Mid Evaluation Report

Project: Model-Agnostic Meta-Learning (MAML)

Name: Pradeep Meena
Roll no : 240757

1 Introduction to Machine Learning Models

The main focus of the latter part of the course was on understanding how models learn from data, how training works, why models fail, and how neural networks extend basic learning models from the single perceptrons to the multi-layer perceptrons.

2 Types of Learning

We began by studying the main categories of machine learning.

2.1 Unsupervised Learning

Unsupervised learning works without labels. The goal is to discover patterns or structure in data. Clustering methods such as k-means belong to this category, where similar data points are grouped together.

2.2 Supervised Learning

Supervised learning uses labeled data. Given an input vector x, the model learns to predict an output y. This includes:

2.2.1 Regression and Classification

machine learning problems such as regression and classification can be understood from a probabilistic perspective. instead of directly minimizing an error, we assume an underlying probability distribution for the data and estimate the model parameters that best explain the observed samples.

3 Linear Regression

Linear regression models the relationship between inputs and output using a linear equation:

$$y = w^T x + b \tag{1}$$

where x is the feature vector, w is the weight vector, and b is the bias. The objective of training is to find values of w and b that minimize prediction error.

3.1 Loss Function and Derivation

We use mean squared error (MSE) as the loss function:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2}$$

Substituting $\hat{y}_i = w^T x_i + b$:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - (w^T x_i + b)\right)^2 \tag{3}$$

This loss function is smooth and differentiable, which allows optimization using gradient-based methods.

3.2 Matrix Form

Let X be the data matrix, y the target vector, and w the weight vector. The MSE can be written as:

$$\text{MSE} = \frac{1}{n}(y - Xw)^T(y - Xw) \tag{4}$$

This compact form simplifies gradient computation and improves computational efficiency.

3.2.1 Why We Assume the Normal Distribution

In linear regression, it is commonly assumed that the target values are generated from a normal (Gaussian) distribution centered around the model prediction:

$$y \sim \mathcal{N}(w^T x, \sigma^2) \tag{5}$$

This assumption implies that the noise in the data is normally distributed with zero mean and variance σ^2. This probabilistic assumption directly leads to the mean squared error (MSE) loss.

3.2.2 Maximum Likelihood Estimation

For independent samples, the likelihood is:

$$L(\theta) = \prod_i P(y_i \mid x_i, \theta) \tag{6}$$

Taking the logarithm:

$$\hat{\theta} = \arg\max_\theta \log L(\theta) \tag{7}$$

3.2.3 MLE for Linear Regression

Under Gaussian noise:

$$P(y_i \mid x_i, w) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y_i - w^T x_i)^2}{2\sigma^2}\right) \tag{8}$$

Maximizing the log-likelihood reduces to:

$$\arg\min_w \sum_i (y_i - w^T x_i)^2 \tag{9}$$

Thus, maximum likelihood estimation is equivalent to minimizing the mean squared error.

3.2.4 MLE for Classification

For binary classification, the target follows a Bernoulli distribution:

$$P(y_i \mid p_i) = p_i^{y_i}(1 - p_i)^{1-y_i} \tag{10}$$

The log-likelihood becomes:

$$\log L = \sum_i [y_i \log p_i + (1 - y_i) \log(1 - p_i)] \tag{11}$$

This yields the binary cross-entropy loss.

4 Optimization in Linear Regression

The loss function for linear regression is given by:

$$\mathcal{L}(w) = \frac{1}{N} \sum_{i=1}^{N} (y_i - w^T x_i)^2 \tag{12}$$

The gradient of the loss function with respect to the weight vector w is:

$$\nabla_w \mathcal{L} = \frac{2}{N} X^T (Xw - y) \tag{13}$$

5 Logistic Regression

Logistic regression is used for binary classification problems where the output variable can take only two values, typically 0 or 1.

The probability that the output belongs to class 1 is modeled using the sigmoid function:

$$P(y = 1 \mid x) = \sigma(w^T x + b) \tag{14}$$

where the sigmoid function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{15}$$

6 Why Use Gradient Descent?

Training a machine learning model involves minimizing a loss function. Gradient descent is an iterative optimization algorithm used to find the parameters that minimize the loss.

6.1 Gradient of Mean Squared Error

The gradient of the mean squared error loss with respect to the weights is given by:

$$\nabla_w \text{MSE} = \frac{2}{n} X^T (Xw - y) \tag{16}$$

6.2 Gradient Descent Update Rule

The parameters are updated in the direction opposite to the gradient:

$$w_{\text{new}} = w_{\text{old}} - \eta \nabla_w \text{MSE} \tag{17}$$

7 K-Means Clustering (Unsupervised Learning)

K-means is an unsupervised learning algorithm used to partition data into K distinct clusters. Each data point is assigned to the cluster whose mean is closest to it.

7.1 Problem with Random Initialization

The standard K-means algorithm initializes cluster centroids randomly. This can lead to poor clustering results because the algorithm may converge to a local minimum depending on the initial choice of centroids.

7.2 K-Means++ Initialization

K-means++ is an improved initialization technique that aims to spread the initial cluster centroids far apart from each other, leading to better clustering performance and faster convergence.

7.3 K-Means++ Algorithm

1. Choose the first centroid randomly from the data points.
2. For each data point x_i, compute the squared distance to the nearest existing centroid:
$$D(x_i)^2 = \min_j \|x_i - \mu_j\|^2$$
3. Choose the next centroid with probability proportional to $D(x_i)^2$:
$$P(x_i) = \frac{D(x_i)^2}{\sum_k D(x_k)^2}$$
4. Repeat steps 2 and 3 until K centroids are selected.

5. Proceed with the standard K-means algorithm using these initialized centroids.

8 Underfitting and Overfitting

A machine learning model can fail to generalize well to unseen data due to underfitting or overfitting.

8.1 Underfitting

Underfitting occurs when the model is too simple to capture the underlying structure of the data. In this case, the model performs poorly on both training and testing data.

$$\text{MSE}_{\text{train}} \uparrow, \quad \text{MSE}_{\text{test}} \uparrow \tag{18}$$

8.2 Overfitting

Overfitting occurs when the model is too complex and learns the noise in the training data instead of the true pattern. This results in very low training error but poor performance on unseen data.

$$\text{MSE}_{\text{train}} \approx 0, \quad \text{MSE}_{\text{test}} \uparrow \tag{19}$$

8.3 Regularization

Regularization is a technique used to prevent overfitting by adding a penalty term to the loss function. In L2 regularization, the loss function becomes:

$$\mathcal{L}_{\text{reg}} = \text{MSE} + \lambda \sum_i w_i^2 \tag{20}$$

8.4 Feature Scaling

Feature scaling is performed to ensure that all input features have comparable ranges, which helps gradient-based optimization algorithms converge faster.

One commonly used feature scaling method is standardization, defined as:

$$x' = \frac{x - \mu}{\sigma} \tag{21}$$

9 Multilayer Perceptron (MLP)

A multilayer perceptron is a class of feedforward artificial neural networks that consists of multiple layers of neurons. Each neuron performs a weighted sum of its inputs followed by a non-linear activation function.

9.1 Neuron Model

The basic computation performed by a neuron is given by:

$$z = w^T x + b \tag{22}$$
$$h = \phi(z) \tag{23}$$

where $\phi(\cdot)$ is an activation function.

A commonly used activation function is the Rectified Linear Unit (ReLU), defined as:

$$\text{ReLU}(x) = \max(0, x) \tag{24}$$

9.2 Forward Propagation

In a multilayer perceptron with two hidden layers, the forward propagation equations are:

$$h_1 = \text{ReLU}(W_1 x + b_1) \tag{25}$$
$$h_2 = \text{ReLU}(W_2 h_1 + b_2) \tag{26}$$
$$\hat{y} = W_o h_2 + b_o \tag{27}$$

9.3 Loss Function

The mean squared error loss for regression using an MLP is given by:

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{28}$$

9.4 Parameter Update

The network parameters are updated using gradient descent as follows:

$$w \leftarrow w - \eta \frac{\partial \mathcal{L}}{\partial w} \tag{29}$$

9.5 Training Algorithm

The training process of an MLP can be summarized as follows:

Training a Multilayer Perceptron epoch = 1 to E each training sample (x, y) Perform forward propagation Compute the loss Perform backpropagation Update network parameters

10 Convolutional Neural Networks (CNN)

Convolutional Neural Networks are a specialized class of neural networks designed to process grid-like data such as images. CNNs exploit the spatial structure of the input data by using convolutional layers instead of fully connected layers.

10.1 Why CNNs are Used for Image Data

Traditional artificial neural networks treat every input pixel independently, ignoring the spatial relationship between neighboring pixels. CNNs overcome this limitation by using convolution operations that capture local spatial patterns.

10.2 Key Characteristics of CNN

Local Connectivity: Each neuron in a convolutional layer is connected only to a small region of the input, known as the receptive field.

Weight Sharing: The same set of weights (filters) is used across different spatial locations, significantly reducing the number of parameters.

Translational Invariance: CNNs are robust to small translations in the input image, meaning that an object can be recognized regardless of its position.

Dimensionality Reduction: Pooling layers reduce the spatial dimensions of the feature maps, leading to lower computational cost.

Non-linearity: Non-linear activation functions allow CNNs to model complex patterns in data.

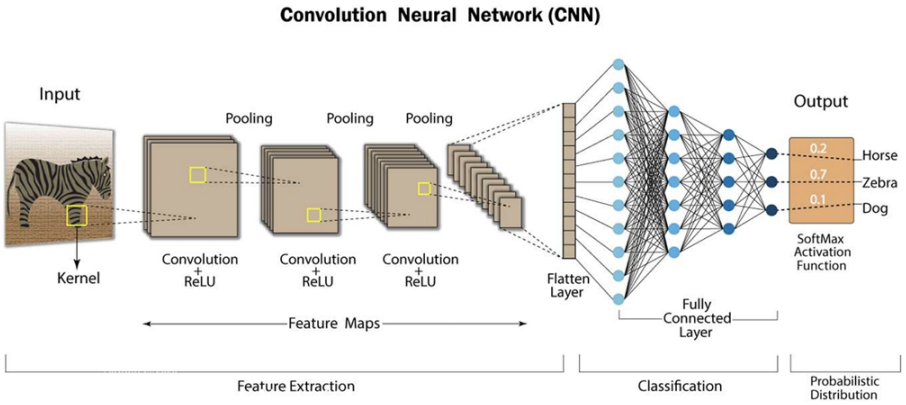

Figure 2: A standard Convolutional Neural Network (CNN) architecture, showing the feature extraction layers (Convolution, ReLU, Pooling) and the final classification layers.

Figure 1: A standard convolutional Neural network (CNN) architeture , showing the feature extraction layer (convolution , Relu , polling) aand the final classification layers.

10.3 CNN vs Artificial Neural Networks

Compared to traditional artificial neural networks, CNNs offer the following advantages:

- Fewer parameters due to weight sharing
- Better performance on image data
- Preservation of spatial information
- Improved computational efficiency

11 Convolutional Neural Networks (CNN)

A Convolutional Neural Network (CNN) is a specialized deep learning architecture designed primarily for processing data with a grid-like topology, such as images. Unlike standard Multi-Layer Perceptrons (MLPs), which connect every input neuron to every output neuron resulting in "many weights" and high computational cost, CNNs use convolutional layers to process data more efficiently.

A typical CNN architecture is shown in Figure 2.

Key defining characteristics include:

Local Connectivity (Filters): Instead of dense matrix multiplication, CNNs use small filters (kernels) that slide across the input to perform mathematical operations (convolution).

Translational Invariance: The network is designed to recognize features regardless of their position in the image.

Dimensionality Reduction: CNNs employ Max Pooling layers to downsample the fea- ture maps, reducing the spatial dimensions and computational load.

Non-Linearity: They typically use ReLU (Rectified Linear Unit) activation functions ($a = \text{ReLU}[Wx + b]$) to introduce non-linearity into the model.

12 Model-Agnostic Meta-Learning (MAML)

This lecture focused on **Model-Agnostic Meta-Learning (MAML)**, a meta-learning framework proposed by Finn et al. that enables neural networks to adapt quickly to new tasks using only a small amount of data. The core idea of meta-learning is to *learn how to learn*, rather than learning a single task directly.

12.1 Motivation

In many real-world problems, collecting large labeled datasets for every new task is impractical. Humans, however, can adapt to new tasks using only a few examples. MAML aims to replicate this ability by training models that can achieve fast adaptation with only a few gradient updates.

Unlike task-specific meta-learning methods, MAML is **model-agnostic**, meaning it can be applied to any model trained using gradient descent, including fully connected networks, convolutional neural networks, and reinforcement learning policies.

12.2 Meta-Learning Problem Setup

In MAML, we consider a distribution over tasks $p(\mathcal{T})$. Each task $\mathcal{T}_i$ has its own loss function $\mathcal{L}_{\mathcal{T}_i}$. The objective is to learn a shared model initialization that can quickly adapt to a new task sampled from this distribution.

Meta-learning treats entire tasks as training examples. During training, the model is exposed to many tasks so that it learns an initialization that generalizes well across tasks.

12.3 Inner Loop: Task-Specific Adaptation

Given a task $\mathcal{T}_i$ and model parameters θ, MAML performs one or more gradient descent steps using a small task-specific dataset:

$$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta) \tag{30}$$

where:

- θ are the meta-parameters,
- θ_i' are the adapted parameters for task $\mathcal{T}_i$,
- α is the inner-loop learning rate.

This step represents fast adaptation to a new task.

12.4 Outer Loop: Meta-Optimization

After adaptation, the model is evaluated on new data from the same task $\mathcal{T}_i$. The meta-objective is to minimize the loss after adaptation:

$$\min_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}) \tag{31}$$

The meta-parameters θ are updated using gradient descent:

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}) \tag{32}$$

where β is the meta learning rate.

This update involves computing gradients *through* the gradient update, which requires second-order derivatives.

12.5 MAML Algorithm

The overall MAML procedure can be summarized as follows:

1. Sample a batch of tasks from $p(\mathcal{T})$.
2. For each task, perform one or more gradient steps to obtain adapted parameters.
3. Evaluate the adapted model on new data from each task.
4. Update the shared initialization parameters using the meta-gradient.

12.6 Applications of MAML

MAML has been successfully applied to multiple domains:

- **Few-shot classification** (e.g., Omniglot, MiniImagenet),
- **Few-shot regression** (e.g., sinusoid regression),
- **Reinforcement learning**, where agents adapt quickly to new environments.

12.7 Key Insights

- MAML learns a **good initialization**, not a task-specific model.
- Fast learning is achieved using standard gradient descent.
- The framework is simple, flexible, and architecture-independent.
- A first-order approximation of MAML can be used to reduce computation cost.

12.8 Conclusion

This lecture demonstrated how MAML provides a general and powerful framework for meta-learning. By optimizing model parameters for rapid adaptability, MAML enables efficient learning in low-data regimes and serves as a strong foundation for modern few-shot learning and adaptive systems.

13 (Python & Scientific Libraries)

- **Assignment 1 Highlights:** This was a refresher on Python basics—things like list comprehension, dictionary mapping, and basic control flow. We focused heavily on **NumPy** for array manipulation (essential for weights) and **Pandas** for managing datasets.
- **Visualization:** Using **Matplotlib**, we practiced plotting histograms and line graphs to see how data is distributed before feeding it into a model.
- **OOP Practice:** We worked on Object-Oriented Programming (OOP) by building class hierarchies (like a `Human Being` parent class). This is crucial because, in **PyTorch**, every neural network is built as a class.

14 Classical ML (Supervised Learning)

In Assignment 2, we moved from basic data to actual "learning" algorithms. We focused on the two main paradigms: Regression and Classification.

- **Linear Regression from Scratch:** Instead of just using a library, we wrote our own `fit` and `predict` methods to understand how weights are updated.
- **Optimization with SGD:** We compared standard Batch Gradient Descent with **Stochastic Gradient Descent (SGD)**. I found that while SGD is "noisy" and erratic, it's much better for large datasets because it doesn't wait to see the whole dataset before making a change.
- **Logic-Based Models:** We also calculated **Entropy** and **Information Gain** by hand. It showed how a Decision Tree "cleans up" a messy dataset by choosing the best questions to split the data.
- **SVM Theory:** We looked at how **Support Vector Machines** try to find the "widest street" (the maximum margin) to separate classes.

15 Deep Learning & MAML Intuition

The final phase was the jump into **PyTorch** and the core logic behind **Model-Agnostic Meta-Learning (MAML)**.

- **Assignment 3 Tasks:** We used **MNIST** but didn't train it the normal way. We split it into five different tasks (like digits 0–1 as Task A, 2–3 as Task B). This mimics real-world wireless problems where conditions change constantly.
- **The "Few-Shot" Problem:** We tested how a model performs with very few examples (the support set). Normal models usually fail here (overfitting), but **MAML** is different—it doesn't learn a specific task; it learns an **initialization** that is ready to learn *anything* fast.
- **CNN Implementation:** We built a Convolutional Neural Network with layers for feature extraction (conv layers) and classification.
- **Wireless Applications:** We discussed how this applies to **Channel Reconstruction** and **Trajectory Prediction for UAVs**, where we need models to adapt using 90% fewer samples than normal.