Practica 1. Ejercicio 3

Cruz Gallegos Ramsés Aarón

Morales Zepeda Ivan Yutlanih

Sem. de Sol. de Problemas de Inteligencia Artificial II

Sección: D05

Índice

Introducción

El algoritmo de retropropagación es una técnica para el entrenamiento de redes neuronales que permite ajustar los pesos de las conexiones entre las neuronas para minimizar el error entre las salidas deseadas y las salidas producidas por la red. Consiste en propagar el error desde la capa de salida hacia atrás a través de la red, ajustando los pesos en función de la contribución de cada neurona al error total.

Desarrollo

Se desarrolló una implementación de un Perceptrón Multicapa utilizando el lenguaje de programación Python. Algunas partes clave del desarrollo son:

- **Definición de funciones de activación:** Las funciones de activación son fundamentales en las redes neuronales para introducir no linealidad en el modelo. En este caso, se implementan la función sigmoidal y su derivada, que se utilizan en la capa de salida y en el cálculo de los gradientes durante la retropropagación. Para la modificación a la retropropagación, se optó por probar el algoritmo con una función de activación diferente, ReLU.

```
# Definición de funciones de activación sigmoid
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# Definición de funciones de activación relu
def relu(x):
    return np.maximum(0, x)

def relu_derivative(x):
    return np.where(x <= 0, 0, 1)
```

- **Definición de la clase NeuralNetwork**: Una clase que representa la red neuronal, con métodos para inicializar los pesos y sesgos, propagar hacia adelante, retropropagar el error y entrenar la red.

```
class NeuralNetwork:
    def __init__(self, layer_sizes):
        self.layer_sizes = layer_sizes
        self.num_layers = len(layer_sizes)

        # Inicialización aleatoria de pesos y sesgos
        self.weights = [np.random.rand(layer_sizes[i],   layer_sizes[i+1]) for i in range(self.num_layers - 1)]
        self.biases = [np.random.rand(1, layer_sizes[i+1]) for i in range(self.num_layers - 1)]

    def forward(self, X):
        # Propagación hacia adelante
        self.activations = [X]
        for i in range(self.num_layers - 1):
            weighted_input = np.dot(self.activations[-1], self.weights[i]) + self.biases[i]
            self.activations.append(relu(weighted_input))
        return self.activations[-1]

    def backward(self, X, y, output, learning_rate):
        # Retropropagación
        error = y - output
        deltas = [error * relu_derivative(output)]

        for i in range(self.num_layers - 2, 0, -1):
            error = deltas[-1].dot(self.weights[i].T)
            deltas.append(error *relu_derivative(self.activations[i]))
        deltas.reverse()

        # Actualización de pesos y sesgos
        for i in range(self.num_layers - 1):
            self.weights[i] += self.activations[i].T.dot(deltas[i]) * learning_rate
            self.biases[i] += np.sum(deltas[i], axis=0) * learning_rate

    def train(self, X, y, epochs, learning_rate):
        for epoch in range(epochs):
            output = self.forward(X)
            self.backward(X, y, output, learning_rate)

    def predict(self, X):
        return np.round(self.forward(X))
```

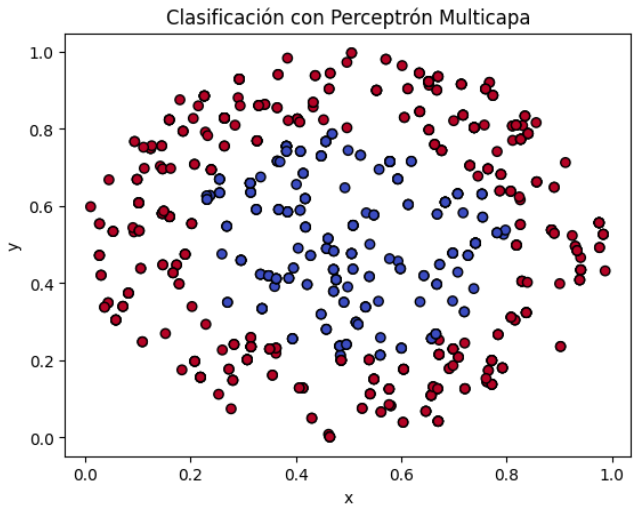
- **Entrenamiento de la red:** Se carga el conjunto de datos y se define la arquitectura de la red neuronal. Luego, se crea una instancia del modelo y se entrena utilizando el algoritmo de retropropagación durante un número determinado de épocas.

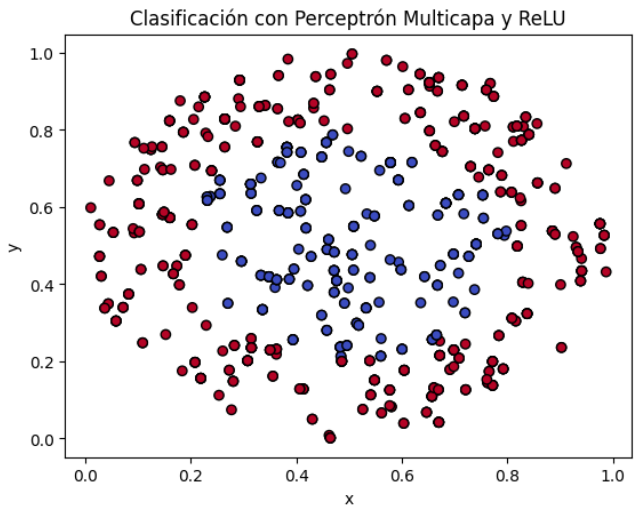
```
def train(self, X, y, epochs, learning_rate):
        for epoch in range(epochs):
            output = self.forward(X)
            self.backward(X, y, output, learning_rate)
```

Durante el entrenamiento, se propagan los datos hacia adelante a través de la red neuronal y se calculan las predicciones. Luego, se retropropaga el error para ajustar los pesos y sesgos en función de la diferencia entre las predicciones y los valores reales.

Resultados

Se obtuvieron resultados satisfactorios en la clasificación de datos utilizando el Perceptrón Multicapa y la técnica de retropropagación. Sin embargo, no se observaron cambios significativos en las gráficas al cambiar la función de activación sigmoidal por la función de activación ReLU. Es posible que la diferencia no sea lo suficientemente significativa como para alterar la apariencia de la frontera de decisión en el espacio de características.





Conclusiones

La implementación del Perceptrón Multicapa utilizando el algoritmo de retropropagación resultó satisfactoria. Aunque en este caso no se observaron diferencias significativas entre el uso de funciones de activación sigmoidal y ReLU, consideramos que es importante experimentar con diferentes arquitecturas de red y funciones de activación para encontrar el mejor rendimiento del modelo y obtener los mejores resultados. Finalmente, concluimos que la retropropagación es una técnica poderosa para la clasificación de datos.