Actividad 2 Algoritmo del perceptrón

Alumno: José Osvaldo Farfán de León

Codigo: 214796622

Materia: IA ll

Profesor: Carlos Alberto Villaseñor Padilla

Sección: “D03”

Algoritmo de McCulloch-Pitts

El Modelo de McCulloch-Pitts, desarrollado por Warren McCulloch y Walter Pitts en 1943, es un modelo teórico que sienta las bases para comprender el funcionamiento de las redes neuronales artificiales. Aunque este modelo es bastante simple en comparación con las redes neuronales modernas, fue un punto de partida importante en el campo de la inteligencia artificial y la neurociencia computacional.

El Modelo de McCulloch-Pitts se basa en la idea de que las neuronas biológicas pueden ser aproximadas por unidades computacionales simples. Estas unidades, llamadas "neuronas artificiales" o "células de McCulloch-Pitts", toman múltiples señales de entrada binarias (0 o 1) y generan una única señal de salida binaria (0 o 1) en función de una regla predefinida.

Las características clave del modelo son las siguientes:

- Entradas binarias: Cada neurona artificial recibe múltiples señales de entrada, que pueden ser binarias, es decir, activadas (1) o desactivadas (0).
- Pesos sinápticos: Cada entrada está asociada con un peso sináptico que indica la importancia relativa de esa entrada para la neurona.
- Función de activación: La neurona aplica una función de activación a la suma ponderada de las entradas y sus pesos sinápticos. Si esta suma supera un umbral predefinido, la neurona se activa y emite una señal de salida (1); de lo contrario, permanece inactiva (0).
- Composición en redes: El poder del modelo radica en su capacidad para conectarse en redes. Varias neuronas McCulloch-Pitts se pueden interconectar formando capas de procesamiento, lo que permite la creación de redes neuronales artificiales más complejas.

Algoritmo del Perceptron

El algoritmo del perceptrón está basado en en el modelo de para representar la neurona de McCulloch-Pitts. Los parámetros son un vector x de tamaño n que representa los datos (incluido 1), un vector w que representa los pesos (incluido el bias), b que representa el bias, un vector y que representa la respuesta real, una vector d que representa la respuesta deseada y η que representa el factor de aprendizaje.

- Se comienza por inicializar w en la posición 0 en 0. Después se entra en un bucle desde n=1 hasta el tamaño de las entradas.

- En la iteración actual se activa el perceptrón calculando en x y d en la posición n.
- Se calcula la respuesta real y se almacena en la posición n de y utilizando la función signum.

Se adapta w en la posición n+1 como w en posición n más η multiplicado por la suma de d en la posición n menos la y en la posición n multiplicado por el dato en x en n, o vector de pesos en posición n más el factor de aprendizaje multiplicado por la suma de la respuesta deseada en n menos la respuesta real en la posición n multiplicada por el dato entrada en la posición n. Esto donde +1 si x en n pertenece a la primera clase o -1 si x en n pertenece a la segunda clase.

- Finalmente se incrementa n en 1 y se repiten los pasos anteriores.

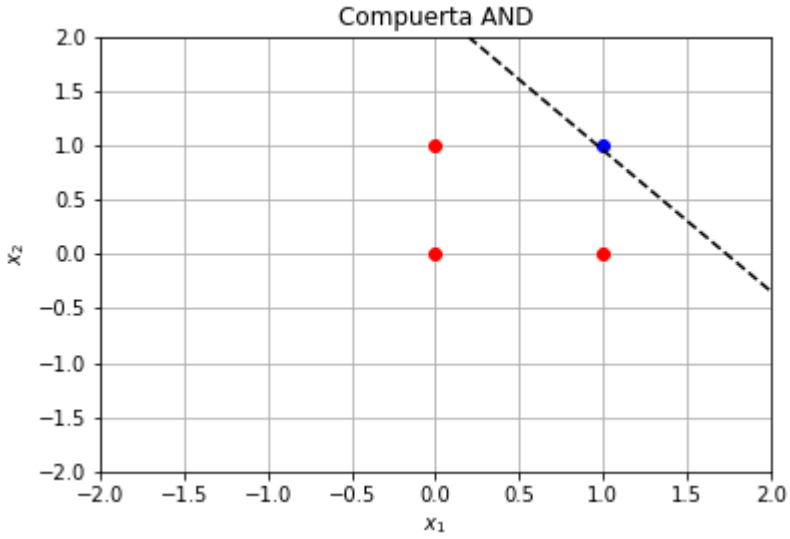
Codigo:

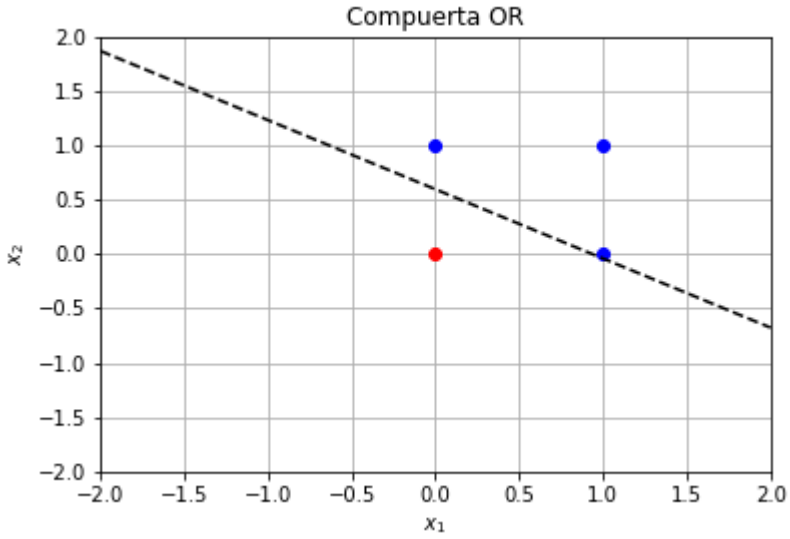
```
  import numpy as np

class Perceptron:
    def __init__(self, n_input, learning_rate):
        self.w = -1 + np.random.rand(n_input)
        self.b = -1 + 2 * np.random.rand()
        self.eta = learning_rate

    def predict(self, X):
        p = X.shape[1]
        Y_est = np.zeros(p)
        for i in range(p):
            Y_est[i] = np.dot(self.w, X[:,i]) + self.b
            if Y_est[i] >= 0:
                Y_est[i] = 1
            else:
                Y_est[i] = 0
        return Y_est

    def fit(self, X, Y, epochs=50):
        p = X.shape[1]
        for _ in range(epochs):
            for i in range(p):
                Y_est = self.predict(X[:,i].reshape(-1, 1))
                self.w += self.eta * (Y[i] - Y_est) * X[:,i]
                self.b += self.eta * (Y[i] - Y_est)
```

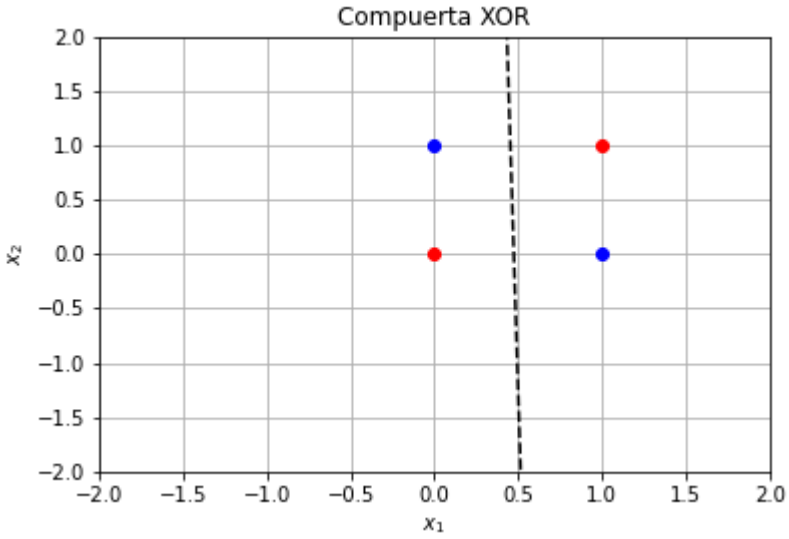
```
import numpy as np
import matplotlib
```

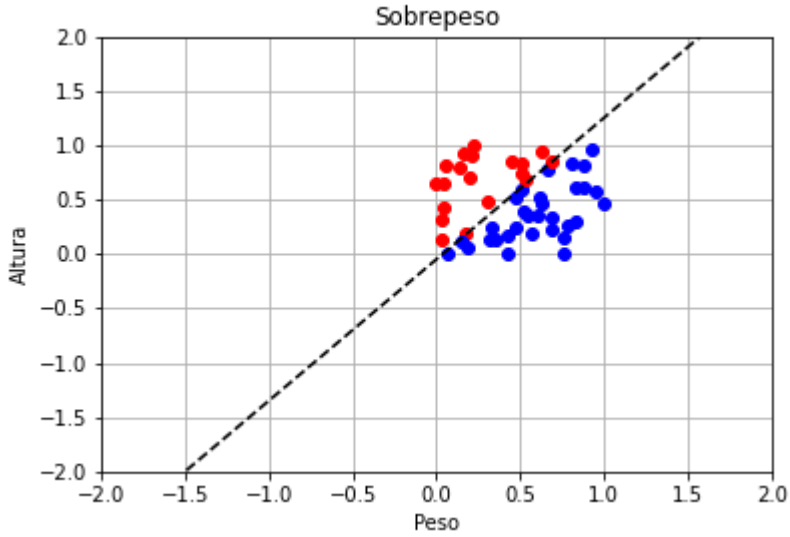
```python
import matplotlib.pyplot as plt
from perceptron import Perceptron

matplotlib.use('TkAgg')

def draw_2d(title, X, Y, model, xlabel = r'$x_1$', ylabel = r'$x_2$'):
    plt.title(title)
    plt.grid(True)
    plt.xlim([-2, 2])
    plt.ylim([-2, 2])
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)

    _, p = X.shape
    for i in range(p):
        if Y[i] == 0:
            plt.plot(X[0,i], X[1,i], 'or')
        else:
            plt.plot(X[0,i], X[1,i], 'ob')

    w1, w2, b = model.w[0], model.w[1], model.b
    li, ls = -2, 2
    plt.plot(
        [li, ls],
        [(1/w2) * (-w1*(li)-b), (1/w2)*(-w1*(ls)-b)],
        '--k'
    )


def main():
    neuron = Perceptron(2, 0.1)
    X = np.array([
        [0, 0, 1, 1],
        [0, 1, 0, 1]
    ])

    # Compuertas

    # Compuerta AND
    Y = np.array([0, 0, 0, 1])

    neuron.fit(X, Y)
    draw_2d('Compuerta AND', X, Y, neuron)
    plt.savefig('Compuerta AND') # plt.show()
```

```
    plt.cla()

    # Compuerta OR
    Y = np.array([0, 1, 1, 1])
    neuron.fit(X, Y)
    draw_2d('Compuerta OR', X, Y, neuron)
    plt.savefig('Compuerta OR') # plt.show()
    plt.cla()

    # Compuerta XOR
    Y = np.array([0, 1, 1, 0])
    neuron.fit(X, Y)
    draw_2d('Compuerta XOR', X, Y, neuron)
    plt.savefig('Compuerta XOR') # plt.show()
    plt.cla()

    # Sobrepeso

    # X = [Weights, Heights]
    X = np.array([
        np.random.default_rng().uniform(50, 150, 50), # Weight KGs
        np.random.default_rng().uniform(1.5, 2.3, 50) # Heigth Ms
    ])
    Y = (X[0] / X[1]**2) >= 25

    # Normalize weights (Min-max)
    X[0] = (X[0] - X[0].min()) / (X[0].max() - X[0].min())

    # Normalize Heights (Min-max)
    X[1] = (X[1] - X[1].min()) / (X[1].max() - X[1].min())

    neuron.fit(X, Y, 250)
    draw_2d('Sobrepeso', X, Y, neuron, 'Peso', 'Altura')
    plt.savefig('Sobrepeso')


if __name__ == '__main__':
    main()
```


Compuerta AND
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
x_2
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x_1


Compuerta OR
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
x_2
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x_1





¿Por que la compuerta XOR no puede ser aprendida por este modelo?
Porque no es linealmente separable, debido a los resultados que nos da la tabla de verdad de dicha compuerta, es decir tenemos dos puntos para cada tipo de clase, ambos tipos de clase tienen a su par de clase en extremo opuesto, por lo cual es imposible que pueda.

¿Que importancia tiene el factor de aprendizaje?
Este es el que controla la velocidad en que aprenda, ya que tenemos que ser cuidadosos con el como usamos los factores para que aprenda porque si lo hacemos muy lento tardara mucho tiempo y si los factores los hacemos muy rápido este no tendrá la precisión que deseamos que tenga y no aprendera bien nuestro algoritmo.