Perceptrón Simple

En este notebook programaremos un perceptron simple utilizando numpy. El objetivo es que comprendamos el funcionamiento del perceptrón y que practiquemos la programación en Python. En la siguiente figura se encuentra una representación del perceptrón.

@juan1rving

```
import numpy as np
from IPython.display import Image
```

Calcular producto punto

El primer paso es calcular el logit, *h*, a partir del producto punto. La fórmula explícita es la siguiente:

$$h = WX + b$$

```
def transponer_matrix(matriz: np.ndarray) -> np.ndarray:
    """
    Transpone una matriz dada.

    Args:
        matriz (np.ndarray): Matriz a transponer. Puede ser un vector o una ma

    Returns:
        np.ndarray: Matriz transpuesta.
    """
    if matriz.ndim == 1:
        matriz_transpuesta = matriz.reshape(-1, 1)
    else:
        filas, columnas = matriz.shape
        matriz_transpuesta = np.zeros((columnas, filas), dtype=matriz.dtype)
        for i in range(filas):
            for j in range(columnas):
                matriz_transpuesta[j, i] = matriz[i, j]
    return matriz_transpuesta

def multiplicar_matrix(m1: np.ndarray, m2: np.ndarray) -> np.ndarray:
    """
    Multiplica dos matrices.

    Args:
        m1 (np.ndarray): Primera matriz.
        m2 (np.ndarray): Segunda matriz.
```

```
    Returns:
        np.ndarray: Resultado de la multiplicación de m1 por m2.

    Raises:
        AssertionError: Si las dimensiones de las matrices no son compatibles
    """
    if m1.ndim == 1:
        m1 = m1.reshape(1, -1)
    if m2.ndim == 1:
        m2 = m2.reshape(-1, 1)

    assert m1.shape[1] == m2.shape[0], "Dimensiones incompatibles."

    resultado = np.zeros((m1.shape[0], m2.shape[1]), dtype=np.result_type(m1,
    for i in range(m1.shape[0]):
        for j in range(m2.shape[1]):
            for k in range(m1.shape[1]):
                resultado[i, j] += m1[i, k] * m2[k, j]

    if resultado.shape == (1, 1):
        return resultado[0, 0]
    else:
        return resultado

def function_h(X: np.ndarray, W: np.ndarray, b: np.float64) -> np.float64:
    """
    Calcula la función h para el perceptrón, combinando inputs, pesos y sesgo.

    Args:
        X (np.ndarray): Vector de entrada.
        W (np.ndarray): Vector de pesos.
        b (float): Sesgo.

    Returns:
        np.ndarray: Resultado de aplicar la función h.
    """
    return multiplicar_matrix(np.append(W,b), transponer_matrix(np.append(X,1)
```

Función de activación

Para este ejemplo utilizaremos la función sigmoide como función de activación.

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

```
def aplicar_escalon(h: np.float64) -> int:
    """
    Aplica la función escalón a un valor dado.

    Args:
        h (float): Valor a evaluar con la función escalón.
```

```
    Returns:
        int: Resultado de aplicar la función escalón, 1 si h >= 0, de lo contr
    """
    if h >= 0 :
        return 1
    else :
        return 0

def perceptron(inputs: np.ndarray, weights: np.ndarray, bias: float) -> None:
    """
    Simula un perceptrón simple, realizando cálculos con inputs, pesos y sesgo

    Args:
        inputs (np.ndarray): Vector de entrada.
        weights (np.ndarray): Vector de pesos.
        bias (float): Sesgo.
    """
    print(f"Inputs: {tuple(list(inputs))}")
    print(f"Parámetros: {tuple(list(weights))}")
    h = function_h(inputs, weights, bias)
    print(f"Resultado de la neurona: {aplicar_escalon(h)}")
```

Probar inferencia

Ahora definamos unos pesos y veamos el resultado de una pasada frontal (fordward pass).

```
print("\t\t\t***PERCEPTRÓN SIMPLE***")
inputs = np.array([0.7 , -0.3 ])
weights = np.array([0.1 , 0.8 ])
bias = 0.5
perceptron(inputs, weights, bias)
```

```
                            ***PERCEPTRÓN SIMPLE***
    Inputs: (0.7, -0.3)
    Parámetros: (0.1, 0.8)
    Resultado de la neurona: 1
```

Perceptrón OR

```
print("\n\n\n\n\n\n\t\t\t***PERCEPTRÓN SIMPLE - OR***")
inputs = np.array([0, 0])
weights = np.array([1 ,1])
bias = -1
perceptron(inputs, weights, bias)
```

```
                        ***PERCEPTRÓN SIMPLE - OR***
Inputs: (0, 0)
Parámetros: (1, 1)
Resultado de la neurona: 0
```

Conclusiones

A través de esta práctica, hemos profundizado en la comprensión de los fundamentos del deep learning, específicamente en la operación y aplicación de un perceptrón simple. Implementamos funciones para realizar operaciones básicas de álgebra lineal, como la transposición y multiplicación de matrices, y las aplicamos en el contexto de un perceptrón para modelar el proceso de toma de decisiones lineales basado en entradas, pesos, y un sesgo. Esta aproximación nos ha permitido apreciar cómo un modelo tan elemental puede efectivamente realizar clasificaciones binarias, sirviendo como el bloque constructor para redes neuronales más complejas.