Perceptron

By: **Annamalai A** with **GPT** assist

Biological Inspiration

A biological neuron has:

- **Dendrites** → receive input signals
- **Cell body** → processes signals
- **Axon** → sends output signals

The perceptron mathematically models this behavior:

- Inputs → dendrites
- Weighted sum → cell body
- Activation function → firing decision
- Output → axon

McCulloch Pitts Neuron

The McCulloch Pitts (MCP) neuron, introduced in 1943 by Warren McCulloch and Pitts, is a simplified mathematical model of a biological neuron. It receives binary inputs (0 or 1), applies fixed weights and a threshold, and produces a binary output. The MCP model was foundational in early neural network theory and logic representation.

Model description

- Inputs: $x_1, x_2, \ldots, x_n \in \{0, 1\}$
- Weights: $w_1, w_2, \ldots, w_n$ (often 0 or 1 in the classical MCP)
- Threshold θ (a non-negative integer)
- Output: $y = 1$ if $\sum_i w_i x_i \geq T$, otherwise $y = 0$

Key point

MCP neurons implement logical functions (AND, OR, NOT combinations) using threshold logic. They are binary, deterministic, and do not learn; weights and thresholds are set manually.

Introduction to the Perceptron

The **Perceptron** (Rosenblatt, 1957) is a later, learnable version of the MCP neuron. It generalizes MCP by allowing real-valued weights and a trainable bias, and it can learn these parameters from data using a simple update rule.

Perceptron Architecture

A perceptron has:

- Input features $x_1, x_2, \ldots, x_n$
- Weights $w_1, w_2, \ldots, w_n$ (real-valued)
- Bias b (or weight for an input fixed as 1)
- Activation function (commonly the step function for classification)

Equation

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

$$\hat{y} = f(\theta) = \begin{cases} 1 & \text{if } \theta > 0 \\ 0 & \text{otherwise} \end{cases}$$

Perceptron Learning Rule

The perceptron updates weights only on mistakes. For a training example (x, y):

- Predict $\hat{y}$ using current weights
- Compute error $\delta = y - \hat{y}$ (which is in $\{-1, 0, 1\}$)
- Update:

$$w_i = w_i + \eta\, \delta\, x_i$$

$$b = b + \eta\,\delta$$

where η is the learning rate.

Key properties:

- If $\hat{y} = y$ → no update
- If $\hat{y} = 0, y = 1$ → weights are increased toward that example
- If $\hat{y} = 1, y = 0$ → weights are decreased

Decision Boundary and Geometry

The perceptron defines a linear decision boundary (hyperplane) given by:

$$\{x : w^T x + b = 0\}$$

This hyperplane separates feature space into two half-spaces for class 0 and class 1. Training moves the hyperplane to better separate classes.

For 2D, decision boundary is a line

For 3D, decision boundary is a plane

For n-D, a hyperplane

Convergence Theorem

If the training data is **linearly separable**, the perceptron learning algorithm converges to a weight vector that correctly classifies all examples in a finite number of steps (assuming a fixed learning rate). If the data is not linearly separable, the algorithm does not converge and may cycle.

Limitations

- Can only solve **linearly separable** problems (e.g., fails on XOR)
- Binary outputs only (basic perceptron)
- Sensitive to feature scaling and noisy labels
- No probabilistic outputs (gives hard 0/1 predictions)

Advantages

- Simple and fast to train
- Easy to implement and interpret
- Foundation for modern neural networks

Variants and Extensions

- **Multi-layer Perceptron (MLP)**: stacks perceptrons with non-linear activations to solve non-linear problems.
- **Perceptron with different activations**: using sigmoid or ReLU and training via gradient descent leads to logistic regression or neural network models.

Here is an example for AND and OR gate using MCP/Perceptron:

- **AND**: weights = [1,1], threshold = 2 (or bias = -1.5)
- **OR**: weights = [1,1], threshold = 1 (or bias = -0.5)