Universidad de Guadalajara
Centro Universitario de Ciencias Exactas e Ingenierías
Seminario de Solución de Problemas de Inteligencia Artificial II
Dr. Diego Oliva
Depto. De Ciencias Computacionales
Practica 1
Perceptrón Simple y Multicapa

Ejercicio 3: perceptrón multicapa

Mercado Manzo Luis Alfonso

Código:212559704

Objetivo:

Implementar el algoritmo de retropropagación para un perceptrón multicapa de forma que se puedan elegir libremente la cantidad de capas de la red y la cantidad de neuronas para cada capa.

1. Para entrenar y probar el algoritmo se debe usar el dataset concentlite.csv, el cual contiene dos clases distribuidas de forma concéntrica (Figura 2). Debe representarse gráficamente con diferentes colores el resultado de la clasificación hecha por el perceptrón multicapa.

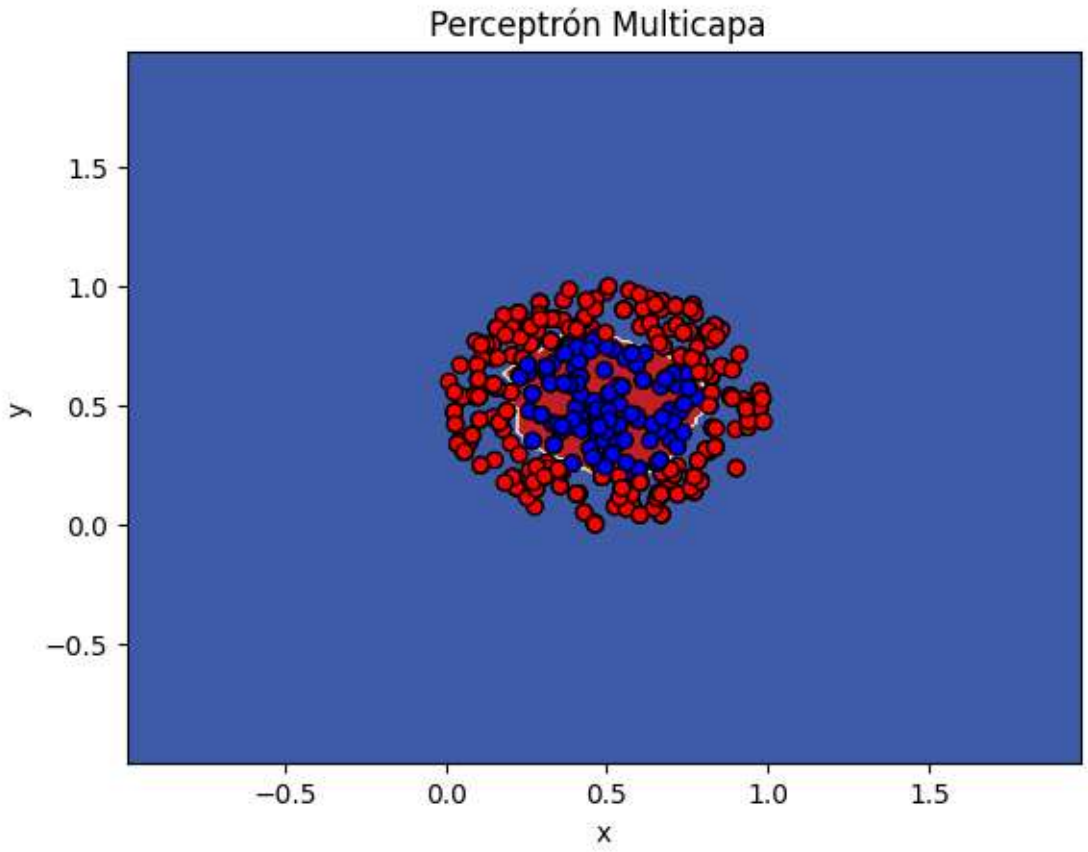
2. Probar otra regla se aprendizaje o alguna modificación a la retropropagación

Codigo:

```
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 12 21:20:18 2023

@author: luis mercado
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier

# Lectura del archivo de datos
datos = pd.read_csv('concentlite.csv', header=None)

# Extracción de las entradas y las salidas de los datos
entradas = datos.iloc[:, :-1].values
salidas = datos.iloc[:, -1].values

# Definición de parámetros
num_capas = 5  # Número de capas ocultas (ajusta según tus
necesidades)
neuronas_por_capa = [10, 5, 3]  # Número de neuronas para cada capa
oculta (ajusta según tus necesidades)
tasa_aprendizaje = 0.01  # Tasa de aprendizaje (ajusta según tus
necesidades)
```

```
max_iter = 5000  # Número máximo de iteraciones (ajusta según tus
necesidades)

# Creación del perceptrón multicapa
mlp = MLPClassifier(hidden_layer_sizes=tuple(neuronas_por_capa),
learning_rate_init=tasa_aprendizaje, max_iter=max_iter)

# División del conjunto de datos en entrenamiento y prueba
entradas_entrenamiento, entradas_prueba, salidas_entrenamiento,
salidas_prueba = train_test_split(
    entradas, salidas, test_size=0.1, random_state=42
)

# Entrenamiento del perceptrón multicapa
mlp.fit(entradas_entrenamiento, salidas_entrenamiento)

# Evaluación del perceptrón multicapa en los datos de prueba
porcentaje_acierto = mlp.score(entradas_prueba, salidas_prueba)
print("Porcentaje de acierto en los datos de prueba:",
porcentaje_acierto * 100, "%")

# Clasificación de todos los puntos del plano para visualizar la
superficie de decisión
x_min, x_max = entradas[:, 0].min() - 1, entradas[:, 0].max() + 1
y_min, y_max = entradas[:, 1].min() - 1, entradas[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min,
y_max, 0.02))
Z = mlp.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Gráfico de la superficie de decisión y los puntos de datos
plt.figure()
plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu)
plt.scatter(entradas[:, 0], entradas[:, 1], c=salidas, edgecolors='k',
cmap=plt.cm.bwr)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Perceptrón Multicapa ')
plt.show()
```

resultados:



Porcentaje de acierto en los datos de prueba: 98.80952380952381 %

Conclusión:

La retropropagación es un componente fundamental en el entrenamiento de redes neuronales y ha sido clave para el resurgimiento del interés en el aprendizaje profundo (deep learning) en las últimas décadas. Permite a las redes neuronales aprender representaciones jerárquicas de datos y es la base de muchas aplicaciones exitosas en campos como el procesamiento de imágenes, el procesamiento de lenguaje natural y más.