On First-Order Meta-Learning Algorithms

Alex Nichol, Joshua Achiam, and John Schulman

Presented by Francis Yu, Zhenhailong Wang
Outline

- Motivation
- Recap: MAML
- Algorithm: Reptile
- Analysis: Why Reptile works
- Experiment
- Conclusion
Motivation

- Meta Learning: Finding a good initialization of a network that can quickly adapt to new tasks
- Problem of MAML: Needs to compute Hessian wrt model parameter; GPU out-of-memory
- Build a meta-learning algorithm based on first-order gradients
Recap: MAML

- **Objective:**
  
  \[
  \minimize_{\phi} \mathbb{E}_\tau[L_{\tau,B}(U_{\tau,A}^k(\phi))]
  \]

  \(\tau\): sampled task
  
  \(\phi\): model parameter
  
  \(A\): support set
  
  \(B\): query set
  
  \(U_{\tau,A}^k\): the operator that updates \(\phi\) for \(k\) steps on support set
  
  \(L_{\tau,B}\): Loss on query set

- **Gradient:**
  
  \[
  g_{\text{MAML}} = \frac{\partial}{\partial \phi} L_{\tau,B}(U_{\tau,A}(\phi)) = U'_{\tau,A}(\phi)L'_{\tau,B}(\tilde{\phi})
  \]

  \(\tilde{\phi} : U_{\tau,A}(\phi) = \phi + g_1 + g_2 + \cdots + g_k\)
Algorithm: Reptile

- No support/query split
- No calculation of loss function/derivative using $\tilde{\phi}$

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 $\tilde{\phi}$  

Compute $\tilde{\phi} = U^k_\tau(\phi)$, denoting $k$ steps of SGD or Adam  

Update $\phi \leftarrow \phi + \epsilon(\tilde{\phi} - \phi)$  

end for
Algorithm: Reptile

No support/query split
No calculation of loss function/derivative using $\tilde{\phi}$

Algorithm 1 Reptile (serial version)

Initialize $\phi$, the vector of initial parameters

for iteration = 1, 2, ... do

Sample task $\tau$, corresponding to loss $L_{\tau}$ on weight vectors $\tilde{\phi}$

Compute $\tilde{\phi} = U_{\tau}^k(\phi)$, denoting $k$ steps of SGD or Adam

Update $\phi \leftarrow \phi + \epsilon(\tilde{\phi} - \phi)$

end for
Algorithm: Reptile

Algorithm 1 Reptile (serial version)

Initialize $\phi$, the vector of initial parameters

for iteration = 1, 2, ... do

Sample task $\tau$, corresponding to loss $L_\tau$ on weight vectors $\phi$

Compute $\tilde{\phi} = U_\tau^k(\phi)$, denoting $k$ steps of SGD or Adam

Update $\phi \leftarrow \phi + \epsilon(\tilde{\phi} - \phi)$

end for

batch version

$\phi \leftarrow \phi + \frac{1}{n} \sum_{i=1}^{n} (\tilde{\phi}_i - \phi)$

[1]
Algorithm: Relation to MAML|FOMAML|iMAML

Meta-gradient:

\[ g_{MAML} = U'_{\tau, A}(\phi) L'_{\tau, B}(\phi) \]
Algorithm: Relation to MAML|FOMAML|iMAML

- Updating $\phi$ using only the final parameters $\tilde{\phi}_i$

Meta-graident:

$$g_{MAML} = U'_{\tau,A}(\phi) L'_{\tau,B}(\tilde{\phi})$$
Algorithm: Reptile

MAML:

\[ \tilde{\phi}_i = U_{\tau_i,A}(\phi), \forall i = 1 \text{ to } n \]

\[ \phi \leftarrow \phi - \beta \nabla_{\phi} \sum_{i=1}^{n} L_{\tau_i,B}(\tilde{\phi}_i) \]

Reptile (batch version):

\[ \phi \leftarrow \phi + \epsilon \frac{1}{n} \sum_{i=1}^{n} (\tilde{\phi}_i - \phi) \] [1]

- “Reptile Gradient” is defined as:

\[ g_{\text{Reptile}} := (\phi - \tilde{\phi})/\alpha \]

where \( \alpha \) is the stepsize used by SGD operation

- Alternatively, instead of simply updating \( \phi \) in the direction of \( (\tilde{\phi} - \phi) \), we can plug the Reptile Gradient into an adaptive algorithm such as Adam.
Algorithm: Relation to SGD

- Similar to joint training on the expected loss \( \mathbb{E}_\tau [L_\tau] \)
- \( k = 1 \), Reptile corresponds to SGD on the expected loss
- \( k > 1 \) the update will include important terms coming from second-and-higher order derivatives of each task.

\[
\begin{align*}
\nabla \text{Reptile}, k=1 &= \mathbb{E}_\tau [(\phi - \tilde{\phi})/\alpha] \\
&= \mathbb{E}_\tau [\phi - U_\tau(\phi)]/\alpha \\
&= \mathbb{E}_\tau [\phi - (\phi + g_1)]/\alpha \\
&= \mathbb{E}_\tau [\phi - (\phi - \alpha \nabla_\phi L_\tau(\phi))] / \alpha \\
&= \mathbb{E}_\tau [\alpha \nabla_\phi L_\tau(\phi)] / \alpha \\
&= \mathbb{E}_\tau [\nabla_\phi L_\tau(\phi)]
\end{align*}
\]
Algorithm: Relation to SimuParallel SGD

- Other than stepsize parameter and task sampling, batch version Retile = SimuParallel SGD
- SimuParallel SGD: sequential SGD -> parallel SGD

Algorithm 3 SimuParallelSGD(Examples \( \{ c^1, \ldots, c^m \} \), Learning Rate \( \eta \), Machines \( k \))

- Define \( T = \lfloor m/k \rfloor \)
- Randomly partition the examples, giving \( T \) examples to each machine.
- **for all** \( i \in \{1, \ldots, k\} \) **parallel do**
  - Randomly shuffle the data on machine \( i \).
  - Initialize \( w_{i,0} = 0 \).
  - **for all** \( t \in \{1, \ldots, T\} \) **do**
    - Get the \( t \)th example on the \( i \)th machine (this machine), \( c^{i,t} \)
    - \( w_{i,t} \leftarrow w_{i,t-1} - \eta \partial_w c^i(w_{i,t-1}) \)
  - **end for**
- **end for**
- Aggregate from all computers \( v = \frac{1}{k} \sum_{i=1}^{k} w_{i,t} \) and **return** \( v \).

\[ \phi \leftarrow \phi + \frac{1}{n} \sum_{i=1}^{n} (\tilde{\phi}_i - \phi) \]
Why Reptile Works?

- Short Answer: Both MAML and Reptile have the same leading-order terms in the meta-gradient, approximated using Taylor series expansion.

\[
\mathbb{E}[g_{\text{MAML}}] = (1)\text{AvgGrad} - (2\alpha)\text{AvgGradInner} + O(\alpha^2)
\]

\[
\mathbb{E}[g_{\text{FOMAML}}] = (1)\text{AvgGrad} - (\alpha)\text{AvgGradInner} + O(\alpha^2)
\]

\[
\mathbb{E}[g_{\text{Reptile}}] = (2)\text{AvgGrad} - (\alpha)\text{AvgGradInner} + O(\alpha^2)
\]

- minimizes the expected loss
- maximizes within-task generalization
Leading Order Expansion of the Gradient

We assume each task gives us a sequence of $k$ loss functions $L_1, ..., L_k$. We define the following notations:

- $g_i = L_i'(\phi_i)$ (gradient obtained during SGD)
- $\phi_{i+1} = \phi_i - \alpha g_i$ (sequence of parameter vectors)
- $\overline{g}_i = L_i'(\phi_1)$ (gradient at initial point)
- $\overline{H}_i = L_i''(\phi_1)$ (Hessian at initial point)
Leading Order Expansion of the Gradient

\[ g_i = L'_i(\phi_i) = L'_i(\phi_1) + L''_i(\phi_1)(\phi_i - \phi_1) + O(\|\phi_i - \phi_1\|^2) \]  

(Taylor’s theorem)

\[ = \bar{g}_i + \bar{H}_i (\phi_i - \phi_1) + O(\alpha^2) \]  

(using definition of \( \bar{g}_i, \bar{H}_i \))

\[ = \bar{g}_i - \alpha \bar{H}_i \sum_{j=1}^{i-1} g_j + O(\alpha^2) \]  

(using \( \phi_i - \phi_1 = -\alpha \sum_{j=1}^{i-1} g_j \))

\[ = \bar{g}_i - \alpha \bar{H}_i \sum_{j=1}^{i-1} \bar{g}_j + O(\alpha^2) \]  

(using \( g_j = \bar{g}_j + O(\alpha) \))
Leading Order Expansion of the Gradient: MAML

\[ g_{\text{MAML}} = \frac{\partial}{\partial \phi_1} L_k(\phi_k) \]

\[ = \frac{\partial}{\partial \phi_1} L_k(U_{k-1}(U_{k-2}(\cdots (U_1(\phi_1)))))) \]

\[ = U_1'(\phi_1) \cdots U_{k-1}'(\phi_{k-1}) L_k'(\phi_k) \]

\[ = (I - \alpha L_1''(\phi_1)) \cdots (I - \alpha L_{k-1}''(\phi_{k-1})) L_k'(\phi_k) \]

\[ = \left( \prod_{j=1}^{k-1} (I - \alpha L_j''(\phi_j)) \right) g_k \quad \text{(product notation, definition of } g_k) \]
Leading Order Expansion of the Gradient: MAML

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

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

\[
= \overline{g}_k - \alpha \sum_{j=1}^{k-1} \overline{H}_j \overline{g}_k - \alpha \overline{H}_k \sum_{j=1}^{k-1} \overline{g}_j + O(\alpha^2)
\]
Leading Order Expansion of the Gradient: FOMAML

\[
g_{\text{MAML}} = \left( \prod_{j=1}^{k-1} (I - \alpha H_j) \right) \left( \bar{g}_k - \alpha \bar{H}_k \sum_{j=1}^{k-1} \bar{g}_j \right) + O(\alpha^2)
\]

\[
= \left( I - \alpha \sum_{j=1}^{k-1} H_j \right) \left( \bar{g}_k - \alpha \bar{H}_k \sum_{j=1}^{k-1} \bar{g}_j \right) + O(\alpha^2)
\]

\[
= \bar{g}_k - \alpha \sum_{j=1}^{k-1} H_j \bar{g}_j - \alpha \bar{H}_k \sum_{j=1}^{k-1} \bar{g}_j + O(\alpha^2)
\]
Comparison of the Gradient Updates

Suppose $k = 2$, then we have:

\[
\begin{align*}
g_{\text{MAML}} &= g_2 - \alpha \bar{H}_2 g_1 - \alpha \bar{H}_1 g_2 + O(\alpha^2) \\
g_{\text{FOMAML}} &= g_2 - \alpha \bar{H}_2 g_1 + O(\alpha^2) \\
g_{\text{Reptile}} &= g_1 + g_2 = g_1 + g_2 - \alpha \bar{H}_2 g_1 + O(\alpha^2)
\end{align*}
\]

$g_1$ is simply the gradient to optimize the joint training loss.
Comparison of the Gradient Updates: Minibatch View

We define $\text{AvgGrad}$ to be the gradient of the expected loss and $\text{AvgGradInner}$ as follows:

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

$$\text{AvgGradInner} = \mathbb{E}_{\tau,1,2} \left[ H_2 \bar{g}_1 \right]$$

$$= \mathbb{E}_{\tau,1,2} \left[ H_1 \bar{g}_2 \right]$$

$$= \frac{1}{2} \mathbb{E}_{\tau,1,2} \left[ H_2 \bar{g}_1 + H_1 \bar{g}_2 \right]$$

$$= \frac{1}{2} \mathbb{E}_{\tau,1,2} \left[ \frac{\partial}{\partial \phi} (\bar{g}_1 \cdot \bar{g}_2) \right]$$

Thus, for $k = 2$, we have:

$$\mathbb{E} [g_{\text{MAML}}] = (1) \text{AvgGrad} - (2\alpha) \text{AvgGradInner} + O(\alpha^2)$$

$$\mathbb{E} [g_{\text{FOMAML}}] = (1) \text{AvgGrad} - (\alpha) \text{AvgGradInner} + O(\alpha^2)$$

$$\mathbb{E} [g_{\text{Reptile}}] = (2) \text{AvgGrad} - (\alpha) \text{AvgGradInner} + O(\alpha^2)$$
Comparison of the Gradient Updates: General K

\[ g_{\text{MAML}} = \bar{g}_k - \alpha \bar{H}_k \sum_{j=1}^{k-1} \bar{g}_j - \alpha \sum_{j=1}^{k-1} \bar{H}_j \bar{g}_k + O(\alpha^2) \]

\[ \mathbb{E}[g_{\text{MAML}}] = (1)\text{AvgGrad} - (2(k-1)\alpha)\text{AvgGradInner} \]

\[ g_{\text{FOMAML}} = g_k = \bar{g}_k - \alpha \bar{H}_k \sum_{j=1}^{k-1} \bar{g}_j + O(\alpha^2) \]

\[ \mathbb{E}[g_{\text{FOMAML}}] = (1)\text{AvgGrad} - ((k-1)\alpha)\text{AvgGradInner} \]

\[ g_{\text{Reptile}} = -(\phi_{k+1} - \phi_1)/\alpha = \sum_{i=1}^{k} g_i = \sum_{i=1}^{k} \bar{g}_i - \alpha \sum_{i=1}^{k} \sum_{j=1}^{i-1} \bar{H}_i \bar{g}_j + O(\alpha^2) \]
2. Solution Manifolds Interpretation

Let $\phi$ denote the network initialization, and let $W_\tau$ denote the set of optimal parameters for task $\tau$. We want to find $\phi$ such that the distance $D(\phi, W_\tau)$ is small for all tasks.

$$\min_{\phi} E_\tau \left[ \frac{1}{2} D(\phi, W_\tau)^2 \right]$$

Given a non-pathological set $S \subset \mathbb{R}^d$, then for almost all points $\phi \in \mathbb{R}^d$ the gradient of the squared distance $D(\phi, S)^2$ is $2(\phi - P_S(\phi))$, where $P_S(\phi)$ is the projection (closest point) of $\phi$ onto $S$. Thus,

$$\nabla_{\phi} E_\tau \left[ \frac{1}{2} D(\phi, W_\tau)^2 \right] = E_\tau \left[ \frac{1}{2} \nabla_{\phi} D(\phi, W_\tau)^2 \right] = E_\tau \left[ \phi - P_{W_\tau}(\phi) \right],$$

where $P_{W_\tau}(\phi) = \arg\min_{p \in W_\tau} D(p, \phi)$.

Each iteration of Reptile corresponds to sampling a task $\tau$ and performing a stochastic gradient update

$$\phi \leftarrow \phi - \epsilon \nabla_{\phi} \frac{1}{2} D(\phi, W_\tau)^2$$

$$= \phi - \epsilon(\phi - P_{W_\tau}(\phi))$$

$$= (1 - \epsilon)\phi + \epsilon P_{W_\tau}(\phi).$$
Datasets

- Sinusoid curve sampled points
- Mini-ImageNet
- Omniglot
Experiments: Sinusoid Curve Approximation

Data: Randomly sample data points on interval [-5, 5]
Goal: approximate the original sinusoid curve on interval [-5, 5]
Experiments: Few-Shot Classification

Table 1: Results on Mini-ImageNet

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>1-shot 5-way</th>
<th>5-shot 5-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAML + Transduction</td>
<td>48.70 ± 1.84%</td>
<td>63.11 ± 0.92%</td>
</tr>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt;-order MAML + Transduction</td>
<td>48.07 ± 1.75%</td>
<td>63.15 ± 0.91%</td>
</tr>
<tr>
<td>Reptile</td>
<td>47.07 ± 0.26%</td>
<td>62.74 ± 0.37%</td>
</tr>
<tr>
<td>Reptile + Transduction</td>
<td>49.97 ± 0.32%</td>
<td>65.99 ± 0.58%</td>
</tr>
</tbody>
</table>

Table 1: Results on Omniglot

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>1-shot 5-way</th>
<th>5-shot 5-way</th>
<th>1-shot 20-way</th>
<th>5-shot 20-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAML + Transduction</td>
<td>98.7 ± 0.4%</td>
<td>99.9 ± 0.1%</td>
<td>95.8 ± 0.3%</td>
<td>98.9 ± 0.2%</td>
</tr>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt;-order MAML + Transduction</td>
<td>98.3 ± 0.5%</td>
<td>99.2 ± 0.2%</td>
<td>89.4 ± 0.5%</td>
<td>97.9 ± 0.1%</td>
</tr>
<tr>
<td>Reptile</td>
<td>95.39 ± 0.09%</td>
<td>98.90 ± 0.10%</td>
<td>88.14 ± 0.15%</td>
<td>96.65 ± 0.33%</td>
</tr>
<tr>
<td>Reptile + Transduction</td>
<td>97.68 ± 0.04%</td>
<td>99.48 ± 0.06%</td>
<td>89.43 ± 0.14%</td>
<td>97.12 ± 0.32%</td>
</tr>
</tbody>
</table>
Experiments: Task Gradient Combinations

Gradient combinations on 5-shot 5-way Omniglot
Conclusion

1. Reptile is a simple but efficient first order meta-learning algorithm
2. The strength of Reptile is that the implementation is simple and the computation and memory cost is significantly lower than MAML since it does not need to compute the Hessian but the performance is comparable.
3. Using two-step task gradient descent is theoretically equivalent to first order meta-learning gradient update with an extra jointly training gradient, and the empirical results also support this analysis.
4. Cons: Task formulation is not clear
Reference


Thanks for Your Attention!
Motivation --- zhenhailong --- 30s
Algorithm --- zhenhailong --- ?min
Relation to SGD --- zhenhailong --- ?min
Relation to SimuParallel --- zhenhailong --- ?min
sinwave case study --- hangyu --- 2min
why Reptile works? --- hangyu --- 5-8min?
experiments --- hangyu ---
  ○ “transfuction” setting
  ○ results
conclusion --- hangyu 0.5 - 1 min