Runtime / Change Runtime Type / GPU

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_gaussian\_quantiles

#CREAR VALORES ALEATORIOS

N=1000

gaussian\_quantiles=make\_gaussian\_quantiles(mean=None,

                    cov=0.1,

                    n\_samples = N,

                    n\_features = 2,

                    n\_classes = 2,

                    shuffle = True,

                    random\_state=None

             )

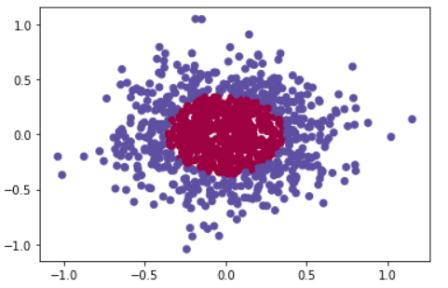
X,Y = gaussian\_quantiles

X.shape  #(1000,2) 2 features

Y.shape  #(1000,)

Y= Y[:,np.newaxis]

plt.scatter(X[:,0], X[:,1], c=Y[:,0], cmap=plt.cm.Spectral)



Valores de -1, 1

Que red neural detecto la clase del centro y la exterior

FUNCIONES

#FUNCION ACTIVACION SIGMOID, CON SU DERIVADA

def sigmoid(x, derivate = False):

    if derivate:

        return np.exp(-x)/(( np.exp(-x) +1)\*\*2)

    else:

        return 1 / (1 + np.exp(-x))

#FUNCION ACTIVACION RELU, CON SU DERIVADA

def relu(x, derivate = False):

    if derivate:

        x[x <= 0] = 0

        x[x > 0 ] = 1

        return x

    else:

        return np.maximum(0, x)

#FUNCION DE PERDIDA MSE, CON SU DERIVADA

def mse(y, y\_hat, derivate = False):

    if derivate:

        return (y\_hat - y)

    else:

        return np.mean((y\_hat - y)\*\*2)

ESTRUCTURA DE RED, SUS PESOS Y BIAS

def initialize\_parameters\_deep(layers):

    parameters = {}

    L = len(layers)

    for l in range(0,L-1):

        #np.random.rand(10,3) 10x3 numeros de 0-1

**Rand1 = np.random.rand( layers[l], layers[l+1])**

**Rand2 = np.random.rand( 1, layers[l+1])**

        #PESO W0 = Rand(Capas , 1)   W1 = Rand(Capas, 2)  ... W9 = Rand(Capas, 10)

        parameters['W' + str(l+1)] = (Rand1 \* 2) -1

        # BIAS b0=Rand(1,1)  b1=Rand(1,2)... b9=Rand(1,10)

        parameters['b' + str(l+1)] = (Rand2 \* 2) -1

    return parameters

capas=[2,4,8,1]

params = initialize\_parameters\_deep(capas)

params

OUTPUT

{'W1': array([[ 0.51341609, -0.32344696, 0.17906696, 0.51893131],

[-0.01576041, -0.83575883, -0.3785694 , 0.2146555 ]]),

'W2': array([[-0.66552311, -0.51020037, 0.19650767, 0.64775838, -0.51583197,

0.70873147, -0.31181365, 0.33386521],

[-0.42514933, 0.23822352, -0.5048245 , -0.05213759, -0.94212684,

-0.1274211 , 0.78998783, -0.40597563],

[-0.14165635, 0.8810552 , 0.80015575, 0.30594527, -0.1261536 ,

-0.50107521, -0.11410558, -0.17319435],

[ 0.36392107, 0.55692844, 0.78517082, 0.40034523, 0.14745944,

0.18968421, 0.30897946, -0.31757332]]),

'W3': array([[-0.81252073],

[-0.99330012],

[-0.12161478],

[-0.96696329],

[ 0.2736685 ],

[ 0.89157228],

[-0.678955 ],

[ 0.6962366 ]]),

'b1': array([[ 0.23815174, -0.31086849, -0.99076906, 0.95719333]]),

'b2': array([[-0.50217654, 0.28183487, -0.10243542, -0.67834142, -0.6163654 ,

-0.20368903, 0.7336713 , -0.23134855]]),

'b3': array([[0.82989528]])}

# w1[2x4] w2[4x8] w3[8x1]

# b1[2x4] b2[4x8] b3[8x1]

**#FORWARD**

def trainData(X,params,lr,training=True):

**#FORWARD**

params['A0'] = X # entrada de los valores

**#El producto punto de los parametros con el peso  mas el bias**

params['Z1']= np.matmul(params['A0'], params['W1']) + params['b1']

params['A1'] =relu(params['Z1'])

# Recibi el analisis de la anterior neurona

params['Z2']= np.matmul(params['A1'], params['W2']) + params['b2']

params['A2'] =relu(params['Z2'])

#El producto punto de los parametros con el peso  mas el vayas

params['Z3']= np.matmul(params['A2'], params['W3']) + params['b3']

params['A3'] = sigmoid(params['Z3'])

output = params['A3']

#params['A0'].shape #1000,2

#params['W1'].shape #2,4

#params['b1'].shape #1,4

#X.shape #1000,2

#y.shape #1000,1

    if training:

        #BACKPROPAGATION

#producto de Y, derivada X parámetro con derivada de sigmoid

        params['dZ3'] = mse(Y,output,True) \* sigmoid(params['A3'],True)

        params['dW3'] = np.matmul(params['A2'].T,params['dZ3'])

        params['dZ2'] = np.matmul(params['dZ3'], params['W3'].T) \* relu(params['A2'],True)

        params['dW2'] = np.matmul(params['A1'].T,params['dZ2'])

        params['dZ1'] = np.matmul(params['dZ2'], params['W2'].T) \* relu(params['A1'],True)

        params['dW1'] = np.matmul(params['A0'].T,params['dZ1'])

        #Gradiente descenso

        params['W3'] = params['W3'] - params['dW3'] \* 0.0001

        params['b3'] = params['b3'] - (np.mean(params['dW3'],axis=0,keepdims=True)) \* lr

        params['W2'] = params['W2'] - params['dW2'] \* 0.0001

        params['b2'] = params['b2'] - (np.mean(params['dW2'],axis=0,keepdims=True)) \* lr

        params['W1'] = params['W1'] - params['dW1'] \* 0.0001

        params['b1'] = params['b1'] - (np.mean(params['dW1'],axis=0,keepdims=True)) \* lr

    return output

capas=[2,4,8,1]

params = initialize\_parameters\_deep(capas)

#params

errors=[]

for \_ in range(60000):

    output  = trainData(X, params, 0.0001)

    if \_ % 25 ==0: #cada 25 iteraciones imprima y guarde

        #print(mse(Y,output))

        errors.append(mse(Y,output))

plt.plot(errors)