Practica 1 Ejercicio 3
Implementación del Algoritmo de Retropropagación para un Perceptrón Multicapa

---

Universidad de Guadalajara, Centro Universitario de Ciencias exactas e Ingenierías.

Alumno:
Cárdenas Pérez Calvin Cristopher Código: 214798706

Carrera:
Ingeniería En Computación (INCO)
Materia:
Seminario de Solución de Problemas de Inteligencia artificial II
Maestro:
Dr. Diego Oliva
Ciclo:
2024 A
Sección:
D05
Fecha de Entrega:
lunes 15 de abril de 2024

Introducción

En el ámbito del aprendizaje automático, el algoritmo de retropropagación para un perceptrón multicapa es fundamental para entrenar redes neuronales artificiales y realizar clasificaciones precisas en conjuntos de datos complejos. En este informe, exploraremos la implementación y análisis de este algoritmo en Python. Comenzaremos por comprender los fundamentos teóricos del perceptrón multicapa y la retropropagación, para luego presentar su implementación práctica y analizar su desempeño en la clasificación de datos reales. Este análisis nos proporcionará una comprensión más profunda de cómo el algoritmo de retropropagación influye en el rendimiento del perceptrón multicapa y su aplicabilidad en diferentes escenarios de aprendizaje automático.

Desarrollo

Importation de bibliotecas

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

numpy se utiliza para realizar operaciones numéricas eficientes.
pandas se utiliza para cargar y manipular conjuntos de datos, en este caso, para cargar el conjunto de datos desde un archivo CSV.
matplotlib.pyplot se utiliza para graficar los datos y visualizar los resultados.

Definición de la función de activación sigmoidal

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

Esta función toma un valor numérico y devuelve su valor sigmoidal calculado utilizando la fórmula que se muestra.

Definición de la derivada de la función de activación sigmoidal

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

Esta función toma un valor numérico que ya ha pasado por la función sigmoidal y devuelve su derivada, que se calcula como $x \times (1-x)$.

Definición de la clase NeuralNetwork

```
class NeuralNetwork:
    def __init__(self, layers):
```

```
        self.layers = layers
        self.weights = [np.random.randn(layers[i], layers[i+1]) for i in
range(len(layers)-1)]
        self.biases = [np.random.randn(1, layers[i+1]) for i in
range(len(layers)-1)]

    def feedforward(self, inputs):
        self.activations = [inputs]
        for i in range(len(self.layers)-1):
            inputs = sigmoid(np.dot(inputs, self.weights[i]) +
self.biases[i])
            self.activations.append(inputs)
        return inputs

    def backpropagation(self, inputs, targets, learning_rate):
        output = self.feedforward(inputs)[-1]
        error = targets - output
        deltas = [error * sigmoid_derivative(output)]

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

        deltas.reverse()

        for i in range(len(self.layers)-1):
            self.weights[i] += self.activations[i].T.dot(deltas[i]) *
learning_rate
            self.biases[i] += np.sum(deltas[i], axis=0, keepdims=True) *
learning_rate
```

Esta clase define una red neuronal multicapa. Contiene métodos para inicializar los pesos y sesgos de la red, realizar el paso de avance (feedforward) y el paso de retropropagación (backpropagation) para entrenar la red.

Carga de los datos

```
data = pd.read_csv('concentlite.csv', header=None)
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values.reshape(-1, 1)
```

En esta parte del código, cargamos el conjunto de datos desde un archivo CSV llamado concentlite.csv. Se espera que este archivo contenga datos en formato tabular, donde la última columna sea la etiqueta de clase y las columnas anteriores sean características.

Definición de la arquitectura de la red neuronal

```
input_size = X.shape[1]
hidden_layers = [5, 5]  # Por ejemplo, dos capas ocultas con 5 neuronas cada una
output_size = 1
layers = [input_size] + hidden_layers + [output_size]
```

Esta sección define la arquitectura de la red neuronal, especificando el número de neuronas en cada capa. En este ejemplo, se define una red con una capa de entrada, dos capas ocultas con 5 neuronas cada una y una capa de salida.

Creación y entrenamiento de la red neuronal

```
# Crear la red neuronal
nn = NeuralNetwork(layers)

# Entrenamiento de la red
epochs = 1000
learning_rate = 0.1

for epoch in range(epochs):
    nn.backpropagation(X, y, learning_rate)
```

Aquí creamos una instancia de la clase NeuralNetwork con la arquitectura definida anteriormente.

En este bucle, entrenamos la red neuronal utilizando el algoritmo de retropropagación durante un número fijo de épocas. En cada época, llamamos al método backpropagation para actualizar los pesos y sesgos de la red en función de los datos de entrada y las etiquetas de clase proporcionadas.

Clasificación de los datos:

```
predictions = np.round(nn.feedforward(X))
```

Utilizamos la red neuronal entrenada para realizar predicciones sobre los datos de entrada X.

Separación de los datos según la clase

```
class_0 = X[y.flatten() == -1]
class_1 = X[y.flatten() == 1]
```

En esta parte, separamos los datos de entrada en dos conjuntos basados en las etiquetas de clase.

Graficación de los resultados

```
plt.plot(class_0[:, 0], class_0[:, 1], 'rs', markerfacecolor='none',
markersize=6)
plt.plot(class_1[:, 0], class_1[:, 1], 'kx', markersize=6)
```

Utilizamos matplotlib para graficar los datos. Las clases son representadas por 'X' negras y cuadrados rojos sin relleno en el gráfico.

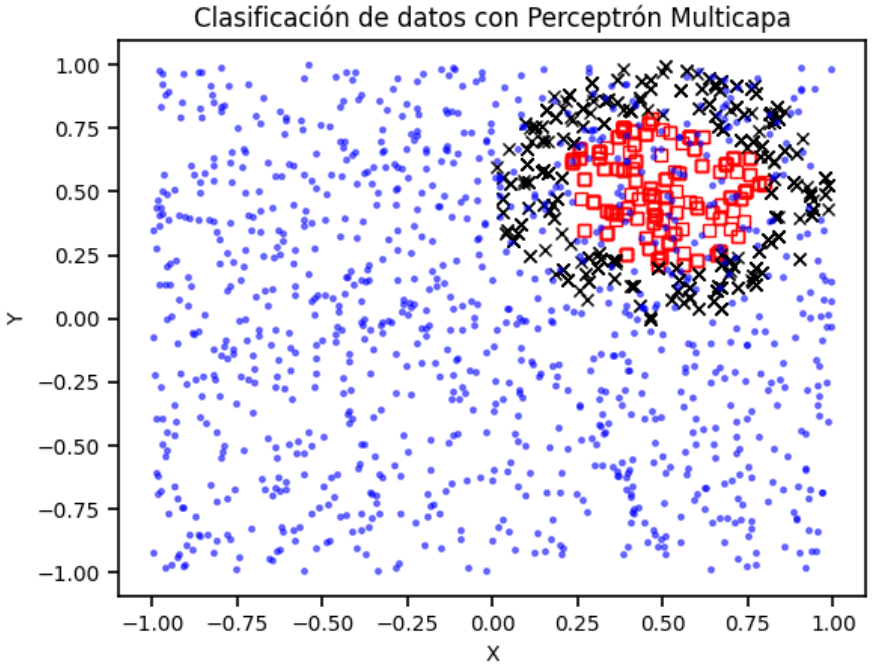
Generación de puntos aleatorios y su Graficación

```
random_points = np.random.rand(1000, 2) * 2 - 1  # Genera puntos
aleatorios en el rango [-1, 1]
plt.plot(random_points[:, 0], random_points[:, 1], 'b.', alpha=0.5,
markersize=5)
```

Generamos 1000 puntos aleatorios en el rango [-1, 1] y los graficamos en azul como puntos con transparencia.

Resultados

Por último tenemos los resultados, como se muestra tenemos los datos del csv en x negras y cuadros rojos están en pequeño ya que se incluye también los 1000 puntos que se piden



Conclusión

La implementación del algoritmo de retropropagación para un perceptrón multicapa fue emocionante y aprendí algo diferente que eso es lo que me gusta a final de cuentas de las materias de la universidad.

A través de esta práctica, pude comprender mejor el funcionamiento interno de las redes neuronales y su capacidad para realizar clasificaciones en conjuntos de datos complejos. A pesar de los obstáculos encontrados, la experiencia me permitió fortalecer mis habilidades en programación y adquirir un conocimiento más profundo sobre el aprendizaje automático.
Ojalá que esto no se complique tanto y me gustaría seguir aprendiendo de este campo y aplicar lo aprendido en proyectos futuros.