Práctica 9. Aplicando retropropagación en Python – parte 1.

Individual***

*Adjunta tu código completo agregando comentarios de lo que realiza cada parte****

La retropropagación (*backpropagation* en inglés) es un algoritmo fundamental en el entrenamiento de redes neuronales artificiales. Es utilizado para calcular los gradientes de la función de pérdida con respecto a los pesos de la red, lo que permite ajustar los pesos de manera que la red minimice la pérdida en el conjunto de datos de entrenamiento.

1. Captura y ejecuta el siguiente ejercicio que implementa la retropropagación en Python para entrenar una red neuronal utilizando bibliotecas como NumPy:

Es una red neuronal con una capa oculta que utiliza la función de activación sigmoide.

Durante el entrenamiento, se calcula el error mediante la diferencia entre la salida real y la salida predicha. Luego, se utiliza la **retropropagación** para **ajustar los pesos y los sesgos** de la red con el fin de **minimizar el error**. Este **proceso** se repite durante un número fijo de épocas o hasta que se alcance un cierto criterio de convergencia.

```
import numpy as np


# Función de activación sigmoide
def sigmoid(x):
    return 1 / (1 + np.exp(-x))


# Derivada de la función de activación sigmoide
def sigmoid_derivative(x):
    return x * (1 - x)



# Datos de entrada y salida
X = np.array([[0, 0],
```

```
        [0, 1],
        [1, 0],
        [1, 1]])

y = np.array([[0],
        [1],
        [1],
        [0]])

# Inicialización de pesos y bias
np.random.seed(1)
input_neurons = 2
hidden_neurons = 3
output_neurons = 1

weights_input_hidden = np.random.uniform(size=(input_neurons, hidden_neurons))
weights_hidden_output = np.random.uniform(size=(hidden_neurons, output_neurons))
```

```
bias_hidden = np.random.uniform(size=(1, hidden_neurons))
bias_output = np.random.uniform(size=(1, output_neurons))

# Hiperparámetros
epochs = 10000
learning_rate = 0.1

# Entrenamiento
for epoch in range(epochs):
    # Feedforward
    input_hidden = np.dot(X, weights_input_hidden) + bias_hidden
    output_hidden = sigmoid(input_hidden)

    input_output = np.dot(output_hidden, weights_hidden_output) + bias_output
    output = sigmoid(input_output)
```

```
    # Retropropagación
    # Calcular el error
    error = y - output

    # Calcular los deltas y ajustar los pesos
    delta_output = error * sigmoid_derivative(output)
    error_hidden = delta_output.dot(weights_hidden_output.T)
    delta_hidden = error_hidden * sigmoid_derivative(output_hidden)

    # Actualizar pesos y bias
    weights_hidden_output += output_hidden.T.dot(delta_output) * learning_rate
    bias_output += np.sum(delta_output, axis=0, keepdims=True) * learning_rate
    weights_input_hidden += X.T.dot(delta_hidden) * learning_rate
    bias_hidden += np.sum(delta_hidden, axis=0, keepdims=True) * learning_rate

# Resultados
print("Resultado después del entrenamiento:")
print(output)
```

***Imprime los resultados obtenidos.

*Adjunta tu código completo agregando comentarios de lo que realiza cada parte****

Anota tus conclusiones.