

# Perceptron compuertas lógicas

Victor Gerardo Rodríguez Barragán

29 de Octubre de 2023



# 1 Descripción del ejercicio y solución

## 1.1 Ejercicio

El ejercicio se centra en la implementación y entrenamiento de un perceptrón para resolver la compuerta lógica AND. Una compuerta AND es una operación lógica que toma dos entradas booleanas (0 o 1) y devuelve una salida que es 1 solo cuando ambas entradas son 1; de lo contrario, devuelve 0. El objetivo del perceptrón es aprender una función que pueda replicar este comportamiento lógico.

## 1.2 Solucion

1. **Inicializacion:** Se inicializan de manera aleatoria los pesos y el bias con una tasa fija de aprendizaje de 0.01.
2. **Funcion de activacion:** Se utiliza una función de activación simple que activa la salida si la suma ponderada de las entradas es mayor que cero. La función de activación devuelve 1 en ese caso y 0 en caso contrario.
3. **Entrenamiento:** Se entrena el perceptrón con los datos de entrada y salida esperados.
4. **Iteracion:** El proceso de entrenamiento se repite 10,000 veces para garantizar que el perceptrón tenga suficientes oportunidades para aprender la función.
5. **Prueba:** Se prueba el perceptrón con los datos de entrada y salida esperados.

# 2 Código

```
use rand::Rng;

struct Perceptron {
    weights: [f64; 2],
    bias: f64,
    learning_rate: f64,
}

impl Perceptron {
    fn new() -> Self {
        let mut rng = rand::thread_rng();
        let weights = [rng.gen_range(-1.0..1.0), rng.gen_range(-1.0..1.0)];
        let bias = rng.gen_range(-1.0..1.0);
        let learning_rate = 0.01;
    }
}
```

```

        Perceptron {
            weights,
            bias,
            learning_rate,
        }
    }

    fn activate(&self, sum: f64) -> i32 {
        if sum > 0.0 {
            1
        } else {
            0
        }
    }

    fn feed_forward(&self, inputs: &[i32; 2]) -> i32 {
        let weighted_sum = (inputs[0] as f64 * self.weights[0])
            + (inputs[1] as f64 * self.weights[1])
            + self.bias;
        self.activate(weighted_sum)
    }

    fn train(&mut self, inputs: &[i32; 2], target: i32) {
        let guess = self.feed_forward(inputs);
        let error = target - guess;
        for i in 0..2 {
            self.weights[i] += self.learning_rate * (error as f64) * (inputs[i] as f64);
        }
        self.bias += self.learning_rate * (error as f64);
    }
}

fn main() {
    // Perceptrón para la compuerta lógica AND
    let mut and_perceptron = Perceptron::new();
    let and_training_data = [
        ([0, 0], 0),
        ([0, 1], 0),
        ([1, 0], 0),
        ([1, 1], 1),
    ];

    // Entrenamiento para la compuerta lógica AND
    for _ in 0..10000 {
        for &(inputs, target) in &and_training_data {
            and_perceptron.train(&inputs, target);
        }
    }
}

```

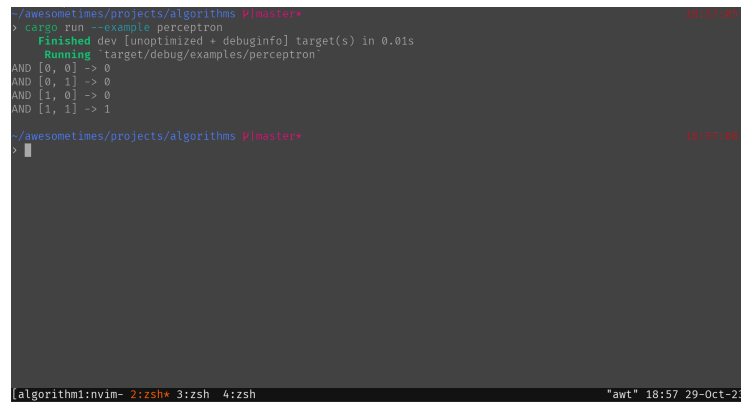
```

    }
}

// Pruebas para AND
for &(inputs, _) in &and_training_data {
    let result = and_perceptron.feed_forward(&inputs);
    println!("AND {:?} -> {}", inputs, result);
}
}

```

### 3 Resultados



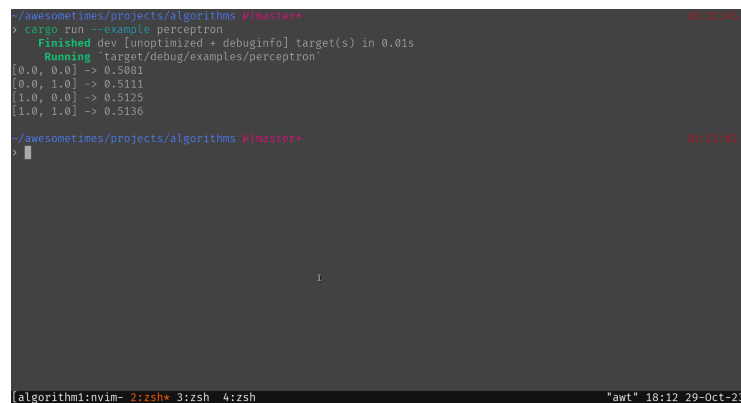
```

~/awesometimes/projects/algorithms $ cargo run --example perceptron
> cargo run --example perceptron
    Finished dev [unoptimized + debuginfo] target(s) in 0.01s
    Running `target/debug/examples/perceptron`
AND [0, 0] -> 0
AND [0, 1] -> 0
AND [1, 0] -> 0
AND [1, 1] -> 1

~/awesometimes/projects/algorithms $

```

#### 3.1 Intento de hacer la compuerta XOR



```

~/awesometimes/projects/algorithms $ cargo run --example perceptron
> cargo run --example perceptron
    Finished dev [unoptimized + debuginfo] target(s) in 0.01s
    Running `target/debug/examples/perceptron`
[0.0, 0.0] -> 0.5081
[0.0, 1.0] -> 0.5111
[1.0, 0.0] -> 0.5125
[1.0, 1.0] -> 0.5136

~/awesometimes/projects/algorithms $

```

## **4 Conclusion**

### **4.1 Aprendizajes**

Se aprendieron los conceptos básicos de un perceptrón, que es la unidad fundamental de las redes neuronales artificiales.

Se experimentó con la capacidad del perceptrón para aprender una función lógica simple, en este caso, las compuertas AND y XOR.

### **4.2 Implementacion del algoritmo**

Se puede implementar un perceptrón para resolver cualquier problema de clasificación binaria. Sin embargo, el perceptrón no puede resolver problemas que no son linealmente separables, como la compuerta XOR a menos que se utilicen múltiples perceptrones.

### **4.3 Retos y dificultades**

Sin duda el reto mas grande fue el de implementar la compuerta XOR, ya que no se podia resolver con un solo perceptron, lo intente implementar de diferentes maneras pero no pude resolverlo.