Bilevel Optimization in MAML: Summary Notes

Transcribed from notes by Adithi Samudrala

November 19, 2025

---

1 Core Concepts

- `meta-learner`: Manages the meta-parameters, denoted by θ.
- `fast-learner`: Copies the meta-parameters θ at the start of each task and adapts them. The adapted parameters are denoted as θ'. The adaptation rule is:

$$\theta' = \theta - \alpha \frac{d\mathcal{L}_{\text{train}}}{d\theta}$$

- `meta-steps`: The number of times the entire algorithm runs; this is the number of outer loop updates. In our case, this is 2000.
- **Outer Loop Process:**
  1. Sample a batch of tasks from a distribution $p(\mathcal{T})$.
  2. For each task, compute the validation loss on the initial meta-parameters θ. This is the **pre-update loss**.
  3. Perform k inner loop steps on each task's training data ($D_{\mathcal{T}}^{\text{train}}$) to produce the adapted parameters, θ'.
  4. Compute the validation loss on the adapted parameters θ'. This is the **post-update loss**.

2 The MAML Process: A Step-by-Step Breakdown

For each method, the process involves an inner loop for task adaptation and an outer loop for meta-updating.

2.1 Inner Loop (Task Adaptation)

The goal is to adapt the initial parameters θ to a specific task $\mathcal{T}$. This is done over k steps of gradient descent on the task's training (support) set.

1. Start with the initial parameters: $\theta^{(0)} = \theta$.
2. For $j = 0, \ldots, k-1$:
$$\theta^{(j+1)} = \theta^{(j)} - \alpha \nabla_\theta \mathcal{L}_{\text{train}}(\theta^{(j)})$$
3. The final adapted parameters are $\theta' = \theta^{(k)}$.

After adaptation, we evaluate the performance on the validation (query) set to get the validation loss: $\mathcal{L}_{\text{val}}(\theta')$. This loss tells us how good the initial parameters θ were at enabling rapid adaptation.

2.2 Outer Loop (Meta-Update)

The goal is to update the meta-parameters θ based on the performance of the adapted parameters.

1. **For each task, compute the meta-gradient:** $\nabla_\theta \mathcal{L}_{\text{val}}(\theta')$. This is the core of MAML and is calculated differently by each solver.
2. **Average the meta-gradients** across the batch of tasks to get the final meta-gradient, g_{meta}:
$$g_{\text{meta}} = \frac{1}{B} \sum_{\mathcal{T}_i} \nabla_\theta \mathcal{L}_{\text{val}}(\theta_i')$$
3. **Update the meta-parameters** using this averaged gradient and a meta-learning rate, β:
$$\theta \leftarrow \theta - \beta \cdot g_{\text{meta}}$$

This entire process constitutes **one meta-iteration**.

3 Meta-Gradient Calculation by Solver

3.1 Explicit MAML (Full MAML)

The meta-gradient is found by applying the chain rule through all k inner-loop steps. For a single step, the dependency is $\theta'(\theta) = \theta - \alpha \nabla_\theta \mathcal{L}_{\text{train}}(\theta)$. The gradient involves the Hessian:

$$\nabla_\theta \mathcal{L}_{\text{val}}(\theta') = \nabla_{\theta'} \mathcal{L}_{\text{val}}(\theta') \cdot \left(I - \alpha \nabla_\theta^2 \mathcal{L}_{\text{train}}(\theta)\right) \tag{1}$$

3.2 First-Order MAML (FOMAML)

This is a first-order approximation that drops the computationally expensive Hessian term:

$$\nabla_\theta \mathcal{L}_{\text{val}}(\theta') \approx \nabla_{\theta'} \mathcal{L}_{\text{val}}(\theta') \tag{2}$$

3.3 Implicit MAML (iMAML)

This method treats the adapted parameters θ^* as the solution to a regularized optimization problem:

$$\theta^* = \arg\min_{\theta'} \left(\mathcal{L}_{\text{train}}(\theta') + \frac{\lambda}{2} ||\theta' - \theta||^2 \right) \tag{3}$$

Differentiating the optimality condition of this problem w.r.t. θ yields a linear system for the meta-gradient:

$$\nabla_\theta \mathcal{L}_{\text{val}}(\theta^*) = \lambda (H_{\text{train}} + \lambda I)^{-1} \nabla_{\theta'} \mathcal{L}_{\text{val}}(\theta^*) \tag{4}$$

This is solved efficiently using the **Conjugate Gradient (CG)** algorithm, which only requires **Hessian-vector products**.

3.4 Reptile

Reptile avoids calculating a true gradient. After k inner loops to get the adapted parameters ϕ, the outer update is:

$$\theta \leftarrow \theta + \epsilon(\phi - \theta) \tag{5}$$

3.5 Neumann Series Approximation

This approximates the inverse Hessian term from Explicit MAML using a geometric series (the Neumann series):

$$(I - \alpha H)^{-1} \approx I + \alpha H + \alpha^2 H^2 + \dots \tag{6}$$

The full meta-gradient is then approximated as:

$$\nabla_\theta \mathcal{L}_{\text{val}}(\phi) \approx g_{\text{val}} + \alpha H g_{\text{val}} + \alpha^2 H^2 g_{\text{val}} + \dots \tag{7}$$

This is computed iteratively and relies on **Hessian-vector products**.

3.6 Penalty Method

This method reframes the bilevel problem into a single-level meta-objective by adding a penalty term for not satisfying the inner-loop optimality condition:

$$\mathcal{L}_{\text{meta}}(\theta) = \mathcal{L}_{\text{val}}(\phi) + \lambda ||\nabla_\phi \mathcal{L}_{\text{train}}(\phi)||^2 \tag{8}$$

The meta-gradient is then the gradient of this entire combined objective. This calculation also uses **Hessian-vector products**.