MACHINE LEARNING REFRESHER

Prepared by

MAHARAVO
Tefinjanahary Anicet

Supervised by

Pr. Belcaid Anass

Date

09/25/2024

Lecture 2

TABLE OF CONTENT

MACHINE LEARNING AS DATA-DRIVEN PROGRAMMING

Machine Learning is a type of programming where algorithms automatically learn from data by **extracting important features** to make decisions or predictions, rather than following explicitly defined rules and logic as in traditional programming.

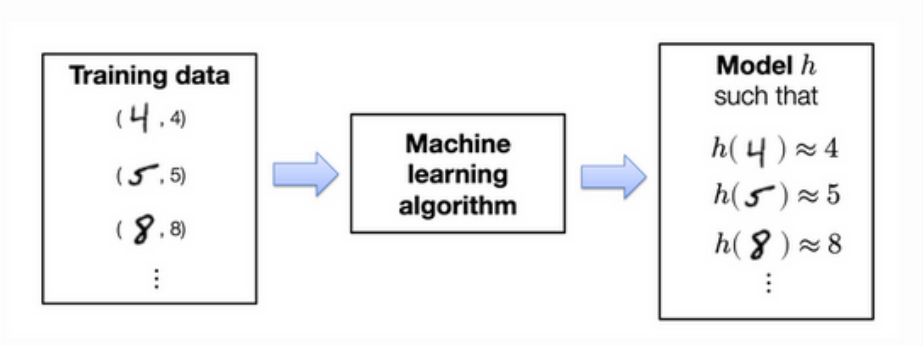
Example of applications: Optical Character Recognition (OCR)

Machine learning models are trained on the labeled data from the MNIST dataset, learning to associate pixel patterns with specific digits. Once trained, these models can generalize to accurately classify new images of digits, effectively automating the OCR task.

AlexNet 2012 Paper by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton
MNIST Dataset

ELEMENTS OF AN ML ALGORITHM

In supervised machine learning, we start by collecting a set of **labeled images**. These images are then used as input for the algorithm, which learns to create a program that solves the task.

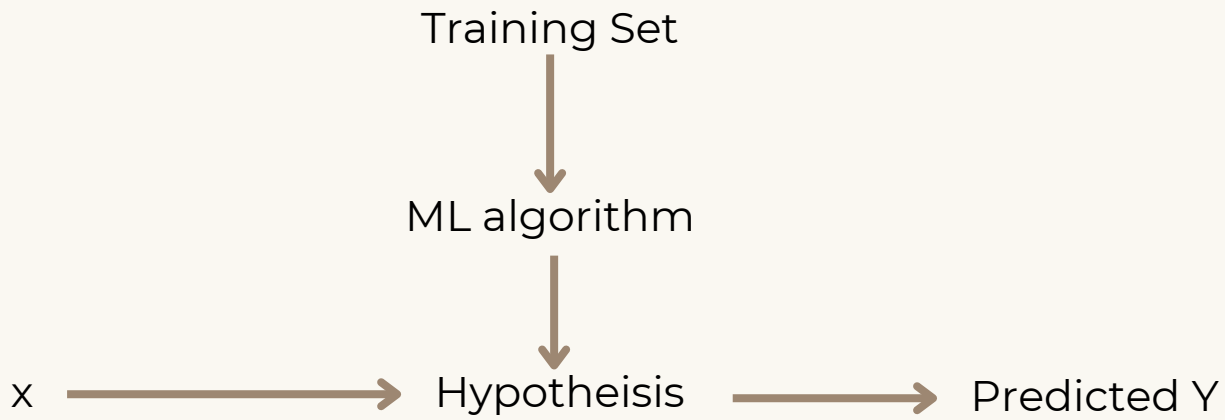


Three different elements:

- **Hypothesis Class**
- **Loss Function**
- **Optimization**

HYPOTHESIS CLASS

The hypothesis class consists of all the possible functions or structures that a given model can learn from the data. This includes the **different ways the model can map inputs** (such as digit images) to outputs (class labels) based on its parameters.



The flexibility of the hypothesis class is important because **it influences the patterns the model can learn**. A more complex hypothesis class can capture intricate relationships in the data but may also lead to overfitting, where the model learns noise instead of the actual patterns.

Example 1: Linear Classifier Hypothesis Class

In a linear classifier, the hypothesis class consists of linear functions that separate two different classes. It seeks a **straight line (or hyperplane)** to differentiate the classes based on features.

Simple Hypothesis Class: It can only accurately classify digits if they are **linearly separable**. For example, 1 and 7 based on simple features like stroke orientation.

Example 2: Neural Network Hypothesis Class

Neural networks have a complex hypothesis class with multiple layers that can learn **non-linear patterns**.

Complex Hypothesis Class: This allows them to recognize intricate shapes in data. For instance, a neural network can easily tell the difference between digits “8” and “3,” which have curves that a simpler model might miss.

LOSS FUNCTION

The loss function measures the difference between the model's predictions and actual labels in the training data. It indicates **the model's performance during training**. The objective of the algorithm is to adjust the model parameters to **minimize** this loss, leading to improved accuracy in predicting new, unseen data.

1: Classification Error

The Classification Error is the simplest loss function for classification tasks. It indicates whether the classifier made an error in its predictions

$$\text{err}(h(x), y) = \begin{cases} 0 & \text{if } \arg\max_i h_i(x) = y \\ 1 & \text{otherwise} \end{cases}$$

This loss function is useful for assessing classifier quality. However, it is not ideal for optimization because it is **not differentiable.**

2: Cross entropy

Softmax or Cross Entropy is more appropriate as it transforms the output of the hypothesis function (H) into probabilities by exponentiating the values and normalizing them to ensure they sum to one:

$$z_i = p(\text{label} = i) = \frac{\exp(h_i(x))}{\sum_j \exp\left(h_j(x)\right)}$$

i: the specific class of interest for which the probability is being calculated.

The **Cross Entropy loss** is calculated as the negative log of the predicted probability for the ***correct class (i = y)*:**

- y is the true class label (the actual class the data point belongs to
- Number of class

$$L_i = -\log(z_i)$$

$$L_{CE}(h(x), y) = -\log(p(\text{label} = y)) = -h_y(x) + \log\left(\sum_{j=1}^{k} e^{h_j(x)}\right)$$

Softmax function and Cross Entropy loss are differentiable. **Differentiability** allows the use of optimization methods like gradient descent to adjust the model's parameters. These functions provide a useful gradient that helps correct prediction errors and improve the model during training.

Error vs Loss

Error evaluates overall performance

Loss guides model optimization.

OPTIMIZATION

The objective is to minimize the loss function.

$$\text{Minimize}_\theta \frac{1}{m} \sum_{i=1}^{m} l\left(h_\theta(x^i), y^i\right)$$

For linear Classifier Model:

$$\text{Minimize}_\theta \frac{1}{m} \sum_{i=1}^{m} l\left(\theta^T x, y^i\right)$$

Optimization Methods

Gradient Descent:
A classical method for optimization that computes the gradient of the loss function to update model parameters.

Advanced Optimizers:
Variants of gradient descent such as Adam, RMSprop, and Adagrad improve convergence speed and reliability by **adaptively adjusting the learning rate and incorporating momentum.**

Gradient descent

To minimize the function, the gradient descent algorithm proceeds by iteratively taking steps in the direction of the negative gradient:

$$\theta = \theta - \alpha \nabla_\theta f(\theta)$$

θ : represents the model parameters,
∇f(θ) : the gradient of the loss function with respect to the parameters,
α : the step size (learning rate).

You can visualize this in the provided link, which demonstrates the evolution of the algorithms with various step sizes. *Comparison of Gradient Descent*

$$\nabla_\theta f(\theta) \in \mathbb{R}^{n \times k} = \begin{bmatrix} \frac{df(\theta)}{d\theta_{11}} & \cdots & \frac{df(\theta)}{d\theta_{1k}} \\ \vdots & \ddots & \vdots \\ \frac{df(\theta)}{d\theta_{n1}} & \cdots & \frac{df(\theta)}{d\theta_{nk}} \end{bmatrix}$$

X: a matrix of shape (N,D), where:
N is the number of samples (data points).
D is the number of features (dimensions) per sample
.
θ: a matrix of shape (D,C), where:
C is the number of classes (or output dimensions)

$h(X;\theta)=X \cdot \theta$ shape = (N,C)

gradient shape = shape θ = (D,C)

Stochastic Gradient Descent

Update the parameters using the Batch Gradient:

$$\theta = \theta - \frac{\alpha}{B} \sum_{i=1}^{B} \nabla_\theta l(h_\theta(x^i), y^i)$$

B: batch size

Gradient Cross Entropy

Gradient of softmax function

$$\begin{aligned} \frac{\partial l_{ce}(h, y)}{\partial h_i} &= \frac{\partial}{\partial h_i}\Big(- h_y + \log \sum_{j=1}^{k} \exp h_j \Big) \\ &= -1\{i = y\} + \frac{\exp h_i}{\sum_{j=1}^{k} \exp h_j} \end{aligned}$$

$$\nabla_h l_{ce}(h, y) = z - e_y$$

a vector of dimension k
z is the normalisation of the vector exp(h).

The gradient for the full loss using the ***chain rule***

$$\begin{aligned} \frac{\partial}{\partial \theta} l_{ce}(\theta^T x, y) &= \frac{\partial l_{ce}(\theta^T x, y)}{\partial \theta^T x} \frac{\theta^T x}{\partial \theta} \\ &= (z - e_y)(x) \end{aligned}$$

The previous formula is **wrong**. Typically, the shapes of θ and the gradient should match; however, in this case, θ has a shape of (n, k) while the gradient is (k, n).

x : Input vector (shape ***n×1***).

z−ey : Gradient of the loss with respect to the scores (shape **k×1**).

Transpose (z−ey): **Transpose** of the gradient (shape **1×k**).

x · (z−ey)**T** : The result is a matrix of shape **n×k**, which matches the **shape of θ**

So the correct formula is :

$$\frac{\partial}{\partial \theta} l_{ce}(\theta^T x, y) = x(z - e_y)^T$$

REFRENCES AND MATERACIALS___

https://anassbelcaid.github.io/deeplearning/

https://yann.lecun.com/exdb/mnist/

https://losslandscape.com

https://github.com/jiupinjia/Visualize-Optimization-Algorithms

Email

a.belcaid@uae.ac.ma