Overview of Meta Learning: Learning to Learn

Jiao Sun
Computer Science Department
University of Southern California
`jiaosun@usc.edu`

Abstract

Meta-learning problems are that when there is a distribution of tasks, and we would like to obtain an agent that performs well (i.e., learns quickly) when presented with a previously unseen task sampled from this distribution. We analyze a family of optimization-based algorithms for learning a parameter initialization that can be fine-tuned quickly on a new task, which consists of MAML, first-order MAML, iMAML and the main idea of Auto-Meta.

1 Introduction to Meta Learning

Thrun and Pratt [2012] stated that, given one task to solve, an algorithm is learning "if its performance at the task improves with experience", while, given a family of tasks to solve, an algorithm is learning to learn if "its performance at each task improves with experience and with the number of tasks". We refer to the last one as a meta-learning algorithm. For example, we evaluate the meta-learning on German shepherd, golden retriever and pugs It does not learn how to solve a specific task. It successively learns to solve many tasks, and each time it learns a new task, it becomes better at learning new tasks: it learns to learn. There are several ways that meta learning solves a few-shot classification task. Metric learning learns a distance function between data point and labeled ones.

2 Model-Agnostic Meta-Learning: MAML

Finn et al. [2017] claimed that the goal of the trained model is to quickly learn a new task from a small amount of new data, and the model is trained by the meta-learner to be able to learn on a large number of different tasks. The key idea underlying their method is to train the model's initial parameters such that the model has maximal performance on a new task after the parameters have been updated through one or more gradient steps computed with a small amount of data from that new task. This work does not expand the number of learned parameters nor place constraints on the model architecture (by requiring a recurrent model or Siamese network [Koch et al., 2015]). it can be readily combined with fully connected, convolutional, or recurrent neural networks. It can also be used with a variety of loss functions.

2.1 Meta-learning Problem Set-up

The goal of few-shot meta-learning is to train a model that can quickly adapt to a new task using only a few data points and training iterations. The problem should be applied to a variety of learning problems. Consider a model, denoted by f, that maps observations x to outputs a. A generic notion of a learning task is below. Formally, for each task $T = \{\mathcal{L}(x_1, a_1, ..., x_H, a_H), q(x_1), q(x_{t+1}|x_t, a_t), H\}$: a loss function $\mathcal{L}$, a distribution over the initial observation $q(x_1$, a transition distribution $q(x_{t+1}|x_t, a_t)$, and an episode length H. The model may generate samples of length H by choosing an output a_t at each time t. In order to explicitly encourage a neural network which learns internal features that are applicable for all the tasks. Since

For CSCI-699.

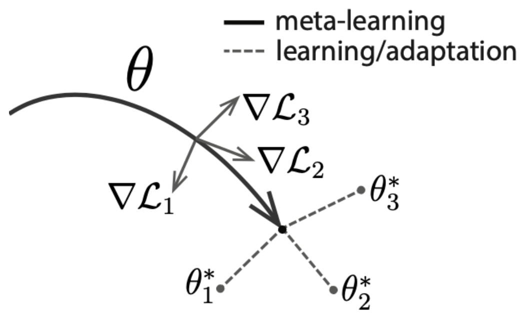

Figure 1: Diagram of the model-agnostic meta-learning algorithm (MAML), which optimizes for a representation θ that can quickly adapt to new tasks.

the model will be fine-tuned using a gradient-based learning rule on a new task. The diagram of MAML is shown in Figure 1. They aim to find model parameters that are sensitive to changes in the task, such that small changes in the parameters will produce large improvements on the loss function of any task drawn from $p(\mathcal{T})$, when altered in the direction of the gradient of that loss.

2.2 A Model-Agnostic Meta-Learning Algorithm

The assumption is that the model is represented by a parameterized function f_θ with parameters θ, the loss function is smooth enough in θ so we can use gradient-based learning techniques. When adapting to a new task T_i, θ becomes $\theta^{'}$, and it is calculated by:

$$\theta_i^{'} = \theta - \alpha \bigtriangledown_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$$

The step size α may be fixed as a hyperparameter or meta-learned here we only consider one gradient update. More concretely, the meta-objective is as follows:

$$\min_\theta \sum_{T_i \sim p(T)} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i^{'}}) = \sum_{T_i \sim p(T)} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \bigtriangledown_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)})$$

The meta-optimization is performed over the model parameter θ, whereas the objective is computed using the updated model parameters $\theta^{'}$. The meta-optimization across tasks is performed via SGD.

$$\theta \leftarrow \theta - \beta \bigtriangledown_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i^{'}})$$

2.3 Species of MAML

They discuss the specific instantiations of meta-learning algorithm for supervised learning and reinforcement learning. For supervised learning, the loss function can be mean-squared error or cross-entropy loss. For reinforcement learning, the model being learned, f_θ, is a policy that maps from states x_t to a distribution over actions at a_t at each timestep $t \in \{1, ..., H\}$, the loss function for task $\mathcal{T}_i$ and model f_ϕ is:

$$\mathcal{L}_{\mathcal{T}_i} = -\mathbb{E}_{x_t, a_t \sim f_\phi, q_{\mathcal{T}_i}} [\sum_{t=1}^{H} R_i(x_t, a_t)]$$

Then we use these customized loss functions in different scenarios to update the gradient in the above accordingly. Note that the MAML meta-gradient update involves a gradient through a gradient. Computationally, this requires an additional backward pass through f to compute Hessian-vector, which brings the significant computational cost.

The authors of MAML also proposed a variant called first-order MAML (FOMAML), which is defined by ignoring the second derivative terms, avoiding this problem but at the expense of losing

Algorithm 1 Reptile (serial version)

```
Initialize $\phi$, the vector of initial parameters
for iteration $= 1, 2, \ldots$ do
    Sample task $\tau$, corresponding to loss $L_\tau$ on weight vectors $\widetilde{\phi}$
    Compute $\widetilde{\phi} = U_\tau^k(\phi)$, denoting $k$ steps of SGD or Adam
    Update $\phi \leftarrow \phi + \epsilon(\widetilde{\phi} - \phi)$
end for
```

some gradient information. Surprisingly, though, they found that FOMAML worked nearly as well as MAML on the Mini ImageNet dataset, which indicates that most of the improvement in MAML comes from the gradients of the objective at the post-update parameter values, rather than the second order updates from differentiating through the gradient update. It also led to roughly 33% peed-up in the network.

3 On First-order Meta-Learning Algorithms: FMAML

Consider the optimization problem of MAML: find an initial set of parameters θ, such that from a randomly sampled task $\mathcal{T}$ with the corresponding loss $L_{\mathcal{T}}$, the learner will have low loss after k updates. That is:

$$\min_{\phi} \mathbb{E}_{\mathcal{T}}[L_{\mathcal{T}}(U_{\mathcal{T}}^k(\phi))]$$

where $U_{\mathcal{T}}^k$ is the operator that updates θk times using data sampled from $\mathcal{T}$. MAML achieves it with the inner-loop optimization uses training samples A, whereas the loss is computed using test samples B. This way, MAML optimizes for generalization.

$$\min_{\phi} \mathbb{E}_{\mathcal{T}}[L_{\mathcal{T},\mathcal{B}}(U_{\mathcal{T},A}(\phi))]$$

MAML works by optimizing this loss through stochastic gradient descent, i.e., computing

$$g_{MAML} = \frac{\partial}{\partial \phi} L_{\mathcal{T},B}(U_{\mathcal{T},A}(\phi)) = U'_{\mathcal{T},A}(\phi) L'_{\mathcal{T},B}(\widetilde{\phi})$$

where $\widetilde{\phi} = U_{\mathcal{T},A}(\phi))$, $U'_{\mathcal{T},A}(\phi))$ is the Jacobian matrix of the update operation $U_{\mathcal{T},A}(\phi)) = \phi + g_1 + g_2 + ... + g_k$. FOMAML treats these gradients as constants, thus $g_{FOMAML} = L_{\mathcal{T},B}(\widetilde{\phi})$. Reptile is also a first-order gradient-based meta-learning algorithm shown in the above.

While MAML calculates the gradient with the loss function in the test data, Reptile updates k steps of gradients in sample tasks, and use the difference between current gradient and initial gradient to update. In the last step, instead of simply updating ϕ in the direction $\widetilde{\phi} - \phi$, we can treat $\widetilde{\phi} - \phi$ as a gradient and plug it into an adaptive algorithm such as Adam.

Unlike in the discussion and analysis of MAML, they do not consider a training set and test set from each task; instead, they assume that each task gives us a sequence of k loss functions $L1, L2, ..., Lk$; for example, classification loss on different minibatches. They use the following definitions:

$$g_i = L'_i(\phi_i) \text{ gradient obtained during SGD}$$

$$\phi_{i+1} = \phi_i - \alpha g_i$$

$$\bar{g}_i = L'_i(\phi_1)$$

$$\bar{H}_i = L''_i(\phi_1)$$

To analyze why Reptile works, they approximate the update using a Taylor series.

First, let's calculate the SGD gradients to $O(\alpha^2)$ as follows.

$$g_i = L_i'(\phi_i) = L_i'(\phi_1) + L_i''(\phi_1)(\phi_i - \phi_1) + \underbrace{O(\|\phi_i - \phi_1\|^2)}_{=O(\alpha^2)} \quad \text{(Taylor's theorem)} \tag{13}$$

$$= \overline{g}_i + \overline{H}_i(\phi_i - \phi_1) + O(\alpha^2) \quad \text{(using definition of } \overline{g}_i, \overline{H}_i) \tag{14}$$

$$= \overline{g}_i - \alpha \overline{H}_i \sum_{j=1}^{i-1} g_j + O(\alpha^2) \quad \text{(using } \phi_i - \phi_1 = -\alpha \sum_{j=1}^{i-1} g_j) \tag{15}$$

$$= \overline{g}_i - \alpha \overline{H}_i \sum_{j=1}^{i-1} \overline{g}_j + O(\alpha^2) \quad \text{(using } g_j = \overline{g}_j + O(\alpha)) \tag{16}$$

Next, we will approximate the MAML gradient. Define U_i as the operator that updates the parameter vector on minibatch i: $U_i(\phi) = \phi - \alpha L_i'(\phi)$.

$$g_{\text{MAML}} = \frac{\partial}{\partial \phi_1} L_k(\phi_k) \tag{17}$$

$$= \frac{\partial}{\partial \phi_1} L_k(U_{k-1}(U_{k-2}(\dots(U_1(\phi_1))))) \tag{18}$$

$$= U_1'(\phi_1) \cdots U_{k-1}'(\phi_{k-1}) L_k'(\phi_k) \qquad \text{(repeatedly applying the chain rule)} \tag{19}$$

$$= \big(I - \alpha L_1''(\phi_1)\big) \cdots \big(I - \alpha L_{k-1}''(\phi_{k-1})\big) L_k'(\phi_k) \qquad \text{(using } U_i'(\phi) = I - \alpha L_i''(\phi)) \tag{20}$$

$$= \left(\prod_{j=1}^{k-1} (I - \alpha L_j''(\phi_j)) \right) g_k \qquad \text{(product notation, definition of } g_k) \tag{21}$$

Next, let's expand to leading order

$$g_{\text{MAML}} = \left(\prod_{j=1}^{k-1} (I - \alpha \overline{H}_j) \right) \left(\overline{g}_k - \alpha \overline{H}_k \sum_{j=1}^{k-1} \overline{g}_j \right) + O(\alpha^2) \tag{22}$$

(replacing $L_j''(\phi_j)$ with $\overline{H}_j$, and replacing g_k using Equation (16))

$$= \left(I - \alpha \sum_{j=1}^{k-1} \overline{H}_j \right) \left(\overline{g}_k - \alpha \overline{H}_k \sum_{j=1}^{k-1} \overline{g}_j \right) + O(\alpha^2) \tag{23}$$

$$= \overline{g}_k - \alpha \sum_{j=1}^{k-1} \overline{H}_j \overline{g}_k - \alpha \overline{H}_k \sum_{j=1}^{k-1} \overline{g}_j + O(\alpha^2) \tag{24}$$

For simplicity of exposition, let's consider the $k = 2$ case, and later we'll provide the general formulas.

$$g_{\text{MAML}} = \overline{g}_2 - \alpha \overline{H}_2 \overline{g}_1 - \alpha \overline{H}_1 \overline{g}_2 + O(\alpha^2) \tag{25}$$

$$g_{\text{FOMAML}} = g_2 = \overline{g}_2 - \alpha \overline{H}_2 \overline{g}_1 + O(\alpha^2) \tag{26}$$

$$g_{\text{Reptile}} = g_1 + g_2 = \overline{g}_1 + \overline{g}_2 - \alpha \overline{H}_2 \overline{g}_1 + O(\alpha^2) \tag{27}$$

As we will show in the next paragraph, the terms like $\overline{H}_2 \overline{g}_1$ serve to maximize the inner products between the gradients computed on different minibatches, while lone gradient terms like $\overline{g}_1$ take us to the minimum of the joint training problem.

When we take the expectation of g_{FOMAML}, g_{Reptile}, and g_{MAML} under minibatch sampling, we are left with only two kinds of terms which we will call AvgGrad and AvgGradInner. In the equations below $\mathbb{E}_{\tau,1,2}[\dots]$ means that we are taking the expectation over the task τ and the two minibatches defining L_1 and L_2, respectively.

- AvgGrad is defined as gradient of expected loss.

  $$\text{AvgGrad} = \mathbb{E}_{\tau,1}[\overline{g}_1] \tag{28}$$

  $(-\text{AvgGrad})$ is the direction that brings ϕ towards the minimum of the "joint training" problem; the expected loss over tasks.

- The more interesting term is AvgGradInner, defined as follows:

  $$\begin{aligned} \text{AvgGradInner} &= \mathbb{E}_{\tau,1,2}\left[\overline{H}_2\overline{g}_1\right] && (29)\\ &= \mathbb{E}_{\tau,1,2}\left[\overline{H}_1\overline{g}_2\right] && \text{(interchanging indices 1, 2)} \quad (30)\\ &= \tfrac{1}{2}\mathbb{E}_{\tau,1,2}\left[\overline{H}_2\overline{g}_1 + \overline{H}_1\overline{g}_2\right] && \text{(averaging last two equations)} \quad (31)\\ &= \tfrac{1}{2}\mathbb{E}_{\tau,1,2}\left[\frac{\partial}{\partial\phi_1}(\overline{g}_1 \cdot \overline{g}_2)\right] && (32) \end{aligned}$$

  Thus, $(-\text{AvgGradInner})$ is the direction that increases the inner product between gradients of different minibatches for a given task, improving generalization.

Recalling our gradient expressions, we get the following expressions for the meta-gradients, for SGD with $k = 2$:

$$\begin{aligned} \mathbb{E}[g_{\text{MAML}}] &= (1)\text{AvgGrad} - (2\alpha)\text{AvgGradInner} + O(\alpha^2) && (33)\\ \mathbb{E}[g_{\text{FOMAML}}] &= (1)\text{AvgGrad} - (\alpha)\text{AvgGradInner} + O(\alpha^2) && (34)\\ \mathbb{E}[g_{\text{Reptile}}] &= (2)\text{AvgGrad} - (\alpha)\text{AvgGradInner} + O(\alpha^2) && (35) \end{aligned}$$

In practice, all three gradient expressions first bring us towards the minimum of the expected loss over tasks, then the higher-order AvgGradInner term enables fast learning by maximizing the inner product between gradients within a given task.

Finally, we can extend these calculations to the general $k \geq 2$ case:

$$\begin{aligned} g_{\text{MAML}} &= \overline{g}_k - \alpha\overline{H}_k\sum_{j=1}^{k-1}\overline{g}_j - \alpha\sum_{j=1}^{k-1}\overline{H}_j\overline{g}_k + O(\alpha^2) && (36)\\ \mathbb{E}[g_{\text{MAML}}] &= (1)\text{AvgGrad} - (2(k-1)\alpha)\text{AvgGradInner} && (37)\\ g_{\text{FOMAML}} &= g_k = \overline{g}_k - \alpha\overline{H}_k\sum_{j=1}^{k-1}\overline{g}_j + O(\alpha^2) && (38)\\ \mathbb{E}[g_{\text{FOMAML}}] &= (1)\text{AvgGrad} - ((k-1)\alpha)\text{AvgGradInner} && (39)\\ g_{\text{Reptile}} &= -(\phi_{k+1} - \phi_1)/\alpha = \sum_{i=1}^{k} g_i = \sum_{i=1}^{k}\overline{g}_i - \alpha\sum_{i=1}^{k}\sum_{j=1}^{i-1}\overline{H}_i\overline{g}_j + O(\alpha^2) && (40)\\ \mathbb{E}[g_{\text{Reptile}}] &= (k)\text{AvgGrad} - \left(\tfrac{1}{2}k(k-1)\alpha\right)\text{AvgGradInner} && (41) \end{aligned}$$

As in the $k = 2$, the ratio of coefficients of the AvgGradInner term and the AvgGrad term goes MAML $>$ FOMAML $>$ Reptile. However, in all cases, this ratio increases linearly with both the stepsize α and the number of iterations k. Note that the Taylor series approximation only holds for small αk.

They show that the Reptile update maximizes the inner product between gradients of different minibatches from the same task, corresponding to improved generalization. This finding may have implications outside of the meta-learning setting for explaining the generalization properties of SGD. Our analysis suggests that Reptile and MAML perform a very similar update, including the same two terms with different weights.

4 Meta-Learning with Implicit Gradients: iMAML

MAML process requires higher-order derivatives, imposes a non-trivial computational and memory burden, and can suffer from vanishing gradients. They also differentiate through the optimization path as shown in the Figure 2. By leveraging the implicit differentiation approach, they derive an analytical expression for the meta (or outer level) gradient that depends only on the solution to the inner optimization and not the path taken by the inner optimization algorithm.

iMAML reformulates the goal of MAML as to learn models from of the form $h\phi(x) : \mathcal{X} \to \mathcal{Y}$. Performance on a task is specified by a loss function $\mathcal{L}(\phi, \mathcal{D})$ as a function of parameter function and dataset. The goal for task T_i is to learn task-specific parameters ϕ using D_i^{tr} such that we can minimize the population or test loss of the task $\mathcal{L}(\phi_i, \mathcal{D}_i^{test})$. In the general bi-level metal-learning setup, the goal of meta-learning is to learn meta-parameters that produce good task specific parameters after adaptation, as specified below:

$$\theta^*_{ML} := \operatorname*{arg\,min}_{\theta \in \Theta} F(\theta), \text{where } F(\theta) = \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}(Alg(\theta, \mathcal{D}_i^{tr}), \mathcal{D}_i^{test}).$$

we typically interpret $Alg(\theta, \mathcal{D}_i^{tr}$ as either explicitly or implicitly solving an underlying optimization problem, so that we can get a generalization performance by using the adaptation procedure with the meta-learned parameters as $\phi_j = Alg(\theta^*_{ML}, \mathcal{D}_j^{tr})$.

To avoid over-fitting in the inner level, they consider a more explicitly regularized algorithm:

$$Alg^\star(\theta, \mathcal{D}_i^{tr}) = \operatorname*{arg\,min}_{\phi' \in \Phi} \mathcal{L}(\phi', \mathcal{D}_i^{tr}) + \frac{\lambda}{2}||\phi' - \theta||^2$$

With notation

$$\mathcal{L}_i(\phi) := \mathcal{L}(\phi, \mathcal{D}_i^{test}), \widehat{\mathcal{L}} := \mathcal{L}\phi, \mathcal{D}_i^{tr}), Alg_i(\theta) := Alg(\theta, \mathcal{D}_i^{tr})$$

Then we can rewrite the bi-level meta-learning problem as:

$$Alg_i^\star(\theta) := \operatorname*{arg\,min}_{\phi' \in \Phi} G_i(\phi', \theta), \text{where } G_i(\phi', \theta) = \widehat{\mathcal{L}}_i(\phi') + \frac{\lambda}{2}||\phi' - \theta||^2 \text{ for inner level, and}$$

$$\theta^*_{ML} := \operatorname*{arg\,min}_{\theta \in \Theta} F(\theta), \text{where } F(\theta) = \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}_i(Alg^\star(\theta)) \text{ for outer level}$$

4.1 The iMAML algorithm

Our aim is to solve bi-level meta-learning problem using an iterative gradient based algorithm of the form. The gradient descent update be expanded using the chain rule as:

$$\theta \leftarrow \theta - \eta\frac{1}{M}\sum_{i=1}^{M} \frac{dAlg_i^\star(\theta)}{d\theta} \bigtriangledown_\phi \mathcal{L}_i(Alg_i^\star(\phi))$$

Here $\bigtriangledown_\phi \mathcal{L}_i(Alg_i^\star(\phi)$ is simply $\bigtriangledown_\phi \mathcal{L}_i(\phi)|_{\phi = Alg^\star(\theta)}$ which can be easily obtained in practice via automatic differentiation. We must compute $\frac{dAlg_i^\star(\theta)}{d\theta}$.

Consider $Alg_i^\star(\phi)$ for task T_i, let $\phi_i = Alg_i^\star(\theta)$ be the result of $Alg_i^\star(\phi)$. If $(I + \frac{1}{\lambda}\bigtriangledown^2_\phi \widehat{\mathcal{L}}_i(\phi_i))^{-1}$ is invertible, then the derivative Jacobian is

$$\frac{dAlg_i^\star(\theta)}{d\theta} = (I + \frac{1}{\lambda}\bigtriangledown^2_\phi \widehat{\mathcal{L}}_i(\phi_i))^{-1}$$

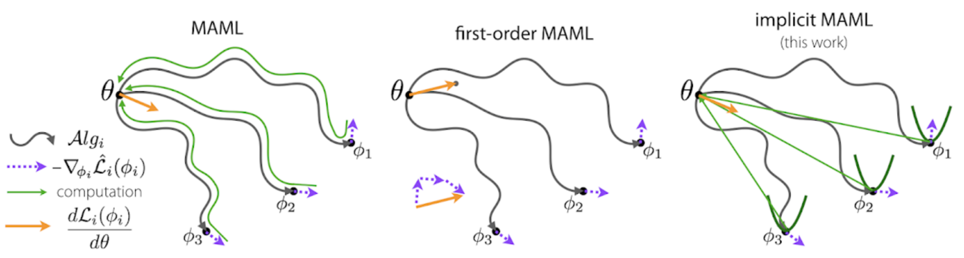

Figure 2: To compute the meta-gradient $\sum_i \frac{d\mathcal{L}_i(\phi_i)}{d\theta}$, the MAML algorithm differentiates through the optimization path, as shown in green, while first-order MAML computes the meta-gradient by approximating $\frac{d\phi_i}{d\theta}$ as I. Our implicit MAML approach derives an analytic expression for the exact meta-gradient without differentiating through the optimization path by estimating local curvature.

Note that the derivative (Jacobian) depends only on the final result of the algorithm, and not the path taken by the algorithm. Thus, in principle any approach of algorithm can be used to compute $Alg_i^\star(\theta)$, thereby decoupling the meta-gradient computation from choice of inner level optimizer. However, it is difficult to use it directly in practice since the meta-gradients require computation of $Alg_i^\star(\theta)$, which is the exact solution to the inner optimization problem, we can maybe only get the approximate solutions.

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

Definition 1. (δ-approx. algorithm) Let$Alg_i(\theta)$ be a δ-accurate approximation of $Algo_i^\star(\theta)$, i.e.,

$$\|Alg_i(\theta) - Alg_i^\star(\theta)\| \leq \delta$$

We will perform a partial or approximate matrix inversion given by:

Definition 2. ($\delta^{'}-$ approximates Jacobian-vector product)let g_i be a vector such that

$$\|g_i - (I + \frac{1}{\lambda}\bigtriangledown_\phi^2 \widehat{\mathcal{L}}_i(\phi_i))^{-1}) \bigtriangledown_\phi \mathcal{L}_i(\phi_i)\| \leq \delta^{'}$$

where $\phi_i = Alg_i(\theta)$ and Alg_i is based on the definition 1, therefore g_i in the above equation is an approximation of the meta-gradient for task T_i. Observe that g_i can be obtained as an approximate solution to the optimization problem according to the CG algorithm:

$$\min_w w^T(I + \frac{1}{\lambda}\bigtriangledown_\phi^2 \widehat{\mathcal{L}}_i(\phi_i))w - w^T \bigtriangledown_\phi \mathcal{L}i(\phi_i)$$

Then we get the iMAML algorithm as shown in Figure 3.

5 Auto-Meta: Automated Gradient Based Meta Learner Search

The goal in Kim et al. [2018] is to automatically find the optimal network architecture for gradient-based meta-learners. Considering the loss function $\mathcal{L}$ for given tasks j represented by training and test data sets (D_j^{tr}, D_j^{test}), this can be for formulated as:

$$\min_{A,\theta} \sum_j \mathcal{L}D_j^{test}, U(D_j^{tr}, \theta; A))$$

where A and θ are the neural network architecture and its parameters, respectively. U denotes the computation of parameter updates using one or more gradient descent steps. A natural and simply way to solve this problem is to minimize the loss $\mathcal{L}$ over parameters, and keep the candidate architectures fixed. Then based on some promising architectures, more complicated architectures are searched progressively. By repeating these two steps, we can obtain a good approximate solution to Equation above.

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

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

Algorithm 2 Implicit Meta-Gradient Computation

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

Figure 3: Algorithm for iMAML

As the gradient-based meta-learning algorithm, they adopt Reptile. As the network architecture search method, they use the PNAS algorithm [Liu et al., 2018] where three layers (i.e., block, cell, and network) of abstraction for representing a neural network topology were defined. At most B blocks which represent a combination operators applied to two inputs are included in a cell. This cell is then stacked a certain number of times to create a full CNN. During the architecture search, the cells "progressively" get more complicated by adding a block to themselves. Without expensive training procedure, the performance of each cell is evaluated with a surrogate predictor, such as LSTM to rank all expanded candidate cells. Then, CNNs with the top K cells are trained and evaluated. We continue in this way until each cell has the maximum number of blocks.

This gradient based meta learners with automatically search architectures have much better results than other meta-learners with human-crafted models on some few shot image classification tasks and compatible results with state-of-the-art techniques which employed more sophisticated auxiliary components such as encoder and decoder networks for the tasks.

References

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-Volume 70*, pages 1126–1135. JMLR. org, 2017.

J. Kim, S. Lee, S. Kim, M. Cha, J. K. Lee, Y. Choi, Y. Choi, D.-Y. Cho, and J. Kim. Auto-meta: Automated gradient based meta learner search. *arXiv preprint arXiv:1806.06927*, 2018.

G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2, 2015.

C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

S. Thrun and L. Pratt. *Learning to learn*. Springer Science & Business Media, 2012.