26 DE FEBRERO DE 2024

# PERCEPTRON SIMPLE - XOR

## PRACTICA 1 – EJERCICIO 1

CRUZ GALLEGOS RAMSES AARÓN

SEM IA 2

MTRO: CAMPOS PEÑA DIEGO

**Introducción:**

El siguiente código implementa un perceptrón simple para la clasificación de patrones binarios. El perceptrón es un modelo básico de red neuronal con aplicaciones en problemas de clasificación lineal. En este caso, se utiliza para realizar la prueba de clasificación de patrones utilizando el conjunto de datos de entrada y salida XOR.

El programa incluye funciones para la lectura de datos desde archivos CSV, entrenamiento del perceptrón, prueba del modelo entrenado en un conjunto de datos de prueba, y la visualización gráfica de los patrones y la recta separadora resultante.

**Desarrollo:**

Las principales funcionalidades con las que cuenta el perceptrón son las siguientes:

1. **Funciones del Perceptrón:**
    - **perceptron_activation(summation):** En esta función, defino la activación del perceptrón, asignando 1 si la suma ponderada de las entradas supera un umbral y 0 en caso contrario.
    - **perceptron(inputs, weights, bias):** Aquí implemento el perceptrón utilizando la función de activación. Calculo la suma ponderada de las entradas, aplico la función de activación y obtengo la predicción binaria.
2. **Lectura de Datos:**
    - **read_data(file):** Para cargar datos desde los archivos CVS, utilizo esta función. Separo las entradas y salidas para su posterior uso.
3. **Entrenamiento del Perceptrón:**
    - **train_perceptron(inputs, outputs, learning_rate, max_epochs, convergence_criterion):** Aquí se entrena el perceptrón, ajustando los pesos y el sesgo.
4. **Prueba del Perceptrón:**
    - **test_perceptron(inputs, weights, bias):** En esta función, realiza la prueba del perceptrón en un conjunto de datos de entrada utilizando los pesos y el sesgo previamente entrenados.
5. **Cálculo de Precisión:**
    - **calculate_accuracy(outputs_real, outputs_predictions):** Para evaluar la precisión del perceptrón, se calcula la proporción de predicciones correctas en comparación con las salidas reales.
6. **Visualización Gráfica:**
    - **plot_graph(inputs, outputs, weights, bias):** Genero una gráfica que muestra visualmente los patrones de entrada, así como la recta separadora aprendida por el perceptrón.
7. **Main:**

    Aquí básicamente se hace uso de todas las funciones en conjunto:

    - Leo datos de entrenamiento y prueba desde archivos CSV.
    - Entreno el perceptrón y evalúo su rendimiento en el conjunto de datos de prueba.
    - Calculo y muestro la precisión del perceptrón.
    - Visualizo gráficamente los patrones y la recta separadora.

**Código:**

```
import numpy as np
import matplotlib.pyplot as plt

# PERCEPTRON
def perceptron_activation(summation):
    return 1 if summation >= 0 else 0

def perceptron(inputs, weights, bias):
    summation = np.dot(inputs, weights) + bias
    return perceptron_activation(summation)

# LEER CSV(EXCEL CON LOS DATOS)
def read_data(file):
    data = np.genfromtxt(file, delimiter=',')
    inputs = data[:, :-1]
    outputs = data[:, -1]
    return inputs, outputs

# ENTRENAMIENTO DEL PERCEPTRON
def train_perceptron(inputs, outputs, learning_rate, max_epochs,
convergence_criterion):
    num_inputs = inputs.shape[1]
    num_patterns = inputs.shape[0]

    weights = np.random.rand(num_inputs)
    bias = np.random.rand()
    epochs = 0
    convergence = False

    while epochs < max_epochs and not convergence:
        convergence = True
        for i in range(num_patterns):
            input_pattern = inputs[i]
            output_prediction = outputs[i]
            output_received = np.dot(weights, input_pattern) + bias
            error = output_prediction - output_received

            if abs(error) > convergence_criterion:
                convergence = False
                weights += learning_rate * error * input_pattern
                bias += learning_rate * error
        epochs += 1
    return weights, bias

# TEST
def test_perceptron(inputs, weights, bias):
    output_received = np.dot(inputs, weights) + bias
```

```
    return np.vectorize(perceptron_activation)(output_received)

# CALCULO DE PRECISIÓN
def calculate_accuracy(outputs_real, outputs_predictions):
    correct_predictions = np.sum(outputs_real == outputs_predictions)
    total_predictions = len(outputs_real)
    accuracy = correct_predictions / total_predictions
    return accuracy

def plot_graph(inputs, outputs, weights, bias):
    plt.figure(figsize=(8, 6))
    # GRAFICAR PATRONES
    plt.scatter(inputs[:, 0], inputs[:, 1], c=outputs, s=100)

    # GRAFICAR RECTA
    x_min, x_max = inputs[:, 0].min() - 1, inputs[:, 0].max() + 1
    y_min, y_max = inputs[:, 1].min() - 1, inputs[:, 1].max() + 1

    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.01), np.arange(y_min, y_max,
0.01))
    Z = test_perceptron(np.c_[xx.ravel(), yy.ravel()], weights, bias)
    Z = Z.reshape(xx.shape)

    plt.contour(xx, yy, Z, colors='k', linestyles=['-'], levels=[0])
    plt.title('Patrones y Recta Separadora')
    plt.xlabel('Entrada X1')
    plt.ylabel('Entrada X2')
    plt.grid(True)
    plt.show()

# MAIN
if __name__ == "__main__":
    training_file = 'XOR_trn.csv'
    test_file = 'XOR_tst.csv'

    inputs_train, outputs_train = read_data(training_file)
    inputs_test, outputs_test = read_data(test_file)

    # PARAMETROS
    max_epochs = 100
    learning_rate = 0.1
    convergence_criterion = 0.01  # ALTERACIONES ALEATORIAS 5%

    # ENTRENAMIENTO
    trained_weights, trained_bias = train_perceptron(inputs_train, outputs_train,
learning_rate,
    max_epochs, convergence_criterion)
    print("Perceptrón entrenado con éxito.")
    # PERCEPTRON CON DATOS DE ENTRADA
    outputs_predictions = test_perceptron(inputs_test, trained_weights, trained_bias)
```

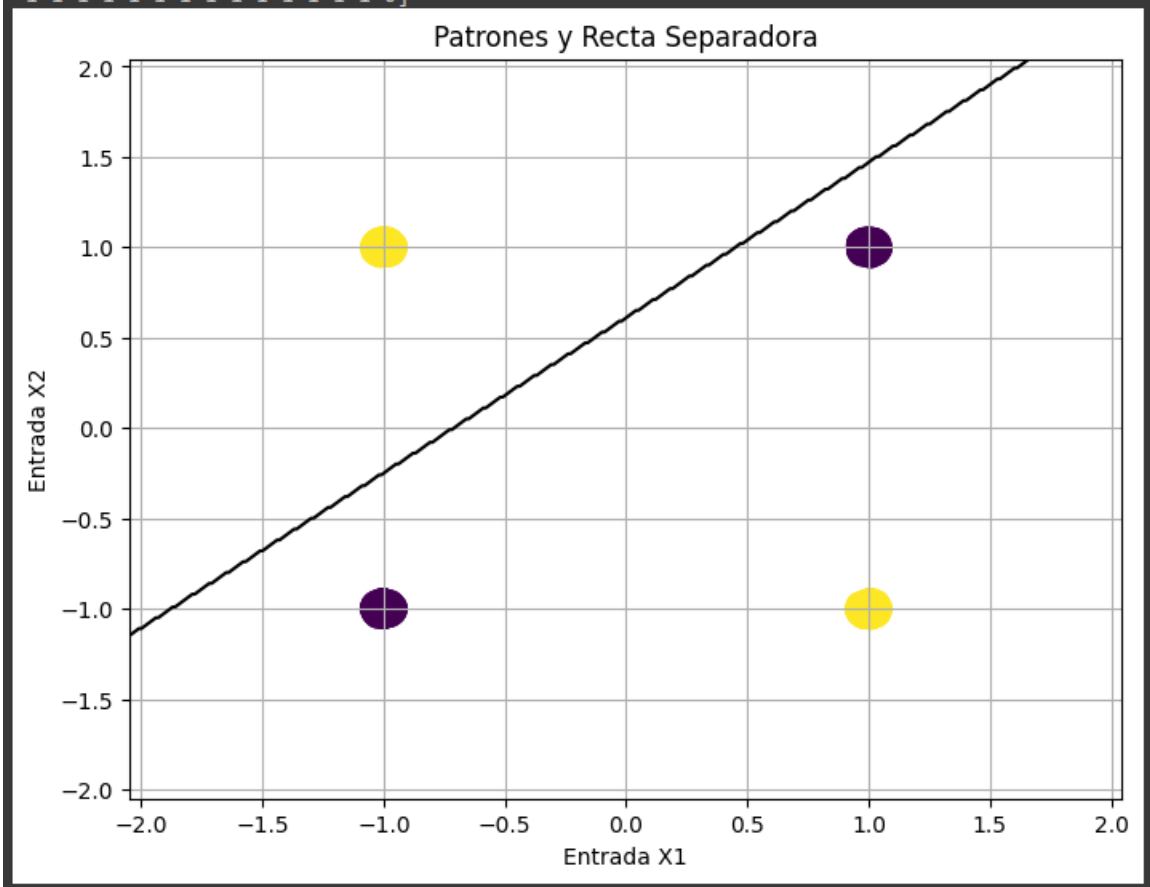
```
# SACAR PRESICIÓN
accuracy = calculate_accuracy(outputs_test, outputs_predictions)
print("Precisión del perceptrón en datos de prueba (Accuracy):", accuracy)

# RESULTADOS
print("Salidas en prueba:")
print(outputs_test)
print("Salidas predichas por el perceptrón:")
print(outputs_predictions)

plot_graph(inputs_train, outputs_train, trained_weights, trained_bias)
```

**Resultados:**

```
Perceptrón entrenado con éxito.
Precisión del perceptrón en datos de prueba (Accuracy): 0.18
Salidas en prueba:
[-1.  1. -1. -1. -1. -1.  1.  1. -1.  1. -1. -1.  1.  1.  1. -1. -1.  1.
  1. -1.  1. -1.  1. -1.  1.  1.  1. -1. -1. -1. -1.  1. -1.  1. -1. -1.
 -1.  1. -1. -1. -1.  1. -1. -1.  1.  1.  1.  1. -1. -1.  1. -1. -1. -1.
  1.  1.  1. -1. -1.  1. -1. -1. -1.  1. -1.  1. -1. -1. -1. -1.  1. -1.
 -1. -1.  1. -1.  1. -1.  1. -1. -1. -1.  1.  1. -1.  1.  1.  1. -1.  1.
 -1.  1.  1.  1. -1.  1.  1. -1. -1. -1. -1. -1.  1.  1.  1. -1. -1. -1.
  1. -1.  1. -1.  1. -1.  1.  1. -1. -1.  1.  1. -1. -1. -1.  1.  1.  1.
 -1.  1. -1.  1.  1. -1.  1.  1. -1.  1. -1. -1. -1. -1. -1.  1. -1. -1.
 -1.  1. -1. -1.  1. -1.  1.  1.  1.  1.  1. -1. -1.  1. -1.  1.  1. -1.
  1.  1. -1.  1.  1. -1.  1.  1. -1. -1.  1. -1. -1. -1. -1.  1. -1.  1.
 -1.  1.  1. -1. -1. -1. -1. -1.  1. -1. -1. -1. -1. -1. -1. -1. -1.  1.
 -1.  1.]
Salidas predichas por el perceptrón:
[1 0 1 1 1 1 0 0 1 0 1 1 0 0 0 1 1 0 1 1 0 1 0 1 0 0 0 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1
 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 0 1 1 1 1 0 1 0
 1 1 1 0 1 1 1 0 1 1 1 1 0 1 0 1 0 1 0 0 1 0 0 1 0 1 1 1 1 1 0 1 1 1 0 1 1
 0 1 0 1 1 0 1 1 1 0 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0]
```



**Conclusión:**

El programa presenta de forma gráfica el funcionamiento del perceptrón simple, en este caso hice pruebas XOR para verificar su funcionamiento y el porque este tipo de perceptrón no funciona para funciones que no sean lineales, por lo tanto para proyectos grandes este tipo de perceptrón no es muy útil,  como tal el perceptrón funciona de la manera esperada y logre cumplir el objetivo de la misma que es comprender y aplicar el perceptrón simple.

**Github:**

https://github.com/CRUZITO4O4/SEM-IA-2