Meta-Learning - Learning to learn

[MAML]

Use MAML to train a Model (Mfine-tune). goal: classfy un-labeled pics.

class: $P_1 \sim P_5$ (5 labeled sample to train, 15 un-labeled sample to test)

Training data: besides P_1-P_5 labeled samples, have 10-class (each class has 30 labeled samples) to help train Mmeta.

Brief Algo: 1° use C_1-C_{10} to train Mmeta, 2° use P_1-P_5 to fine-tune 3° Final model: Mfine-tune.

$C_1 \sim C_{10}$: meta-train classes, 30 samples, ie Dmeta-train, to train Mmeta

$P_1 \sim P_5$: meta-test classes, 100 samples, ie Dmeta-test to train and test Mfine-tune.

Based on 5-way, 5-shot:

In Mmeta-training: From $C_1 \sim C_{10}$, randomly pick 5 classes; each class, randomly picks 20 labeled samples to create one Task T. From this, the 5 labeled samples is T's support set, other 15 samples are T's query set. This "Task T" is "one data", repeatedly pick many "Task T" to have a batch to SGD. Meanwhile, Mfine-tune training process is the same as Mmeta.

Algorithm 2 Model-Agnostic Meta-learning (Pre-training, to get Mmeta)

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

Require: α, β: step size (LR) hyperparameters

1: randomly init θ

2: while not done do

3: Sample batch of task $T_i \sim p(T)$

1° 4: for all T_i do

5: Evaluate $\nabla_\theta L_{T_i}(f_\theta)$ w.r.t. K samples

Gradient by 6: Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta L_{T_i}(f_\theta)$

7: end for

Gradient 2° 8: Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(T)} L_{T_i}(f_{\theta_i'})$

9: end while

4~7: per task → per temporary model

8: original model

main idea: It trains a model so it can adapt a new tasks with very little data. During training, you don't just shuffle examples — your sample tasks and practice test adaption on each one so that at test time, the model can update quickly on a handful of examples ("few-shot")

Different from related ideas:

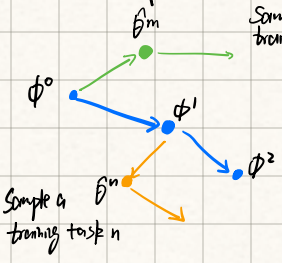
- Transfer Learning: pretrain on big data then fine-tune
  Meta-learning trains for rapid fine-tuning by simulating few-shot episodes, often needs far few examples.
- Multitask Learning: trains one model on many tasks jointly
  Meta-learning expecially optimizes the adaption step.
- AutoML/hyperparam Search: picks models/hyperparams
  Meta-learning learns an init, metric space or optimizer that adapts fast.

↳ What happens in meta-train vs. fine-tune

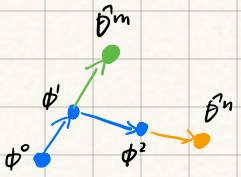
- Each task = (support set, query set)
  Support is the tiny "adaption" set; Query is the "evaluation" set.

- Meta-train (2-stage update)
  1° inner (task-specific) step on support:
  Copy the model, update on the task's support to get adapted params θ'.
  2° Outer (meta) step on query
  Evaluate on the task's query; sum query losses across tasks in the batch and backpropagate to update the original θ
  Intuition: practice fast adaption on support, then reviewed initializations that generalize to query - discouraging overfitting and encouraging across-task generalization.
- Fine-tune (1-stage)
  Use the support to directly update the original model for the new task. The query acts as a held-out test set only; no second (meta) update there.
- Why not just "sum losses over tasks and descend on θ" (plain transfer)?
  That minimizes only an across-task empirical risk w/o modeling task-specific adaption.
  MAML explicity optimizes for rapid per-task adaption, which typically yields better few-shot performance than plain pretrain and finetune.

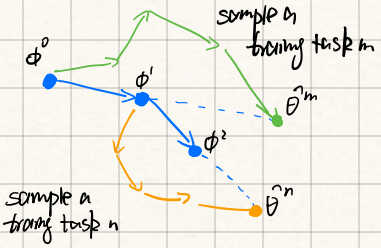
Reptile — On First-Order Meta-Learning Algorithms



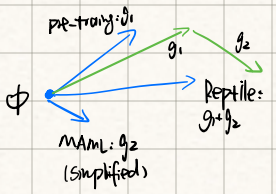
MAML



Model Pre-Training



Reptile



Relation among those 3

§ TL;DR:
- Significant difference between MAML and Reptile
  MAML meta-updates by a query loss and backprop through inner steps (2nd order)
  Reptile moves θ to task-adapted weights (1st order)
- Why Reptile "better (= simpler + cheaper + similar acc)":
  It update $\theta \leftarrow \theta + \beta(U_k(\theta) - \theta)$ approximates the MAML objective's meta-gradient using only 1st order information and gradient alignment across batches within each task.

§ Key difference:

MAML:
- Use a support → inner updates to get θ', then evaluates θ' on a query set and backprogates through the inner updates to update the initialization θ.
- Full MAML needs 2nd-order derivatives (Hessian) since $\nabla_\theta L(\text{query}; \theta')$ depends on θ through $\theta' = U_k(\theta)$

Reptile:
- For a sampled task, run k ordinary SGD steps on that task: $\theta' = U_k(\theta)$
- No query loss, no backprop through the inner loop. Simply move the initialization to the adapted weights: $\theta \leftarrow \theta + \beta(\theta' - \theta)$
- Entirely 1st-order (no 2nd-order) and the "support/query" split is optional. (it emerges if each inner step uses a fresh mini-batch from the task).