
**Actividad 4 Perceptrón multicapa**

**Alumno: José Osvaldo Farfán de León**

**Codigo: 214796622**

**Materia: IA ll**

**Profesor: Carlos Alberto Villaseñor Padilla**

**Sección: “D03”**

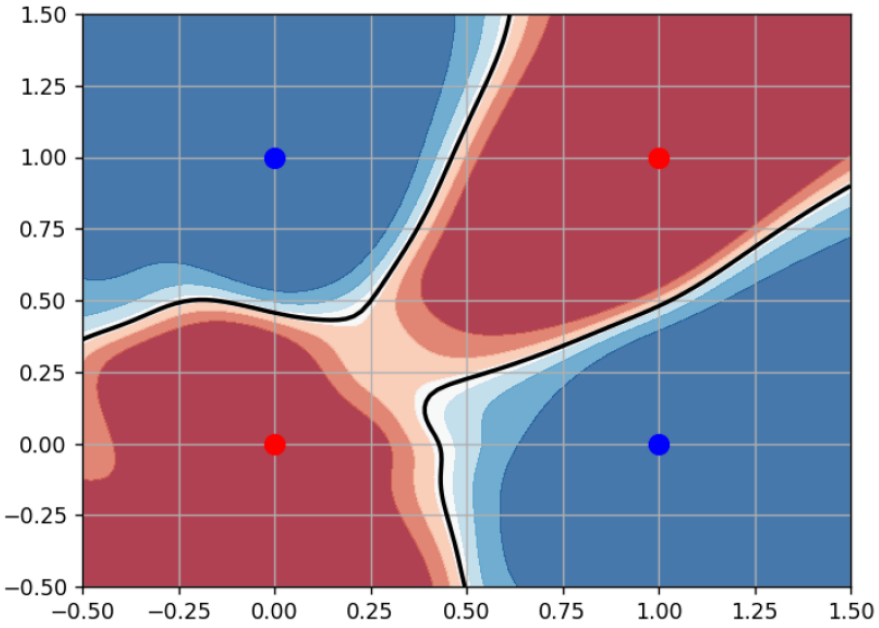
Para el conjunto de datos siguiente usa el código generado en clase para clasificar los datos correctamente. Gráfica los datos como se vio en clase para verificar tu arquitectura.

```
from activations import *
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

class MLP:
    def __init__(self, layers_dims,
                 hidden_activation=tanh,
                 output_activation=logistic):
        #Attributes
        self.L = len(layers_dims) - 1
        self.w = [None] * (self.L + 1)
        self.b = [None] * (self.L + 1)
        self.f = [None] * (self.L + 1)

        for l in range(1, self.L+1):
            self.w[l] = -1 + 2 * np.random.rand(layers_dims[l],
                                                layers_dims[l-1])
            self.b[l] = -1 + 2 * np.random.rand(layers_dims[l], 1)
            if l == self.L:
                self.f[l] = output_activation
            else:
                self.f[l] = hidden_activation

    def predict(self, X):
        A = X
        for l in range(1, self.L + 1):
            Z = self.w[l] @ A + self.b[l]
            A = self.f[l](Z)
        return A

    def fit(self, X, Y, epochs=500, lr=0.1):
        p = X.shape[1]
        for _ in range(epochs):
            # Initialize containers
            A = [None] * (self.L + 1)
            dA = [None] * (self.L + 1)
            lg = [None] * (self.L + 1)
            #Propagation ---------------------------------
            A[0] = X
            for l in range(1, self.L + 1):
                Z = self.w[l] @ A[l-1] + self.b[l]
                A[l], dA[l] = self.f[l](Z, derivative=True)

            #Backpropagation -------------------
            for l in range(self.L, 0, -1):
                if l == self.L:
                    lg[l] = (Y - A[l]) * dA[l]
                else:
                    lg[l] = (self.w[l+1].T @ lg[l+1]) * dA[l]

            #Weight and bias update ---------------------------
            for l in range(1, self.L+ 1):
                self.w[l] += (lr/p) * (lg[l] @ A[l-1].T)
                self.b[l] += (lr/p) * np.sum(lg[l])
```

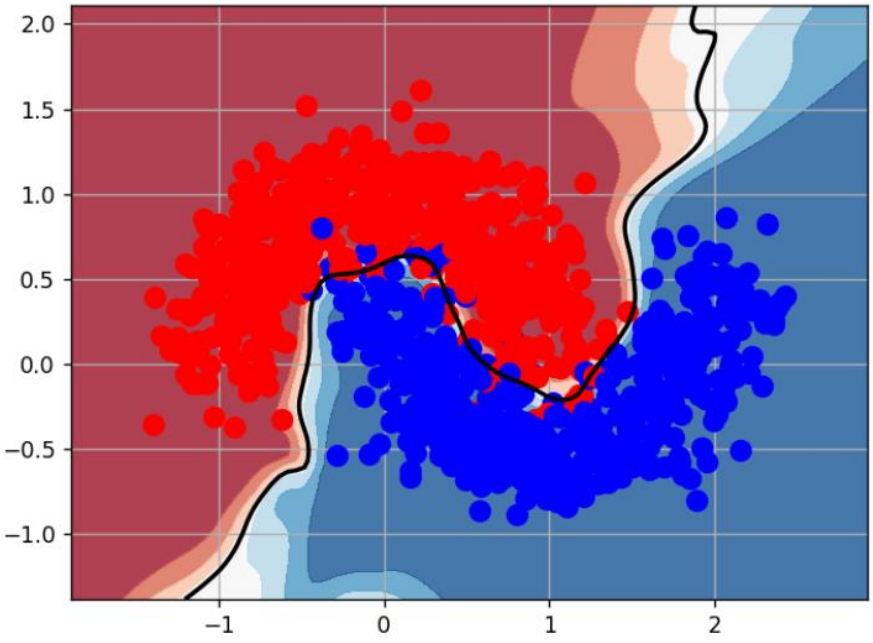
```python
def MLP_binary_class_2d(X, Y, net):
    plt.figure()
    for i in range(X.shape[1]):
        if Y[0, i] == 0:
            plt.plot(X[0,i], X[1,i], 'ro', markersize=9)
        else:
            plt.plot(X[0,i], X[1, i], 'bo', markersize=9)
    xmin, ymin = np.min(X[0,:])-0.5, np.min(X[1,:]) - 0.5
    xmax, ymax = np.max(X[0,:])+0.5, np.max(X[1,:]) + 0.5
    xx, yy = np.meshgrid(np.linspace(xmin, xmax, 100),
                         np.linspace(ymin, ymax, 100))
    data = [xx.ravel(), yy.ravel()]
    zz = net.predict(data)
    zz = zz.reshape(xx.shape)
    plt.contour(xx,yy,zz, [0.5], colors='k', linestyle='--', linewidths=2)
    plt.contourf(xx,yy,zz, alpha=0.8, cmap=plt.cm.RdBu)
    plt.xlim([xmin, xmax])
    plt.ylim([ymin, ymax])
    plt.grid()
    plt.show()
```

```python
#XOR
file = 'XOR.csv'

#moonsS
# file = 'moons.csv'

#blobs
# file = 'blobs.csv'

#circles
# file = 'circles.csv'

df = pd.read_csv(file)

X = np.array([df['x1'].values, df['x2'].values])
Y = np.array([df['y'].values])



net = MLP((2,100,50,30,1))
net.fit(X, Y)
print(net.predict(X))
MLP_binary_class_2d(X, Y, net)
```

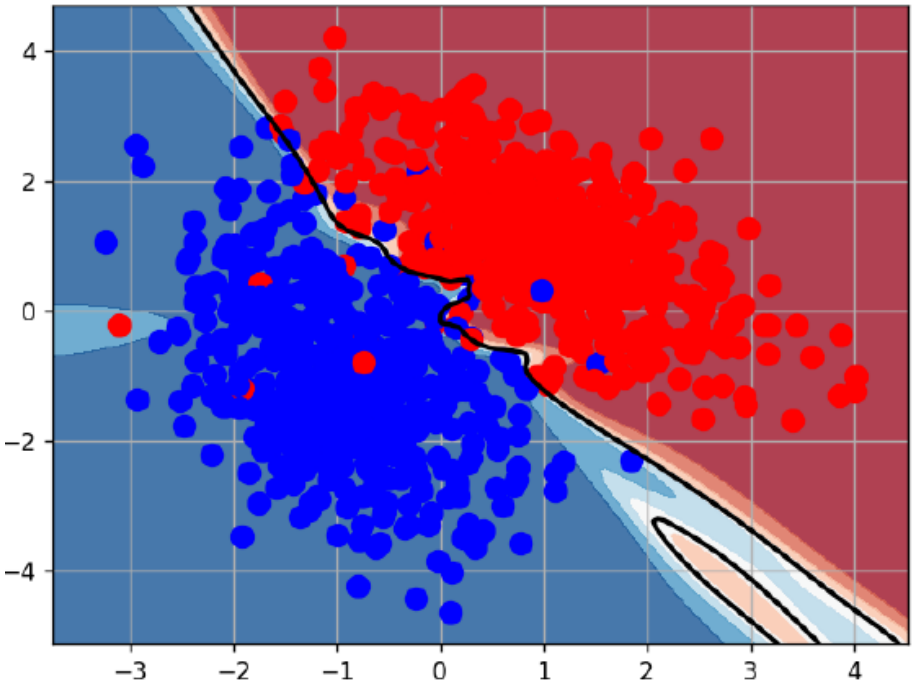
XOR


1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
1.50

MOONS


2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1
0
1
2

BLOBS




CIRCLES