Meta Learning (Part 1)
Hung-yi Lee
Introduction

• Meta learning = Learn to learn

- Learning task 1
- Learning task 2
- ... (Tasks 2 to 100)
- Learning task 100

I can learn task 101 better because I learn some learning skills

Be a better learner

Life-long: one model for all the tasks
Meta: How to learn a new model

Task 1: speech recognition
Task 2: image recognition
Task 100: text classification
Meta Learning

It is also a function.

Can machine find $F$ from data?

\[ f^* = F(D_{\text{train}}) \]
Meta Learning

Machine Learning ≈ 根據資料找一個函數 f 的能力

Meta Learning

≈ 根據資料找一個找一個函數 f 的函數 F 的能力

Training Data

\[ F(\text{cat, dog, cat, dog}) = f^* \]
Machine Learning is Simple

Step 1: define a set of function

Step 2: goodness of function

Step 3: pick the best function

Function $f$ → Learning algorithm $F$

就好像把大象放進冰箱 ......
Meta Learning

• Define a set of learning algorithm

Different decisions in the red boxes lead to different algorithms. What happens in the red boxes is decided by humans until now.

(limit to gradient descent based approach)
Meta Learning

\[ L(F) = \sum_{n=1}^{N} l^n \]

- Defining the goodness of a function \( F \)

Task 1
- \( f_1 \)
- \( l_1 \)

Task 2
- \( f_2 \)
- \( l_2 \)
Meta Learning

Widely considered in few-shot learning

Training Tasks

Task 1

Task 2

Testing Tasks

Sometimes you need validation tasks
Meta Learning

- Defining the goodness of a function $F$

$$L(F) = \sum_{n=1}^{N} l^n$$

- Find the best function $F^*$

$$F^* = \arg\min_{F} L(F)$$
Omniglot

https://github.com/brendenlake/omniglot

• 1623 characters
• Each has 20 examples
Omniglot – Few-shot Classification

- **N-ways K-shot** classification: In each training and test tasks, there are **N classes**, each has **K examples**.

<table>
<thead>
<tr>
<th>Training set (Support set)</th>
<th>Testing set (Query set)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training set (Support set)</td>
<td>Testing set (Query set)</td>
</tr>
<tr>
<td>Training set (Support set)</td>
<td>Testing set (Query set)</td>
</tr>
<tr>
<td>Training set (Support set)</td>
<td>Testing set (Query set)</td>
</tr>
<tr>
<td>Training set (Support set)</td>
<td>Testing set (Query set)</td>
</tr>
</tbody>
</table>

- **20 ways**
- **1 shot**

Each character represents a class

• Split your characters into training and testing characters
  • Sample N training characters, sample K examples from each sampled characters → one training task
  • Sample N testing characters, sample K examples from each sampled characters → one testing task

Demo of Reptile: https://openai.com/blog/reptile/
Techniques Today

• **MAML**

• **Reptile**
  • Alex Nichol, Joshua Achiam, John Schulman, On First-Order Meta-Learning Algorithms, arXiv, 2018
MAML

Loss Function:

\[ L(\phi) = \sum_{n=1}^{N} l^n(\hat{\theta}^n) \]

\( \hat{\theta}^n \): model learned from task \( n \)

\( l^n(\hat{\theta}^n) \): loss of task \( n \) on the testing set of task \( n \)

Learning Algorithm (Function \( F \))

Only focus on initialization parameter \( \phi \)
**MAML**

Loss Function:

\[ L(\phi) = \sum_{n=1}^{N} l^n(\hat{\theta}^n) \]

\( \hat{\theta}^n \): model learned from task \( n \)

\( \hat{\theta}^n \) depends on \( \phi \)

\( l^n(\hat{\theta}^n) \): loss of task \( n \) on the testing set of task \( n \)

How to minimize \( L(\phi) \)? Gradient Descent

\[ \phi \leftarrow \phi - \eta \nabla_{\phi} L(\phi) \]

---

**Model Pre-training**

Widely used in transfer learning

Loss Function:

\[ L(\phi) = \sum_{n=1}^{N} l^n(\phi) \]
MAML

\[ L(\phi) = \sum_{n=1}^{N} l^n(\hat{\theta}^n) \]

\[ l^1 \text{ (Loss of task 1)} \]

\[ l^2 \text{ (Loss of task 2)} \]

\[ \text{Small } l^1(\hat{\theta}^1) \]

\[ \text{Small } l^2(\hat{\theta}^2) \]

\[ \hat{\theta}^1 \]

\[ \hat{\theta}^2 \]

\[ \phi \]

我们不在意 \( \phi \) 在 training task 上表現如何

我们在意用 \( \phi \) 訓練出來的 \( \hat{\theta}^n \) 表現如何
Model Pre-training

\[ L(\phi) = \sum_{n=1}^{N} l^n(\phi) \]

\[ l^2(\hat{\theta}^2) \]

\[ l^1 \text{ (Loss of task 1)} \]

\[ l^2 \text{ (Loss of task 2)} \]

Finding the best \( \phi \) in all tasks doesn't guarantee good \( \hat{\theta}^n \) after training.
**MAML**

Loss Function:

\[ L(\phi) = \sum_{n=1}^{N} l^n(\hat{\theta}^n) \]

\( \hat{\theta}^n \): model learned from task \( n \)

\( \hat{\theta}^n \) depends on \( \phi \)

Loss Function:

\[ l^n(\hat{\theta}^n) \): loss of task \( n \) on the testing set of task \( n \)

How to minimize \( L(\phi) \)? Gradient Descent

\[ \phi \leftarrow \phi - \eta \nabla_{\phi} L(\phi) \]

Find \( \phi \) achieving good performance **after training**

**Model Pre-training**

Widely used in transfer learning

Find \( \phi \) achieving good performance
MAML

- Fast ... Fast ... Fast ...
- Good to truly train a model with one step. 😊
- When using the algorithm, still update many times.
- Few-shot learning has limited data.

Considering one-step training:

\[
L(\phi) = \sum_{n=1}^{N} l^n(\hat{\theta}^n)
\]

\[
\phi \leftarrow \phi - \eta \nabla_{\phi} L(\phi)
\]

Only focus on initialization parameter \(\phi\)
Toy Example

Each task:

• Given a target sine function $y = a \sin(x + b)$
• Sample K points from the target function
• Use the samples to estimate the target function

Sample $a$ and $b$ to form a task
Toy Example

Model Pre-training

Source of images
https://towardsdatascience.com/paper-repro-deep-metalearning-using-maml-and-reptile-fd1df1cc81b0

MAML
# Omniglot & Mini-ImageNet

<table>
<thead>
<tr>
<th>Model</th>
<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>
<td>1-shot</td>
</tr>
<tr>
<td>MANN, no conv (Santoro et al., 2016)</td>
<td>82.8%</td>
<td>94.9%</td>
<td>–</td>
</tr>
<tr>
<td>MAML, no conv (ours)</td>
<td>89.7 ± 1.1%</td>
<td>97.5 ± 0.6%</td>
<td>–</td>
</tr>
<tr>
<td>Siamese nets (Koch, 2015)</td>
<td>97.3%</td>
<td>98.4%</td>
<td>88.2%</td>
</tr>
<tr>
<td>matching nets (Vinyals et al., 2016)</td>
<td>98.1%</td>
<td>98.9%</td>
<td>93.8%</td>
</tr>
<tr>
<td>neural statistician (Edwards &amp; Storkey, 2017)</td>
<td>98.1%</td>
<td>99.5%</td>
<td>93.2%</td>
</tr>
<tr>
<td>memory mod. (Kaiser et al., 2017)</td>
<td>98.4%</td>
<td>99.6%</td>
<td>95.0%</td>
</tr>
<tr>
<td>MAML (ours)</td>
<td>98.7 ± 0.4%</td>
<td>99.9 ± 0.1%</td>
<td>95.8 ± 0.3%</td>
</tr>
</tbody>
</table>

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

Warning of Math
\[ \phi \leftarrow \phi - \eta \nabla_{\phi} L(\phi) \]

\[ L(\phi) = \sum_{n=1}^{N} l^n(\hat{\theta}^n) \]

\[ \hat{\theta} = \phi - \varepsilon \nabla_{\phi} l(\phi) \]

\[ \nabla_{\phi} L(\phi) = \nabla_{\phi} \sum_{n=1}^{N} l^n(\hat{\theta}^n) = \sum_{n=1}^{N} \nabla_{\phi} l^n(\hat{\theta}^n) \]

\[ \frac{\partial l(\hat{\theta})}{\partial \phi_i} = \sum_{j} \frac{\partial l(\hat{\theta})}{\partial \hat{\theta}_j} \frac{\partial \hat{\theta}_j}{\partial \phi_i} \]

\[ \nabla_{\phi} l(\hat{\theta}) = \begin{bmatrix} \frac{\partial l(\hat{\theta})}{\partial \phi_1} \\ \frac{\partial l(\hat{\theta})}{\partial \phi_2} \\ \vdots \\ \frac{\partial l(\hat{\theta})}{\partial \phi_i} \end{bmatrix} \]

\[ \phi_i \rightarrow \hat{\theta}_1 \rightarrow \hat{\theta}_2 \rightarrow \cdots \rightarrow \hat{\theta}_j \rightarrow \cdots \rightarrow l(\hat{\theta}) \]
\[
\phi \leftarrow \phi - \eta \nabla \phi L(\phi)
\]

\[
L(\phi) = \sum_{n=1}^{N} l^n(\hat{\theta}^n)
\]

\[
\hat{\theta} = \phi - \varepsilon \nabla \phi l(\phi)
\]

\[
\nabla \phi L(\phi) = \nabla \phi \sum_{n=1}^{N} l^n(\hat{\theta}^n) = \sum_{n=1}^{N} \nabla \phi l^n(\hat{\theta}^n)
\]

\[
\frac{\partial l(\hat{\theta})}{\partial \phi_i} = \sum_{j} \frac{\partial l(\hat{\theta})}{\partial \hat{\theta}_j} \frac{\partial \hat{\theta}_j}{\partial \phi_i} \approx \frac{\partial l(\hat{\theta})}{\partial \hat{\theta}_i}
\]

\[
\hat{\theta}_j = \phi_j - \varepsilon \frac{\partial l(\phi)}{\partial \phi_j}
\]

\[
i \neq j: \quad \frac{\partial \hat{\theta}_j}{\partial \phi_i} = -\varepsilon \frac{\partial l(\phi)}{\partial \phi_i \partial \phi_j} \approx 0
\]

\[
i = j: \quad \frac{\partial \hat{\theta}_j}{\partial \phi_i} = 1 - \varepsilon \frac{\partial l(\phi)}{\partial \phi_i \partial \phi_j} \approx 1
\]
\[
\phi \leftarrow \phi - \eta \nabla \phi L(\phi)
\]

\[
L(\phi) = \sum_{n=1}^{N} l^n(\hat{\theta}^n)
\]

\[
\hat{\theta} = \phi - \varepsilon \nabla \phi l(\phi)
\]

\[
\nabla_{\phi} L(\phi) = \nabla \phi \sum_{n=1}^{N} l^n(\hat{\theta}^n) = \sum_{n=1}^{N} \nabla_{\phi} l^n(\hat{\theta}^n)
\]

\[
\frac{\partial l(\hat{\theta})}{\partial \phi_i} = \sum_j \frac{\partial l(\hat{\theta})}{\partial \hat{\theta}_j} \frac{\partial \hat{\theta}_j}{\partial \phi_i} \approx \frac{\partial l(\hat{\theta})}{\partial \hat{\theta}_i}
\]

products, which is supported by standard deep learning libraries such as TensorFlow (Abadi et al., 2016). In our experiments, we also include a comparison to dropping this backward pass and using a first-order approximation, which we discuss in Section 5.2.

\[
\nabla_{\phi} l(\hat{\theta}) = \begin{bmatrix}
\frac{\partial l(\hat{\theta})}{\partial \phi_1} \\
\frac{\partial l(\hat{\theta})}{\partial \phi_2} \\
\vdots \\
\frac{\partial l(\hat{\theta})}{\partial \phi_i} \\
\end{bmatrix} = \begin{bmatrix}
\frac{\partial l(\hat{\theta})}{\partial \hat{\theta}_1} \\
\frac{\partial l(\hat{\theta})}{\partial \hat{\theta}_2} \\
\vdots \\
\frac{\partial l(\hat{\theta})}{\partial \hat{\theta}_i} \\
\end{bmatrix} = \nabla_{\hat{\theta}} l(\hat{\theta})
\]
End of Warning
MAML – Real Implementation

Sample a training task $m$

$\hat{\theta}^m$

$\phi^0$

$\phi^1$

$\phi^2$

Model Pre-training

Sample a training task $n$

$\hat{\theta}^n$

$\phi^0$

$\phi^1$

$\phi^2$

$\hat{\theta}^m$

$\hat{\theta}^n$
Translation

18 training tasks: 18 different languages translating to English
2 validation tasks: 2 different languages translating to English

Ro = Romanian

Fi = Finnish

Techniques Today

• MAML

• Reptile
  • Alex Nichol, Joshua Achiam, John Schulman, On First-Order Meta-Learning Algorithms, arXiv, 2018

https://openai.com/blog/reptile/
You might be thinking “isn’t this the same as training on the expected loss $\mathbb{E}_\tau [L_\tau]$?” Indeed, if the partial minimization consists of a single gradient step, then this algorithm corresponds to minimizing the expected loss:

\[
\phi^0 
\xrightarrow{\text{Sample a training task } m} \phi^1 
\xrightarrow{\text{Sample a training task } n} \phi^2 
\xrightarrow{\text{Sample a training task } m} \hat{\theta}^m 
\xrightarrow{\hat{\theta}^n} \hat{\theta}^n
\]
Reptile

Pre-train: $g_1$

MAML: $g_2$

(simplified)

Reptile: $g_1 + g_2$

Pre-train:

Accuracy vs. Iteration

$g_1$

$\frac{1}{2} \times (g_1 + g_2)$

$g_1 + g_2$

$g_2$

$\frac{1}{3} \times (g_1 + g_2 + g_3)$

$g_1 + g_2 + g_3$

$g_3$

$\frac{1}{4} \times (g_1 + g_2 + g_3 + g_4)$

$g_1 + g_2 + g_3 + g_4$

$g_4$
Training a network (by RL) to determine ...

Architecture & Activation

How to update

Network Structure

Init

Update

θ¹

θ²

Compute Gradient

∇θ

Training Data

MAML, Reptile

Learning Algorithm (Function $F$)

More ...

Video: https://www.youtube.com/watch?v=c10nxBcSH14

Training a network (by RL) to determine ...
Turtles all the way down ...... ?

• We learn the initialization parameter $\phi$ by gradient descent

• What is the initialization parameter $\phi^0$ for initialization parameter $\phi$?
Crazy Idea?

- How about learning algorithm beyond gradient descent?

Just a network

Design network architecture = Design training algorithm?

Learn an even bigger function

Learning Algorithm (Function $F$)

Training Data

Testing Data

cat

dog

$\hat{\theta}$