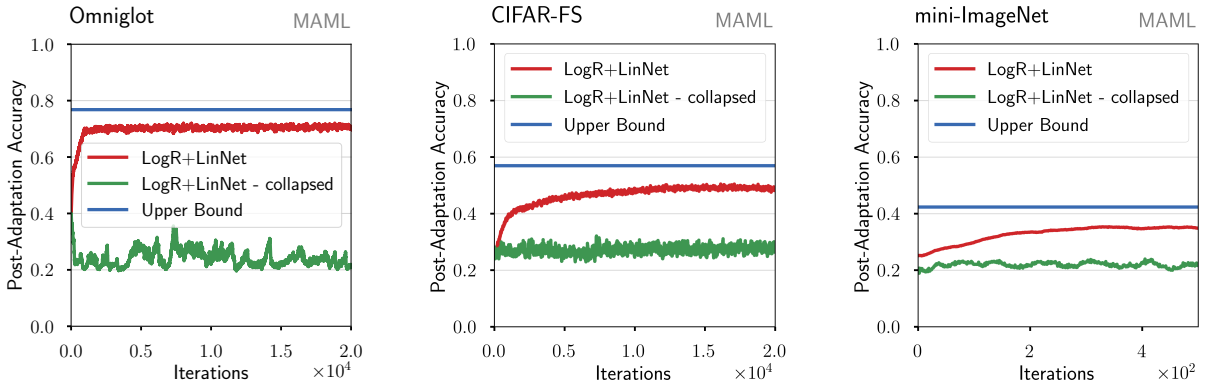

Figure B.3: Collapsing multiple linear layers into a single layer before adapting to new tasks (green curves) leads to poor adaptation results.

But after meta-training, can we collapse those multiple linear layers into a single linear layer so as to reduce the model size?

Fig. B.3 shows that collapsing erases the model's ability to adapt to new tasks. This suggests that **the solutions identified in models with additional linear layers need to stay in the overparameterized space to be effective and they cannot be collapsed before adaptation**.

B.4 Does SGD-induced implicit regularization for linear networks matter for meta-learning?

Implicit regularization refers to the bias towards solutions with different generalization abilities when stochastic gradient descent (SGD) is used to optimize overparameterized models Gidel et al. (2019).

One might want to argue that implicit regularization explains why additional linear layers in our experiments (Fig. 1 of the main text, Fig. B.1) help to meta-learn. This is an interesting hypothesis, though we do not think it can fully explain what is observed.

First, as those experiments have shown, for models augmented with linear nets, there is very little difference between training accuracies and meta-testing accuracies. Moreover, for models without linear nets, there is very little difference between the two phases either. Thus, while SGD could lead to solutions with different generalization abilities in regular supervised learning settings, our experiments do not show there is an overfitting issue in the experiments here for meta-learning.

The difference between parsimonious and overparameterized models (with linear layers) could be due to the difference in solutions. However, the solutions are in space of different dimensionality. Fig. B.3 shows that **after** we collapse the overparameterize models such that the solutions are in the same space, the solutions from the collapsing becomes as ineffective (in terms of leading to poor adaptation results) as the original models. This suggests that SGD is **not** identifying a solution in the overparameterized space that could have been identified in the original space. In other words, SGD is not biasing towards any solution in the overparameterized space that could have made difference in the original space.

B.5 ANIL with Meta-Optimizers

In §5.3, we showed that MAML augmented with meta-optimizers was able to meta-learn in shallow non-linear networks. Table B.4 shows similar results when using ANIL as the meta-learning algorithm. (Note: this table omits T-Nets, as it is not compatible with ANIL.) We observe that ANIL always benefits from the combination with a meta-optimizer, and specifically that it benefits most from KFO.

Table B.4: Meta-Optimizers Improve ANIL Meta-Learnability on CNN(2)

Dataset	ANIL	ANIL w/		
		MSGD	MC	KFO
Omniglot	91.00	92.47	92.67	**94.40**
CIFAR-FS	66.10	67.55	67.43	**68.81**
mini-ImageNet	56.42	57.45	59.07	**62.65**

B.6 Effect of Number of Layers

Figure B.4 complements the results in §5.3 when varying the number of layers in the convolutional network. (We include again results on Omniglot and CIFAR-FS for convenience.) As we observed in the main text, shallower models greatly benefit from the additional meta-optimization parameters which alleviates the burden of learning how to adapt. But, this benefit dampens as we increase the number of convolutional layers – the gap between META-KFO and MAML shrinks – since the additional upper layers implicitly transform the gradient of lower layers, thus enabling successful meta-learning.

C Details on KFO

C.1 Other factorization schemes

A technical issue arises when expressing the meta-optimizer U_ξ as a neural network: the dimensionality of modern (model) network architectures ranges in the tens of thousands, if not more. To address those computational

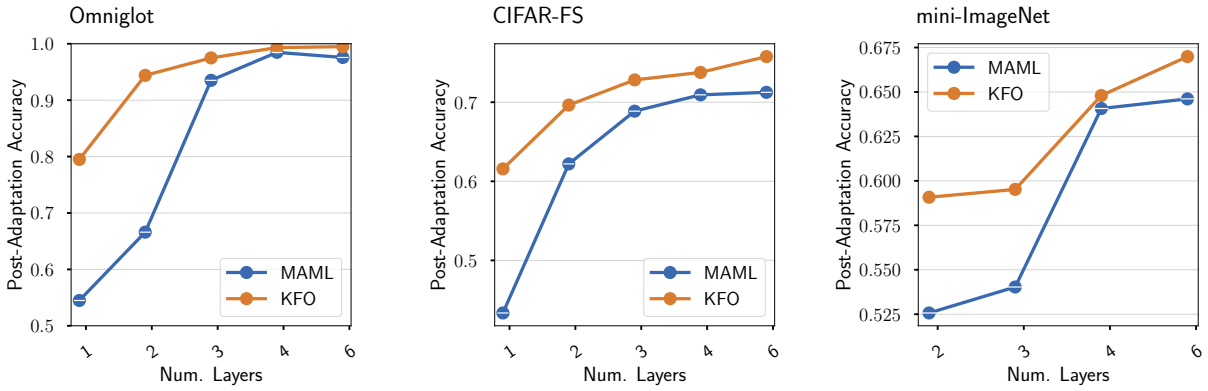

Figure B.4: Complementing Fig. 4, we vary the number of convolutional layers in the model. As the model size increases the performance of both methods improves, which can be attributed to the model's increased capacity to learn the target task. However, the net gain from META-KFO over MAML has diminishing returns as the number of layers increases since the benefit of the external meta-optimizer reduces as the upper layers of the larger models have more capacity to meta-learn to control their own bottom layers.

and memory issues, we learn one meta-optimizer network per matrix parameter in the model network and express the weights of the optimizer neural network as a Kronecker factorization such that for each weight $W \in \mathbb{R}^{m \times n} : W = R^\top \otimes L$, where $R \in \mathbb{R}^{m \times m}$ and $L \in \mathbb{R}^{n \times n}$.

Many matrix factorization schemes could be used and would result in different modeling and computational trade-offs. For example, using a low-rank Cholesky factorization $A = LL^T$ where $L \in \mathbb{R}^{k \times r}$ allows to interpolate between computational complexity and decomposition rank by tuning the additional hyper-parameter r. The Cholesky decomposition might be preferable to the Kronecker one in memory-constrained applications, since r can be used to control the memory requirements of the meta-optimizer. Moreover, such a decomposition imposes symmetry and positiveness on A, which might be desirable when approximating the Hessian or its inverse.

In this work, we preferred the Kronecker decomposition over alternatives for three reasons: (1) the computational and memory cost of the Kronecker-product are acceptable, (2) $R^\top \otimes L$ is full-rank whenever L, R are full-rank, and (3) the identity matrix lies in the span of Kronecker-factored matrices. In particular, this last motivation allows meta-optimizers to recover the gradient descent update by letting R, L be the identity.

C.1.1 Schematics and Pseudo-code

Pseudo-code for meta-optimizers is provided in Algorithm 1, and a schematic of the model-optimizer loop in Figure C.5.

Algorithm 1 Meta-Learning with Meta-Optimizers

Require: Fast learning rate α, Initial parameters $\theta^{(\text{init})}$ and $\xi^{(\text{init})}$, Optimizer `Opt`

while $\theta^{(\text{init})}, \xi^{(\text{init})}$ not converged **do**
  Sample task $\tau \sim p(\tau)$
  $\theta_1 = \theta^{(\text{init})}, \quad \xi_1 = \xi^{(\text{init})}$
  for step $t = 1, \dots, T$ **do**
    Compute loss $\ell_\tau(\theta_t)$
    Compute gradients $\nabla_{\theta_t} \ell_\tau(\theta_t)$ and $\nabla_{\xi_t} \ell_\tau(\theta_t)$
    Update the meta-optimizer parameters $\xi_{t+1} = \xi_t - \alpha \nabla_{\xi_t} \ell_\tau(\theta_t)$
    Compute the model update $U_{\xi_{t+1}}(\nabla_{\theta_t} \ell_\tau(\theta_t))$
    Update the model parameters $\theta_{t+1} = \theta_t - U_{\xi_{t+1}}(\nabla_{\theta_t} \ell_\tau(\theta_t))$
  Update model and meta-optimizer initializations
    $\theta^{(\text{init})} \leftarrow \theta^{(\text{init})} - \texttt{Opt}(\nabla_{\theta^{(\text{init})}} \ell_\tau(\theta_T))$
    $\xi^{(\text{init})} \leftarrow \xi^{(\text{init})} - \texttt{Opt}(\nabla_{\xi^{(\text{init})}} \ell_\tau(\theta_T))$

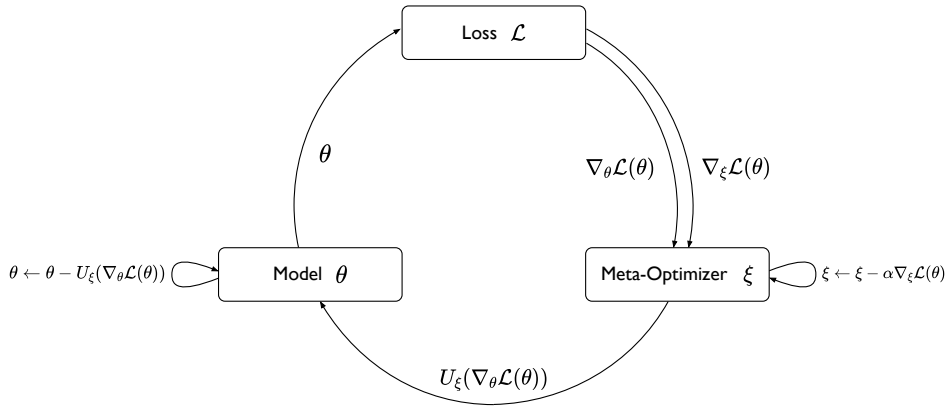

Figure C.5: Schematic of model-optimizer loop

Table D.5: **Adaptation learning rates** for the linear model experiments in the main text and the Suppl. Material

Dataset	Model	Meta learning rate	Adaptation learning rate
Synthetic	LR	0.01	0.900
Synthetic	LR+LinNet(2)	0.1	0.900
Synthetic	LR+LinNet(3)	0.1	0.900
Omniglot	LR	0.0005	1.0
Omniglot	LR+LinNet	0.0005	0.08
CIFAR-FS	LR	0.0005	0.01
CIFAR-FS	LR+LinNet	0.0001	0.02
mini-ImageNet	LR	0.0005	0.01
mini-ImageNet	LR+LinNet	0.0005	0.01

D Hyperparameters and details for reproducibility

We report the hyper-parameter values for the experiments of the main text. Each experimental setup is tuned independently based on post-adaptation accuracies obtained on validation tasks.

All experiments are implemented on top of PyTorch Paszke et al. (2019) and learn2learn Arnold et al. (2020). Moreover, our work relied on tools from the Python scientific ecosystem Van Rossum and Drake (1995), including numpy Hunter (2007), matplotlib Harris et al. (2020), and scipy Virtanen et al. (2020). We provide example implementations of Meta-KFO at: `https://github.com/Sha-Lab/kfo`

Linear Models On the 1D linear regression, both SHALLOW and DEEP models use a fast-adaptation learning rate set to 0.1. Their parameters are optimized by SGD with a meta learning rate set to 0.01, and a momentum term set to 0.9. On the vision datasets (Omniglot, CIFAR-FS, mini-ImageNet), the meta and adaptation learning rates are given in Table D.5. For those experiments, the added linear network consists of 256-128-64-64 hidden units. To simplify computations on mini-ImageNet, we downsample images to 32x32 pixels.

Nonlinear Models All methods in Tables 1, 2, and 3 are tested on the standard 5-ways 5-shots setting. For Omniglot and CIFAR-FS, the networks are the original CNN from Finn et al. (2017) with 32 hidden units, while for mini-ImageNet we used 64 hidden units. We use Adam to optimize the networks, with default values of (0.9, 0.999) for β and 1e-8 for ϵ, and process tasks by batches of 16. Meta and adaptation learning rates for all methods are reported in Tables D.6, D.7.

Table D.6: **Adaptation learning rates** for the two-layer CNN experiments in the main text and the Suppl. Material

	MAML w/					ANIL w/				
Dataset	(null)	LinNet	MSGD	MC	KFO	(null)	LinNet	MSGD	MC	KFO
Omniglot	0.08	0.05	0.5	0.5	0.1	0.5	0.5	0.05	0.05	0.1
CIFAR-FS	0.07	0.01	0.7	0.7	0.1	0.5	0.5	0.1	0.1	0.1
mini-ImageNet	0.001	0.001	0.1	0.05	0.001	0.5	0.5	0.1	0.1	0.1

Table D.7: **Meta learning rates** for the two-layer CNN experiments in the main text and the Suppl. Material

	MAML w/					ANIL w/				
Dataset	(null)	LinNet	MSGD	MC	KFO	(null)	LinNet	MSGD	MC	KFO
Omniglot	0.003	0.003	0.0005	0.0005	0.001	0.001	0.003	0.001	0.001	0.001
CIFAR-FS	0.003	0.003	0.001	0.003	0.003	0.002	0.001	0.001	0.001	0.001
mini-ImageNet	0.0005	0.0005	0.001	0.001	0.0001	0.001	0.001	0.001	0.001	0.001