Lecture 23: Meta-Learning and Model Adaptation

Contents

1 Approaches to Model Adaptation

Before entering the domain of Meta-Learning, it is crucial to understand the standard paradigms for adapting a pre-trained model to a new task.

1.1 Standard Adaptation Strategies

When presented with a new task (e.g., a new classification problem), there is a hierarchy of adaptation strategies:

1. **Prompting:** If the model is promptable (like an LLM), simply provide instructions or examples.
2. **Linear Probing:** Use the pre-trained model as a fixed feature extractor (embedder).
   - Train a new task-specific head (e.g., linear classifier or regression layer) on top of the fixed embeddings.
   - **Advantage:** Easy to do; requires no data beyond the specific task data.
   - **Disadvantage:** Might not perform as well as full fine-tuning if the embeddings are not linearly separable for the specific task.
3. **Full Fine-Tuning:** Train all parameters (Head ϕ_w and Model Θ).
   - Uses the pre-trained model as an initial condition.
   - **Advantage:** Typically yields higher performance.
   - **Disadvantage:** Computationally expensive; high risk of overfitting (forgetting).
4. **Parameter-Efficient Fine-Tuning (PEFT):** Approaches like LoRA (Low-Rank Adaptation) or Soft-prompting combined with a new head.

1.2 Practical Tips for Head Initialization

When performing a full fine-tune for a new task, the initialization of the new task-specific head is critical.

- **Bad Practice:** Do not initialize the head randomly and immediately begin full fine-tuning. Random gradients from the untrained head can degrade the pre-trained features in the early steps (sending "noise" back into the model).
- **Better Practice:** Initialize the head weights to **zero**. This ensures that initially, no gradients flow back to distort the backbone model.
- **Best Practice:** Use a subset of data to train *only* the head first (Linear Probing), then unfreeze the rest of the model for full fine-tuning.

2 Meta-Learning Fundamentals

Definition: Meta-learning is the process of making a model better at being fine-tuned for tasks. It is often described as "learning to learn."

2.1 Problem Setup

To perform meta-learning, we require:

1. **A distribution of tasks:** A collection of tasks from a specific family.
2. **Data Split per Task:** For each task i, we have:
   - Training set D_i (Support Set).
   - Test/Held-out set $\tilde{D}_i$ (Query Set).
   - Loss functions $\mathcal{L}_{train}$ and $\mathcal{L}_{test}$.
3. **Goal:** Find an initialization Θ_0 such that performing fine-tuning on D_i leads to low loss on $\tilde{D}_i$ across all tasks.

2.2 Key Insight: The "RNN" Perspective

The process of fine-tuning via Stochastic Gradient Descent (SGD) can be viewed as the unrolling of a Recurrent Neural Network (RNN).

- The "hidden state" of this RNN is the model parameters Θ.
- The "input" at each step is the task data D_i.
- Since RNNs are differentiable, the entire fine-tuning process is differentiable. We can back-propagate through the optimization steps.

3 Model-Agnostic Meta-Learning (MAML)

MAML is a general algorithm compatible with any model trained with gradient descent. The core idea is to train the initialization weights.

3.1 The Algorithm

The training process involves two loops: an inner loop (task adaptation) and an outer loop (meta-update).

3.1.1 1. Inner Loop (Adaptation)

For a sampled task i starting at model parameters Θ_0:

1. Compute gradient on support set D_i: $\nabla_\Theta \mathcal{L}(D_i, \Theta_0)$.
2. Update parameters (e.g., one or more gradient steps) to obtain task-specific parameters Θ_i':

$$\Theta_i' = \Theta_0 - \alpha \nabla_\Theta \mathcal{L}(D_i, \Theta_0) \tag{1}$$

   where α is the inner learning rate.

1. 内循环：短期特训（Fast Stage / 快速适应）

目标： 让运动员（模型）用最快的速度学会一个**特定任务**。

元素	解释
运动员	初始参数 θ。这是运动员的**"底子"**。
任务 T	比如"在雪地上滑雪"，或"在泥泞跑道上赛跑"。
训练数据	任务 T 的**少量训练样本**（K 样本）。就像教练只给你 5 分钟热身。
动作	**一步或几步梯度下降**。运动员立即开始特训，调整自己的肌肉记忆。
结果	运动员 θ 变成了 θ_i'。这是**"特训成果"**。

2. 外循环：长期进化（Meta Stage / 慢速优化）

目标： 调整运动员的**"先天底子"（初始 θ），让它能更好地适应**所有可能的任务**。

元素	解释
评委	元学习算法。
评估数据	任务 T 的**少量测试样本**。评委要看你特训后的成果 θ_i' 在实战中表现如何。
动作	**计算元梯度**（梯度穿透）。评委不是直接奖励 θ_i'，而是要追溯：**"如果一开始的底子 θ 更好一点，这次特训的效果 θ_i' 是不是会更好？"**
结果	**初始底子 θ** 被更新成了 θ_{new}。

3.1.2 2. Outer Loop (Meta-Optimization)

Evaluate the adapted parameters Θ_i' on the query set $\tilde{D}_i$:

1. Compute the test loss: $\mathcal{L}_{test}(\tilde{D}_i, \Theta_i')$.
2. The objective is to minimize the sum of these test losses with respect to the *original* initialization Θ_0:
$$\min_{\Theta_0} \sum_{i \sim p(\mathcal{T})} \mathcal{L}_{test}(\tilde{D}_i, \Theta_i') \tag{2}$$
3. Compute meta-gradients:
$$\Theta_0 \leftarrow \Theta_0 - \beta \nabla_{\Theta_0} \sum_{i} \mathcal{L}_{test}(\tilde{D}_i, \Theta_i') \tag{3}$$
where β is the outer learning rate.

3.2 Computational Challenges

Calculating $\nabla_{\Theta_0} \mathcal{L}_{test}(\tilde{D}_i, \Theta_i')$ requires differentiating through the inner loop update Θ_i'. Since Θ_i' itself depends on the gradient of Θ_0, this involves computing **Hessian-vector products** (second-order derivatives). This is computationally expensive and memory-intensive.

4 Reptile: A First-Order Approximation

Reptile is a variant that avoids the explicit calculation of second-order derivatives ("backprop through backprop").

4.1 The Intuition

Instead of explicitly optimizing for the sensitivity of the loss to the initialization, Reptile simply moves the initialization towards the parameters obtained after fine-tuning on a task.

4.2 The Algorithm

1. Sample a task i.
2. Perform k steps of SGD on task i starting from Θ_0 to reach Θ_{final} (Standard training).
3. Update the initialization Θ_0 by interpolating towards Θ_{final}:
$$\Theta_0 \leftarrow \Theta_0 + \beta(\Theta_{final} - \Theta_0) \tag{4}$$
Here, $(\Theta_{final} - \Theta_0)$ serves as a proxy for the meta-gradient.

5 Meta-Learning for Linear Probes (ANIL / MetaOptNet)

Sometimes we do not want to fine-tune the whole model at test time, but rather learn a feature extractor ϕ such that a linear head w can be learned very efficiently (Linear Probing).

5.1 Differentiable Closed-Form Solvers

For regression tasks (and arguably classification via approximations), the optimal linear head w^* has a closed-form solution given the features Φ and labels Y.

5.1.1 Ridge Regression Example

For a task with inputs X and labels Y, let $\Phi(X)$ be the features extracted by the model. The optimal head w^* for Ridge Regression is:

$$w^* = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T Y \tag{5}$$

5.2 Optimization Process

Since the calculation of w^* involves only differentiable matrix operations (matrix multiplication, inversion):

1. Pass support set X through model to get Φ.
2. Compute w^* using the closed-form formula (Eq. 5).
3. Evaluate loss on query set using w^* and query features.
4. Backpropagate the query loss *through* the closed-form calculation of w^* to update the feature extractor parameters Θ.

This allows for calculating gradients without iterative unrolling of SGD, making it highly efficient.

6 Catastrophic Forgetting

Definition: When adapting a model to a new task, it often forgets how to perform previous tasks.

6.1 Conceptual Analogy

Think of machine learning as blowing up a "leaky balloon."

- The data acts as the air pressure inflating the balloon (the model's capabilities).
- Weight decay and loss are leaks.
- If you stop inflating one part (Task A) to inflate another (Task B), the first part deflates. Without the pressure of the original data, the weights drift, and knowledge is lost.

6.2 Practical Solution

The standard solution is **Replay/Mixing**:

- During fine-tuning, mix in a small percentage (e.g., 10%) of the original pre-training data.
- If the pre-training task had distinct heads, keep them active and computing loss during fine-tuning.
- This provides a gradient signal that anchors the shared weights, preventing them from drifting too far from the generalizable representation.