Noname manuscript No.
(will be inserted by the editor)

Stateless Neural Meta-Learning using Second-Order Gradients

Mike Huisman · Aske Plaat · Jan N. van Rijn

Received: date / Accepted: date

Abstract Meta-learning can be used to learn a good prior that facilitates quick learning; two popular approaches are MAML and the meta-learner LSTM. These two methods represent important and different approaches in meta-learning. In this work, we study the two and formally show that the meta-learner LSTM subsumes MAML, although MAML, which is in this sense less general, outperforms the other. We suggest the reason for this surprising performance gap is related to second-order gradients. We construct a new algorithm (named TURTLE) to gain more insight into the importance of second-order gradients. TURTLE is simpler than the meta-learner LSTM yet more expressive than MAML and outperforms both techniques at few-shot sine wave regression and 50% of the tested image classification settings (without any additional hyperparameter tuning) and is competitive otherwise, at a computational cost that is comparable to second-order MAML. We find that second-order gradients also significantly increase the accuracy of the meta-learner LSTM. When MAML was introduced, one of its remarkable features was the use of second-order gradients. Subsequent work focused on cheaper first-order approximations. On the basis of our findings, we argue for more attention for second-order gradients.

Keywords Meta-learning · Few-shot learning · Deep learning · Transfer Learning

Mathematics Subject Classification (2020) 68T07 · 68T45

Mike Huisman · Aske Plaat · Jan N. van Rijn
LIACS, Leiden University, The Netherlands
Niels Bohrweg 1, 2333CA, Leiden
E-mail: m.huisman@liacs.leidenuniv.nl

1 Introduction

Humans learn new tasks quickly. While deep neural networks have demonstrated human or even super-human performance on various tasks such as image recognition [Krizhevsky et al., 2012, He et al., 2015] and game-playing [Mnih et al., 2015, Silver et al., 2016], learning a new task is often slow and requires large amounts of data [LeCun et al., 2015]. This limits their applicability in real-world domains where few data and limited computational resources are available.

Meta-learning [Schmidhuber, 1987, Schaul and Schmidhuber, 2010] is one approach to address this issue. The idea is to learn at two different levels of abstraction: at the *outer-level* (across tasks), we learn a prior that facilitates faster learning at the *inner-level* (single task) [Vilalta and Drissi, 2002, Vanschoren, 2018, Hospedales et al., 2020, Huisman et al., 2021]. The prior that we learn at the outer-level can take on many different forms, such as the learning rule [Andrychowicz et al., 2016, Ravi and Larochelle, 2017] and the weight initialization [Nichol et al., 2018, Finn et al., 2017].

MAML [Finn et al., 2017] and the meta-learner LSTM [Ravi and Larochelle, 2017] are two well-known techniques that focus on these two types of priors. More specifically, MAML aims to learn a good weight initialization from which it can learn new tasks quickly using regular gradient descent. In addition to learning a good weight initialization, the meta-learner LSTM [Ravi and Larochelle, 2017] attempts to learn the optimization procedure in the form of a separate LSTM network. The meta-learner LSTM is more general than MAML in the sense that the LSTM can learn to perform gradient descent (see Section 4) or something better.

This suggests that the performance of MAML can be mimicked by the meta-learner LSTM on few-shot image classification. However, our experimental results and those by Finn et al. [2017] show that this is not necessarily the case. The meta-learner LSTM fails to find a solution in the *meta-landscape* that learns as well as gradient descent.

In this work, we aim to investigate the performance gap between MAML and the meta-learner LSTM. We hypothesize that the underperformance of the meta-learner LSTM could be caused by (i) the lack of second-order gradients, or (ii) the fact that an LSTM is used as an optimizer. To investigate these hypotheses, we introduce TURTLE, which is similar to the meta-learner LSTM but uses a fully-connected feed-forward network as an optimizer instead of an LSTM and, in addition, uses second-order gradients. Although both MAML and the meta-learner LSTM are by now surpassed by other state-of-the-art techniques, such as LEO [Rusu et al., 2019] and MetaOptNet [Lee et al., 2019] (see Section 2), they are still relevant and widely used. The aim of this paper is to gain insight into the performance gap between the meta-learner LSTM and MAML. Our contributions are:

- We formally show that the meta-learner LSTM subsumes MAML.
- We formulate a new meta-learning algorithm called TURTLE to overcome two potential shortcomings of the meta-learner LSTM. We demonstrate

that TURTLE successfully closes the performance gap to MAML as it outperforms MAML (and the meta-learner LSTM) on sine wave regression, and various settings involving miniImageNet and CUB by at least 1% accuracy without any additional hyperparameter tuning. TURTLE requires roughly the same amount of computation time as second-order MAML.
- Based on the results of TURTLE, we enhance the meta-learner LSTM by using raw gradients as meta-learner input and second-order information and show these changes result in a performance boost of 1-6% accuracy, indicating the importance of second-order gradients.

2 Related work

The success of deep learning techniques has been largely limited to domains where abundant data and large compute resources are available [LeCun et al., 2015]. The reason for this is that learning a new task requires large amounts of resources. Meta-learning is an approach that holds the promise of relaxing these requirements by learning to learn. The field has attracted much attention in recent years, resulting in many new techniques, which can be divided into metric-based, model-based, and optimization-based approaches [Huisman et al., 2021]. In our work, we focus on an optimization-based approach, which includes both MAML and the meta-learner LSTM (see Figure 1).

MAML [Finn et al., 2017] aims to find a good weight initialization from which new tasks can be learned quickly within several gradient update steps. As shown in Figure 1, many works build upon the key idea of MAML, for example, to decrease the computational costs [Nichol et al., 2018, Rajeswaran et al., 2019], increase the applicability to online and active learning settings [Grant et al., 2018, Finn et al., 2018], or increase the expressivity of the algorithm [Li et al., 2017, Park and Oliva, 2019, Lee and Choi, 2018]. Despite its popularity, MAML does no longer yield state-of-the-art performance on few-shot learning benchmarks [Lu et al., 2020], as it is surpassed by, for example, latent embedding optimization (LEO) [Rusu et al., 2019] which optimizes the initial weights in a lower-dimensional latent space, and MetaOptNet [Lee et al., 2019], which stacks a convex model on top of the meta-learned initialization of a high-dimensional feature extractor. Although these approaches achieve state-of-the-art techniques on few-shot benchmarks, MAML is elegant and generally applicable as it can also be used in reinforcement learning settings [Finn et al., 2017].

While the meta-learner LSTM [Ravi and Larochelle, 2017] learns both an initialization and an optimization procedure, it is generally hard to properly train the optimizer [Metz et al., 2019]. As a result, techniques that use hand-crafted learning rules instead of trainable optimizers may yield better performance. It is perhaps for this reason that most meta-learning algorithms use simple, hand-crafted optimization procedures to learn new tasks, such as regular gradient descent [Bottou, 2004], Adam [Kingma and Ba, 2015], or RMSprop [Tieleman and Hinton, 2017]. Andrychowicz et al. [2016], show that

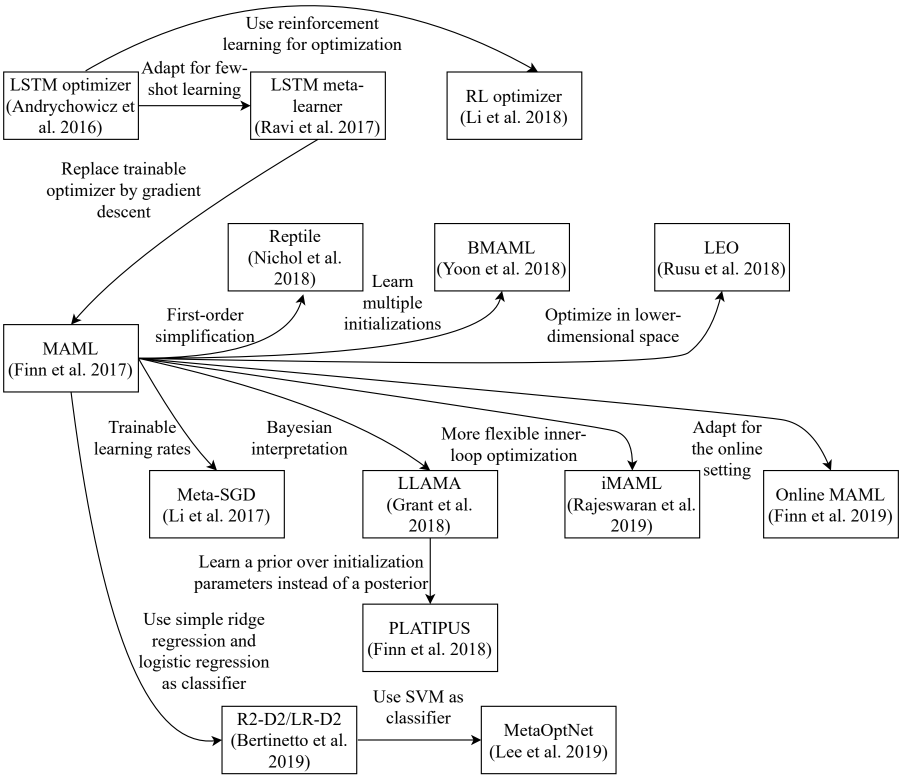

Use reinforcement learning for optimization
LSTM optimizer (Andrychowicz et al. 2016)
Adapt for few-shot learning
LSTM meta-learner (Ravi et al. 2017)
RL optimizer (Li et al. 2018)
Replace trainable optimizer by gradient descent
Reptile (Nichol et al. 2018)
BMAML (Yoon et al. 2018)
LEO (Rusu et al. 2018)
Learn multiple initializations
First-order simplification
Optimize in lower-dimensional space
MAML (Finn et al. 2017)
Trainable learning rates
Bayesian interpretation
More flexible inner-loop optimization
Adapt for the online setting
Meta-SGD (Li et al. 2017)
LLAMA (Grant et al. 2018)
iMAML (Rajeswaran et al. 2019)
Online MAML (Finn et al. 2019)
Learn a prior over initialization parameters instead of a posterior
PLATIPUS (Finn et al. 2018)
Use simple ridge regression and logistic regression as classifier
Use SVM as classifier
R2-D2/LR-D2 (Bertinetto et al. 2019)
MetaOptNet (Lee et al. 2019)

Fig. 1: An overview of the relationships between optimization-based meta-learning techniques [Huisman et al., 2021].

learned optimizers may learn faster and yield better performance than gradient descent.

The goal of our work is to investigate why MAML often outperforms the meta-learner LSTM, while the latter is at least as expressive as the former (see Section 4.1).[1] Finn and Levine [2018] have shown that the reverse also holds: MAML can approximate any learning algorithm. However, this theoretical result only holds for sufficiently deep base-learner networks. Thus, for a given network depth, it does not say that MAML subsumes the meta-learner LSTM. In contrast, our result that the meta-learner LSTM subsumes MAML holds for any base-learner network and depth.

In order to investigate the performance gap between the meta-learner LSTM and MAML, we propose TURTLE which replaces the LSTM module from the meta-learner LSTM with a feed-forward neural network. Note that Metz et al. [2019] also used a regular feed-forward network as an optimizer. However, they were mainly concerned with understanding and correcting the difficulties that

[1] Link to our code: `https://github.com/mikehuisman/revisiting-learned-optimizers`

arise from training an optimizer and do not learn a weight initialization for the base-learner network as we do. Baik et al. [2020] also use a feed-forward network on top of MAML but its goal is to generate a per-step learning rate and weight decay coefficients. The feed-forward network in TURTLE, in contrast, generates direct weight updates.

3 Preliminaries

In this section, we explain the notation and the concepts of the works that we build upon.

3.1 Few-shot learning

In the context of supervised learning, the few-shot setup is commonly used as a testbed for meta-learning algorithms [Vinyals et al., 2016, Finn et al., 2017, Nichol et al., 2018, Ravi and Larochelle, 2017]. One reason for this is the fact that tasks $\mathcal{T}_j$ are small, which makes learning a prior *across* tasks not overly expensive.

Every task $\mathcal{T}_j$ consists of a support (training) set $D^{tr}_{\mathcal{T}_j}$ and query (test) set $D^{te}_{\mathcal{T}_j}$ [Vinyals et al., 2016, Lu et al., 2020, Ravi and Larochelle, 2017]. When a model is presented with a new task, it tries to learn the associated concepts from the support set. The success of this learning process is then evaluated on the query set. Naturally, this means that the query set contains concepts that were present in the support set.

In classification settings, a commonly used instantiation of the few-shot setup is called N-way k-shot learning [Finn et al., 2017, Vinyals et al., 2016]. Here, given a task $\mathcal{T}_j$, every support set contains k examples for each of the N distinct classes. Moreover, the query set must contain examples from one of these N classes.

Suppose we have a dataset D from which we can extract J tasks. For meta-learning purposes, we split these tasks into three non-overlapping partitions: (i) meta-training, (ii) meta-validation, and (iii) meta-test tasks [Ravi and Larochelle, 2017, Sun et al., 2019]. These partitions are used for training the meta-learning algorithm, hyperparameter tuning, and evaluation, respectively. Note that non-overlapping means that every partition is assigned some class labels which are unique to that partition.

3.2 MAML

As mentioned before, MAML [Finn et al., 2017] attempts to learn a set of initial neural network parameters $\boldsymbol{\theta}$ from which we can quickly learn new tasks within T steps of gradient descent, for a small value of T. Thus, given a task $\mathcal{T}_j =$

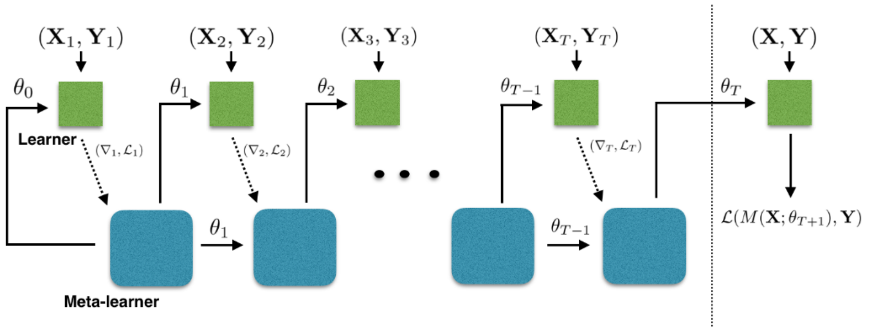

$(\mathbf{X}_1, \mathbf{Y}_1)$
$(\mathbf{X}_2, \mathbf{Y}_2)$
$(\mathbf{X}_3, \mathbf{Y}_3)$
$(\mathbf{X}_T, \mathbf{Y}_T)$
$(\mathbf{X}, \mathbf{Y})$
θ_0
θ_1
θ_2
θ_{T-1}
θ_T
Learner
$(\nabla_1, \mathcal{L}_1)$
$(\nabla_2, \mathcal{L}_2)$
$(\nabla_T, \mathcal{L}_T)$
θ_1
θ_{T-1}
$\mathcal{L}(M(\mathbf{X}; \theta_{T+1}), \mathbf{Y})$
Meta-learner

Fig. 2: The workflow of the meta-learner LSTM [Ravi and Larochelle, 2017]. The base-learner parameters are updated by an LSTM meta-learner network. The base-learner is denoted as M. (X_t, Y_t) are support sets, whereas (X, Y) is the query set. Note that the figure uses subscripts to indicate time steps and not tasks, i.e., θ_T are the parameters at step T and not the initial parameters for task T.

$(D^{tr}_{\mathcal{T}_j}, D^{te}_{\mathcal{T}_j})$, MAML will produce a sequence of weights $(\boldsymbol{\theta}^{(0)}_j, \boldsymbol{\theta}^{(1)}_j, \boldsymbol{\theta}^{(2)}_j, ..., \boldsymbol{\theta}^{(T)}_j)$, where

$$\boldsymbol{\theta}^{(t+1)}_j = \boldsymbol{\theta}^{(t)}_j - \alpha \nabla_{\boldsymbol{\theta}^{(t)}_j} \mathcal{L}_{D^{tr}_{\mathcal{T}_j}}(\boldsymbol{\theta}^{(t)}_j). \tag{1}$$

Here, α is the inner learning rate and $\mathcal{L}_D(\boldsymbol{\varphi})$ the loss of the network with weights $\boldsymbol{\varphi}$ on dataset D. Note that the first set of weights in the sequence is equal to the initialization, i.e., $\boldsymbol{\theta}^{(0)}_j = \boldsymbol{\theta}$.

Given a distribution of tasks $p(\mathcal{T})$, we can formalize the objective of MAML as finding the initial parameters

$$\arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathcal{T}_j \backsim p(\mathcal{T})} \left[\mathcal{L}_{D^{te}_{\mathcal{T}_j}}(\boldsymbol{\theta}^{(T)}_j) \right]. \tag{2}$$

Note that the loss is taken with respect to the query set, whereas $\boldsymbol{\theta}^{(T)}_j$ is computed on the support set $D^{tr}_{\mathcal{T}_J}$.

The initialization parameters $\boldsymbol{\theta}$ are updated by optimizing this objective in Equation 2, where the expectation over tasks is approximated by sampling a batch of tasks. Importantly, updating these initial parameters requires backpropagation through the optimization trajectories on the tasks from the batch. This implies the computation of second-order derivatives, which is computationally expensive. However, Finn et al. [2017] have shown that first-order MAML, which ignores these higher-order derivatives and is computationally less demanding, works just as well as the complete, second-order MAML version.

3.3 Meta-learner LSTM

The meta-learner LSTM by Ravi and Larochelle [2017] can be seen as an extension of MAML as it does not only learn the initial parameters $\boldsymbol{\theta}$ but also the optimization procedure which is used to learn a given task. Note that MAML only uses a single base-learner network, while the meta-learner LSTM uses a separate meta-network to update the base-learner parameters, as shown in Figure 2. Thus, instead of computing $(\boldsymbol{\theta}_j^{(0)}, \boldsymbol{\theta}_j^{(1)}, \boldsymbol{\theta}_j^{(2)}, ..., \boldsymbol{\theta}_j^{(T)})$ using regular gradient descent as done by MAML, the meta-learner LSTM learns a procedure that can produce such a sequence of updates, using a separate meta-network.

This trainable optimizer takes the form of a special LSTM module, which is applied to every weight in the base-learner network after the gradients and loss are computed on the support set. The idea is to embed the base-learner weights into the cell state $\boldsymbol{c}$ of the LSTM module. Thus, for a given task $\mathcal{T}_j$, we start with cell state $\boldsymbol{c}_j^{(0)} = \boldsymbol{\theta}$. After this initialization phase, the base-learner parameters (which are now inside the cell state) are updated as

$$\begin{aligned} \boldsymbol{c}_j^{(t+1)} = \; & \underbrace{\sigma\left(\boldsymbol{W_f} \cdot [\nabla_{\boldsymbol{\theta}_j^{(t)}}, \mathcal{L}_{D^{tr}_{\mathcal{T}_j}}, \boldsymbol{\theta}_j^{(t)}, \boldsymbol{f}_j^{(t-1)}] + \boldsymbol{b_f}\right)}_{\text{weight decay}} \odot \boldsymbol{c}_j^{(t)} \\ & + \underbrace{\sigma\left(\boldsymbol{W_i} \cdot [\nabla_{\boldsymbol{\theta}_j^{(t)}}, \mathcal{L}_{D^{tr}_{\mathcal{T}_j}}, \boldsymbol{\theta}_j^{(t)}, \boldsymbol{i}_j^{(t-1)}] + \boldsymbol{b_i}\right)}_{\text{learning rate}} \odot \bar{\boldsymbol{c}}_j^{(t)}, \end{aligned} \tag{3}$$

where $\odot$ is the element-wise product, the two sigmoid factors σ are the parameterized forget gate $\boldsymbol{f}_j^{(t)}$ and learning rate $\boldsymbol{i}_j^{(t)}$ vectors that steer the learning process, $\nabla_{\boldsymbol{\theta}_j^{(t)}} = \nabla_{\boldsymbol{\theta}_j^{(t)}} \mathcal{L}_{D^{tr}_{\mathcal{T}_j}}(\boldsymbol{\theta}_j^{(t)})$, $\mathcal{L}_{D^{tr}_{\mathcal{T}_j}} = \mathcal{L}_{D^{tr}_{\mathcal{T}_j}}(\boldsymbol{\theta}_j^{(t)})$, and $\bar{\boldsymbol{c}}_j^{(t)} = -\nabla_{\boldsymbol{\theta}_j^{(t)}} \mathcal{L}_{D^{tr}_{\mathcal{T}_j}}(\boldsymbol{\theta}_j^{(t)})$. Both the learning rate and forget gate vectors are parameterized by weight matrices $\boldsymbol{W}_f, \boldsymbol{W}_i$ and bias vectors $\boldsymbol{b_f}$ and $\boldsymbol{b_i}$, respectively. These parameters steer the inner learning on tasks and are updated using regular, hand-crafted optimizers after every meta-training task. As noted by Ravi and Larochelle [2017], this is equivalent to gradient descent when $\boldsymbol{c}^{(t)} = \boldsymbol{\theta}_j^{(t)}$, and the sigmoidal factors are equal to $\mathbf{1}$ and $\boldsymbol{\alpha}$, respectively.

In spite of the fact that the LSTM module is applied to every weight individually to produce updates, it does maintain a separate hidden state for each of them. In a similar fashion to MAML, updating the initialization parameters (and LSTM parameters) would require propagating backwards through the optimization trajectory for each task. To circumvent the computational costs associated with this expensive operation, the meta-learner LSTM assumes that input gradients and losses are *independent* of the parameters in the LSTM.

4 Towards stateless neural meta-learning

In this section, we study the theoretical relationship between MAML and the meta-learner LSTM. Based on the resulting insight, we formulate a new meta-learning algorithm called TURTLE (s<u>t</u>ateless ne<u>ur</u>al me<u>t</u>a-<u>le</u>arning) which is simpler than the meta-learner LSTM and more expressive than MAML.

4.1 Theoretical relationship

There is a subsumption relationship between MAML and the LSTM meta-learner. The gradient update rule used by MAML uses a fixed learning rate and no weight decay. The LSTM meta-learner, on the other hand, can learn a dynamic weight decay and learning rate schedule. These observations gives rise to the following theorem:

Theorem 1 *The meta-learner LSTM subsumes MAML*

Proof We prove this theorem by showing that there is a parameterization of the LSTM meta-learner such that it updates the base-learner weights using gradient descent with a fixed learning rate α and without weight decay. In other words, we show that there exist $\boldsymbol{W}_f, \boldsymbol{b_f}, \boldsymbol{W_i}, \boldsymbol{b_i}$ such that the update made by the LSTM meta-learner is equivalent to that made by MAML

$$\boldsymbol{c}_j^{(t+1)} = \boldsymbol{\theta}_j^{(t)} - \alpha \nabla_{\boldsymbol{\theta}_j^{(t)}} \mathcal{L}_{D^{tr}_{\mathcal{T}_j}}(\boldsymbol{\theta}_j^{(t)}) \tag{4}$$

$$= \mathbf{1} \odot \boldsymbol{\theta}_j^{(t)} + \alpha \mathbf{1} \odot -\nabla_{\boldsymbol{\theta}_j^{(t)}} \mathcal{L}_{D^{tr}_{\mathcal{T}_j}}(\boldsymbol{\theta}_j^{(t)}). \tag{5}$$

The update of the meta-learner LSTM (Equation 3) satisfies this relationship when $\boldsymbol{c}_j^{(0)} = \boldsymbol{\theta}$ (satisfied by construction), the weight decay is equal a vector of ones $\mathbf{1}$, and the learning rate to $\alpha\mathbf{1}$. The weight decay condition condition can be met by setting $\boldsymbol{W_f}$ to a matrix of zeros and $\mathbf{b}_f$ to vector of sufficiently large values to push the output of the sigmoid near its saturation point (1). Since the learning rate $0 < \alpha < 1$ falls within the codomain of the sigmoid function, the learning rate condition can also be met by setting $\boldsymbol{W_i}$ to a matrix of zeros and $\boldsymbol{b}_f = -\ln(\frac{1-\alpha}{\alpha})\mathbf{1}$. Thus, we have shown that it is possible to parameterize the LSTM meta-learner so that it mimics gradient descent with any learning rate $0 < \alpha \leq 1$.

4.2 Potential problems of the meta-learner LSTM

The theoretical insight that meta-learner LSTM subsumes MAML is not congruent with empirical findings which show that MAML outperforms the meta-learner LSTM on the miniImageNet image classification benchmark [Finn et al., 2017, Ravi and Larochelle, 2017], indicating that LSTM is unable to successfully navigate the error landscape to find a solution at least as good as the one found by MAML.

A potential cause is that the meta-learner LSTM attempts to learn a *stateful* optimization procedure, allowing it to employ dynamic weight decay and learning rate schedules for learning new tasks. While this gives the technique more flexibility in learning an optimization algorithm, it may also negatively affect meta-optimization as it may be harder to find a good dynamic optimization algorithm than a static one because the space of dynamic algorithms is less constrained. In addition, we conjecture that the loss landscape for dynamic algorithms is less smooth because the weight decay and learning rate schedules, which can have a large influence on the performance, depend on parameter trajectories (paths from initialization to task-specific parameters). We hypothesize that removing the stateful nature of the trainable optimizer may smoothen the meta-landscape as it constrains the space of possible solutions and removes the dependency of the learning rate on parameter trajectories, which can stabilize learning. For this reason, we replace the LSTM module in TURTLE with a regular fully-connected, feed-forward network, which is stateless.

Another potential cause of the underperformance could be the first-order assumption made by the meta-learner LSTM, which we briefly mentioned in Section 3.3. Effectively, this disconnects the computational graph by stating that weight updates made at time step t by the meta-network do not influence the inputs that this network receives at future time steps $t < t' < T$. Consequently, the algorithm ignores curvature information which can be important for stable training. While first-order MAML achieves similar performance to MAML, we think that the loss landscape of the LSTM meta-learner is less smooth (for reasons mentioned above), which can exacerbate the harmful effect of the first-order assumption. To overcome this issue, we use second-order gradients by default in TURTLE and investigate the effect of making the first-order assumption.

4.3 TURTLE

In an attempt to make the meta-landscape easier to navigate, we introduce a new algorithm, TURTLE, which trains a feed-forward meta-network to update the base-learner parameters. TURTLE is simpler than the meta-learner LSTM as it uses a *stateless* feed-forward neural network as a trainable optimizer, yet more expressive than MAML as its meta-network can learn to perform gradient descent.

The trainable optimizer in TURTLE is thus a fully-connected feed-forward neural network. We denote the batch of inputs that this network receives at time step t in the inner loop for task $\mathcal{T}_j$ as $I_j^{(t)} \in \mathbb{R}^{n \times d}$, where n and d are the number of base-learner parameters and the dimensionality of the inputs, respectively. The exact inputs that this network receives will be determined empirically, but two choices, inspired by the meta-learner LSTM, are: (i) the gradients with respect to all parameters and (ii) the current loss (repeated n times for each parameter in the base-network).

Moreover, we could mitigate the absence of a state in the meta-network by including a time step $t \in \{0, 1, ..., T-1\}$ and/or historical information such as a moving average of previous gradients or updates made by the meta-network. We denote the latter by $\boldsymbol{h}_j^{(t)}$ which is updated by

$$\boldsymbol{h}_j^{(t+1)} = \beta \boldsymbol{h}_j^{(t)} + (1-\beta)\boldsymbol{v}_j^{(t)}, \tag{6}$$

where $0 \leq \beta \leq 0$ is a constant that determines the time span over which previous inputs affect the new state $\boldsymbol{h}_j^{(t+1)}$, and $\boldsymbol{v}_j^{(t)} \in \mathbb{R}^n$ is the new information (either the updates or gradients at time step t). When using previous updates, we initialize $\boldsymbol{h}_j^{(0)}$ by a vector of zeros.

Weight updates are then computed as follows

$$\boldsymbol{\theta}_j^{(t+1)} = \boldsymbol{\theta}_j^{(t)} + \boldsymbol{\alpha} \odot g_{\boldsymbol{\phi}}(I_j^{(t)}), \tag{7}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^n$ is a vector of learning rates per parameter. Note that this weight update equation is simpler than the one used by the meta-learner LSTM (see Equation 3) as our meta-network $g_{\boldsymbol{\phi}}$ is stateless. Therefore, we do not have parameterized forget and input gates. Moreover, the learning rates per parameter in $\boldsymbol{\alpha}$ are not constrained to be within the interval $[0, 1]$ as is the case for the meta-learner LSTM due to the use of the sigmoid function.

In Algorithm 1 we show, in different colors, the code for MAML (red), the meta-learner LSTM (blue), and TURTLE (green). Although the code structure of the three meta-learners is similar, the update rules are quite different. Both the base- and meta-learner parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are updated by backpropagation through the optimization trajectories (line 11).

Algorithm 1 MAML meta-learner LSTM TURTLE

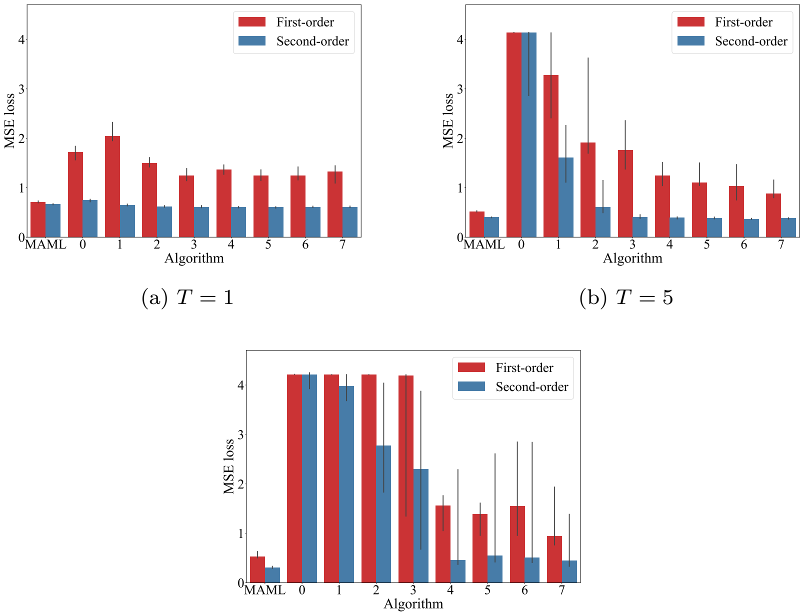
```
1: Initialize parameters $\boldsymbol{\Theta}$ = $\{\boldsymbol{\theta}\}$ $\{\boldsymbol{\theta}, \boldsymbol{\phi}\}$ $\{\boldsymbol{\theta}, \boldsymbol{\phi}\}$
2: Initialize $g_{\boldsymbol{\phi}}$ as N.A. LSTM feed-forward network
3: repeat
4:     Sample batch of J tasks $B = \{\mathcal{T}_j \backsim p(\mathcal{T})\}_{j=1}^{J}$
5:     for $\mathcal{T}_j = (D^{tr}_{\mathcal{T}_j}, D^{te}_{\mathcal{T}_j})$ in $B$ do
6:         $\boldsymbol{\theta}_j^{(0)} = \boldsymbol{\theta}$
7:         for $t = 1, ..., T$ do
8:             Update $\boldsymbol{\theta}_j^{(t)}$ using Eq. 1 Eq. 3 Eq. 7
9:         end for
10:    end for
11:    Update $\boldsymbol{\Theta}$ using $\sum_{\mathcal{T}_j \in B} \mathcal{L}_{D^{te}_{\mathcal{T}_j}}(\boldsymbol{\theta}_j^{(T)})$
12: until convergence
```

5 Experiments

In this section, we describe our experimental setup and the results that we obtained.

5.1 Hyperparameter analysis on sine-wave regression

Here, we investigate the effect of the order of information (first- versus second-order), the number of updates T per task, and further increasing the number of layers of the meta-network on the performance of TURTLE on 5-shot



(a) $T = 1$

(b) $T = 5$

(c) $T = 10$

Fig. 3: Influence of the order, number of update steps, and number of hidden layers (horizontal axis) on the meta-validation performance of TURTLE on 5-shot sine wave regression. We also plot the performance of first- and second-order MAML for comparison. Note that a lower MSE loss corresponds to better performance. The vertical bars indicate the 95% confidence intervals.

sine wave regression. The results are displayed in Figure 3. Note that in this experiment, we fixed the learning rate vector $\boldsymbol{\alpha}$ to be a vector of ones, which means that the updates proposed by the meta-network are directly added to the base-learner parameters without any scaling. Moreover, the only input that the meta-network receives is the gradient of the loss on the support set with respect to a base-learner parameter, and every hidden layer of the meta-network consists of 20 nodes followed by ReLU nonlinearities.

As we can see, the difference between first- and second-order MAML is relatively small, which was also found by Finn et al. [2017]. In contrast, this is not the case for TURTLE, where the first-order variant fails to achieve a similar performance as second-order TURTLE. Furthermore, we see that the stability of TURTLE decreases as T increases as the confidence intervals become larger and the performance with fewer hidden layers deteriorates. Lastly, we find that 5 or 6 hidden layers yield the best performance across different values of T. For this reason, all further TURTLE experiments will be conducted with a meta-network of 5 hidden layers.

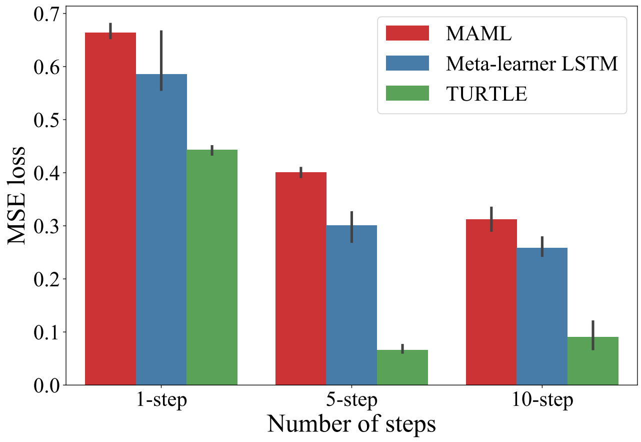


Fig. 4: Median meta-test performance of MAML, the meta-learner LSTM, and TURTLE on 5-shot sine wave regression. Note that a lower error indicates better performance. The vertical bars indicate the 95% confidence intervals.

5.2 Few-shot sine wave regression

First, we compare the performance of TURTLE to that of MAML and the meta-learner LSTM on sine wave regression, which was originally proposed by Finn et al. [2017]. We follow their experimental setup and use 70K, 1K, and 2K tasks for training, validation, and testing respectively. All experiments are performed 30 times with different random weight initializations of the base- and meta-learner networks. We perform meta-validation every 2.5K tasks for hyperparameter tuning. The meta-test results of this experiment are displayed in Figure 4. As we can see, TURTLE, which uses second-order gradients, systematically outperforms the meta-learner LSTM, which uses first-order gradients.

5.3 Few-shot image classification

Without additional hyperparameter tuning, we now investigate the performance of 5-step TURTLE on few-shot image classification tasks, following the setup used in Chen et al. [2019]. In addition, we investigate the importance of second-order gradients in this setting. For this, we use miniImageNet [Vinyals et al., 2016] (with the class splits proposed by Ravi and Larochelle [2017]) and CUB [Wah et al., 2011]. We use the same base-learner network used by Snell et al. [2017] and Chen et al. [2019]. Each algorithm is run with 5 different random weight initializations.

Table 1: Median meta-test accuracy scores and 95% confidence intervals over 5 runs of 5-way image classification on miniImageNet (left) and CUB (right). The best performance is displayed in bold font. Note that a higher accuracy indicates better performance.

	miniImageNet		CUB	
	1-shot	5-shot	1-shot	5-shot
TrainFromScratch	0.29 ± 0.00	0.40 ± 0.00	0.30 ± 0.00	0.46 ± 0.00
Finetuning	0.38 ± 0.00	0.56 ± 0.00	0.33 ± 0.01	0.53 ± 0.01
Baseline++	0.44 ± 0.00	0.58 ± 0.00	0.36 ± 0.01	0.53 ± 0.01
Meta-learner LSTM	0.45 ± 0.01	0.61 ± 0.00	0.50 ± 0.00	0.65 ± 0.01
Meta-learner LSTM[2]	**0.48** ± 0.01	**0.63** ± 0.01	**0.53** ± 0.01	0.71 ± 0.00
FO-MAML	0.46 ± 0.01	**0.63** ± 0.00	0.52 ± 0.00	0.72 ± 0.01
MAML	0.47 ± 0.01	**0.63** ± 0.00	0.52 ± 0.00	**0.73** ± 0.01
First-order TURTLE	0.43 ± 0.01	0.59 ± 0.04	0.50 ± 0.00	0.64 ± 0.03
TURTLE	**0.48** ± 0.01	0.62 ± 0.01	**0.53** ± 0.00	0.72 ± 0.01

We compare the performance against three simple transfer-learning models, following Chen et al. [2019]: train from scratch, finetuning, and baseline++. Based on our hyperparameter experiments for TURTLE, we also investigate an enhanced version of the meta-learner LSTM which uses raw gradients as meta-learner input and second-order information. The meta-test accuracy scores on 5-way miniImageNet and CUB classification are displayed in Table 1. Note that we use the best-reported hyperparameters for MAML and the meta-learner LSTM on miniImageNet, while we use the best hyperparameters found on sine wave regression for TURTLE. Despite this, TURTLE and second-order meta-learner LSTM outperform MAML and other techniques in 50% of the tested scenarios while they yield the competitive performance in the other scenarios. As we can see, the performances of all models are better on 5-shot classification compared with 1-shot classification. Looking at the results for miniImageNet, we see that the addition of second-order gradients increases the performance of both the meta-learner LSTM and TURTLE. An overview of the exact hyperparameter values that were used for all techniques can be found in Appendix A.

5.4 Cross-domain few-shot learning

We also investigate the robustness of the meta-learning algorithms when a task distribution shift occurs. For this, we train the techniques on miniImageNet and evaluate their performance on CUB tasks (and vice versa), following Chen et al. [2019]. This is a challenging setting that requires a more general learning ability than for the experiments above. The results are shown in Table 2. Also in these

[2] Our enhanced version of the meta-learner LSTM, which takes raw gradients as inputs, uses second-order gradients, and makes 8 updates per task.

Table 2: Median meta-test accuracy scores in 5-way cross-domain classification (train on tasks from one dataset and evaluate on tasks from another dataset). The median accuracy and 95% confidence intervals were computed over 5 runs. The meta-learner LSTM^2 refers to our enhanced version of the meta-learner LSTM, which takes raw gradients as inputs, uses second-order gradients, and makes 8 updates per task.

	miniImageNet → CUB		CUB → miniImageNet	
	1-shot	5-shot	1-shot	5-shot
TrainFromScratch	0.30 ± 0.00	0.46 ± 0.00	0.29 ± 0.00	0.40 ± 0.00
Finetuning	0.33 ± 0.00	0.52 ± 0.01	0.29 ± 0.00	0.41 ± 0.00
Baseline++	0.35 ± 0.01	0.52 ± 0.01	0.26 ± 0.00	0.31 ± 0.01
Meta-learner LSTM	0.34 ± 0.01	0.52 ± 0.01	0.29 ± 0.01	0.37 ± 0.00
Meta-learner LSTM^2	0.37 ± 0.00	0.55 ± 0.01	**0.32** ± 0.01	0.43 ± 0.01
FO-MAML	0.34 ± 0.01	0.54 ± 0.01	0.31 ± 0.00	0.45 ± 0.01
MAML	0.35 ± 0.00	**0.56** ± 0.01	0.31 ± 0.00	**0.47** ± 0.00
fo-TURTLE	0.36 ± 0.00	0.54 ± 0.02	0.30 ± 0.00	0.31 ± 0.01
TURTLE	**0.38** ± 0.00	**0.56** ± 0.01	0.30 ± 0.00	0.44 ± 0.00

challenging scenarios, second-order gradients are important to increase the performance of both the meta-learner LSTM and TURTLE. More specifically, the omission of second-order gradients can lead to large performance penalties, ranging from 1% to 5% accuracy.

5.5 Running time comparison

Lastly, we compare the running times of MAML, the meta-learner LSTM, and TURTLE on miniImageNet and CUB. A run comprises the time it costs to perform meta-training, meta-validation, and meta-testing on miniImageNet, and evaluation on CUB. We measure the average time in full hours across 5 runs on nodes with a Xeon Gold 6126 2.6GHz 12 core CPU and PNY GeForce RTX 2080TI GPU. The results are displayed in Figure 5. As we can see, the first-order algorithms are the fastest, while the second-order algorithms are slower (so-MAML and TURTLE). However, the performance of the first-order meta-learner LSTM and first-order TURTLE is worse than that of the second-order variants, indicating the importance of second-order gradients. For MAML, we do not observe such a difference between the first- and second-order variants. TURTLE is, despite its name, not much slower than the second-order MAML (SO-MAML), indicating that the time complexity is dominated by learning the base-learner initialization parameters. In fact, we observe that TURTLE is slightly faster than MAML, indicating that our implementation of the latter is not optimally efficient. In addition, we note that TURTLE is faster than the second-order (enhanced) LSTM meta-learner.

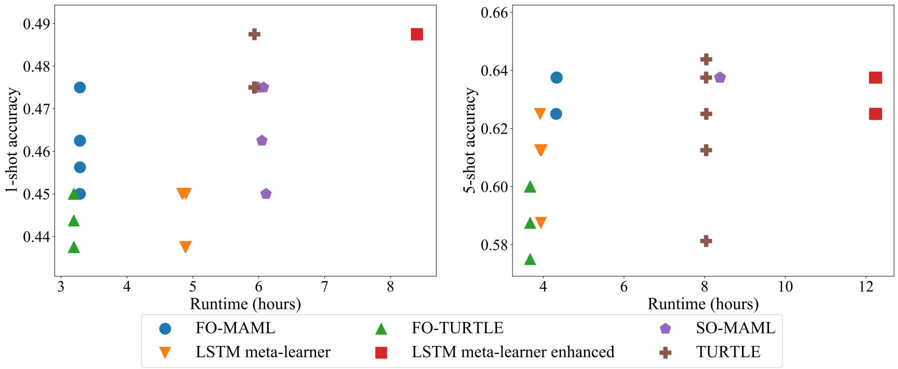


Fig. 5: The running times and few-shot learning accuracy scores on 1-shot (left) and 5-shot (right) miniImageNet image classification of the different techniques for 5 runs with different random seeds.

6 Discussion and future work

In this work, we have formally shown that the meta-learner LSTM [Ravi and Larochelle, 2017] subsumes MAML [Finn et al., 2017]. Experiments of Finn et al. [2017] and ourselves, however, show that MAML outperforms the meta-learner LSTM. We formulated two hypotheses for this surprising finding and, in turn, we formulated a new meta-learning algorithm named TURTLE, which is simpler than the meta-learner LSTM as it is stateless, yet more expressive than MAML because it can learn the weight update rule as it features a separate meta-network.

We empirically demonstrate that TURTLE is capable of outperforming both MAML and the (first-order) meta-learner LSTM on sine wave regression and—without additional hyperparameter tuning—on the frequently used miniImageNet benchmark. This shows that better update rules exist for fast adaptation than regular gradient descent, which is in line with findings by Andrychowicz et al. [2016]. Moreover, we enhanced the meta-learner LSTM by using raw gradients as meta-learner input and second-order gradient information, as they were found to be important for TURTLE. Our results indicate that this enhanced version of the meta-learner LSTM systematically outperforms the original technique by $1-6\%$ accuracy.

In short, these results show that second-order gradients are important for maximizing the few-shot image classification performance of the meta-learner LSTM and TURTLE, at the cost of additional runtime. In contrast, first-order MAML is a good approximation to second-order MAML as it yields similar performance [Finn et al., 2017]. This finding supports our hypothesis that the loss landscape of MAML is smoother than that of meta-learning techniques that learn both the initialization parameters and a gradient-based optimization procedure.

Limitations and open challenges

While TURTLE and the enhanced meta-learner LSTM were shown to yield good performance, it has to be noted that this comes at the cost of increased computational expenses compared with first-order algorithms. That is, these second-order algorithms perform backpropagation through the entire optimization trajectory which requires storing intermediate updates and the computation of second-order gradients. While this is also the case for MAML, it has been shown that first-order MAML achieves a similar performance whilst avoiding this expensive backpropagation process, yielding an excellent trade-off between performance and computational costs. For TURTLE, however, this is not the case, which means that other approaches should be investigated in order to reduce the computational costs. Future research may draw inspiration from Rajeswaran et al. [2019] who approximated second-order gradients in order to speed up MAML.

Our experiments also show that the training stability of TURTLE deteriorates as the number of inner updates increases. This is a known problem of meta-learning techniques that aim to learn the optimization algorithm. Metz et al. [2019] show that this instability is due to the fact that the meta-loss landscape becomes increasingly pathological as the number of inner updates increases. Future work is required to make it feasible to train such techniques for a large number of updates. Moreover, we note that we have only investigated the performances of MAML, TURTLE, and the LSTM meta-learner in 1- and 5-shot settings It would be interesting to investigate in future work how well MAML, TURTLE, and the LSTM meta-learner perform when more shots and ways (classes) are available per task. It may be possible that the performances of these techniques converge to the same point as the amount of available data increases. The intuition behind this is that given enough data, there is no need for a meta-learned prior to successfully learn the task.

Successfully using meta-learning algorithms in scenarios where task distribution shifts occur remains an important open challenge in the field of meta-learning. Our cross-domain experiment demonstrates that the learned optimization procedure by TURTLE generalizes to different tasks than the ones seen at training time, which is in line with findings by Andrychowicz et al. [2016]. For this reason, we think that learned optimizers may be an important piece of the puzzle to broaden the applicability of meta-learning techniques to real-world problems. Future work can further investigate this hypothesis.

Our findings further show the benefit of learning an optimizer in addition to the initialization weights and highlight the importance of second-order gradients.

Acknowledgements This work was performed using the compute resources from the Academic Leiden Interdisciplinary Cluster Environment (ALICE) provided by Leiden University.

Declarations

6.1 Conflicts of Interest

Funding Not applicable: no funding was received for this work.

Employment All authors declare that there is no recent, present, or anticipated employment by any organization that may gain or lose financially through publication of this manuscript.

Interests All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

6.2 Compliance with Ethical Standards

Not applicable: this research did not involve human participants, nor did it involve animals.

6.3 Consent for publication

Not applicable: this research does not involve personal data, and publishing of this manuscript will not result in the disruption of any individual's privacy.

6.4 Availability of data and material

All data that was used in this research have been published as benchmarks by Deng et al. [2009], Vinyals et al. [2016] (miniImageNet) and Wah et al. [2011] (CUB), and is publicly available. The data generator for sine wave regression experiments can be found in the provided code (see below).

6.5 Code availability

All code that was used for this research is made publicly available at `https://github.com/mikehuisman/revisiting-learned-optimizers`.

6.6 Authors' contributions

MH has conducted the research presented in this manuscript. AP and JvR have regularly provided feedback on the work, contributed towards the interpretation of results, and have critically revised the whole.

All authors approve the current version to be published and agree to be accountable for all aspects of the work in ensuring that questions related to the accuracy or integrity of any part of the work are appropriately investigated and resolved.

References

M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems 29*, NIPS'16, pages 3988–3996. Curran Associates Inc., 2016.

S. Baik, M. Choi, J. Choi, H. Kim, and K. M. Lee. Meta-learning with adaptive hyperparameters. In *Advances in Neural Information Processing Systems 33*, NIPS'20, 2020.

L. Bottou. Stochastic Learning. In *Advanced lectures on machine learning*, pages 146–168. Springer, 2004.

W.-Y. Chen, Y.-C. Liu, Z. Kira, Y.-C. F. Wang, and J.-B. Huang. A closer look at few-shot classification. In *International Conference on Learning Representations*, ICLR'19, 2019.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

C. Finn and S. Levine. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. In *International Conference on Learning Representations*, ICLR'18, 2018.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning*, ICML'17, page 1126–1135. PMLR, 2017.

C. Finn, K. Xu, and S. Levine. Probabilistic Model-Agnostic Meta-Learning. In *Advances in Neural Information Processing Systems 31*, NIPS'18, pages 9516–9527, 2018.

E. Grant, C. Finn, S. Levine, T. Darrell, and T. Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. In *International Conference on Learning Representations*, ICLR'18, 2018.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*, 2020.

M. Huisman, J. N. van Rijn, and A. Plaat. A survey of deep meta-learning. *Artificial Intelligence Review*, 2021. ISSN 0269-2821. doi: 10.1007/s10462-021-10004-4.

D. P. Kingma and J. L. Ba. Adam: A method for stochastic gradient descent. In *International Conference on Learning Representations*, ICLR'15, 2015.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, NIPS'12, pages 1097–1105, 2012.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015.

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

Y. Lee and S. Choi. Gradient-based meta-learning with learned layerwise metric and subspace. In *Proceedings of the 35th International Conference on Machine Learning*, ICML'18, pages 2927–2936. PMLR, 2018.

Z. Li, F. Zhou, F. Chen, and H. Li. Meta-SGD: Learning to Learn Quickly for Few-Shot Learning. *arXiv preprint arXiv:1707.09835*, 2017.

J. Lu, P. Gong, J. Ye, and C. Zhang. Learning from very few samples: A survey. *arXiv preprint arXiv:2009.02653*, 2020.

L. Metz, N. Maheswaranathan, J. Nixon, D. Freeman, and J. Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In *Proceedings of the 36th International Conference on Machine Learning*, ICML'19, pages 4556–4565. PMLR, 2019.

V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

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

E. Park and J. B. Oliva. Meta-curvature. In *Advances in Neural Information Processing Systems 32*, NIPS'19, pages 3314–3324, 2019.

A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine. Meta-Learning with Implicit Gradients. In *Advances in Neural Information Processing Systems 32*, NIPS'19, pages 113–124, 2019.

S. Ravi and H. Larochelle. Optimization as a Model for Few-Shot Learning. In *International Conference on Learning Representations*, ICLR'17, 2017.

A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell. Meta-learning with latent embedding optimization. In *International Conference on Learning Representations*, ICLR'19, 2019.

T. Schaul and J. Schmidhuber. Metalearning. *Scholarpedia*, 5(6):4650, 2010.

J. Schmidhuber. Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook. Master's thesis, Technische Universität München, 1987.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

J. Snell, K. Swersky, and R. Zemel. Prototypical Networks for Few-shot Learning. In *Advances in Neural Information Processing Systems 30*, NIPS'17, pages 4077–4087. Curran Associates Inc., 2017.

Q. Sun, Y. Liu, T.-S. Chua, and B. Schiele. Meta-transfer learning for few-shot learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 403–412, 2019.

T. Tieleman and G. Hinton. Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. *Technical Report.*, 2017.

J. Vanschoren. Meta-Learning: A Survey. *arXiv preprint arXiv:1810.03548*, 2018.

R. Vilalta and Y. Drissi. A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2):77–95, 2002.

O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra. Matching Networks for One Shot Learning. In *Advances in Neural Information Processing Systems 29*, NIPS'16, pages 3637–3645, 2016.

C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

A Appendix

A.1 Used hyperparameters

For all techniques mentioned below, we performed meta-validation after every 2,500 training tasks. The best-resulting configuration was evaluated at meta-test time.

For sine wave regression, we use the same base-learner network as Finn et al. [2017], i.e., a fully-connected feed-forward network consisting of a single input node followed by two hidden layers with 40 ReLU nodes each and a final single-node output layer.

For few-shot image classification problems, we use the same base-learner network as used by Snell et al. [2017] and Chen et al. [2019]. This network is a stack of four identical convolutional blocks. Each block consists of 64 convolutions of size 3×3, batch normalization, a ReLU nonlinearity, and a 2D max-pooling layer with a kernel size of 2. The resulting embeddings of the $84 \times 84 \times 3$ input images are flattened and fed into a dense layer with N nodes (one for every class in a task). The base-learner is trained to minimize the cross-entropy loss on the query set, conditioned on the support set.

Transfer learning baselines Note that these models (TrainFromScratch, finetuning, baseline++) pre-trained on minibatches of size 16 sampled from the joint data obtained by merging all meta-training tasks. At test time, they were trained for 100 steps on mini-batches of size 4 sampled from new tasks following Chen et al. [2019]. Every 25 steps, we evaluated their performance on the entire support set to select the best configuration to test on the query set.

LSTM meta-learner For selecting the hyperparameters of the LSTM meta-learner[3], we followed Ravi and Larochelle [2017]. That is, we use a 2-layer architecture, and Adam as meta-optimizer with a learning rate of 0.001. The batch size was set equal to the size of the task. Meta-gradients were clipped to have a norm of at most 0.25, following. The meta-network receives four inputs obtained by preprocessing the loss and gradients using in similar fashion to Andrychowicz et al. [2016] and Ravi and Larochelle [2017]. On miniImageNet and CUB, the LSTM optimizer is set to perform 12 updates per task when the number of examples per class is $k = 1$ and 5 updates when $k = 5$.

[3] Used code: `https://github.com/markdtw/meta-learning-lstm-pytorch`.

MAML Again, we follow Finn et al. [2017] for selecting the hyperparameters, except for the meta-batch size on sine wave regression as we found it not to help performance. This means that the inner learning rate was set to 0.01 and the outer learning rate to 0.001, with Adam as meta-optimizer. These settings hold for both sine wave regression and image classification. When $T > 1$, we use gradient value clipping with a threshold of 10. On image classification, MAML was set to optimize the initial parameters based on $T = 5$ update steps, but an additional 5 steps were made afterwards to further increase the performance. Moreover, we used a meta-batch size of 4 and 2 for 1- and 5-shot image classification respectively.

TURTLE We performed many experiments with the hyperparameters of TURTLE on sine wave regression. Here, we only report the settings that were found to give the best performance, which were also used on the image classification problems. That is, the meta-network consists of 5 hidden layers of 20 nodes each. Every hidden node is followed by a ReLU nonlinearity. The input consists of a raw gradient, a historical real-valued number indicating the moving average of the previous input gradients with a (with a beta decay of 0.9), and a time step integer $t \in \{0, ..., T-1\}$. The output layer consists of a single node which corresponds to the proposed weight update. For training, we used meta-batches of size 2. Additionally, TURTLE maintains a separate learning rate for all weights in the base-learner network. Lastly, TURTLE uses second-order gradients and Adam as meta-optimizer with a learning rate of 0.001.