

Mid-Term Evaluation Report

Model-Agnostic Meta-Learning (MAML) — Foundation and Implementations

Yashita Bhorla
241207

yashita24@iitk.ac.in

Abstract

This report summarizes progress on a meta-learning project centered on Model-Agnostic Meta-Learning (MAML). Over the first four weeks, work moved from Python and scientific computing essentials to classical supervised learning methods (linear regression, logistic regression, KNN), optimization concepts (gradient descent variants, SGD, bias–variance), and key model families (decision trees and SVMs). These foundations were then connected to meta-learning through transfer learning and initialization-focused experiments using a CNN training and fine-tuning pipeline, highlighting how a strong initialization can enable fast adaptation.

A major portion of this report has been dedicated to an introduction of MAML, including its task-distribution formulation, the inner-loop/outer-loop optimization structure, supervised and reinforcement learning instantiations, and practical considerations such as second-order gradients and first-order approximations. The central outcome is a clear understanding that MAML explicitly trains an initialization such that a small number of gradient steps on a small amount of task data yields strong generalization on new tasks.

1. Introduction

Meta-learning aims to train systems that can learn new tasks rapidly, often from very few examples. MAML provides a general gradient-based framework that is *model-agnostic*: it can be applied to any model trained with gradient descent, and it targets the ability to adapt quickly to new tasks drawn from a task distribution.

This mid-term report presents: (i) week-wise progress across assignments, and (ii) a detailed paper-focused section on MAML as the main technical component.

2. Topics Covered So Far

- **Python + Tools:** core Python programming, NumPy vectorization, Pandas DataFrames, Matplotlib plotting.
- **Supervised Learning:** linear regression (MSE), logistic regression (classification, sigmoid, decision boundary), KNN (distance metrics, choosing K).
- **Optimization:** gradient descent, stochastic gradient descent (SGD), mini-batching intuition, learning-rate effects.
- **Generalization:** overfitting/underfitting, bias–variance tradeoff, evaluation mindset.
- **Classical Models:** decision trees (entropy, information gain), SVM (maximum margin, kernel intuition).
- **Deep Learning Pipeline:** CNN training loop, evaluation, fine-tuning procedures.
- **Transfer Learning:** training from scratch vs. pretraining vs. merged-data pretraining; fine-tuning on target tasks.
- **Meta-Learning (Paper Study):** task distribution $p(\mathcal{T})$, support/query split, inner-loop adaptation, outer-loop meta-update, second-order gradients, first-order MAML.

3. Week-wise Progress

3.1. Week 1: Python and Scientific Computing Foundations

3.1.1. Programming Essentials

Week 1 focused on building the programming base required to implement ML algorithms and run experiments reliably. Core Python concepts were revised and practiced, including functions (modularity), control flow (if/else, loops), and standard data structures (lists, dictionaries, tuples). This foundation was essential for writing clean training loops and reusable components later in the project.

3.1.2. Numerical Computing with NumPy

A major emphasis was placed on using NumPy arrays for efficient computation. The key idea was shifting from slow Python loops to **vectorized operations**, enabling faster and cleaner implementations of ML computations such as dot products, broadcasting, and element-wise transformations.

3.1.3. Data Handling with Pandas

Pandas was used to work with tabular datasets via DataFrames. This included loading datasets, basic cleaning/manipulation, and selecting/transforming features. These steps are crucial before model training in almost every ML pipeline.

3.1.4. Visualization with Matplotlib

Matplotlib was used for plotting and visualization (e.g., trends, distributions, basic result plots). Visualization is important for debugging, interpreting results, and communicating findings (e.g., loss curves and accuracy plots).

Outcome: Established a complete workflow for implementing algorithms, handling datasets, and visualizing experiments.

3.2. Week 2: Supervised Learning I — Regression, Classification, and Parameters

3.2.1. What Supervised Learning Means

This week introduced **supervised learning**, where each input has a corresponding label. The goal is to learn a mapping from input features to outputs such that predictions generalize to unseen examples.

3.2.2. Model Parameters: Weights, Bias, and Initialization

A core theme was understanding learnable parameters:

- **Weights** determine how strongly each input feature influences the output.
- **Bias** shifts the prediction baseline and allows flexibility even when inputs are zero.
- **Initialization** refers to the starting values of weights/biases. While simple models may be less sensitive, in deeper models initialization can significantly affect training stability and speed. This becomes especially relevant in meta-learning, where the goal is to learn an initialization that adapts quickly.

3.2.3. Linear Regression (Continuous Prediction)

Linear regression was studied as a first parametric model. The model predicts a continuous value using a linear combination of inputs, and training is framed as minimizing the mean squared error (MSE). This provided a clean introduction to how a loss function drives learning.

3.2.4. Logistic Regression (Binary Classification)

Logistic regression extended the same parameterized linear model to classification. The linear score is mapped through a sigmoid function to obtain probabilities, and training is done using a classification loss (typically cross-entropy). This established the concept of decision boundaries and probabilistic interpretation of outputs.

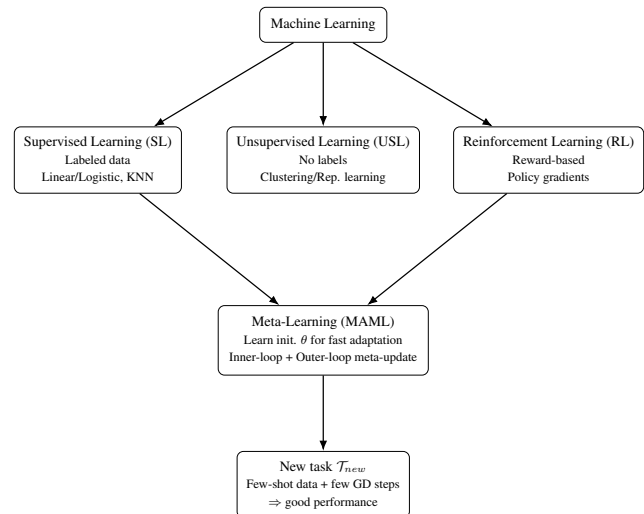


Figure 1. High-level view of SL/USL/RL and where MAML fits in.

3.2.5. K-Nearest Neighbors (KNN) and Choosing K

KNN was explored as a non-parametric baseline that does not learn explicit weights and biases. Instead, it predicts based on proximity to training examples.

- **Role of K :** K controls the smoothness of the classifier. Small K is sensitive to noise (high variance), while large K can oversmooth (high bias).
- **Distance metric:** performance depends on how similarity is measured (e.g., Euclidean distance).
- **Model selection:** choosing K is typically done via validation.

Outcome: Developed a strong understanding of supervised learning, parameterized models (weights/bias), and the importance of hyperparameters such as K in KNN.

3.3. Week 3: Optimization, Generalization, and Classical Models

3.3.1. Optimization: Gradient Descent vs. SGD

This section focused on how models learn by minimizing a loss function:

- **Gradient Descent** uses the full dataset to compute gradients, leading to stable but expensive updates.
- **Stochastic Gradient Descent (SGD)** uses mini-batches, producing noisier but faster updates, and often works better in large-scale learning.
- The role of the **learning rate** was discussed as a tradeoff between speed and stability of convergence.

3.3.2. Generalization: Underfitting/Overfitting and Bias–Variance

Generalization was analyzed through the bias–variance tradeoff:

- **Underfitting** occurs when the model is too simple (high bias).
- **Overfitting** occurs when the model is too complex and memorizes noise (high variance).
- Evaluation strategies were introduced to judge performance on unseen data.

3.3.3. Decision Trees (Interpretability and Greedy Splitting)

Decision trees were studied as interpretable models constructed using greedy splitting rules:

- Splitting criteria based on **entropy** and **information gain**.
- Building a tree top-down and stopping criteria controlling complexity.

3.3.4. Support Vector Machines (Margins and Kernels)

SVMs were studied from the geometric viewpoint:

- The **maximum-margin** principle for robust classification.
- **Kernel intuition** for nonlinear decision boundaries via implicit feature mappings.

3.3.5. Where Unsupervised Learning Fits (Conceptual Placement)

Although the primary implementations so far were supervised, the week also positioned **unsupervised learning** in the ML landscape. Unsupervised learning learns structure without labels (e.g., clustering, representation learning) and is relevant because strong representations or pretraining can improve initializations and downstream adaptation—a theme that connects naturally to meta-learning.

Outcome: Built a solid foundation in optimization (SGD), generalization (bias–variance), and diverse classical model families (trees and SVMs), while situating supervised vs. unsupervised learning conceptually.

3.3.6. Transfer Learning and Initialization Experiments (Bridge to Meta-learning)

- Implemented an end-to-end CNN training pipeline: dataset handling, training loop, and evaluation.
- Explored strategies aligned with the “good initialization enables fast learning” principle:
 1. Training from scratch and fine-tuning on a target task.
 2. Pretraining on one task/domain and fine-tuning on another.
 3. Pretraining on merged/combined data and fine-tuning on the target.

Outcome: Empirically validated that initialization strongly influences adaptation speed and final performance—a direct precursor to MAML’s objective.

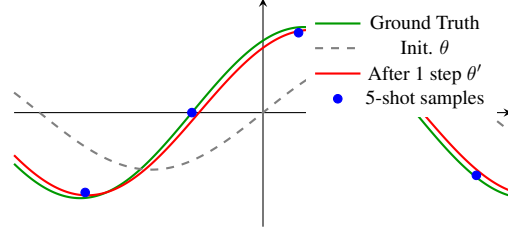


Figure 2. Sinusoid regression (MAML-style): ground-truth task, initialization θ , and adapted parameters θ' after one gradient step using 5-shot samples.

4. Research Paper Study: Model-Agnostic Meta-Learning (MAML)

This section summarizes and analyzes the MAML paper [1], which provides a key insight into what is MAML and how it is useful, and perhaps better than the usual deep learning models.

4.1. Problem Setting: Learning to Adapt Across Tasks

MAML assumes a distribution over tasks, $p(\mathcal{T})$. Each task \mathcal{T}_i defines its own data distribution (supervised learning) or environment (reinforcement learning). The goal is to learn parameters that can be adapted to a new sampled task using only a few gradient steps and a small amount of data (few-shot learning) [1].

4.2. Core Idea: Optimize for Post-Update Performance

Unlike standard training that directly minimizes loss at the current parameters θ , MAML optimizes the model such that *after* a small number of gradient steps, the adapted parameters achieve low loss on the task. Informally, MAML learns an initialization θ that is “easy to fine-tune”.

4.3. Inner Loop: Task-specific Adaptation

For a sampled task \mathcal{T}_i , MAML performs one (or a few) gradient descent steps on task training data:

$$\theta'_i = \theta - \alpha \nabla_{\theta} \mathcal{L}_{\mathcal{T}_i}(f_{\theta}), \quad (1)$$

where α is the inner-loop learning rate, and $\mathcal{L}_{\mathcal{T}_i}$ is the task loss [1].

4.4. Outer Loop: Meta-Optimization Across Tasks

The meta-objective minimizes the loss *after* adaptation, averaged over tasks:

$$\min_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}). \quad (2)$$

The meta-update is then:

$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}), \quad (3)$$

where β is the meta learning rate [1].

4.5. Why MAML Requires Second-Order Gradients (Full Version)

Because θ'_i depends on θ via the inner gradient step in Eq. (1), the meta-gradient in Eq. (3) differentiates through the inner update. This introduces second-order terms (Hessian-vector products) in the full MAML formulation [1]. Practically, this increases computational cost and memory usage compared to standard training.

4.6. Supervised Learning Instantiation (Few-shot Classification/Regression)

In supervised meta-learning, each task \mathcal{T}_i typically has:

- a **support set** (few-shot training examples) used for the inner update, and
 - a **query set** used for the meta-objective and outer update.
- Common task losses include mean squared error (regression) and cross entropy (classification) [1]. The key evaluation criterion becomes: *how good is the model on the query set after adapting on the support set?*

4.7. Reinforcement Learning Instantiation (Meta-RL)

MAML also applies to RL when tasks correspond to different MDPs. The loss is defined via expected return, and gradients are estimated using policy gradient methods. In this case, the inner loop requires on-policy rollouts for each adaptation step, making data collection and compute more expensive than in supervised settings [1].

4.8. First-Order Approximations (Practical Speed-up)

The paper discusses first-order approximations (often called First-Order MAML), which ignore second-order terms in the meta-gradient. This can substantially reduce compute while retaining competitive performance in some settings [1]. This is particularly relevant for implementing scalable versions of MAML.

4.9. Key Takeaways from the Paper Study

- MAML learns a **parameter initialization** that is broadly transferable across tasks.
- The meta-objective explicitly targets **post-adaptation performance** (few-step learning).
- Full MAML requires differentiating through inner-loop updates (second-order gradients).
- First-order variants can reduce computation and may be a strong baseline for implementation.

5. Current Status

- Built strong prerequisites for gradient-based meta-learning: classical ML models, losses, and optimization.
- Implemented and compared training/fine-tuning strategies demonstrating the importance of initialization.
- Completed structured understanding of MAML: formulation, algorithms, supervised/RL variants, and compute tradeoffs.

References

- [1] C. Finn, P. Abbeel, and S. Levine, “Model-agnostic meta-learning for fast adaptation of deep networks,” in *Proceedings of the International Conference on Machine Learning (ICML)*, 2017.