UNIVERSIDADE
FEDERAL DO CEARÁ

Perceptron

A Simple Computational Model of a Neuron

Author:
Paulo Henrique Amaro Matias da Silva

Course: Electrical Engineering
Subject: Introduction to Engineering

Department: Department of Electrical Engineering
Center: Center of Technology
Institution: Federal University of Ceará (UFC)

Fortaleza, Brazil
June 2025

Contents

1 Perceptron Model **2**
1.1 Perceptron's Equation 2
1.2 Activation Function 2
1.3 Training Model 2
1.4 Working Principle of the Perceptron 2
1.5 AND Port 3

2 C Code for the Perceptron **3**

1 Perceptron Model

The perceptron, originally introduced by Frank Rosenblatt in 1958, is a foundational model of an artificial neuron that is useful for performing binary classification tasks. It processes multiple *inputs* weighted inputs, sum the results, and applies an activation function to return an output.

1.1 Perceptron's Equation

The output y expressed as:

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

Where:

- x_i: input i
- w_i: weight associated with input x_i
- b: bias term
- f: activation function

1.2 Activation Function

In this paper, the Heaviside step function is employed as the activation function for the perceptron model. It is defined as follows:

$$f(u) = \begin{cases} 1, & \text{if } u \geq 0 \\ 0, & \text{if } u < 0 \end{cases}$$

1.3 Training Model

The training of the perceptron is performed through an iterative adjustment of the weights w_i and the bias b based on the desired output error d relative to the obtained output y:

$$w_i \leftarrow w_i + \eta \cdot (d - y) \cdot x_i$$
$$b \leftarrow b + \eta \cdot (d - y)$$

Where η is a learning rate.

1.4 Working Principle of the Perceptron

The perceptron simulates the behavior of an artificial neuron by assigning weights to the inputs and processing their weighted sum with a bias term. After this calculation, an activation function is applied to determine the model's binary output.

1. Multiply each input x_i by its corresponding weight w_i.
2. Sum the weighted values and add the bias b.
3. Apply the activation function $f(u)$ to generate the output.
4. Compare it with the desired output d. If there is an error, adjust the weights as follows:

$$w_i \leftarrow w_i + \eta \cdot (d - y) \cdot x_i$$

$$b \leftarrow b + \eta \cdot (d - y)$$

1.5 AND Port

Considering the logical AND function, whose output is 1 only when both inputs are 1. The truth table of the function is as follows:

x_1	x_2	AND (x_1, x_2)
0	0	0
0	1	0
1	0	0
1	1	1

The perceptron can be trained to learn this function by adjusting the weights and bias according to the rules described.

2 C Code for the Perceptron

Below is the complete code of a simple perceptron implemented in the C programming language, trained to learn the logic of the AND gate.

```c
#include <stdio.h>

#define EPOCHS 10
#define LEARNING_RATE 0.1

int step_function(float sum) {
    return (sum >= 0) ? 1 : 0;
}

int main() {
    int inputs[4][2] = {
        {0, 0},
        {0, 1},
        {1, 0},
        {1, 1}
    };
    int expected_outputs[4] = {0, 0, 0, 1};

    float w1 = 0.0, w2 = 0.0, bias = 0.0;

    for (int epoch = 0; epoch < EPOCHS; epoch++) {
        printf("Epoca %d\n", epoch + 1);
        for (int i = 0; i < 4; i++) {
            int x1 = inputs[i][0];
            int x2 = inputs[i][1];
            int target = expected_outputs[i];

            float sum = x1 * w1 + x2 * w2 + bias;
            int output = step_function(sum);
            int error = target - output;

            w1 += LEARNING_RATE * error * x1;
            w2 += LEARNING_RATE * error * x2;
            bias += LEARNING_RATE * error;

            printf("Entrada: [%d, %d] -> Saida: %d | Esperado: %d | Erro: %d\n",
                   x1, x2, output, target, error);
        }
        printf("Pesos: w1=%.2f, w2=%.2f, bias=%.2f\n\n", w1, w2, bias);
    }

    printf("=== Teste Final ===\n");
    for (int i = 0; i < 4; i++) {
        int x1 = inputs[i][0];
        int x2 = inputs[i][1];
        float sum = x1 * w1 + x2 * w2 + bias;
        int output = step_function(sum);
        printf("Entrada: [%d, %d] -> Saida final: %d\n", x1, x2, output);
    }

```

```
    return 0;
}
```

Listing 1: Implementation of a Basic Perceptron for the AND Logic Gate