Topic 1: Model-Agnostic Meta-Learning

BY WANG MA

Department of Statistics and Data Science

Southern University of Science and Technology (SUSTech)

E-mail: maw2020@mail.sustech.edu.cn

This seminar is supervised by Prof. Chao Wang, and the main organizer is Shengjie Niu. Learn more about the seminar at 23 Summer Seminar.

1 Model-Agnostic Meta-Learning (MAML)

Fact: Almost all Deep Learnning models learn through backpropagation of ***Gradients***.

But... Those Gradient-based methods are

– neither designed to cope with a small numer of training samples

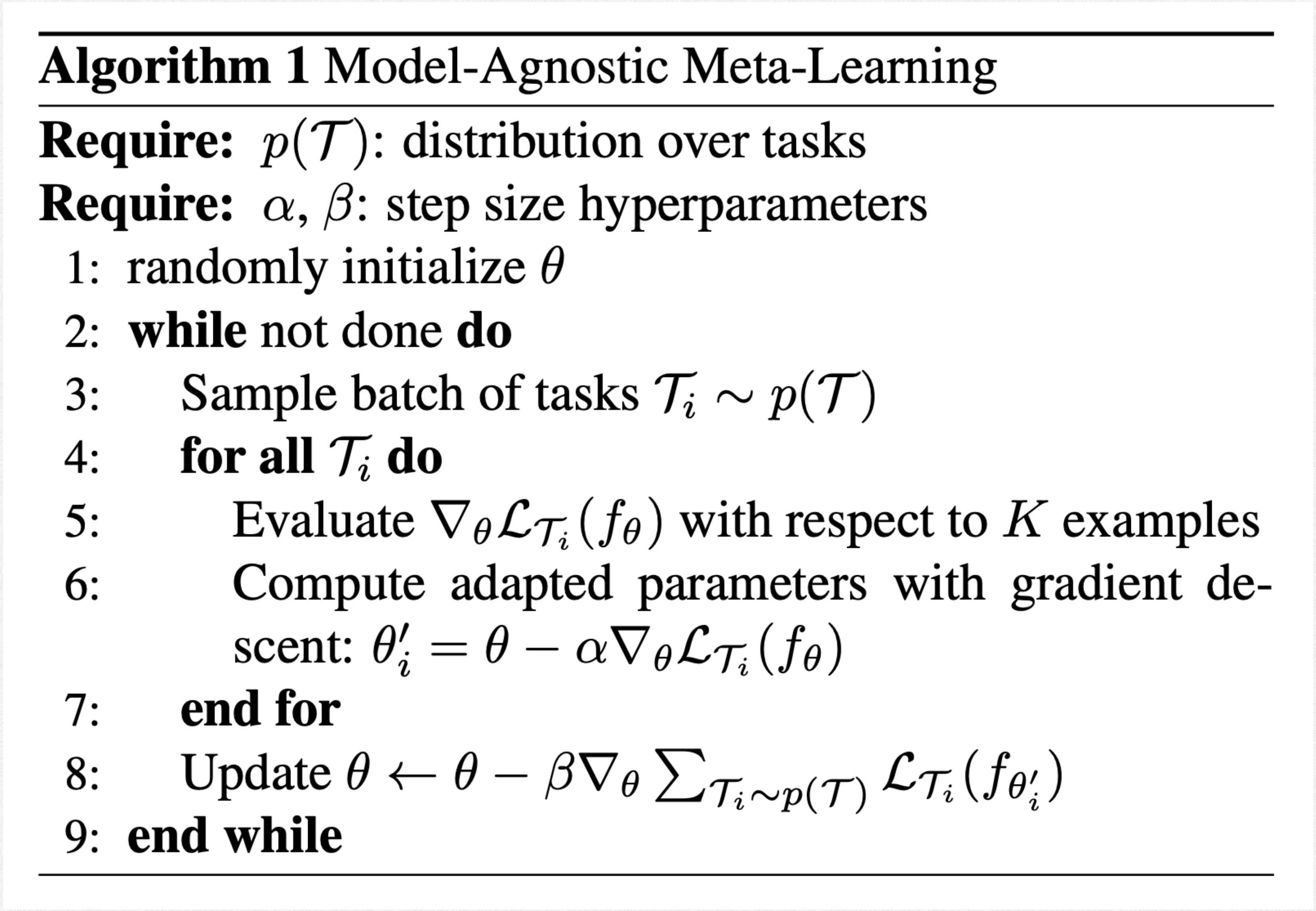
– nor to converge within a small number of optimization steps.

(Lilian Weng, https://lilianweng.github.io/posts/2018-11-30-meta-learning/)

Q: Is there an algorithm such that a small number of gradient updates will lead to fast learning on a new task?

A: Yes! It's MAML.

1.1 The MAML Algorithm



Algorithm 1 Model-Agnostic Meta-Learning

```
Require: $p(\mathcal{T})$: distribution over tasks
Require: $\alpha$, $\beta$: step size hyperparameters
 1: randomly initialize $\theta$
 2: while not done do
 3:     Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
 4:     for all $\mathcal{T}_i$ do
 5:         Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ with respect to $K$ examples
 6:         Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
 7:     end for
 8:     Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$
 9: end while
```

During each training task τ_i (inner loop), we take one gradient update:

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\tau_i}(f_\theta),$$

and the overall loss function is

$$\min_\theta \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\tau_i}(f_{\theta_i'}) = \min_\theta \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\tau_i}(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\tau_i}(f_\theta)}),$$

which is also called meta-objective.

Lastly, the model parameters θ are updated as follows:

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\tau_i \sim p(\tau)} \mathcal{L}_{\tau_i}(f_{\theta_i'}). \tag{1}$$

1.2 Species of MAML

1.2.1 Loss function Used for Supervised Regression and Classification Tasks

1. Supervised Regression Tasks

   Mean-Squared Error = MSE:

$$\mathcal{L}_{\tau_i}(f_\phi) = \sum_{x^{(j)}, y^{(j)} \sim \tau_i} \| f_\phi(x^{(j)}) - y^{(j)} \|_2^2. \tag{2}$$

2. Discrete Classification Tasks

   Cross-Entropy:

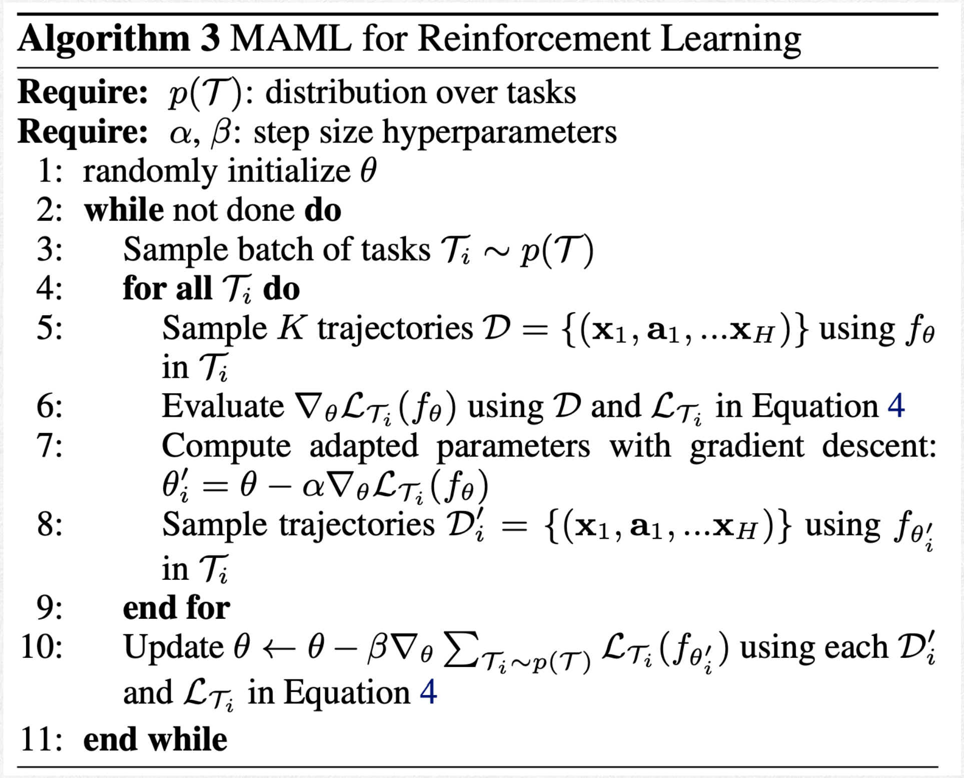
$$\begin{aligned}\mathcal{L}_{\tau_i}(f_\phi) = \sum_{x^{(j)}, y^{(j)} \sim \tau_i} & y^{(j)} \log(f_\phi(x^{(j)})) \\ & + (1 - y^{(j)}) \log(1 - f_\phi(x^{(j)})).\end{aligned}$$

1.2.2 Reinforcement Learning

Task τ_i:

- initial state: $q_i(x_1)$
- transition distribution: $q_i(x_{t+1}|x_t, a_t)$
- loss: $\mathcal{L}_{\tau_i}(f_\phi) = -\mathbb{E}_{x_t, a_t \sim f_\phi, Q_i}[\sum_{t=1}^{H} R_i(x_t, a_t)]$ (negative reward)

We want to learn the function $f_\phi \colon x_t \rightarrow a_t$



Algorithm 3 MAML for Reinforcement Learning

```
Require: $p(\mathcal{T})$: distribution over tasks
Require: $\alpha$, $\beta$: step size hyperparameters
 1: randomly initialize $\theta$
 2: while not done do
 3:     Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$
 4:     for all $\mathcal{T}_i$ do
 5:         Sample $K$ trajectories $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ using $f_\theta$ in $\mathcal{T}_i$
 6:         Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 4
 7:         Compute adapted parameters with gradient descent: $\theta'_i = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
 8:         Sample trajectories $\mathcal{D}'_i = \{(\mathbf{x}_1, \mathbf{a}_1, ...\mathbf{x}_H)\}$ using $f_{\theta'_i}$ in $\mathcal{T}_i$
 9:     end for
10:     Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i})$ using each $\mathcal{D}'_i$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation 4
11: end while
```

2 First-Order MAML

Clearly, the MAML algorithm relies on second derivatives. Here we introduce the **First-Order MAML**, a modified version of MAML omitting second derivatives, in order to make the computation less expensive.

To show why MAML needs second derivative:

Firstly, we consider the inner loop performing k gradient steps, where $k \geqslant 1$. We start with the initial parameter $\theta_0 = \theta_{\text{meta}}$:

$$\begin{aligned}
\theta_0 &= \theta_{\text{meta}} \\
\theta_1 &= \theta_0 - \alpha \nabla_\theta \mathcal{L}(\theta_0) \\
\theta_2 &= \theta_1 - \alpha \nabla_\theta \mathcal{L}(\theta_1) \\
&\dots. \\
\theta_k &= \theta_{k-1} - \alpha \nabla_\theta \mathcal{L}(\theta_{k-1}).
\end{aligned}$$

Then in the outer loop, we update the meta-objective wiht a new batch of data:

$$\theta_{\text{meta}} \leftarrow \theta_{\text{meta}} - \beta g_{\text{MAML}},$$

where

$$\begin{aligned} g_{\text{MAML}} =& \nabla_{\theta} \mathcal{L}(\theta_k) \\ =& \nabla_{\theta_k} \mathcal{L}(\theta_k) \cdot (\nabla_{\theta_{k-1}} \theta_k) \cdots (\nabla_{\theta_0} \theta_1) \cdot (\nabla_{\theta} \theta_0) \\ =& \nabla_{\theta_k} \mathcal{L}(\theta_k) \cdot \left(\prod_{i=1}^{k} \nabla_{\theta_{i-1} \theta_i} \right) \cdot I \\ =& \nabla_{\theta_k} \mathcal{L}(\theta_k) \cdot \prod_{i=1}^{k} \nabla_{\theta_{i-1}} (\theta_{i-1} - \alpha \nabla_{\theta} \mathcal{L}(\theta_{i-1})) \\ =& \nabla_{\theta_k} \mathcal{L}(\theta_k) \cdot \prod_{i=1}^{k} (I - \alpha \nabla_{\theta_{i-1}} (\nabla_{\theta} (\theta_{i-1}))). \end{aligned}$$

The First-Order MAML ignoresthe second derivative part in red. It is simplified as follows:

$$g_{\text{FOMAML}} = \nabla_{\theta_k} \mathcal{L}^{(1)}(\theta_k),$$

where $\mathcal{L}^{(1)}$ implies the loss in outer loop.

3 Reptile

3.1 Reformaling FOMAML

We rewrite the optimization problem of MAML:

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

where U_{τ}^{k} is the operator that updates ϕ k times using data sampled from τ. Omitting the superscript k, this optimization problem can be rewritten as

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

where A represents the training samples in inner loop and B is the test samles used to compute the loss. Then we have

$$\begin{aligned} g_{\mathrm{MAML}} &= \frac{\partial}{\partial \phi} L_{\tau,B}(U_{\tau,A}(\phi)) \\ &= U'_{\tau,A}(\phi) L'_{\tau,B}(\tilde{\phi}), \quad \text{where } \tilde{\phi} = U_{\tau,A}(\phi). \end{aligned}$$

Notice that $U_{\tau,A}$ can be considered as adding a sequence vectors to the initial vector, i.e., $U_{\tau,A} = \phi + g_1 + g_2 + \cdots + g_k$. FOMAML treats each g_i as constants, then $U'_{\tau,A}(\phi) = I$.

This way, FOMAML can be implemented in a particular simple way:

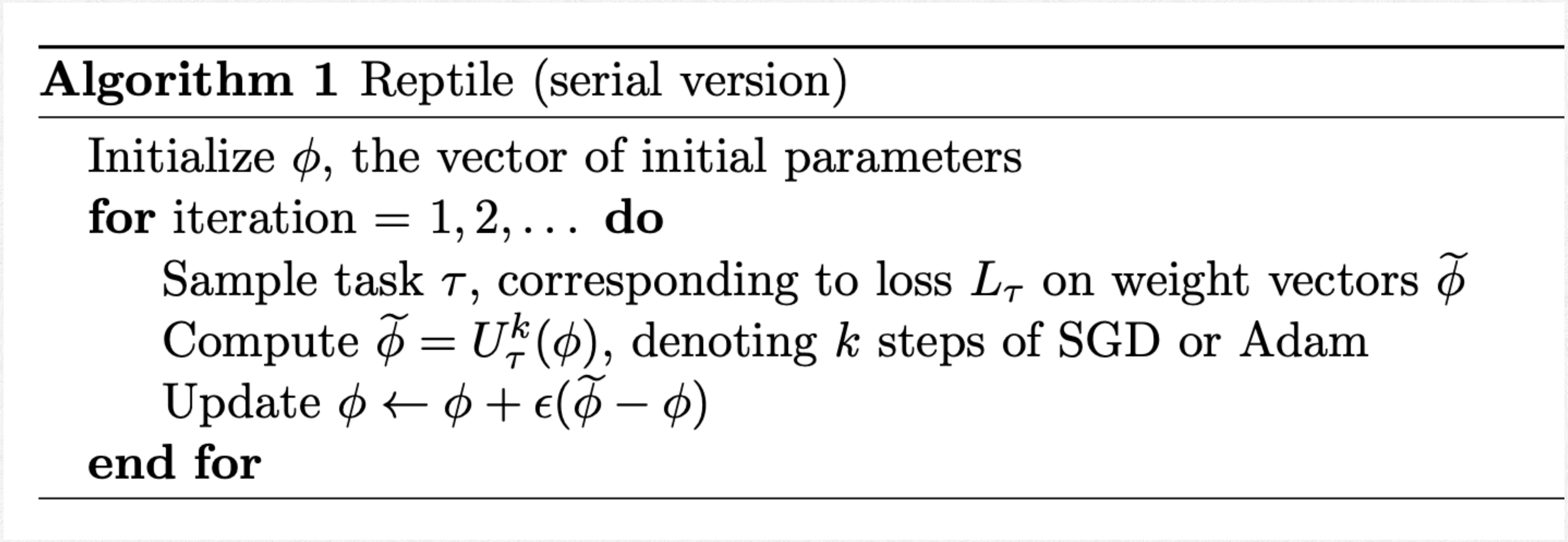
i. sample task τ

ii. apply the update operator, yielding $\tilde{\phi} = U_{\tau,A}(\phi)$

iii. compute the gradient at $\tilde{\phi}$, say, $g_{\mathrm{FOMAML}} = L'_{\tau,B}(\tilde{\phi})$

iv. plug g_{FOMAML} into the outer loop optimizer.

3.2 Reptile Algorithm



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
```

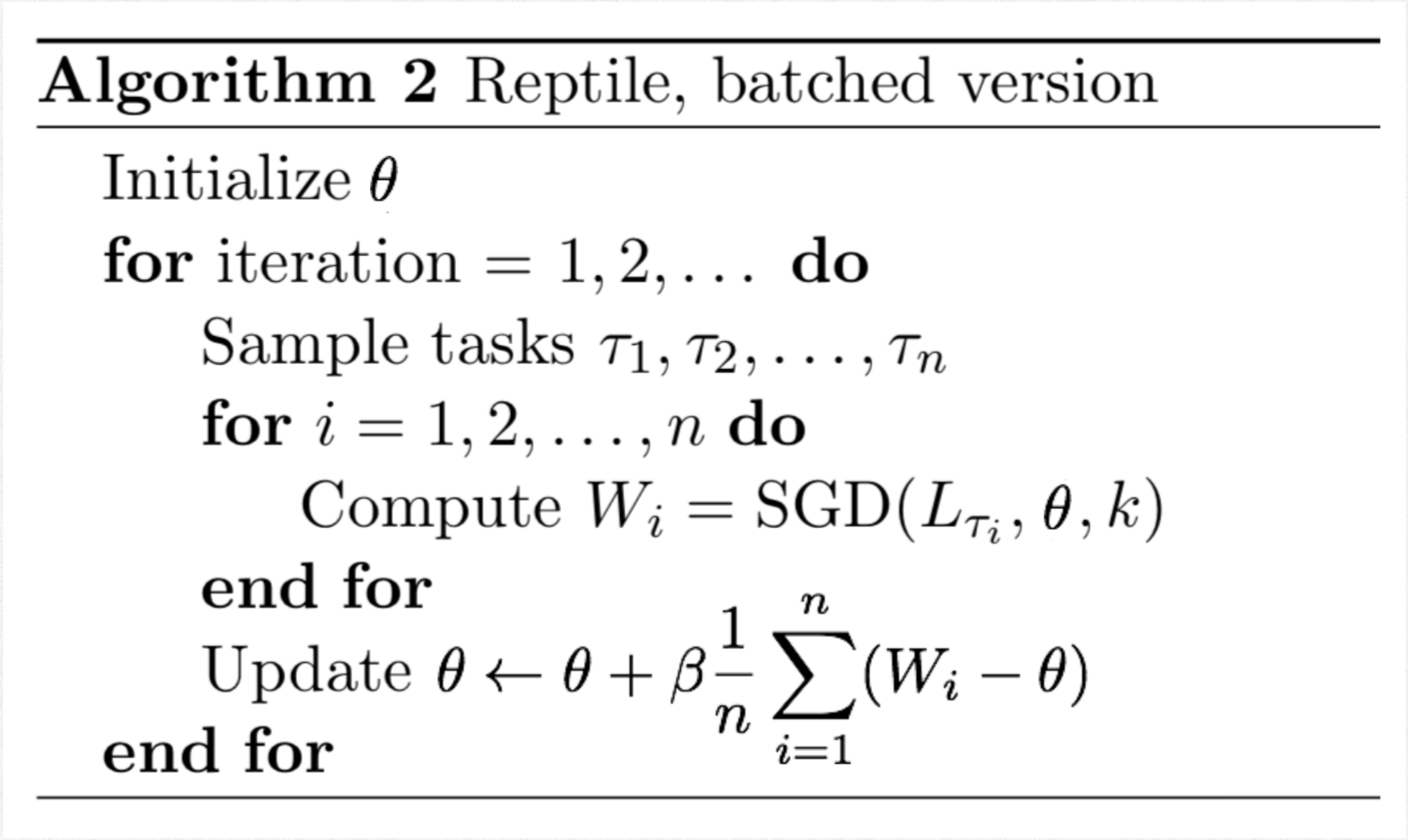
The Reptile works by repeatedly:

i. sampling a task τ

ii. update ϕ multiple steps and get $\tilde{\phi}$

iii. moving ϕ towards $\tilde{\phi}$ with some stepsize.

A batched version:

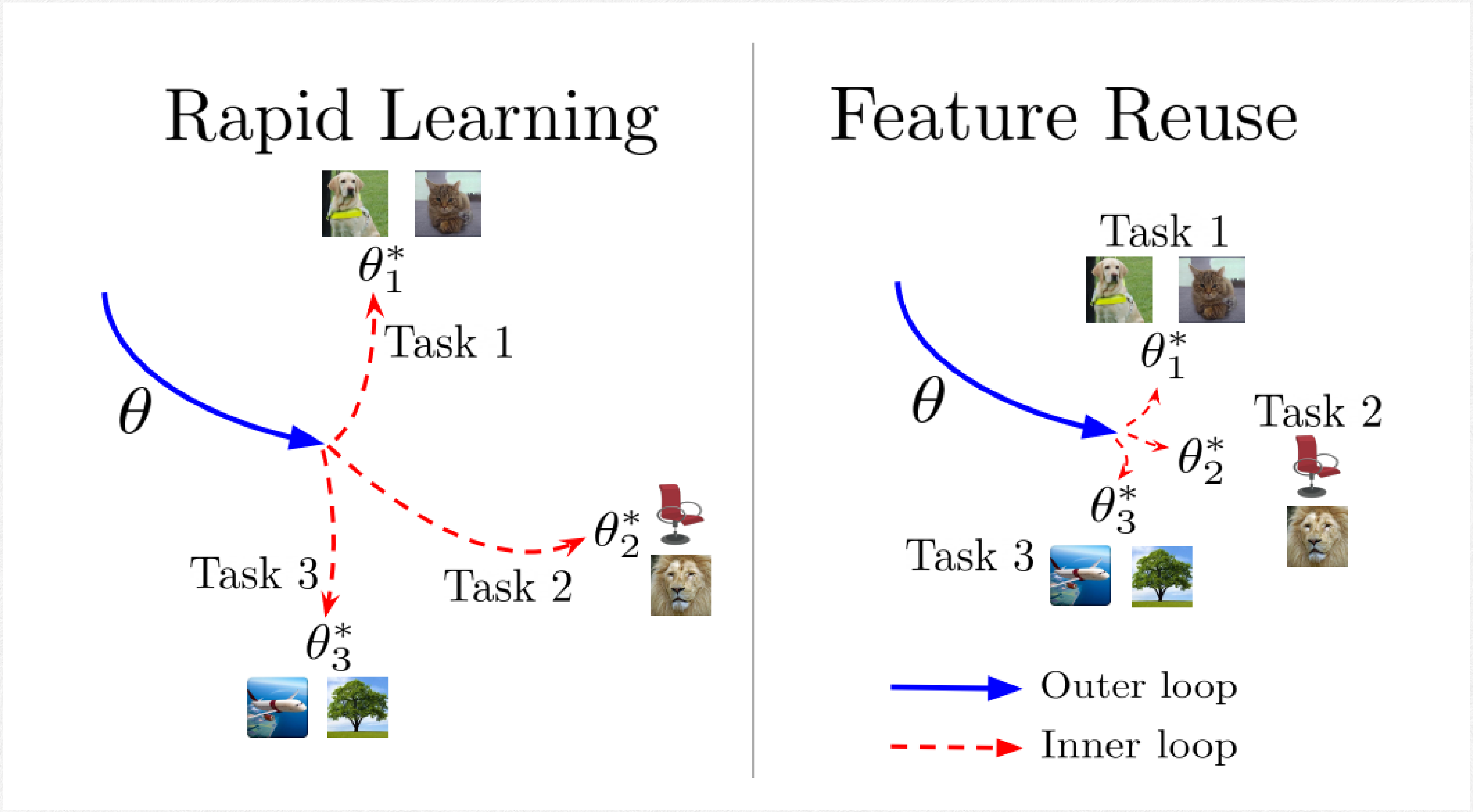


Algorithm 2 Reptile, batched version

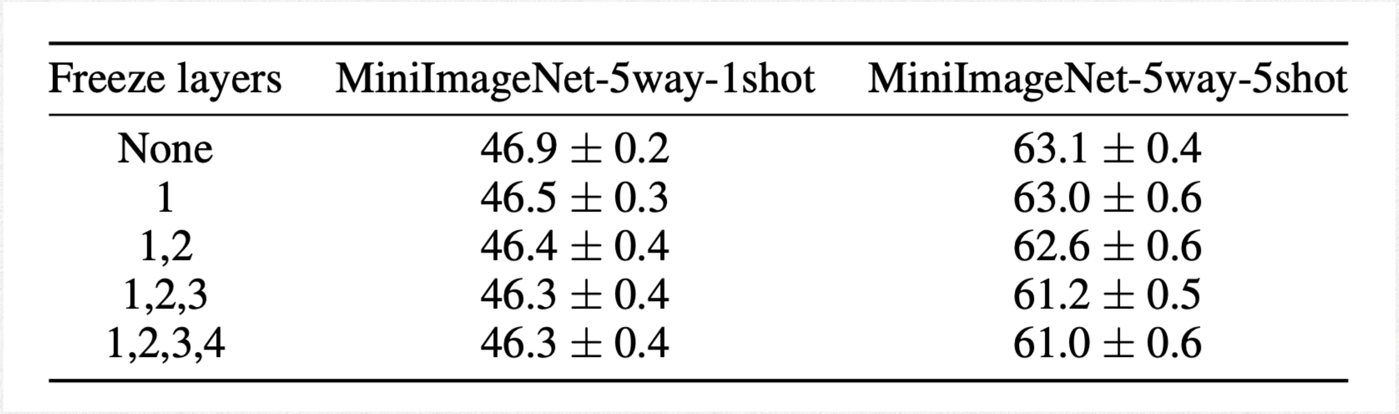
```
Initialize $\theta$
for iteration $= 1, 2, \ldots$ do
    Sample tasks $\tau_1, \tau_2, \ldots, \tau_n$
    for $i = 1, 2, \ldots, n$ do
        Compute $W_i = \mathrm{SGD}(L_{\tau_i}, \theta, k)$
    end for
    Update $\theta \leftarrow \theta + \beta \frac{1}{n} \sum_{i=1}^{n} (W_i - \theta)$
end for
```

$\mathrm{SGD}(L_{\tau_i}, \theta, k)$ performs SGD for k steps on the loss L_{τ_i} starting with the initial parameter θ and returns the final parameter vector, and the reptile gradient is defined as $\frac{\theta - W}{\alpha}$ during the inner loop.

Rapid Learning or Feature Reuse?

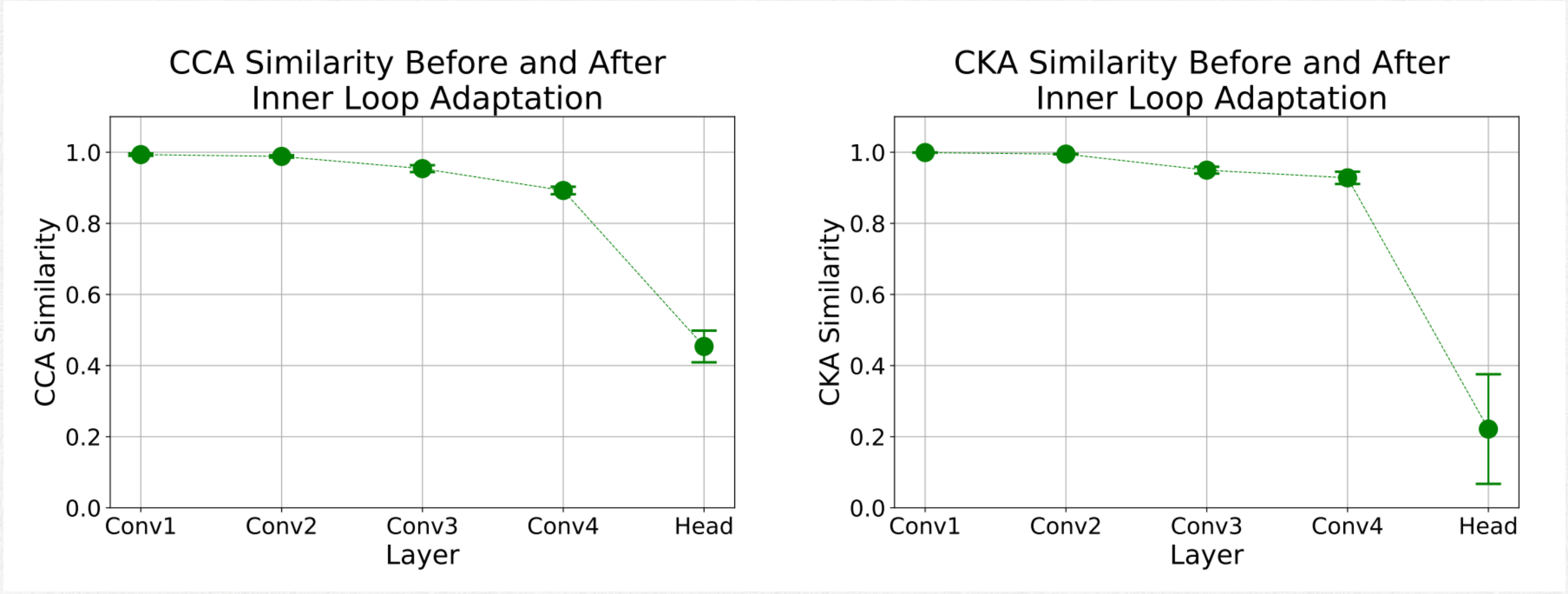


4.1 Without test time inner loop adaptation



Freeze layers	MiniImageNet-5way-1shot	MiniImageNet-5way-5shot
None	46.9 ± 0.2	63.1 ± 0.4
1	46.5 ± 0.3	63.0 ± 0.6
1,2	46.4 ± 0.4	62.6 ± 0.6
1,2,3	46.3 ± 0.4	61.2 ± 0.5
1,2,3,4	46.3 ± 0.4	61.0 ± 0.6

4.2 Dierectly analyze how much the network features change through the inner loop



4.3 Features reuse happens early in Learning

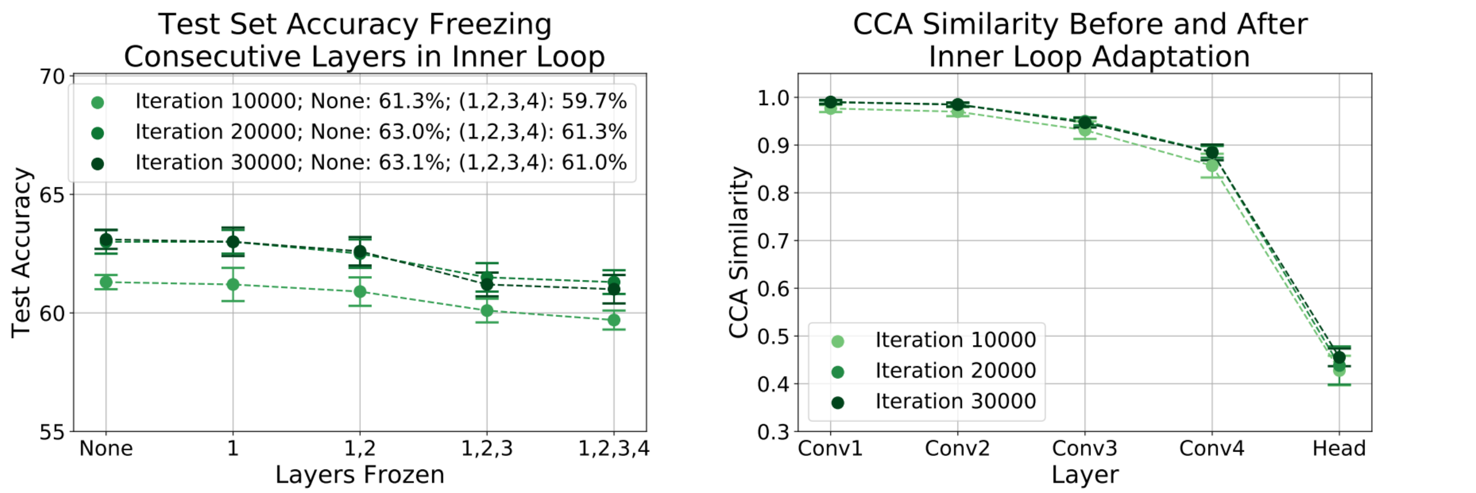
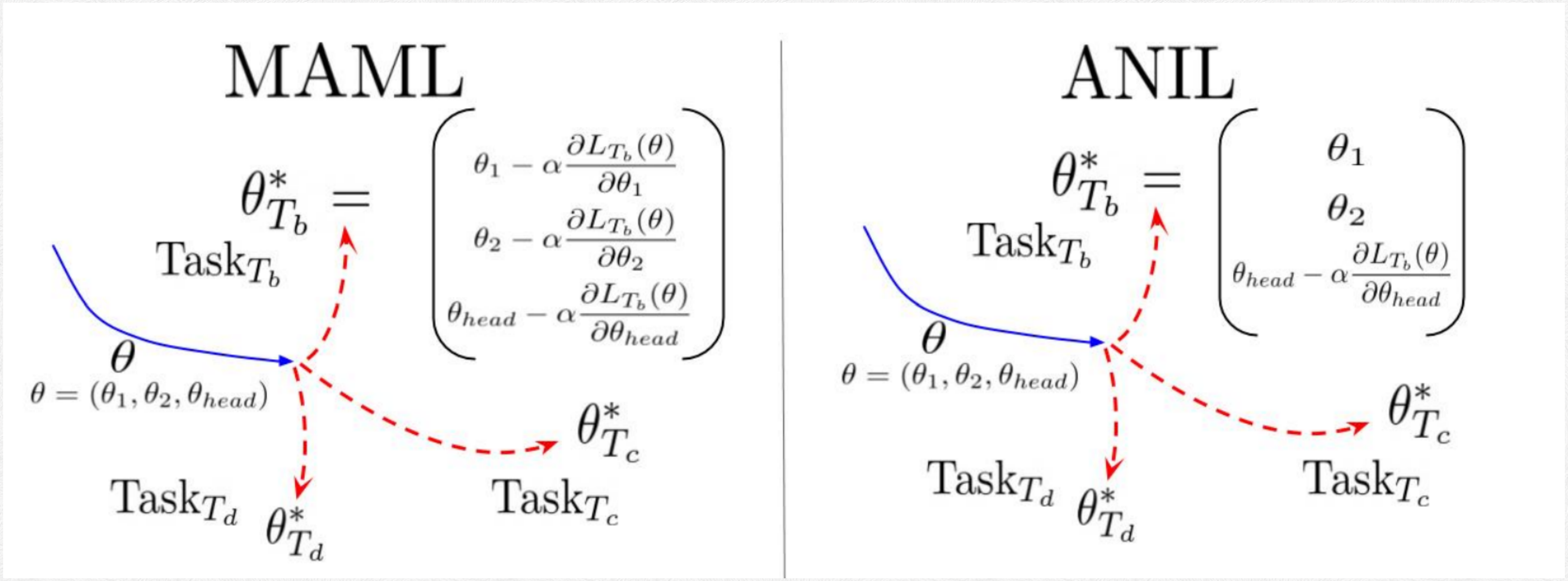


Figure 3: **Inner loop updates have little effect on learned representations from early on in learning.** Left pane: we freeze contiguous blocks of layers (no adaptation at test time), on MiniImageNet-5way-5shot and see almost identical performance. Right pane: representations of all layers except the head are highly similar pre/post adaptation – i.e. features are being reused. This is true from early (iteration 10000) in training.

4.4 The ANIL (Almost No Inner Loop) Algorithm


MAML
$\theta^*_{T_b} = \begin{pmatrix} \theta_1 - \alpha \frac{\partial L_{T_b}(\theta)}{\partial \theta_1} \\ \theta_2 - \alpha \frac{\partial L_{T_b}(\theta)}{\partial \theta_2} \\ \theta_{head} - \alpha \frac{\partial L_{T_b}(\theta)}{\partial \theta_{head}} \end{pmatrix}$
Task_{T_b}
θ
$\theta = (\theta_1, \theta_2, \theta_{head})$
$\theta^*_{T_c}$
Task_{T_d}
$\theta^*_{T_d}$
Task_{T_c}
ANIL
$\theta^*_{T_b} = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \theta_{head} - \alpha \frac{\partial L_{T_b}(\theta)}{\partial \theta_{head}} \end{pmatrix}$
Task_{T_b}
θ
$\theta = (\theta_1, \theta_2, \theta_{head})$
$\theta^*_{T_c}$
Task_{T_d}
$\theta^*_{T_d}$
Task_{T_c}

In ANIL, during training and testing, we remove the inner loop updates for the network body, and apply inner loop adaptation only to the head. The head requires the inner loop to allow it to align to the different classes in each task.

Mathematically, let meta-initialization be $\theta = (\theta_1, \ldots, \theta_l)$ for the l layers of the network, then we have that

$$\theta_m^{(b)} = \Big(\theta_1, \ldots, (\theta_l)_{m-1}^{(b)} - \alpha \nabla_{(\theta_l)_{m-1}^{(b)}} \mathcal{L}_b\big(f_{\theta_{m-1}^{(b)}}\big)\Big),$$

where $\theta_m^{(b)}$ refers the parameters after m inner gradient updates for task τ_b.