



En esta parte del código como se nos pidió. Experimentar con el Código proporcionado de la implementación de la compuerta OR en la presentación Clase 1.

 Generar

 Cerrar

```
import numpy as np

class Perceptron:
    def __init__(self, input_size, learning_rate=0.1, epochs=1000):
        self.W = np.zeros(input_size + 1) # +1 for the bias term
        self.learning_rate = learning_rate
        self.epochs = epochs

    def activation_function(self, x):
        return 1 if x >= 0 else 0

    def predict(self, x):
        x = np.insert(x, 0, 1) # Insert bias term
        z = self.W.T.dot(x)
        return self.activation_function(z)

    def train(self, X, y):
        for _ in range(self.epochs):
            for inputs, label in zip(X, y):
                prediction = self.predict(inputs)
                self.W += self.learning_rate * (label - prediction) * np.insert(inputs, 0, 1)

    def print_weights(self):
        print(f"Pesos después del entrenamiento: {self.W}")

# Datos de entrada y salidas deseadas para la compuerta OR
X = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])
y = np.array([0, 1, 1, 1])

# Crear el perceptron y entrenarlo
perceptron = Perceptron(input_size=2)
perceptron.train(X, y)

# Imprimir los pesos después del entrenamiento
perceptron.print_weights()

# Probar el perceptron
for inputs in X:
    print(f"Entrada: {inputs} > Salida predicha: {perceptron.predict(inputs)}")
```

```
➦ Pesos después del entrenamiento: [-0.1  0.1  0.1]
Entrada: [0 0] > Salida predicha: 0
Entrada: [0 1] > Salida predicha: 1
Entrada: [1 0] > Salida predicha: 1
Entrada: [1 1] > Salida predicha: 1
```

A continuación se implementa lo mismo pero con AND. realizar la implementación de la compuerta AND. Hacer la comprobación manual e identificar qué pasa si modificamos los hiperparametros de la red (learning rate y el número de épocas).

```

import numpy as np

class Perceptron:
    def __init__(self, input_size, learning_rate=0.1, epochs=1000):
        self.W = np.zeros(input_size + 1) # +1 for the bias term
        self.learning_rate = learning_rate
        self.epochs = epochs

    def activation_function(self, x):
        return 1 if x >= 0 else 0

    def predict(self, x):
        x = np.insert(x, 0, 1) # Insert bias term
        z = self.W.T.dot(x)
        return self.activation_function(z)

    def train(self, X, y):
        for _ in range(self.epochs):
            for inputs, label in zip(X, y):
                prediction = self.predict(inputs)
                self.W += self.learning_rate * (label - prediction) * np.insert(inputs, 0, 1)

    def print_weights(self):
        print(f"Pesos después del entrenamiento: {self.W}")

# Datos de entrada y salidas deseadas para la compuerta AND
X = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])
y = np.array([0, 0, 0, 1])

# Crear el perceptron y entrenarlo
perceptron = Perceptron(input_size=2)
perceptron.train(X, y)

# Imprimir los pesos después del entrenamiento
perceptron.print_weights()

# Probar el perceptron
for inputs in X:
    print(f"Entrada: {inputs} > Salida predicha: {perceptron.predict(inputs)}")

```

```

↔ Pesos después del entrenamiento: [-0.2  0.2  0.1]
Entrada: [0 0] > Salida predicha: 0
Entrada: [0 1] > Salida predicha: 0
Entrada: [1 0] > Salida predicha: 0
Entrada: [1 1] > Salida predicha: 1

```

```

# Datos de entrada y salidas deseadas para la compuerta AND
X = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])
y = np.array([0, 0, 0, 1])

```

Para la compuerta XOR

```

# Datos de entrada y salidas deseadas para la compuerta XOR
X = np.array([
    [0, 0],
    [0, 1],
    [1, 0],
    [1, 1]
])
y = np.array([0, 1, 1, 0])

```

un perceptrón simple puede resolver problemas linealmente separables como AND, pero no puede resolver problemas no linealmente separables como XOR. Para resolver XOR, es necesario una red neuronal con al menos una capa oculta. Como se explico en clase.

