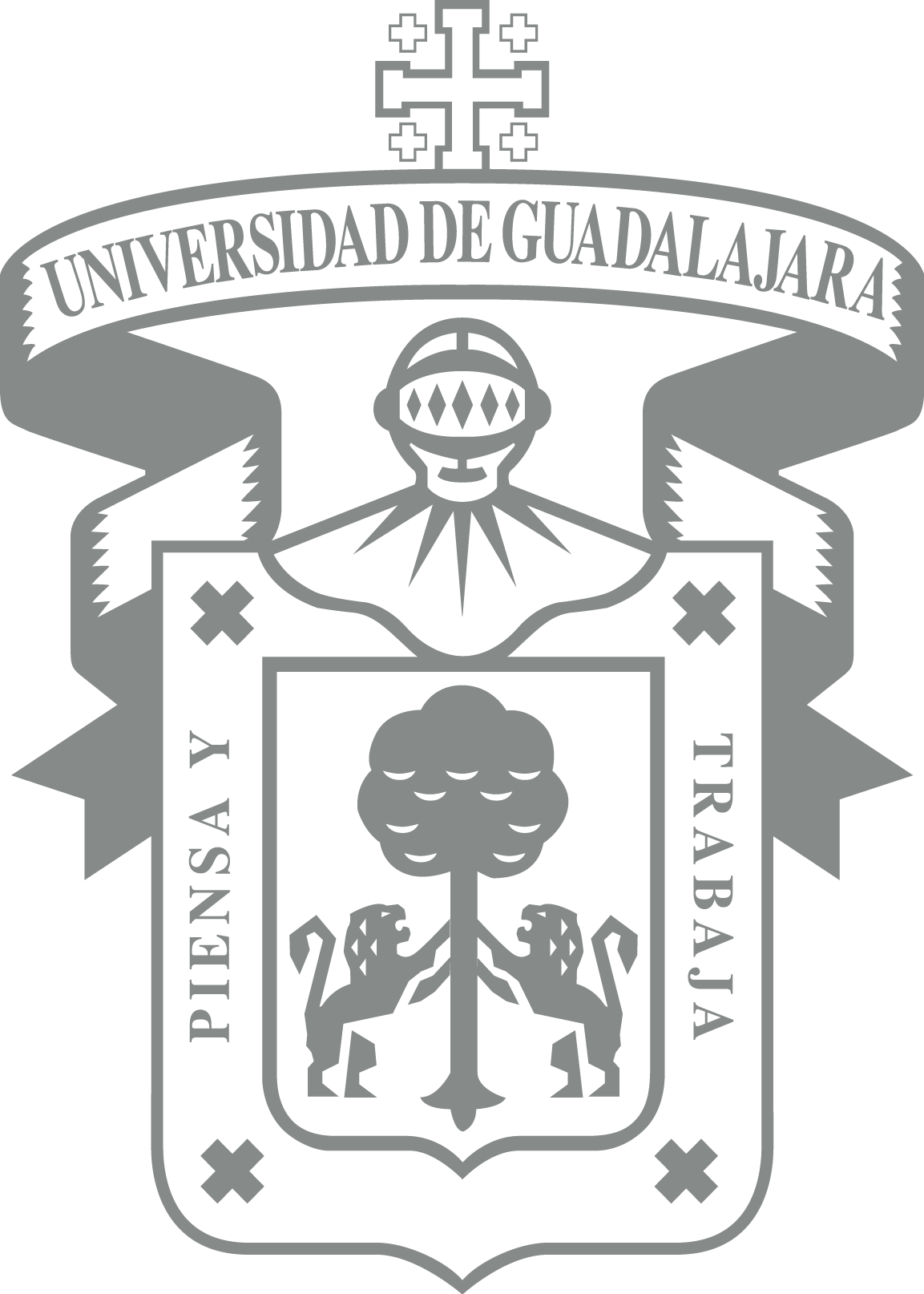
**Universidad de Guadalajara  
Centro Universitario de Ciencias e Ingenierías**



Departamento de Ciencias Computacionales  
Seminario de Solución de Problemas de Inteligencia Artificial ll

Profesor: OLIVA NAVARRO, DIEGO ALBERTO  
Alumno: Torres Hernández David

Código: 215428899 Carrera: INCO Sección: D05 Fecha: 15/04/2024

**Practica 1. Ejercicio 3**

# **Introducción**

Implementar el algoritmo de retro propagación 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 retro propagación.

Gráfico, Gráfico de dispersión

Descripción generada automáticamente

# **Codigo**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# 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)

# Clase Perceptrón Multicapa

class MLP:

    def \_\_init\_\_(self, layers):

        self.layers = layers

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

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

    def feed\_forward(self, inputs):

        activations = [inputs]

        for i in range(len(self.weights)):

            inputs = sigmoid(np.dot(inputs, self.weights[i]) + self.biases[i])

            activations.append(inputs)

        return activations

    def backpropagation(self, inputs, targets, learning\_rate):

        activations = self.feed\_forward(inputs)

        errors = [targets - activations[-1]]

        for i in range(len(self.weights) - 1, 0, -1):

            errors.insert(0, np.dot(errors[0], self.weights[i].T))

        for i in range(len(self.weights)):

            self.weights[i] += learning\_rate \* np.dot(activations[i].T.reshape(-1,1), errors[i] \* sigmoid\_derivative(activations[i+1]))

            self.biases[i] += learning\_rate \* np.sum(errors[i] \* sigmoid\_derivative(activations[i+1]), axis=0)

    def train(self, inputs, targets, epochs, learning\_rate):

        for \_ in range(epochs):

            for i in range(len(inputs)):

                self.backpropagation(inputs[i], targets[i], learning\_rate)

    def predict(self, inputs):

        return np.round(self.feed\_forward(inputs)[-1])

# Cargar datos

data = pd.read\_csv("concentlite.csv")

X = data.iloc[:, :-1].values  # Todas las columnas excepto la última

y = data.iloc[:, -1].values.reshape(-1, 1)  # Última columna como etiquetas

# Definir arquitectura de la red

layers = [2, 5, 1]  # Por ejemplo, 2 neuronas en la capa de entrada, 5 en la capa oculta y 1 en la capa de salida

# Inicializar y entrenar el modelo

mlp = MLP(layers)

mlp.train(X, y, epochs=1000, learning\_rate=0.1)

# Predicción

predictions = mlp.predict(X)

# Definir colores personalizados para los puntos

# rojo para la clase 0 y azul para la clase 1

colors = ['red' if p == 0 else 'blue' for p in predictions.flatten()]

# Graficar resultados con colores personalizados

plt.scatter(X[:,0], X[:,1], c=colors)

plt.title('Clasificación del dataset concentlite')

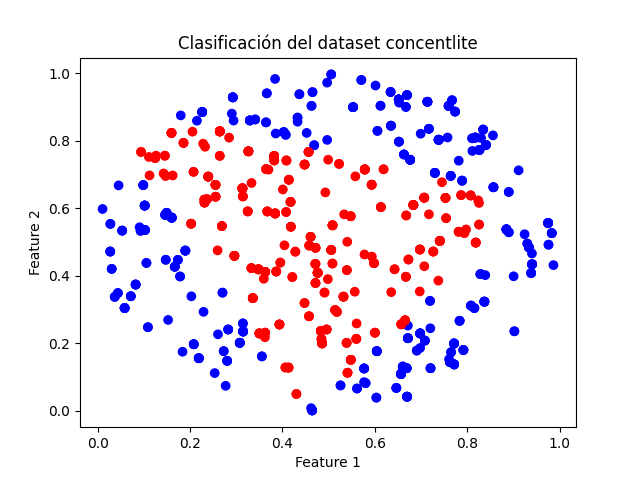
plt.xlabel('Feature 1')

plt.ylabel('Feature 2')

plt.show()

Principio del formulario

Capturas:

Gráfico, Gráfico de dispersión

Descripción generada automáticamenteGráfico, Gráfico de dispersión

Descripción generada automáticamente