Mid-Term Evaluation Report

Akash Baudh
Indian Institute of Technology Kanpur
Kanpur, India
240077

***Abstract*—This report evaluates the progress of the student across three distinct modules: Python & Data Science Fundamentals, Classical Machine Learning Algorithms, and Foundations of Meta-Learning (Transfer Learning). The student has successfully implemented algorithms from scratch, conducted comparative analyses against standard libraries, and demonstrated theoretical understanding of initialization strategies relevant to Model-Agnostic Meta-Learning (MAML). This document synthesizes the experimental results, theoretical insights, and the progression from supervised learning to meta-learning.**

***Index Terms*—Machine Learning, Meta-Learning, MAML, Transfer Learning, Optimization, Deep Learning.**

I. Introduction

A. Motivation and Problem Context

Deep learning models have demonstrated strong performance across a wide range of applications, including computer vision, natural language processing, and recommendation systems. However, a fundamental limitation of standard deep learning is its heavy reliance on large labeled datasets. In many real-world scenarios—such as medical imaging, robotics, and communication systems—acquiring such data is expensive, slow, or impractical. In contrast, humans can rapidly learn new concepts from only a few examples by leveraging prior knowledge.

This project addresses this gap through the framework of **Few-Shot Learning (FSL)**. The central idea is that learning should not be limited to optimizing performance on a single task, but should instead focus on enabling rapid adaptation to new tasks with minimal data. This perspective leads naturally to **Meta-Learning**, often described as "learning to learn," where a model is trained over a distribution of tasks rather than a single dataset.

From an optimization standpoint, traditional supervised learning seeks a single optimal parameter vector θ^* for a fixed task by minimizing the loss over a large dataset $\mathcal{D}$:

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\mathcal{D}; \theta) \tag{1}$$

Meta-learning, in contrast, aims to learn parameters θ that can be quickly adapted to a new task $\mathcal{T}_i$ using only a few gradient updates on a small support set $\mathcal{D}_{\mathcal{T}_i}^{train}$ to produce updated parameters θ_i':

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta) \tag{2}$$

where α is the adaptation learning rate. This shift in objective—from task-specific performance to adaptability—forms the conceptual foundation of **Model-Agnostic Meta-Learning (MAML)**, which is the primary focus of this work.

B. Project Roadmap

The work completed so far follows a structured progression designed to build both theoretical understanding and practical intuition. The project begins with classical machine learning, where regression and classification algorithms are implemented to understand loss functions, gradients, and the bias–variance tradeoff. This is followed by a transition to deep learning, focusing on neural network training, backpropagation, and convolutional architectures.

To bridge deep learning and meta-learning, transfer learning experiments are conducted to study the role of initialization and its impact on adaptation speed in low-data settings. Finally, these insights motivate the study of MAML, which formalizes rapid adaptation as a bi-level optimization problem. Together, these stages establish a coherent path from conventional supervised learning to meta-learning.

II. Assignment 1: Python and Scientific Computing Foundations

The first assignment focused on establishing the computational foundations required for implementing and analyzing machine learning algorithms. Rather than emphasizing model performance, the objective was to build fluency in Python programming and develop an understanding of how numerical computation, data handling, and visualization support the learning process.

Core Python concepts such as variables, control flow, functions, and basic data structures were used to write modular and reusable code. This foundation was essential for implementing training loops and experimental pipelines in later assignments. A key emphasis was placed on **vectorized computation**, where explicit Python loops were replaced by efficient array-based operations using libraries like NumPy.

Numerical computing was carried out using matrix and vector operations, enabling direct implementation of expressions commonly used in machine learning, such as linear combinations of features:

$$\hat{y} = \mathbf{w}^T \mathbf{x} + b \tag{3}$$

and parameter updates. This shift from scalar computation to linear algebra helped build intuition for how models process data internally.

Structured data manipulation was performed using tabular data abstractions, allowing datasets to be loaded, inspected,

filtered, and transformed before training. Data preprocessing steps, including feature scaling and normalization:

$$x_{norm} = \frac{x - \mu}{\sigma} \tag{4}$$

were applied to ensure numerical stability and consistent optimization behavior. Visualization tools were used to plot data distributions and training trends, reinforcing the importance of visual inspection in debugging and interpreting results.

III. Assignment 2 – Part I: Supervised Learning Fundamentals

Assignment 2 – Part I focused on building a rigorous understanding of supervised learning by implementing core regression and classification algorithms and analyzing their behavior through the lens of optimization and generalization. The objective was not to maximize predictive accuracy, but to develop intuition for how models learn from labeled data and how design choices influence training dynamics.

A. Regression

Regression was studied as the simplest form of supervised learning. **Linear Regression** was implemented to model continuous outputs by minimizing the Mean Squared Error (MSE) loss:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{5}$$

This provided a clear illustration of how parameters are updated through gradient descent:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{6}$$

and how loss landscapes govern convergence. **Polynomial Regression** extended this framework to non-linear relationships by introducing polynomial features, highlighting the tradeoff between model expressiveness and generalization. These experiments made the **bias–variance tradeoff** explicit: overly simple models underfit the data (high bias), while overly complex models capture noise and generalize poorly (high variance).

B. Classification

Classification introduced discrete prediction and probabilistic interpretation. **Logistic Regression** demonstrated how a linear model, combined with a sigmoid activation function $\sigma(z) = \frac{1}{1+e^{-z}}$, produces class probabilities and decision boundaries. The cost function used was the log loss (cross-entropy):

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))] \tag{7}$$

C. Instance-Based Learning

Instance-based learning was explored through **K-Nearest Neighbors (KNN)**, where predictions depend on proximity in feature space (e.g., Euclidean distance) rather than learned parameters. Varying the value of K illustrated how hyperparameters directly control model bias and variance: small K leads to complex boundaries (low bias, high variance), while large K smooths boundaries (high bias, low variance).

IV. Assignment 2 – Part II: Optimization Dynamics and Classical Models

Assignment 2 – Part II extended supervised learning by focusing explicitly on optimization behavior and more complex classical models. The central goal was to understand how different optimization strategies and inductive biases influence learning stability and generalization.

A. Stochastic Gradient Descent (SGD)

A key component of this assignment was the study of **Stochastic Gradient Descent (SGD)**. Unlike batch gradient descent, which uses the entire dataset for each update, SGD uses a single example (or a small batch):

$$\theta := \theta - \eta \nabla_\theta L(x^{(i)}, y^{(i)}) \tag{8}$$

By experimenting with different learning rates (η), the assignment demonstrated how improper step sizes can lead to slow convergence (if η is too small), oscillation, or divergence (if η is too large). Comparing **Batch Gradient Descent (BGD)** with SGD revealed a fundamental tradeoff: batch updates yield smoother convergence but are computationally expensive, while stochastic updates introduce gradient noise that improves scalability and, in some cases, helps escape poor local minima.

B. Decision Trees

Decision Trees were studied to understand non-linear decision making and interpretability. Concepts such as **entropy**:

$$H(S) = -\sum_{c} p_c \log_2 p_c \tag{9}$$

and **Information Gain**:

$$IG(S, A) = H(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} H(S_v) \tag{10}$$

were examined to explain how trees select splits to minimize impurity. The role of tree depth as a capacity control mechanism highlighted the susceptibility of trees to overfitting.

C. Support Vector Machines (SVM)

SVMs introduced a geometric perspective on classification through **margin maximization**. The assignment emphasized the role of support vectors, the effect of the regularization parameter C, and the use of **kernel functions** (e.g., RBF kernel) to handle non-linearly separable data by mapping it to a higher-dimensional space:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right) \tag{11}$$

V. Assignment 3: Deep Learning, Transfer Learning, and Initialization

Assignment 3 served as the critical bridge between classical supervised learning and meta-learning by focusing on deep neural networks, transfer learning, and the role of initialization in low-data regimes.

A. Deep Learning Foundations

Deep neural networks were studied as compositions of linear transformations followed by non-linear activations. For a layer l:

$$h^{(l)} = \sigma(W^{(l)}h^{(l-1)} + b^{(l)}) \tag{12}$$

where $\sigma(\cdot)$ introduces non-linearity. For classification, the training minimizes **cross-entropy loss**:

$$L = -\sum_i y_i \log(\hat{y}_i) \tag{13}$$

A key learning was that deep networks obey the same optimization principle as classical models: parameters are updated by gradient descent via **Backpropagation**.

B. Transfer Learning Experiments

The core experimental component involved implementing an end-to-end CNN training pipeline on the MNIST dataset. Transfer learning was explored through three strategies:

1) **Random Initialization:** Training from scratch.
2) **Fine-tuning:** Pretraining on a related task.
3) **Broad Pretraining:** Pretraining on a merged dataset.

Tasks were constructed using binary splits of MNIST (e.g., digits 0-1 as Task A). Empirical evaluation showed that **pretrained models converged significantly faster** and achieved better performance in low-data settings compared to random initialization. This aligns with the optimization view that starting from a better initialization $\theta_{pretrained}$ reduces the distance to the optimum θ^*:

$$\|\theta^* - \theta_{pretrained}\| \ll \|\theta^* - \theta_{random}\| \tag{14}$$

C. Connection to Meta-Learning

Key insights included:

- Initialization matters more than architecture in few-shot settings.
- Transfer learning improves convergence speed.
- Transfer learning is greedy; it optimizes for source-task performance, not explicitly for *adaptability*.

These limitations motivate MAML, which explicitly optimizes an initialization θ such that a small number of gradient steps yields strong performance on new tasks.

VI. Meta-Learning and MAML

A. Task-Based Learning Perspective

Meta-learning reframes learning around a **distribution of tasks** $p(\mathcal{T})$ rather than individual data points. Each task $\mathcal{T}_i$ consists of a small **support set** $\mathcal{D}_{\mathcal{T}_i}^{sup}$ and a **query set** $\mathcal{D}_{\mathcal{T}_i}^{qry}$. The goal is to enable rapid improvement on a new task after observing only a few examples.

B. Core Idea of MAML

MAML optimizes model parameters such that gradient descent itself becomes effective. Mathematically, it performs a bi-level optimization:

1) Inner Loop (Adaptation): For each task $\mathcal{T}_i$, adapt parameters θ to θ_i':

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(\theta) \tag{15}$$

2) Outer Loop (Meta-Update): Update the initialization θ to minimize the sum of losses on the query sets:

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(\theta_i') \tag{16}$$

This structure explicitly optimizes *how parameters should change*, not just where they should end up.

C. Practical Considerations

Because the meta-gradient depends on the inner update, applying the chain rule leads to second-order derivatives:

$$\nabla_\theta \mathcal{L}_{\mathcal{T}_i}(\theta_i') = \nabla_{\theta_i'} \mathcal{L}_{\mathcal{T}_i}(\theta_i') \cdot \left(I - \alpha \nabla_\theta^2 \mathcal{L}_{\mathcal{T}_i}(\theta)\right) \tag{17}$$

To mitigate the computational cost of the Hessian, **First-Order MAML (FOMAML)** approximates this by ignoring the second-order term, often retaining competitive performance.

VII. Insights from the MAML Paper

The paper "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks" establishes that fast adaptation can be learned. By optimizing parameters such that a small number of gradient steps leads to strong task-specific performance, MAML provides a principled bridge between deep learning, transfer learning, and few-shot learning.

Key contributions discussed include:

- **Model-Agnosticism:** MAML applies to classification, regression, and reinforcement learning.
- **Learned Initialization:** MAML learns an initialization that is highly sensitive to informative gradients.
- **Second-Order Optimization:** Differentiating through the adaptation step allows the meta-optimization to account for loss landscape curvature.