Single layer perceptron

Definition

A **perceptron** is the most basic form of an artificial neuron — a single-layer neural network introduced by **Frank Rosenblatt (1958)** for **binary classification** problems (e.g., YES/NO, +1/−1).
It mimics the behavior of a biological neuron by taking inputs, processing them, and producing an output decision.

Mathematically:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

Where:

- x_i = input features
- w_i = weights (importance of each input)
- b = bias (shifts the decision boundary)
- $f(\cdot)$ = activation function (e.g., sign function)
- y = output (+1 or −1)

Structure

A perceptron consists of:

1. **Inputs ($x_1, x_2, ..., x_n$)** — feature values
2. **Weights ($w_1, w_2, ..., w_n$)** — learnable parameters
3. **Bias (b)** — controls the threshold
4. **Summation unit** — computes $z = \mathbf{w} \cdot \mathbf{x} + b$
5. **Activation function** — often the sign function:

$$f(z) = \begin{cases} 1, & z \geq 0 \\ -1, & z < 0 \end{cases}$$

Working Principle

Step 1: Weighted Sum

$$z = \sum_{i=1}^{n} w_i x_i + b$$

Step 2: Activation Function (Step/Sign function in a basic perceptron)

$$\text{output} = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

Step 3: Output

The perceptron outputs **1** or **0**, making a two-class decision.

Training Algorithm

1. **Initialize** weights & bias (small random values or zeros)
2. **For each training sample**:
    - Compute the prediction
    - Update weights & bias if the prediction is wrong
3. **Repeat** until:
    - No errors in a full pass over the data (**convergence**)
    - Or maximum iterations reached

Example

Dataset features:

- x_1 = hours studied
- x_2 = hours slept
- Target: Pass (**1**) or Fail (**0**)

Step 1: Initialize

$$w_1 = 0, \quad w_2 = 0, \quad b = 0$$

Step 2: First example: $(x_1 = 3, x_2 = 5, y = 1)$
Weighted sum:

$$z = (0)(3) + (0)(5) + 0 = 0$$

Prediction: 1 (correct) → No update

Convergence Property

- If data is **linearly separable**, the perceptron is guaranteed to converge to a perfect solution in a finite number of steps.
- If data is **not linearly separable**, the perceptron will never fully converge and will keep oscillating.

Limitations

- Can only solve **linearly separable** problems (fails on XOR, etc.)
- Step activation is **not differentiable** → cannot use gradient descent directly
- No probability output (just hard classification)
- Sensitive to noisy data and outliers