Optimization-based Meta-Learning
Outline

- Brief overview of optim-based meta-learning
  - Meta-learning models aim to learn an **global/initial params** and a **update rule** of how to quickly adapt to individual tasks (i.e., adjusting global initial params to individual tasks)
  - Meta-learning objective is to minimize the expected generalization loss of the meta-learner on the task space
- LSTM-based meta-learning (Meta-LSTM) (Ravi and Larochelle, 2016)
  - Meta-LSTM uses a sequential LSTM update rule
- Meta-SGD (Li et al., 2017)
  - Meta-SGD learns the global learning rate and update direction $\alpha$ addition to the initial params
- Meta-Curvature (Park and Oliva, 2020)
  - Meta-Curvature further improves MAML by adding second-order information into the gradient
    - the additional info is decomposed into 3 component matrices and is multiplied to gradient during inner gradient update steps
- IMO, the current trend is modifying and adding components to the update rules to learn more about individual tasks, but the components are controlled by some global learnable variables
Optim-based Meta-Learning: Intuition

- Meta-learning objective is to minimize the expected generalization loss of the meta-learner on the task space

\[
\min_{\theta_g} \mathbb{E}_{T_e}[L_{Q_e}(\theta_e)] = \mathbb{E}_{T_e}[L_{Q_e}(\theta_g - \alpha \nabla_{\theta_g} L_{S_e}(\theta_g))]
\]

- **inner-loop**: given an individual task \( T_e \), the learner (params \( \theta_e \)) is optimized based on the loss of task support set \( L_{S_e} \)

- **outer-loop**: the loss of task query set \( L_{Q_e} \) is computed with the updated learner and is accumulated to the generalization loss \( L_{\text{global}} = \sum_e L_{Q_e} \). The meta-learner (params \( \theta_g \)) is then optimized based the generalization loss.
Model-Agnostic Meta-Learning (MAML): Model

Algorithm: MAML's training procedure

Require: train meta-set $\mathcal{S}^{train}$
Return: global params $\theta_g$ that are adaptable to many tasks

1: initialize $\theta_g$
2: for epoch in num epochs do
3:   for episode $e$ in num episodes do
4:     sample a local task $\mathcal{T}_e = (\mathcal{S}_e, \mathcal{Q}_e)$ from $\mathcal{S}^{train}$
5:     $\implies$ inner loop: learn local params for the local task via the loss of task support set
6:     set $\theta_0 = \theta_g$
7:     for step $t$ in num steps do
8:       compute loss of the local task support set $L_{S_e} = \sum_{(x, y) \in \mathcal{S}_e} L_{cls}(f(x; \theta_{t-1}), y)$
9:       compute gradient and update local params for the local task $\theta_t = \theta_{t-1} - \alpha \nabla_{\theta_{t-1}} L_{S_e}$
10:      end for
11:     $\implies$ outer loop: learn global params via all the losses of all task query sets
12:     compute loss of the local task query set with local params $L_{Q_e} = \sum_{(x, y) \in \mathcal{Q}_e} L_{cls}(f(x; \theta_T), y)$
13:     accumulate the loss of the local task query set to the global loss $L_{global} += L_{Q_e}$
14:    end for
15:    compute global gradient (w.r.t. sum of all local losses) and update global params $\theta_g = \theta_g - \beta \nabla_{\theta_g} L_{global}$
16: end for
Model-Agnostic Meta-Learning (MAML): Model

**Algorithm**: MAML’s training procedure, inner-loop

**Require**: task support set \(S_e\), global params \(\theta_g\)

**Return**: individual task params \(\theta_T\) after \(T\) update steps

1: \(\mapsto\) **inner loop**: learn local params for the local task via the loss of task support set
2: set \(\theta_0 = \theta_g\)
3: **for** step \(t\) in num steps **do**
4: compute loss of the local task support set
   \[LS_e = \sum_{(x,y) \in S_e} L_{cls}(f(x; \theta_{t-1}), y)\]
5: compute gradient and update local params for the local task
   \[\theta_t = \theta_{t-1} - \alpha \nabla \theta_{t-1} L_{S_e}\]
6: **end for**

**Algorithm**: MAML’s training procedure, outer-loop

**Require**: task query set \(Q_e\), individual task params \(\theta_T\)

**Return**: loss of task query set \(L_{Q_e}\)

1: \(\mapsto\) **outer loop**: learn global params via all the losses of all task query sets
2: compute loss of the local task query set with local params
   \[L_{Q_e} = \sum_{(x,y) \in Q_e} L_{cls}(f(x; \theta_T), y)\]
3: accumulate the loss of the local task query set to the global loss
   \[L_{global} += L_{Q_e}\]
Meta-LSTM vs MAML: Model

Algorithm: MAML’s training procedure, inner-loop

Require: task support set $S_s$, global params $\theta_g$
Return: individual task params $\theta_T$ after $T$ update steps

1: $\rightarrow$ inner loop: learn local params for the local task via the loss of task support set
2: set $\theta_0 = \theta_g$
3: for step $t$ in num steps do
4: compute loss of the local task support set
   $L_{S_s} = \sum_{(x,y) \in S_s} L_{fe}(f(x; \theta_{t-1}), y)$
5: compute gradient and update local params for the local task
   $\theta_t = \theta_{t-1} - \alpha \nabla \theta_{t-1} L_{S_s}$
6: end for

MAML’s objective

$$\min_{\theta_g} \mathbb{E}_{T_e} [L_{Q_e}(\theta_e)] = \mathbb{E}_{T_e} [L_{Q_e}(\theta_{t-1} - \alpha \nabla \theta_{t-1} L_{S_s})]$$

Meta-SGD uses a sequential LSTM update rule, its objective becomes

$$\min_{\theta_g} \mathbb{E}_{T_e} [L_{Q_e}(\theta_e)] = \mathbb{E}_{T_e} [L_{Q_e}(f_{t-1} \odot \theta_{t-1} - i_{t-1} \odot \nabla \theta_{t-1} L_{S_s})]$$
Meta-LSTM vs MAML: Model
Meta-SGD vs MAML: Model

Algorithm: MAML’s training procedure, inner-loop

Require: task support set $S_e$, global params $\theta_g$
Return: individual task params $\theta_T$ after $T$ update steps
1: $\implies$ inner loop: learn local params for the local task via the loss of task support set
2: set $θ_0 = θ_g$
3: for step $t$ in num steps do
4: compute loss of the local task support set $L_{S_e} = \sum_{(x,y) \in S_e} L_{cls}(f(x; \theta_{t-1}), y)$
5: compute gradient and update local params for the local task $\theta_t = \theta_{t-1} - \alpha \nabla \theta_{t-1} L_{S_e}$
6: end for

Algorithm: Meta-SGD’s training procedure, inner-loop

Require: task support set $S_e$, global params $\theta_g$, global lr $\alpha_g$
Return: individual task params $\theta_T$ after $T$ update steps
1: $\implies$ inner loop: learn local params for the local task via the loss of task support set using global params $\theta_g$ and global lr $\alpha_g$
2: set $θ_0 = θ_g$
3: for step $t$ in num steps do
4: compute loss of the local task support set $L_{S_e} = \sum_{(x,y) \in S_e} L_{cls}(f(x; \theta_{t-1}), y)$
5: compute gradient and update local params for the local task $\theta_t = \theta_{t-1} - \alpha_g \nabla \theta_{t-1} L_{S_e}$
6: end for

- **MAML’s objective**

\[
\min_{\theta_g} \mathbb{E}_{T_e} [L_{Q_e}(\theta_e)] = \mathbb{E}_{T_e} [L_{Q_e}(\theta_{t-1} - \alpha \nabla \theta_{t-1} L_{S_e})]
\]

- In addition to learning global initial params as in MAML, Meta-SGD also learns the global learning rate (and update direction) alpha

\[
\min_{\theta_g} \mathbb{E}_{T_e} [L_{Q_e}(\theta_e)] = \mathbb{E}_{T_e} [L_{Q_e}(\theta_{t-1} - \alpha_g \nabla \theta_{t-1} L_{S_e})]
\]
Meta-Curvature vs MAML: Model

Algorithm : MAML’s training procedure, inner-loop

Require: task support set $S_e$, global params $\theta_g$
Return: individual task params $\theta_T$ after $T$ update steps

1: $\implies$ inner loop: learn local params for the local task via the loss of task support set
2: set $\theta_0 = \theta_g$
3: for step $t$ in num steps do
4: compute loss of the local task support set
   $L_{S_e} = \sum_{(x,y) \in S_e} L_{cls}(f(x; \theta_{t-1}), y)$
5: compute gradient and update local params for the local task
   $\theta_t = \theta_{t-1} - \alpha \nabla \theta_{t-1} L_{S_e}$
6: end for

Algorithm : Meta-Curvature’s training procedure, inner-loop

Require: task support set $S_e$, global params $\theta_g$, curvature params $M_f, M_i, M_o$
Return: individual task params $\theta_T$ after $T$ update steps

1: $\implies$ inner loop: learn local params for the local task via the loss of task support set
2: set $\theta_0 = \theta_g$
3: for step $t$ in num steps do
4: compute loss of the local task support set
   $L_{S_e} = \sum_{(x,y) \in S_e} L_{cls}(f(x; \theta_{t-1}), y)$
5: compute gradient and update local params for the local task
   $\theta_t = \theta_{t-1} - \alpha \text{MC}(\nabla \theta_{t-1} L_{S_e}; M_f, M_i, M_o)$
6: end for

- **MAML’s objective**

  $$\min_{\theta_g} \mathbb{E}_{T_e}[L_{Qe}(\theta_e)] = \mathbb{E}_{T_e}[L_{Qe}(\theta_{t-1} - \alpha \nabla \theta_{t-1} L_{S_e})]$$

- **Meta-Curvature further improves MAML by adding second-order information into the gradient**

  $$\min_{\theta_g} \mathbb{E}_{T_e}[L_{Qe}(\theta_e)] = \mathbb{E}_{T_e}[L_{Qe}(\theta_{t-1} - \alpha \text{MC}(\nabla \theta_{t-1} L_{S_e}; M_f, M_i, M_o))]$$
The second-order information is characterized/decomposed by into 3 component matrices, the info is multiplied to the gradient during updates

\[
\text{MC}(\nabla_{\theta_g} L_{S_e(\theta_g)}; M_f, M_i, M_o) = \nabla_{\theta_g} L_{S_e(\theta_g)} \times 3 M_f \times 2 M_i \times 1 M_o
\]

- all component matrices are updated globally similar to global initial params
### FSL Benchmark

<table>
<thead>
<tr>
<th>Model</th>
<th>miniImageNet test accuracy</th>
<th>tieredImageNet test accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1-shot</td>
<td>5-shot</td>
</tr>
<tr>
<td>Matching networks</td>
<td>43.56 ± 0.84%</td>
<td>55.31 ± 0.73%</td>
</tr>
<tr>
<td>(Vinyals et al. 2016)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meta-learner LSTM</td>
<td>43.44 ± 0.77%</td>
<td>60.60 ± 0.71%</td>
</tr>
<tr>
<td>(Ravi &amp; Larochelle 2017)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAML</td>
<td>48.70 ± 1.84%</td>
<td>63.11 ± 0.92%</td>
</tr>
<tr>
<td>(Finn et al. 2017)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LLAMA</td>
<td>49.40 ± 1.83%</td>
<td>-</td>
</tr>
<tr>
<td>(Grant et al. 2018)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>REPTILE</td>
<td>49.97 ± 0.32%</td>
<td>65.99 ± 0.58%</td>
</tr>
<tr>
<td>(Nichol &amp; Schulman 2018)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PLATIPUS (Finn et al. 2018)</td>
<td>50.13 ± 1.86%</td>
<td>-</td>
</tr>
<tr>
<td>Meta-SGD (our features)</td>
<td>54.24 ± 0.03%</td>
<td>70.86 ± 0.04%</td>
</tr>
<tr>
<td>SNAIL</td>
<td>55.71 ± 0.99%</td>
<td>68.88 ± 0.92%</td>
</tr>
<tr>
<td>(Mishra et al. 2018)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Gidaris &amp; Komodakis 2018)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Bauer et al. 2017)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Munkhdalai et al. 2017)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEML+Meta-SGD</td>
<td>56.20 ± 0.86%</td>
<td>73.00 ± 0.64%</td>
</tr>
<tr>
<td>(Zhou et al. 2018)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TADAM</td>
<td>56.30 ± 0.40%</td>
<td>73.90 ± 0.30%</td>
</tr>
<tr>
<td>(Oreshkin et al. 2018)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Qiao et al. 2017)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEO (ours)</td>
<td>61.76 ± 0.08%</td>
<td>77.59 ± 0.12%</td>
</tr>
</tbody>
</table>

Table 1: Test accuracies on miniImageNet and tieredImageNet. For each dataset, the first set of results use convolutional networks, while the second use much deeper residual networks, predominantly in conjunction with pre-training.
Thank you!
