CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS (CUCEI)

DIVISIÓN DE ELECTRÓNICA Y COMPUTACIÓN

DEPARTAMENTO DE CIENCIAS COMPUTACIONALES

Carrera: Ingeniería en Computación

Nombre Materia: Seminario de Solución de Problemas de IA ll

Profesor: Valdez López Julio Esteban

SECCIÓN: D02

Nombre alumno: López Arellano Ricardo David

CODIGO: 217136143

Practica 4

Fecha de entrega: 11/10/2023

CÓDIGO :

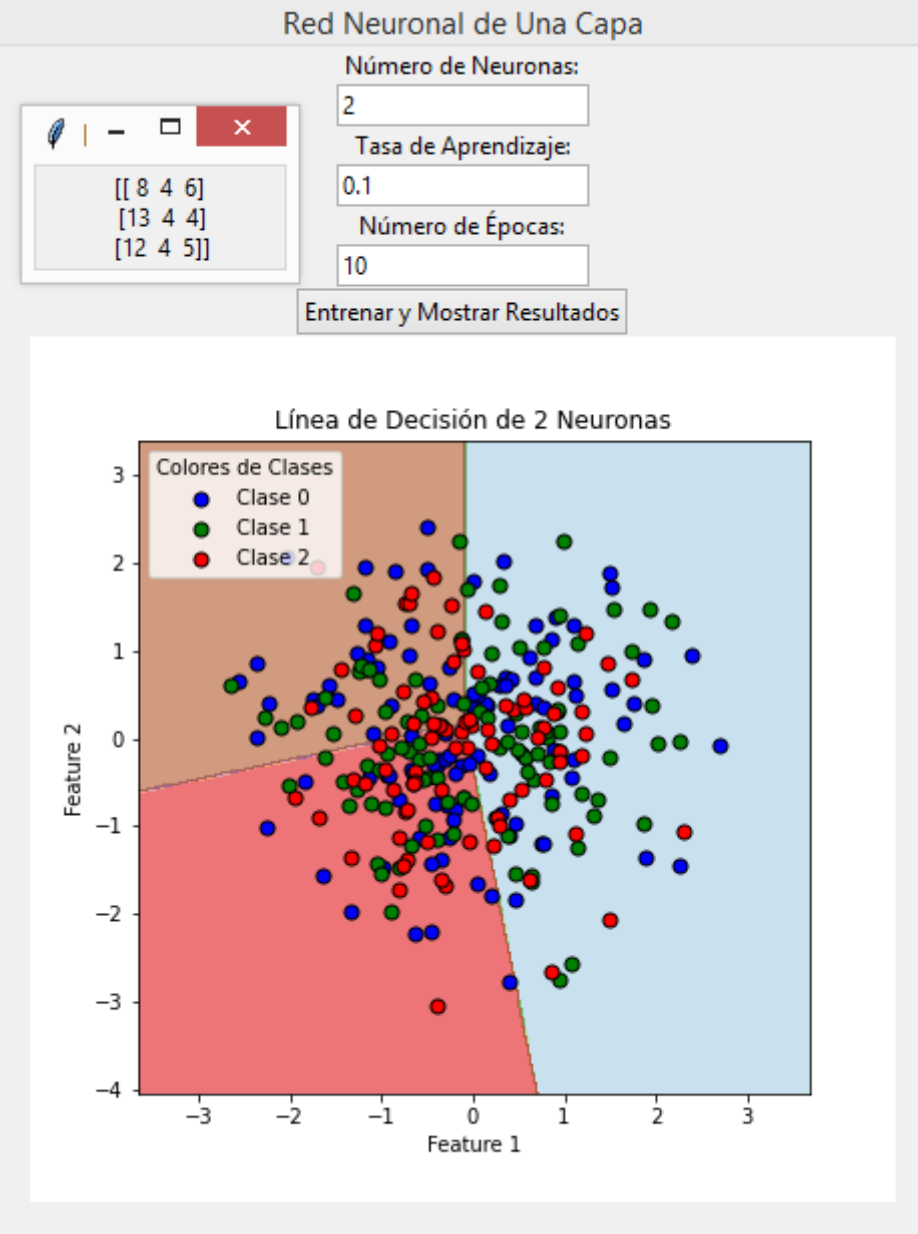
```
import tkinter as tk
from tkinter import ttk
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

# Crear una ventana tkinter
window = tk.Tk()
window.title("Red Neuronal de Una Capa")

# Etiqueta y entrada para el número de neuronas
neuron_label = ttk.Label(window, text="Número de Neuronas:")
neuron_label.pack()
neuron_entry = ttk.Entry(window)
neuron_entry.pack()

# Etiqueta y entrada para la tasa de aprendizaje
learning_rate_label = ttk.Label(window, text="Tasa de Aprendizaje:")
learning_rate_label.pack()
learning_rate_entry = ttk.Entry(window)
learning_rate_entry.pack()

# Etiqueta y entrada para el número de épocas
epochs_label = ttk.Label(window, text="Número de Épocas:")
epochs_label.pack()
epochs_entry = ttk.Entry(window)
epochs_entry.pack()

# Función para entrenar la red neuronal y mostrar resultados
def train_and_display_results():
    try:
        num_neurons = int(neuron_entry.get())
        learning_rate = float(learning_rate_entry.get())
        num_epochs = int(epochs_entry.get())

        # Generar datos de ejemplo
        X, y = generate_data()

        # Dividir los datos en conjuntos de entrenamiento y prueba
        X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

        # Entrenar la red neuronal
        weights, bias = train_single_layer_nn(X_train, y_train,
num_neurons, learning_rate, num_epochs)
```

```
        # Realizar predicciones en el conjunto de prueba
        y_pred = predict(X_test, weights, bias)

        # Calcular la matriz de confusión
        cm = confusion_matrix(y_test, y_pred)

        # Mostrar la matriz de confusión en una ventana emergente
        show_confusion_matrix(cm)

        # Mostrar la línea de decisión de la neurona en un gráfico
        plot_decision_boundary(X, y, weights, bias, num_neurons)

    except ValueError:
        tk.messagebox.showerror("Error", "Ingrese valores válidos.")

# Botón para entrenar la red neuronal y mostrar resultados
train_button = ttk.Button(window, text="Entrenar y Mostrar Resultados",
command=train_and_display_results)
train_button.pack()

# Función para generar datos de ejemplo con tres clases
def generate_data():
    np.random.seed(0)
    X = np.random.randn(300, 2)
    y = np.random.randint(0, 3, size=300)  # Tres clases
    return X, y

# Función para entrenar la red neuronal de una sola capa con tres clases
def train_single_layer_nn(X, y, num_neurons, learning_rate, num_epochs):
    num_samples, num_features = X.shape
    num_classes = len(np.unique(y))
    weights = np.random.randn(num_features, num_neurons, num_classes)
    bias = np.zeros((1, num_neurons, num_classes))

    for epoch in range(num_epochs):
        for c in range(num_classes):
            y_c = (y == c).astype(int)  # One-hot encoding
            # Forward propagation
            z = np.dot(X, weights[:, :, c]) + bias[:, :, c]
            a = 1 / (1 + np.exp(-z))

            # Backpropagation
            dz = a - y_c.reshape(-1, 1)
            dw = np.dot(X.T, dz) / num_samples
            db = np.sum(dz, axis=0) / num_samples

            # Actualizar parámetros
            weights[:, :, c] -= learning_rate * dw
            bias[:, :, c] -= learning_rate * db
```

```
    return weights, bias

# Función para hacer predicciones
def predict(X, weights, bias):
    num_classes = weights.shape[2]
    scores = np.zeros((X.shape[0], num_classes))

    for c in range(num_classes):
        z = np.dot(X, weights[:, :, c]) + bias[:, :, c]
        a = 1 / (1 + np.exp(-z))
        scores[:, c] = a[:, 0]

    y_pred = np.argmax(scores, axis=1)
    return y_pred

# Función para mostrar la matriz de confusión en una ventana emergente
def show_confusion_matrix(cm):
    cm_window = tk.Toplevel(window)
    cm_window.title("Matriz de Confusión")
    cm_label = tk.Label(cm_window, text=str(cm))
    cm_label.pack()

# Función para graficar la línea de decisión con puntos destacados y
leyenda de colores
def plot_decision_boundary(X, y, weights, bias, num_neurons):
    plt.figure(figsize=(6, 6))

    h = .02  # Tamaño de paso en la malla
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min,
y_max, h))

    Z = predict(np.c_[xx.ravel(), yy.ravel()], weights, bias)

    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.6)
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.title(f'Línea de Decisión de {num_neurons} Neuronas')

    # Colores para los puntos
    colors = ['b', 'g', 'r']  # Puedes personalizar los colores
    labels = ['Clase 0', 'Clase 1', 'Clase 2']  # Etiquetas de las clases

    for i in range(len(np.unique(y))):
        plt.scatter(X[y == i, 0], X[y == i, 1], color=colors[i], s=50,
edgecolor='k', label=labels[i])
```

```
    # Leyenda de colores
    plt.legend(title='Colores de Clases', loc='upper left')

    canvas = FigureCanvasTkAgg(plt.gcf(), master=window)
    canvas.get_tk_widget().pack()
    canvas.draw()

# Ejecutar la ventana tkinter
window.mainloop()
```

Capturas de pantalla:

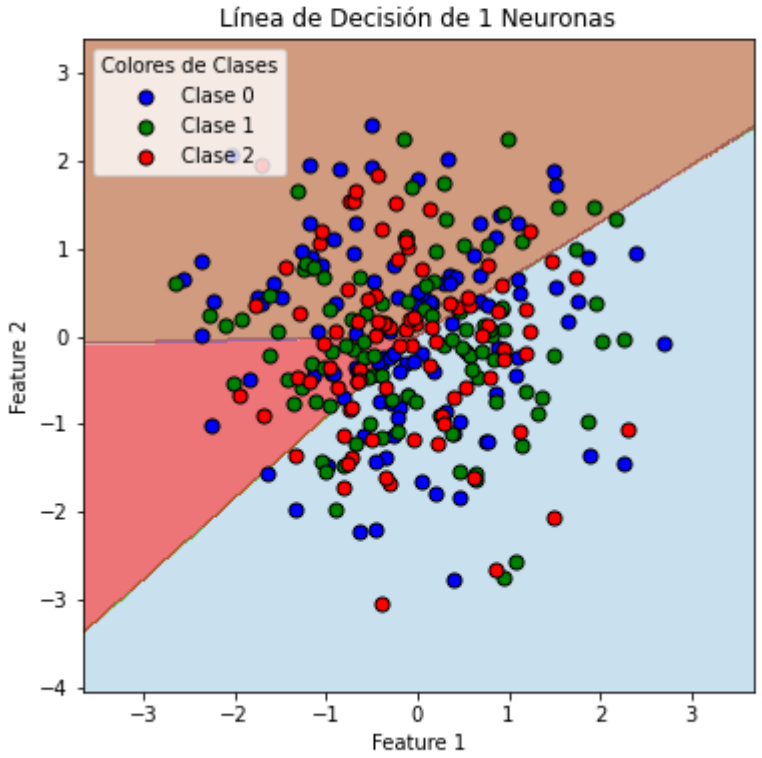


El recuadro pequeño de arriba es la matriz de confusión.

Red Neuronal de Una Capa

[[9 0 9]
[10 2 9]
[8 1 12]]

Número de Neuronas:
1
Tasa de Aprendizaje:
0.001
Número de Épocas:
100
Entrenar y Mostrar Resultados



Red Neuronal de Una Capa

[[12 6 0]
[17 4 0]
[18 3 0]]

Número de Neuronas:
3
Tasa de Aprendizaje:
1
Número de Épocas:
1000
Entrenar y Mostrar Resultados