Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks


Orcun Cetintas
Recent Trends in Automated Machine Learning
Technical University of Munich
July 7th, 2021
Meta-Learning

**Problem:** Deep learning is successful with a large amount of data, but often data is scarce

**Solution:** Use data from other tasks to learn how to learn \(\rightarrow\) **Rapid adaptation** on the new task
Few-Shot Learning

Generalizing to a new task using "few" samples and prior knowledge

https://medium.com/sap-machine-learning-research/deep-few-shot-learning-a1caa289f18

Orcun Cetintas | AutoML
One-Shot Video Object Segmentation [1]
Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks

**Paper's aim:** learning a *model initialization* that can achieve *rapid adaptation*

**Paper proposes:** an algorithm for meta-learning
- model-agnostic
- applicable to different learning problems
Model-Agnostic Meta-Learning (MAML) - Overview

Find model parameters that are **sensitive to changes** in the task
Model-Agnostic Meta-Learning (MAML) - Overview

**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
MAML - Overview

**Inner Loop:** Update the model for a task from an initialization

**Outer Loop:** Optimize for the performance of all inner loop models on all tasks

**Intuition:** We want achieve a low loss after only a few updates on a task
MAML - Notation

- model $f_\theta$ with parameters $\theta$
- distribution over tasks $p(\mathcal{T})$
- sampled task $\mathcal{T}_i$
- task loss $\mathcal{L}_{\mathcal{T}_i}$
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 $\mathcal{T}_i \sim p(\mathcal{T})$
MAML - Inner Loop

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

Inner Loop: Update the model for a task from an initialization

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

Simple gradient update on the sampled task
MAML – Outer Loop

**Algorithm 1** Model-Agnostic Meta-Learning

**Require:** $p(T)$: distribution over tasks

**Require:** $\alpha, \beta$: step size hyperparameters

1: randomly initialize $\theta$
2: while not done do
3: Sample batch of tasks $T_i \sim p(T)$
4: for all $T_i$ do
5: Evaluate $\nabla_{\theta} L_{T_i}(f_\theta)$ with respect to $K$ examples
6: Compute adapted parameters with gradient descent: $\theta'_i = \theta - \alpha \nabla_{\theta} L_{T_i}(f_\theta)$
7: end for
8: Update $\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta'_i})$
9: end while

**Outer Loop:** Optimize for the performance of all inner loop models on all tasks
MAML – Outer Loop

**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**

---

**Meta-objective:**

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

Total loss of all updated models

**Meta-update:**

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

Total loss of all updated models
MAML for Few-Shot Supervised Learning

**Regression:** predict the outputs of a function from only **K datapoints** sampled from that function, after training on many functions with similar statistical properties.

**Classification:** learn to classify an object **only from K examples**, after training on many other types of objects.

**How to use MAML?**
- Simply use the general framework with appropriate **loss functions**!
# MAML for Few-Shot Supervised Learning

## Algorithm 2 MAML for Few-Shot Supervised Learning

**Require:** \( p(T) \): distribution over tasks

**Require:** \( \alpha, \beta \): step size hyperparameters

1. randomly initialize \( \theta \)
2. **while** not done **do**
   3. Sample batch of tasks \( T_i \sim p(T) \)
   4. **for all** \( T_i \) **do**
      5. Sample \( K \) datapoints \( D = \{x^{(j)}, y^{(j)}\} \) from \( T_i \)
      6. Evaluate \( \nabla_\theta \mathcal{L}_{T_i}(f_\theta) \) using \( D \) and \( \mathcal{L}_{T_i} \) in Equation (2) or (3)
      7. Compute adapted parameters with gradient descent: \( \theta'_i = \theta - \alpha \nabla_\theta \mathcal{L}_{T_i}(f_\theta) \)
   8. Sample datapoints \( D'_i = \{x^{(j)}, y^{(j)}\} \) from \( T_i \) for the meta-update
   9. **end for**
10. Update \( \theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} \mathcal{L}_{T_i}(f_{\theta'}_{i}) \) using each \( D'_i \) and \( \mathcal{L}_{T_i} \) in Equation 2 or 3
11. **end while**

### Regression: Mean-squared error (MSE)

\[
\mathcal{L}_{T_i}(f_\phi) = \sum_{x^{(j)}, y^{(j)} \sim T_i} \| f_\phi(x^{(j)}) - y^{(j)} \|_2^2, \tag{2}
\]

### Classification: Cross-entropy loss

\[
\mathcal{L}_{T_i}(f_\phi) = \sum_{x^{(j)}, y^{(j)} \sim T_i} y^{(j)} \log f_\phi(x^{(j)}) + (1 - y^{(j)}) \log(1 - f_\phi(x^{(j)})) \tag{3}
\]
MAML for Reinforcement Learning

Goal: enable an agent to quickly acquire a new task policy using only a small amount of experience

How to use MAML?
• Use policy gradient method for a differentiable framework
• Sample new examples with the new policy
MAML for Reinforcement Learning

**Algorithm 3** MAML for Reinforcement Learning

Require: \( p(T) \): distribution over tasks  
Require: \( \alpha, \beta \): step size hyperparameters  
1: randomly initialize \( \theta \)  
2: while not done do  
3: Sample batch of tasks \( T_i \sim p(T) \)  
4: for all \( T_i \) do  
5: Sample \( K \) trajectories \( \mathcal{D} = \{(x_1, a_1, \ldots, x_H)\} \) using \( f_\theta \) in \( T_i \)  
6: Evaluate \( \nabla_\theta \mathcal{L}_{T_i}(f_\theta) \) using \( \mathcal{D} \) and \( \mathcal{L}_{T_i} \) in Equation 4  
7: Compute adapted parameters with gradient descent:  
   \[ \theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{T_i}(f_\theta) \]  
8: Sample trajectories \( \mathcal{D}_i' = \{(x_1, a_1, \ldots, x_H)\} \) using \( f_{\theta_i'} \) in \( T_i \)  
9: end for  
10: Update \( \theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} \mathcal{L}_{T_i}(f_{\theta_i'}) \) using each \( \mathcal{D}_i' \) and \( \mathcal{L}_{T_i} \) in Equation 4  
11: end while

**Loss**: Negative expected reward

\[
\mathcal{L}_{T_i}(f_\phi) = -\mathbb{E}_{x_t, a_t \sim f_\phi, q_{T_i}} \left[ \sum_{t=1}^{H} R_i(x_t, a_t) \right]. \quad (4)
\]
MAML – Task Overfitting and Memorization

**Task overfitting:** Model aligns too closely to a task and fails to generalize

**Memorization problem:** Meta-learner *memorizes the meta-training tasks* rather than *learning to adapt*

**Example:** Instead of *learning to classify cats*, we want to *learn to rapidly adapting to classify cats*

**Solution:** Per-task random assignment of image classes to N-way classification labels
MAML – Task Overfitting and Memorization

**Task overfitting:** Model aligns too closely to a task and fails to generalize

**Memorization problem:** Meta-learner **memorizes the meta-training tasks** rather than **learning to adapt**

**Example:** Instead of *learning to classify cats*, we want to *learn to rapidly adapting to classify cats*

**Solution:** Per-task random assignment of image classes to N-way classification labels
Experiments – Main Questions

- Can MAML enable **fast learning** of new tasks?
- Can MAML be used for meta-learning in **multiple different domains**?
- Can MAML models **continue to improve** with additional gradient updates?
Experiments - Regression

**Task:** Regressing to a sine wave (varying amplitude and phase) given K data points

**MAML:** Meta-training on all tasks with MAML + fine-tuning on K data points

**Baseline:** Pretraining on all tasks with SGD + fine-tuning on K data points
Experiments - Regression

- **k-shot regression, k=10**
  - MAML (ours)
  - pretrained, step=0.02
  - oracle

- **MAML, K=5**
  - pre-update
  - 1 grad step
  - 10 grad steps
  - ground truth

- **MAML, K=10**
  - pre-update
  - 1 grad step
  - 10 grad steps

- **pretrained, K=5, step size=0.01**
  - used for grad
  - pre-update
  - 1 grad step
  - 10 grad steps

- **pretrained, K=10, step size=0.02**
  - used for grad
  - pre-update
  - 1 grad step
  - 10 grad steps

Orcun Cetintas | AutoML
Experiments - Classification

Task: Few shot classification of N unseen classes with only K instances

- Handwritten character classification on Omniglot
  - 20 instances of 1623 chars from 50 alphabets

- Image classification on MiniImagenet
  - 64 train, 24 val, 12 test classes
# Experiments - Classification

<table>
<thead>
<tr>
<th>Dataset</th>
<th>5-way Accuracy</th>
<th>20-way Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1-shot</td>
<td>5-shot</td>
</tr>
<tr>
<td>Omniglot (Lake et al., 2011)</td>
<td>82.8%</td>
<td>94.9%</td>
</tr>
<tr>
<td>MANN, no conv (Santoro et al., 2016)</td>
<td>89.7 ± 1.1%</td>
<td>97.5 ± 0.6%</td>
</tr>
<tr>
<td>Siamese nets (Koch, 2015)</td>
<td>97.3%</td>
<td>98.4%</td>
</tr>
<tr>
<td>matching nets (Vinyals et al., 2016)</td>
<td>98.1%</td>
<td>98.9%</td>
</tr>
<tr>
<td>neural statistician (Edwards &amp; Storkey, 2017)</td>
<td>98.1%</td>
<td>99.5%</td>
</tr>
<tr>
<td>memory mod. (Kaiser et al., 2017)</td>
<td>98.4%</td>
<td>99.6%</td>
</tr>
<tr>
<td>MAML (ours)</td>
<td><strong>98.7 ± 0.4%</strong></td>
<td><strong>99.9 ± 0.1%</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>5-way Accuracy</th>
<th>20-way Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1-shot</td>
<td>5-shot</td>
</tr>
<tr>
<td>MiniImagenet (Ravi &amp; Larochelle, 2017)</td>
<td>28.86 ± 0.54%</td>
<td>49.79 ± 0.79%</td>
</tr>
<tr>
<td>fine-tuning baseline</td>
<td>41.08 ± 0.70%</td>
<td>51.04 ± 0.65%</td>
</tr>
<tr>
<td>nearest neighbor baseline</td>
<td>43.56 ± 0.84%</td>
<td>55.31 ± 0.73%</td>
</tr>
<tr>
<td>matching nets (Vinyals et al., 2016)</td>
<td>43.44 ± 0.77%</td>
<td>60.60 ± 0.71%</td>
</tr>
<tr>
<td>meta-learner LSTM (Ravi &amp; Larochelle, 2017)</td>
<td><strong>48.07 ± 1.75%</strong></td>
<td><strong>63.15 ± 0.91%</strong></td>
</tr>
<tr>
<td>MAML, first order approx. (ours)</td>
<td><strong>48.70 ± 1.84%</strong></td>
<td><strong>63.11 ± 0.92%</strong></td>
</tr>
<tr>
<td>MAML (ours)</td>
<td><strong>48.70 ± 1.84%</strong></td>
<td><strong>63.11 ± 0.92%</strong></td>
</tr>
</tbody>
</table>

Orcun Cetintas | AutoML
# Experiments - Classification

<table>
<thead>
<tr>
<th>Omniglot (Lake et al., 2011)</th>
<th>5-way Accuracy</th>
<th>20-way Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1-shot</td>
<td>5-shot</td>
</tr>
<tr>
<td>MANN, no conv (Santoro et al., 2016)</td>
<td>82.8%</td>
<td>94.9%</td>
</tr>
<tr>
<td>MAML, no conv (ours)</td>
<td>89.7 ± 1.1%</td>
<td>97.5 ± 0.6%</td>
</tr>
<tr>
<td>Siamese nets (Koch, 2015)</td>
<td>97.3%</td>
<td>98.4%</td>
</tr>
<tr>
<td>matching nets (Vinyals et al., 2016)</td>
<td>98.1%</td>
<td>98.9%</td>
</tr>
<tr>
<td>neural statistician (Edwards &amp; Storkey, 2017)</td>
<td>98.1%</td>
<td>99.5%</td>
</tr>
<tr>
<td>memory mod. (Kaiser et al., 2017)</td>
<td>98.4%</td>
<td>99.6%</td>
</tr>
<tr>
<td>MAML (ours)</td>
<td>98.7 ± 0.4%</td>
<td>99.9 ± 0.1%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MinImagenet (Ravi &amp; Larochelle, 2017)</th>
<th>5-way Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1-shot</td>
</tr>
<tr>
<td>fine-tuning baseline</td>
<td>28.86 ± 0.54%</td>
</tr>
<tr>
<td>nearest neighbor baseline</td>
<td>41.08 ± 0.70%</td>
</tr>
<tr>
<td>matching nets (Vinyals et al., 2016)</td>
<td>43.56 ± 0.84%</td>
</tr>
<tr>
<td>meta-learner LSTM (Ravi &amp; Larochelle, 2017)</td>
<td>43.44 ± 0.77%</td>
</tr>
<tr>
<td><strong>MAML, first order approx. (ours)</strong></td>
<td>48.07 ± 1.75%</td>
</tr>
<tr>
<td>MAML (ours)</td>
<td>48.70 ± 1.84%</td>
</tr>
</tbody>
</table>
Experiments - Reinforcement Learning

**Task:** 2D Navigation - move the point agent to a goal

**MAML:** Meta-training a policy on all tasks with MAML + fine-tuning

**Baseline 1 (pretrained):** Pretraining a policy on all tasks + fine-tuning

**Baseline 2 (random):** Training a policy from scratch
Experiments - Reinforcement Learning
Experiments - Reinforcement Learning

**Tasks:** Locomotion in MuJoCo [2] with two robots (cheetah and ant)

- Run in a particular direction
- Run at a particular velocity

**Baseline 1 (pretrained):** Pretraining a policy on all tasks + fine-tuning

**Baseline 2 (random):** Training a policy from scratch
Experiments - Reinforcement Learning
Experiments – Main Questions

- Can MAML enable **fast learning** of new tasks? **YES!**

- Can MAML be used for meta-learning in **multiple different domains**? **YES!**

- Can MAML models **continue to improve** with additional gradient updates? **Yes, but further exploration is required**

- MAML beats the baselines and achieves the SotA
Discussion

**MAML**: a model-agnostic meta-learning method based on gradient descent

**Pros:**
- Model agnostic
- Only requirement is a differentiable task
- No extra parameters
- Step towards general-purpose meta-learning

**Cons:**
- Learning rate's influence
- Computationally costly
- Hard to train
• **Reptile [3]**: Proposes a new algorithm with only first-order derivatives

• **MAML++ [4]**: Stabilizes MAML training and proposes improvements such as learning the learning rate

• **Meta-SGD [5]**: Learns all components of a meta-optimizer (initialization, update direction and learning rate)
References


Discussion

**MAML**: a model-agnostic meta-learning method based on gradient descent

**Pros:**
- Model agnostic
- Only requirement is a differentiable task
- No extra parameters
- Step towards general-purpose meta-learning

**Cons:**
- Learning rate's influence
- Computationally costly
- Hard to train