Lecture 2
Perceptrons

Contents

1 Introduction

The concept of the perceptron was introduced by **Frank Rosenblatt** in 1957. It was inspired by biological neurons and designed as a simple computational model for classification.

Rosenblatt introduced the idea of **weights** $(w_1, w_2, w_3, \dots)$, which determine the importance of each input x_j in contributing to the output of the perceptron.

Basic Mathematical Model:

$$y = \begin{cases} 1 & \text{if } \sum_j w_j x_j > \text{threshold} \\ 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \end{cases}$$

Here:

- x_j = input values
- w_j = weights associated with each input
- $\sum_j w_j x_j$ = weighted sum of inputs
- *threshold* = a cutoff value that decides whether the perceptron "fires" or not

Thus, the perceptron functions as a **linear classifier** that separates data into two classes based on the weighted sum of its inputs.

1.1 The Universal Model

When it was first introduced, the perceptron was believed to be a **universal computing device**. Researchers assumed it could represent *any Boolean circuit* and perform all kinds of logical reasoning.

Historical Note: In 1958, the *New York Times* described the perceptron as

> "the embryo of an electronic computer that would be able to walk, talk, see, write, reproduce itself, and even be conscious of its own existence."

While this was an exaggeration, it shows how revolutionary the perceptron was considered at the time.

1.2 Linear vs. Affine Functions

- A **linear function** always passes through the origin.
- An **affine function** is just a linear function *shifted* by a constant term.

1 variable:

- Linear: $f(x) = ax$ (line goes through $(0, 0)$).
- Affine: $f(x) = ax + b$ (same slope, but shifted up or down by b).

General Case (vectors):

- Linear: $f(v) = Av$ where A is an $m \times n$ matrix.
- Affine: $f(v) = Av + b$ where b is a constant vector in R^m.

Key Property:

- A function f is **linear** if it preserves vector space operations:

$$f(v_1 + v_2) = f(v_1) + f(v_2), \qquad f(kv) = kf(v)$$

- An **affine** function does the same, but allows a constant shift by b.

Intuition:

- Linear $\rightarrow$ straight line through the origin.
- Affine $\rightarrow$ same straight line, but moved (translated).

1.3 Bias as an Affine Function

In the perceptron:

$$z = \sum_{i=1}^{n} w_i x_i \quad \Rightarrow \quad \text{linear function (decision boundary passes through origin).}$$

Adding a bias term:

$$z = \sum_{i=1}^{n} w_i x_i + b \quad \Rightarrow \quad \text{affine function (decision boundary can shift).}$$

Key Insight:

- Without a bias, the perceptron can only draw decision boundaries that pass through the origin.
- With a bias, the boundary can be moved anywhere in space.
- Bias gives flexibility to position the separating line (or hyperplane) where it best fits the data.

2 Logic Gates in the Perceptron

A perceptron is not limited to just two inputs. It can handle **any number of inputs**, and some of these inputs can be **negated** by assigning negative weights.

General Form:

$$y = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i \geq T \\ 0 & \text{otherwise} \end{cases}$$

- $x_i \in \{0, 1\}$ are the binary inputs.
- w_i are the weights (positive = normal input, negative = negated input).
- T is the threshold.

Examples of Logic Gates

1. AND Gate

$$w_1 = 1,\ w_2 = 1,\ T = 2$$

x_1	x_2	Output y
0	0	0
0	1	0
1	0	0
1	1	1

2. OR Gate

$$w_1 = 1,\ w_2 = 1,\ T = 1$$

x_1	x_2	Output y
0	0	0
0	1	1
1	0	1
1	1	1

3. NOT Gate

$$w_1 = -1,\ T = 0$$

x_1	Output y
0	1
1	0

Key Insights

- Positive weights $\Rightarrow$ treat input normally.
- Negative weights $\Rightarrow$ act as NOT (negation).
- Threshold T controls how many inputs must be "on" for the perceptron to fire.

3 Drawbacks of the Perceptron

- **Binary output only:** The classic perceptron produces only two outputs (0 or 1). It cannot directly handle probabilities or multi-class problems.
- **Works only for linearly separable data:** With just one layer of neurons, it can only learn linear mappings. If the data is not linearly separable, the perceptron fails.

Example: XOR (exclusive OR)

$$\text{XOR}(x_1, x_2) = \begin{cases} 1 & \text{if } (x_1 = 1, x_2 = 0) \text{ or } (x_1 = 0, x_2 = 1) \\ 0 & \text{if } (x_1 = x_2) \end{cases}$$

XOR cannot be computed by a single perceptron because it is **not linearly separable** — no straight line (or hyperplane) can separate the classes.

- **Cannot learn feature interactions:** Each input is weighted independently, so the perceptron cannot capture complex relationships between features.
- **Prone to overfitting:** With limited generalization ability, a simple perceptron may memorize training data but fail on unseen data.

4 MLP as Universal Function Approximation

When multiple perceptrons are combined into layers, they form a **multi-layer perceptron (MLP)** or neural network. Unlike a single perceptron, which can only create a **linear decision boundary**, an MLP can represent highly **non-linear functions**.

- Each hidden layer transforms inputs into new feature spaces.
- Non-linear activation functions (e.g., sigmoid, ReLU) applied after each layer allow the network to capture complex feature interactions.
- Stacking more layers increases the expressive power of the network.

Universal Approximation Theorem (UAT)

- An MLP with at least **one hidden layer** and a non-linear activation can approximate **any continuous function** to arbitrary accuracy, given enough neurons.
- In other words, MLPs are **universal function approximators**.
- Without non-linear activations, even deep networks reduce to a simple linear model.

Why It Matters

- MLPs can learn complex, non-linear relationships between features.
- They approximate predictions closer to the true target compared to linear models.
- This theorem forms the foundation of modern **deep learning**.

5 Activation Functions (Overview)

Why Activation Functions?

- A perceptron computes only a weighted linear sum of inputs.
- Without a non-linear activation, the model can only create linear decision boundaries.

- An activation function $\theta(\cdot)$ maps real-valued outputs to a desired range (e.g., $\{0, 1\}$ for classification).

(Note: Activation functions will be discussed in detail later.)

Examples:

- **Step function:** Produces discrete outputs (0 or 1) $\rightarrow$ binary classification.
- **Sigmoid:** Smooth mapping to range $(0, 1)$ $\rightarrow$ probabilistic outputs.
- **ReLU:** Outputs $\max(0, x)$ $\rightarrow$ efficient and widely used in deep networks.

6 Classification using MLP

- Classification is about finding a **decision boundary** that separates data points from different classes.
- A single perceptron can solve classification only if the classes are **linearly separable**.
- Rosenblatt proved that for linearly separable data, the perceptron learning rule can compute the Boolean function exactly.
- An MLP, with hidden layers and non-linear activations, can represent **any decision boundary**, even in high-dimensional spaces.

7 Regression using MLP

- Regression predicts a **continuous value** instead of a discrete class.
- By using a linear (identity) activation in the output layer, an MLP can perform regression tasks.
- Hidden layers with non-linear activations still allow the model to capture complex, non-linear patterns in the data.
- Examples: predicting house prices, stock market values, or patient health outcomes.