Mid-Evaluation Report: Model-Agnostic Meta-Learning for Wireless Channel Adaptation

Utkarsh Singh - 241117
Department of Electrical Engineering
Indian Institute of Technology Kanpur

***Abstract*—This report summarizes the progress made during the first half of the project, focusing on the transition from classical machine learning optimization to advanced Few-Shot Learning techniques. The work began with an analysis of Stochastic Gradient Descent (SGD) dynamics and classical classifiers (SVM, Decision Trees). Subsequently, rigorous experiments were conducted on the MNIST dataset to evaluate various Transfer Learning strategies, demonstrating the efficacy of feature reuse. Finally, the study progressed to a literature review of Model-Agnostic Meta-Learning (MAML), establishing the theoretical framework for "learning to adapt" as opposed to simple transfer**

***Index Terms*—Meta-Learning,Tranfer Learning, MAML ,SGD, SVM, Few-Shot Learning.**

I. Introduction

A. Project Progression

This report summarizes the technical roadmap followed to achieve this goal, structured around the following learning modules:

- **Foundational Python & OOP:** Developed a modular software architecture using Object-Oriented Programming (OOP) to ensure scalability. Hierarchical class structures were implemented to encapsulate model states, ensuring that inner-loop task updates do not corrupt global meta-parameters.
- **Classical Regression & Optimization:** Implemented Linear and Logistic Regression from scratch (using NumPy) to understand the mechanics of Gradient Descent. Analyzed the impact of feature normalization on convergence speed and the behavior of Stochastic Gradient Descent (SGD) in escaping local minima.
- **Deep Learning Frameworks (PyTorch):** Transitioned to PyTorch to leverage GPU-accelerated Tensors for matrix operations. Utilized Autograd (Automatic Differentiation) to manage dynamic computation graphs required for calculating second-order derivatives (Hessians) in meta-optimization.
- **Transfer Learning (Layer Freezing):** Addressed "data scarcity" by applying Transfer Learning. Implemented Layer Freezing, where initial convolutional layers (feature extractors) are frozen to retain generic signal features, while only the final regression layers are fine-tuned. This strategy mitigates overfitting when adaptation data (pilots) is limited.
- **Model-Agnostic Meta-Learning (MAML):** Learned and Designed the MAML algorithm to optimize the model's *initialization* rather than its final performance, enabling the system to adapt to new wireless environments with minimal gradient steps.

II. Software Architecture

To support the complexity of bi-level optimization loops inherent in Meta-Learning, a robust Object-Oriented Programming (OOP) architecture was established.

A. Hierarchical Abstraction

We implemented a hierarchical class structure to enforce modularity.

- **Encapsulation:** Neural network weights and optimizer states are encapsulated within a `MetaLearner` class to prevent gradient leakage between the inner (task) and outer (meta) loops.
- **Polymorphism:** Derived classes implement specific adaptation strategies (e.g., `MAML` vs. `Reptile`), allowing the simulation engine to switch algorithms dynamically without code refactoring.

III. Mathematical Foundations

Prior to leveraging high-level frameworks, core Machine Learning algorithms were implemented using NumPy to analyze optimization dynamics.

A. Regression and Feature Scaling

A Linear Regression model was implemented using the Mean Squared Error (MSE) cost function:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 \tag{1}$$

The update rule for gradient descent is defined as:

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \tag{2}$$

Insight: Analysis of high-variance datasets highlighted the necessity of Min-Max Normalization. Unscaled features result in elliptical error surfaces where the condition number $\kappa(A)$ is large, causing slow convergence. Normalization transforms this into a spherical geometry ($\kappa \approx 1$), optimizing the gradient descent trajectory.

B. Probabilistic Classification

For classification tasks, we implemented Logistic Regression utilizing the Sigmoid activation $\sigma(z) = \frac{1}{1+e^{-z}}$. The optimization objective is the minimization of the Binary Cross-Entropy (Log-Loss):

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)}))+(1-y^{(i)})\log(1-h_\theta(x^{(i)}))] \tag{3}$$

IV. Optimization Learning Theory

A. Stochastic Gradient Descent (SGD)

We analyzed the convergence properties of Batch GD versus SGD. While Batch GD computes the true gradient $\nabla J(\theta)$, SGD approximates it using a single sample:

$$g_t = \nabla_\theta L(x_{i_t}, y_{i_t}; \theta_t) \tag{4}$$

Since $\mathbb{E}[g_t] = \nabla J(\theta)$, SGD is an unbiased estimator. However, the variance $\text{Var}(g_t)$ introduces noise, allowing the model to escape saddle points—a critical feature for training deep non-convex networks.

B. Support Vector Machines (SVM)

SVMs were analyzed to understand Maximum Margin classification. The optimization problem is formulated as minimizing $||\mathbf{w}||^2$ subject to the constraint:

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \quad \forall i \tag{5}$$

To handle non-linear decision boundaries, we utilized the Radial Basis Function (RBF) kernel:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x}-\mathbf{x}'||^2}{2\sigma^2}\right) \tag{6}$$

This implicitly maps input vectors to an infinite-dimensional feature space, enabling linear separability.

V. Deep Learning Primitives

Transitioning to PyTorch, we focused on computational graph mechanics.

[Image of deep neural network architecture]

- **Tensor Operations:** Utilization of GPU-accelerated tensors for matrix operations.
- **Autograd:** The dynamic computation graph in PyTorch allows for automatic differentiation. This is crucial for MAML, which requires computing the Hessian vector product (HVP) to optimize the initialization parameters.

VI. Transfer Learning Layer Freezing

As a precursor to full Meta-Learning, we explored Transfer Learning techniques to handle data scarcity. This approach assumes that early layers in a Deep Convolutional Neural Network (CNN) learn generic features (e.g., edges, textures) applicable across varied tasks, while deeper layers learn task-specific abstractions.

A. Mechanism of Freezing

In the context of adaptation, we freeze the parameters θ_{base} of the initial layers, rendering them non-trainable during the fine-tuning phase. Only the final fully connected layers (heads) θ_{head} are updated:

$$\nabla_{\theta_{base}}\mathcal{L} = \mathbf{0}, \quad \theta_{head} \leftarrow \theta_{head} - \eta\nabla_{\theta_{head}}\mathcal{L} \tag{7}$$

This significantly reduces the number of trainable parameters, mitigating the risk of overfitting when the target domain has limited data (e.g., few pilot signals).

B. Relevance to Wireless Adaptation

In wireless channel estimation, the "feature extraction" layers can be pre-trained to recognize fundamental signal propagation characteristics (e.g., path loss models, diffraction patterns), which remain consistent across environments. By freezing these layers, the network only needs to adapt the final regression layers to the specific channel impulse response of the new environment, accelerating convergence.

VII. Literature Review: MAML

The core theoretical component of this phase was the analysis of Model-Agnostic Meta-Learning [Finn et al.].

A. Problem Formulation

The goal of MAML is to find a set of initial parameters θ such that one or a few gradient steps on a new task $\mathcal{T}_i$ will produce maximally effective behavior.

B. Algorithmic Derivation

MAML operates via a nested bi-level optimization loop:

1) Inner Loop (Adaptation): For a specific task $\mathcal{T}_i$, we compute task-specific parameters θ_i' using K support samples. This is a standard gradient descent step:

$$\theta_i' = \theta - \alpha\nabla_\theta\mathcal{L}_{\mathcal{T}_i}(f_\theta) \tag{8}$$

2) Outer Loop (Meta-Update): The meta-objective is to minimize the loss of the *adapted* parameters θ_i' on unseen query data. We update θ as follows:

$$\theta \leftarrow \theta - \beta\nabla_\theta \sum_{\mathcal{T}_i\sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}) \tag{9}$$

Expanding the gradient term via the Chain Rule reveals the dependency on the Hessian ∇^2:

$$\nabla_\theta\mathcal{L}(f_{\theta_i'}) = \nabla_{\theta_i'}\mathcal{L}(f_{\theta_i'}) \cdot \left(I - \alpha\nabla_\theta^2\mathcal{L}_{\mathcal{T}_i}(f_\theta)\right) \tag{10}$$

This second-order term ∇_θ^2 (the Hessian) captures the curvature of the loss landscape, guiding the initialization θ to a "flat" region where adaptation is universally fast.

C. First-Order Approximation (FOMAML)

Due to the $O(d^2)$ complexity of computing the Hessian, we investigated First-Order MAML, which approximates the update by assuming $\nabla_\theta^2 \approx 0$. This reduces computational load significantly, making it feasible for IoT edge devices.

VIII. Conclusion and Future Work

This report confirms the completion of the foundational software and theoretical modules. We have successfully bridged the gap between basic optimization theory and the complex bi-level optimization required for Meta-Learning.

Future Roadmap:

1) **Implementation:** Translation of the MAML mathematical derivation into a PyTorch module.
2) **Channel Simulation:** Development of a Ray-Tracing based channel simulator to generate diverse wireless tasks.
3) **Benchmarking:** Quantitative comparison of MAML-based channel estimation against standard pre-training methods.