Knowledge Transfer and Adaptation

Optimization-Based Meta Learning

Antonio Carta

antonio.carta@unipi.it

Outline

- Optimization-based meta-learning
- Model-Agnostic Meta Learner (MAML)
- Implementation, Tips and tricks for MAML
- Some extensions in the literature

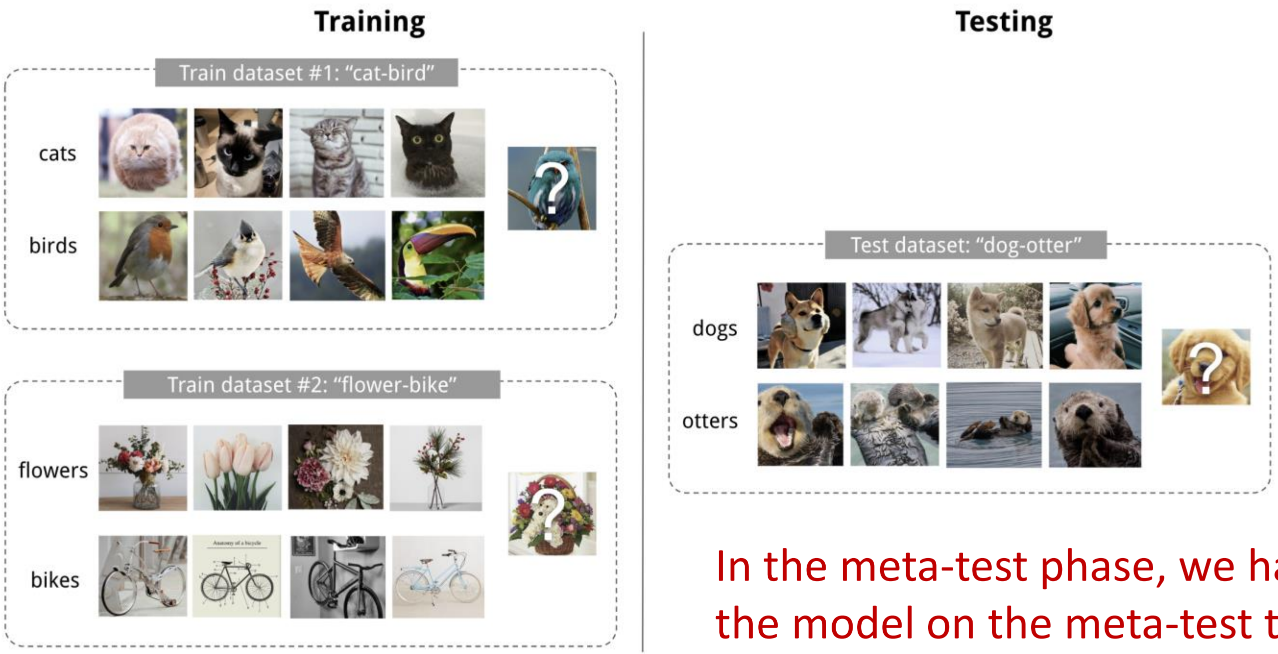
Motivation – Learning to Learn

- **MTL**: Train multiple tasks jointly. Sharing parts of the network encourage positive transfer
  - $\mathcal{A}^{MTL}: \left\{\mathcal{D}_1^{\text{train}}, \dots, \mathcal{D}_N^{\text{train}}\right\} \rightarrow \theta^{MTL}$

- **Meta-learning**: Can we optimize the learning algorithm to solve novel tasks (transfer, low-shot and fast adaptation, hyperparameter search)? *a.k.a. Learning to learn*
  - $\mathcal{A}^{META}: \{\mathcal{D}_1, \dots, \mathcal{D}_N\} \rightarrow \mathcal{A}^*, \mathcal{A}^*: \mathcal{D}_{N+1}^{\text{train}} \rightarrow \theta^{N+1}$

Example: Few-Shot Image Classification

An example of 4-shot 2-class image classification.

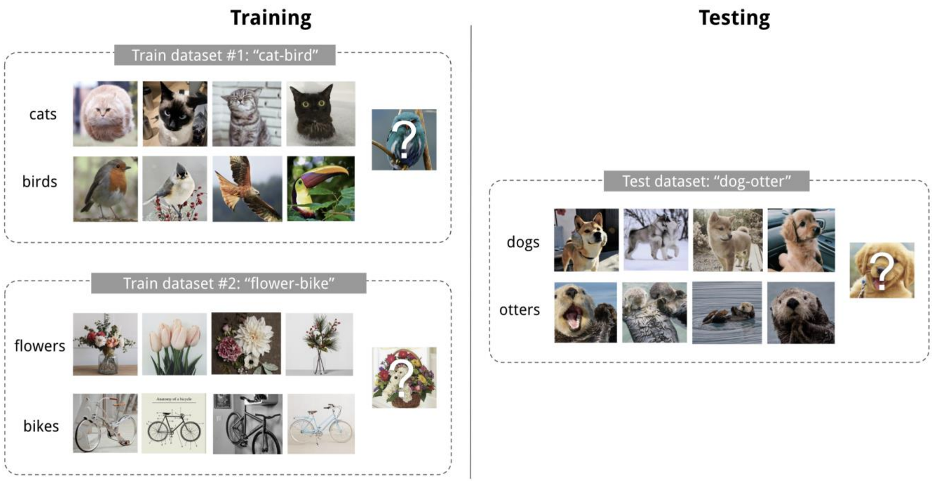


In the meta-test phase, we have to train the model on the meta-test tasks (unseen classes!)

image source: lilianweng.io

Common Terminology

- **support set**: task training set $\mathcal{D}_i^{\text{tr}}$
- **query set**: task test dataset $\mathcal{D}_i^{\text{test}}$
- **meta-training**: training process over the meta-train tasks
- **meta-test**: learning a new task given its support set



learner and meta-learner

Optimization-based methods meta-learn the base algorithm ($\mathcal{A}^*$) hyperparameters

$$\mathcal{A}^{META}: \{\mathcal{D}_1, \ldots, \mathcal{D}_N\} \rightarrow \mathcal{A}^*, \mathcal{A}^*: \mathcal{D}_{N+1}^{\text{train}} \rightarrow \theta^{N+1}$$

learner: a differentiable model such as a deep CNN

meta-learner: a parameterized learning algorithm

- A learned optimizer
- Learned hyperparameters (learning rate, schedule)
- **A learned initialization (our focus today)**

Model-Agnostic Meta Learning

Meta-Learning a Model's Initialization

- Optimization problem:
  - $\theta^* = \min_{\theta} \mathbb{E}_{p(\mathcal{T})} \left[\mathcal{L}_{\mathcal{T}} \left(U_k \left(\theta, D_{\tau}^{Supp} \right), D_{\tau}^{query} \right) \right]$

- $p(\mathcal{T})$: distribution over tasks
  - Few-shot: $U_{\mathcal{T}}$ trains on the (small) support set
- θ model's initialization
- U_k base learning algorithms
  - Fast adaptation: usually U_k is a small number of SGD steps
  - We use k to denote the number of SGD steps
- θ^*: optimal initialization for the family of tasks $p(\mathcal{T})$

Meta-Learning a Model's Initialization

- Optimization problem:
  - $\theta^* = \min_{\theta} \mathbb{E}_{p(\mathcal{T})} \left[\mathcal{L}_{\mathcal{T}} \left(U_k \left(\theta, D_{\tau}^{Supp} \right), D_{\tau}^{query} \right) \right]$

It's a bilevel optimization problem:

- Inner loop (U_k): optimize on a new task starting from θ with algorithm U_k
- Outer loop: optimize initialization θ to improve generalization over the whole family of tasks

Inner and Outer Objective

- $\theta^* = \min_{\theta} \mathbb{E}_{p(\mathcal{T})} \left[\mathcal{L}_{\mathcal{T}} \left(U_k \left(\theta, D_\tau^{Supp} \right), D_\tau^{query} \right) \right]$

Equivalent formulation with separate inner/outer objectives:

- $\theta^* = \min_{\theta} \mathbb{E}_{p(\mathcal{T})} [\mathcal{L}_{\mathcal{T}} (\tilde{\theta}, D_\tau^{query})]$ (outer objective: evaluate on query set)
  - where $\tilde{\theta} = U_k \left(\theta, D_\tau^{Supp} \right)$ (inner objective: train on support set)

MTL vs Meta Learning Initialization

- How can we learn a solution that is better than the multi-task solution $\theta^{MTL} = \min_{\theta} \sum_{\mathcal{T}} \left[\mathcal{L}_{\mathcal{T}}(\theta, D_{\mathcal{T}})\right]$???

Motivating Example: Sine Regression

- Task: a sine wave. Each task has a different phase and amplitude
- Model predicts the output of the sine wave function

Question: What is the optimal MTL solution? What about the optimal meta-learning one?

Sine Regression Results

Answer:

- the **optimal MTL solution** is the one that outputs 0 everywhere (minimum MSE loss for the average of tasks)
- the **optimal meta-learned** solution is the one that quickly adapts the model output to a different phase and amplitude.

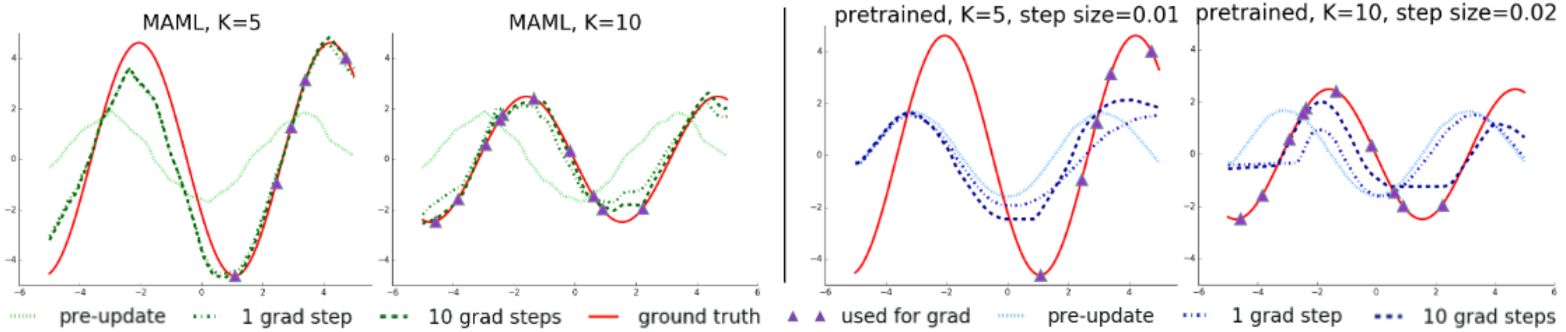


Figure 2. Few-shot adaptation for the simple regression task. Left: Note that MAML is able to estimate parts of the curve where there are no datapoints, indicating that the model has learned about the periodic structure of sine waves. Right: Fine-tuning of a model pretrained on the same distribution of tasks without MAML, with a tuned step size. Due to the often contradictory outputs on the pre-training tasks, this model is unable to recover a suitable representation and fails to extrapolate from the small number of test-time samples.

Figure from: Finn, Chelsea, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks." ICML 2017.

Model-Agnostic Meta Learning

- Optimization problem: $\theta^* = \min_\theta \mathbb{E}_{p(\mathcal{T})}[\mathcal{L}_\mathcal{T}(U_\mathcal{T}(\theta))]$
  - NOTE: we remove the dependency on query/supp sets to simplify the notation, but they are still there

GOAL: we want to optimize θ

- **Meta-train**:
  - Inner loop: optimize solution for each task starting from θ
  - Outer loop: optimize θ
    - The optimizer $U_\mathcal{T}$ needs to be differentiable (SGD methods are ok)
    - Needs to backpropagate on $U_\mathcal{T}$ (e.g. over multiple SGD steps)
- **Meta-test**: use $U_\mathcal{T}$ and θ^* to learn a novel task

MAML – Pseudocode

This is the pseudo code for $U_{\mathcal{T}}$ which is a single SGD step

Algorithm 6 MAML Training episode. $p(\mathcal{T})$ is a distribution over tasks. α, β are the inner and outer learning rates, respectively.

procedure MAMLTRAIN($p(\mathcal{T}), \alpha, \beta$)
    $\theta \leftarrow$ randomly initialize
    while not done **do** ▷ outer loop
        $\mathcal{T}_i \sim p(\mathcal{T})$ ▷ Sample batch of tasks
        $\mathcal{D}_i^{sup}, \mathcal{D}_i^{que} \sim \mathcal{T}_i$ ▷ sample support and query sets
        for i tasks **do** ▷ Inner optimization loop
            $\theta_i' \leftarrow U_k^{\mathcal{T}}(\theta, \mathcal{D}_i^{sup})$ ▷ Inner gradient-based adaptation
        $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{D}_i^{que}} \mathcal{L}_i\left(\theta_i', \mathcal{D}_i^{que}\right)$ ▷ Update initialization

Single-task Solution Found by $U_{\mathcal{T}}$

Notice the different use of suppport and query sets

The gradient backpropagates through θ_i'

FO-MAML – Truncated Gradient

- **FO-MAML** (First Order MAML) is a popular approximation which computes the truncated gradient
- $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ is considered constant during the outer gradient computation (line 10)

Algorithm 2 MAML for Few-Shot Supervised Learning

Require: $p(\mathcal{T})$: distribution over tasks
Require: α, β: step size hyperparameters
1: randomly initialize θ
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 datapoints $\mathcal{D} = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$
6: Evaluate $\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$ using $\mathcal{D}$ and $\mathcal{L}_{\mathcal{T}_i}$ in Equation (2) or (3)
7: Compute adapted parameters with gradient descent: $\theta'_i = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$
8: Sample datapoints $\mathcal{D}'_i = \{\mathbf{x}^{(j)}, \mathbf{y}^{(j)}\}$ from $\mathcal{T}_i$ for the meta-update
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 2 or 3
11: **end while**

pseudocode: Finn, Chelsea, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks." ICML 2017.

PyTorch Functional API

Functional API

- To implement MAML, we will need to backpropagate through the optimizer steps
  - The optimizer needs to be differentiable
  - We need a computational graph of optimizer computation
  - We need the framework to be able to compute second-order derivatives

- `torch.func` is the pytorch functional API (needs torch >= 2.0)
- We will use pytorch but the API are similar across frameworks (e.g. JAX)

https://pytorch.org/docs/stable/func.whirlwind_tour.html

Notebook

- See notebook `demo torch_func.ipynb` in the repository
- Also check the pytorch docs: torch.func Whirlwind Tour — PyTorch 2.2 documentation

Stateful vs functional API

```python
W = torch.randn(1, 2, requires_grad=True)
x = torch.randn(2, requires_grad=True)

# stateful API
W.grad = None  # reset gradient (optimizer.zero_grad)
l = ((W ** 2)@x).sum()
l.backward()
print("stateful API: ", W.grad.tolist())

# Functional API
foo = lambda W: ((W ** 2)@x).sum()
gw = grad(foo)(W)
print("functional API: ", gw.tolist())
```

Back to MAML…

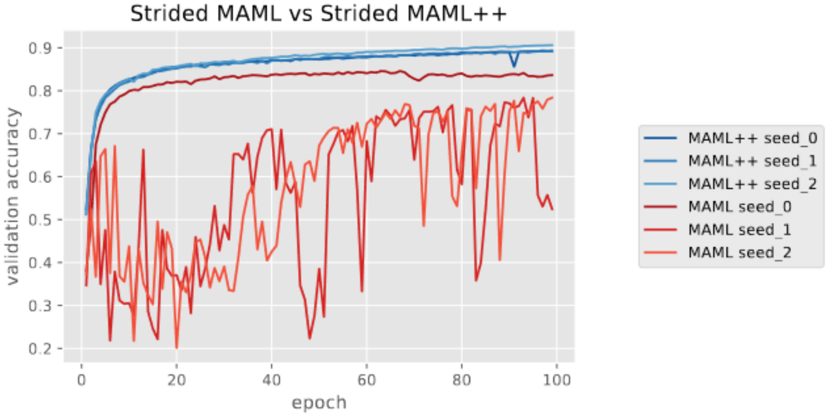
Batch Normalization

- **Batch Normalization** rescales the activations:
  - $y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$
- **Training mode** – statistics computed on the **current mini-batch**:
  - $E\left[x^{(k)}\right] = E_B\left[\mu_B^{(k)}\right]$, and $\mathrm{Var}\left[x^{(k)}\right] = \frac{m}{m-1} E_B\left[\left(\sigma_B^{(k)}\right)^2\right]$
  - Statistics depend on the samples in each mini-batch
  - In MAML, each mini-batch contains samples from a single task
- **Inference mode** – uses EMA of μ, σ seen during training:
  - $y^{(k)} = BN^{inf}_{\gamma^{(k)},\beta^{(k)}}\left(x^{(k)}\right) = \gamma^{(k)} \frac{x^{(k)} - E\left[x^{(k)}\right]}{\sqrt{\mathrm{Var}\left[x^{(k)}\right] + \epsilon}} + \beta^{(k)}$
  - In MAML (and in general in MTL), these statistics will be averaged across ALL THE TASKS, obtaining different values from the ones used during training (single task)!
- **Batch Normalization is often problematic** in multi-task/meta-learning settings due to its dependence on i.i.d. sampling of the data
- MAML implementations often disable the training mode

Training Instability

The bilevel optimization of MAML can be unstable. Some tricks in the literature:

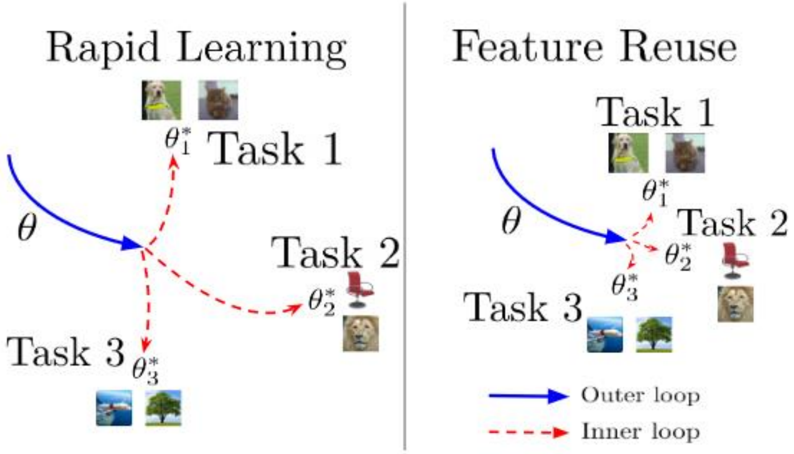
- More stable architectures to avoid exploding gradients (e.g. skip connections)
- Minimize a loss at every step (instead of using only the final model)
- Start with first-order approximation to speed up early training. Then, switch to full gradient
- Per-step and per-layer learning rates
- Cosine annealing to schedule outer learning rate
- Per-step batch normalization statistics



source: Antoniou, Antreas, Harrison Edwards, and Amos Storkey. "How to train your MAML." arXiv preprint arXiv:1810.09502 (2018).

Partial Adaptation

- Two possible goals:
  - **Rapid Learning**: condition the network for fast adaptation
  - **Feature Reuse**: Learn features that generalize across task and get rapid learning as a consequence
- Sometimes, you don't need to adapt the whole network
- **Almost No Inner Loop (ANIL)**: adapts only the final layer in the inner loop



Raghu, Aniruddh, et al. "Rapid learning or feature reuse? towards understanding the effectiveness of maml." arXiv preprint arXiv:1909.09157 (2019).

Extensions and Related Work

- Many works extend MAML by learning the learning rates and other hyperparameters
- It has also been applied in incremental learning settings
- Alternatives without second-order gradients have been proposed (REPTILE)

Take-Home Messages

- **Optimization-Based Meta-learning** is a general framework that works in a large number of settings
- Elegant mathematical formulation that directly optimize the goal of fast adaptation
- Tricky to train and more expensive than normal SGD due to backprop through optimizer

References

- MAML paper: Finn, Chelsea, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks." ICML 2017.
- Reference implementation in the repository

Next Lecture

- Intro to Continual Learning
- The problem of Catastrophic Forgetting
- Notebook