RED NEURONAL MULTIETIQUETA

By: Jared Isaías Monje Flores

4 DE MARZO DE 2024

CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS
UdeG

¿Qué es una red neuronal muticapa?

Una red neuronal multicapa (MLP, por sus siglas en inglés Multilayer Perceptron) es un tipo de red neuronal artificial que consiste en múltiples capas de neuronas interconectadas. Estas redes están compuestas por una capa de entrada, una o más capas ocultas y una capa de salida. Cada capa, excepto la de entrada, está compuesta por un conjunto de neuronas (también llamadas nodos o unidades) que están conectadas a las neuronas de la capa anterior y posterior mediante conexiones ponderadas.

La estructura de capas múltiples permite a las MLP aprender representaciones de datos complejas y no lineales mediante la combinación de transformaciones lineales y no lineales. Las neuronas en cada capa oculta calculan una combinación lineal de las salidas de las neuronas en la capa anterior, seguida de una función de activación no lineal. Estas funciones de activación introducen no linealidades en el modelo, lo que permite a la red aprender y modelar relaciones más complejas en los datos.

Las redes neuronales multicapa se utilizan en una amplia gama de aplicaciones de aprendizaje automático y reconocimiento de patrones, incluyendo clasificación, regresión, reconocimiento de voz, procesamiento de imágenes, procesamiento de lenguaje natural, entre otros. Su capacidad para modelar relaciones complejas en los datos y su flexibilidad las convierten en una herramienta poderosa para resolver una variedad de problemas de aprendizaje automático.

Instrucciones:

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.

RESULTADOS OBTENIDOS:

XOR

Valores de ajuste

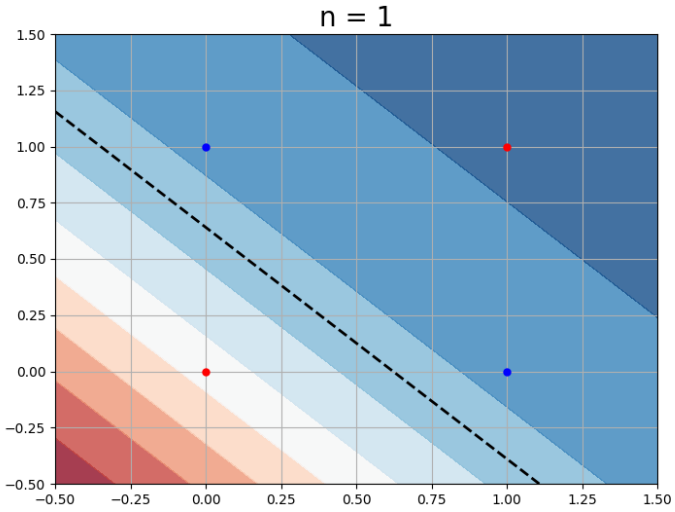
Entradas: 2

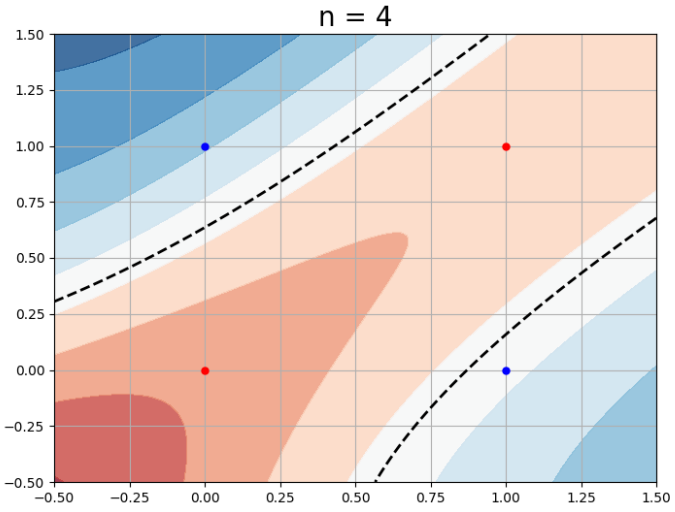
Salidas: 1

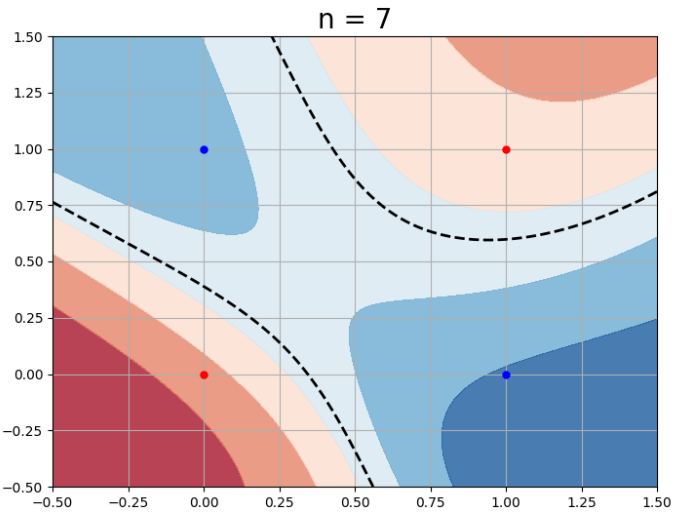
Función de activación: tanh

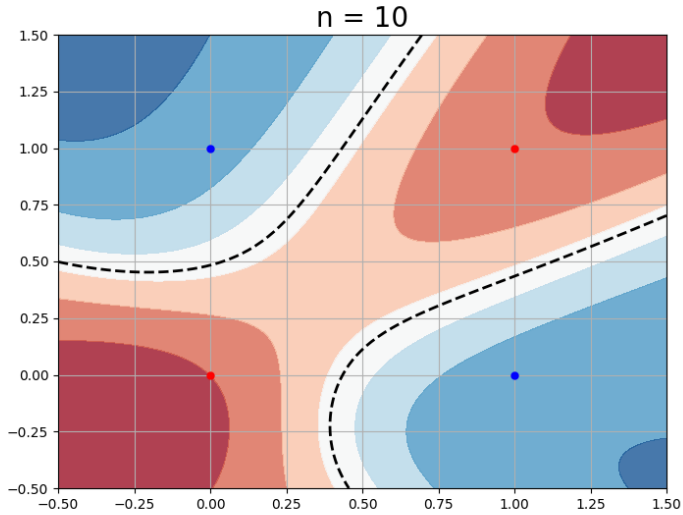
Función de activación salida: logística

Épocas: 300









Blobs

Valores de ajuste

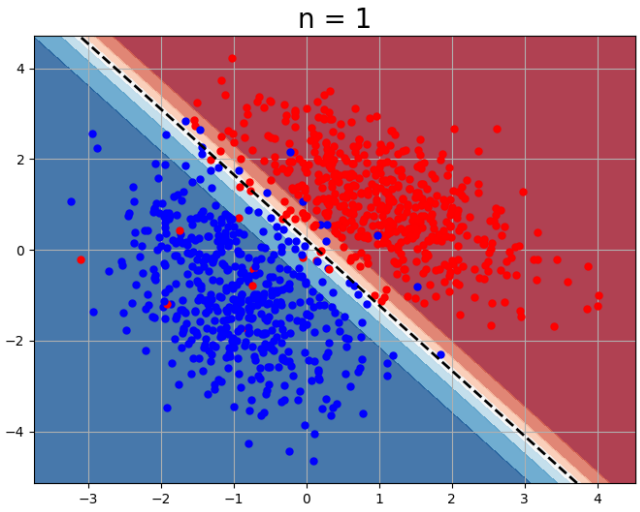
Entradas: 2

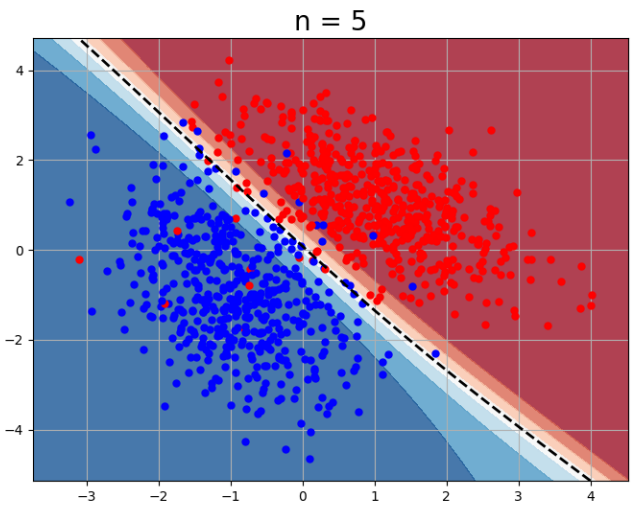
Salidas: 1

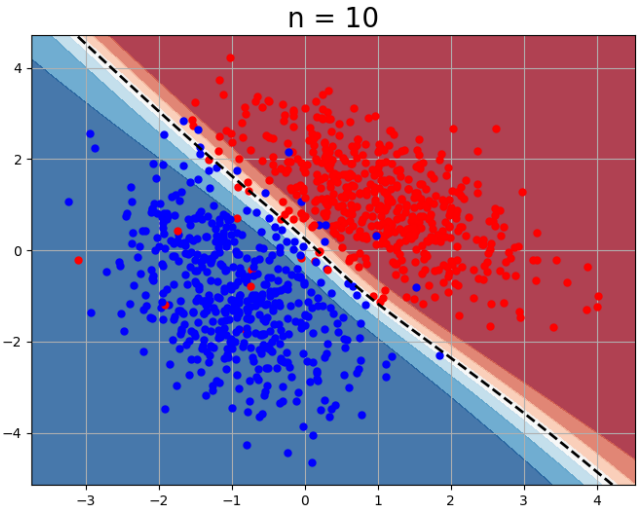
Función de activación: tanh

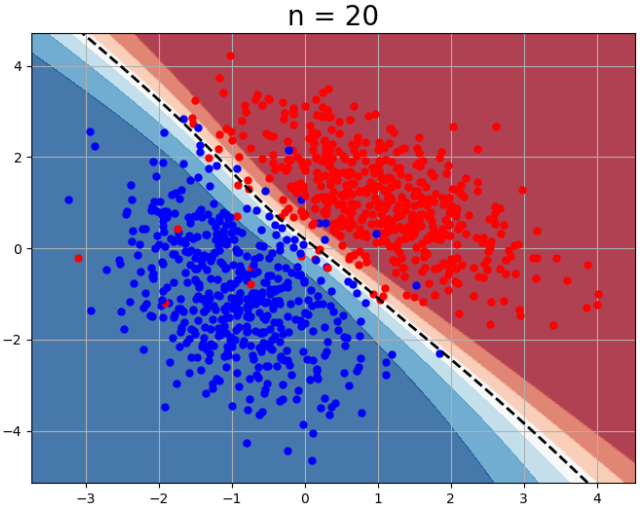
Función de activación salida: logística

Épocas: 300









Circles

Valores de ajuste

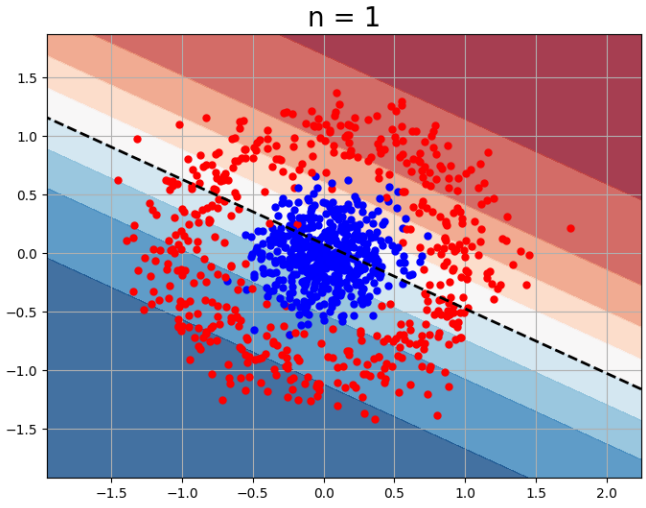
Entradas: 2

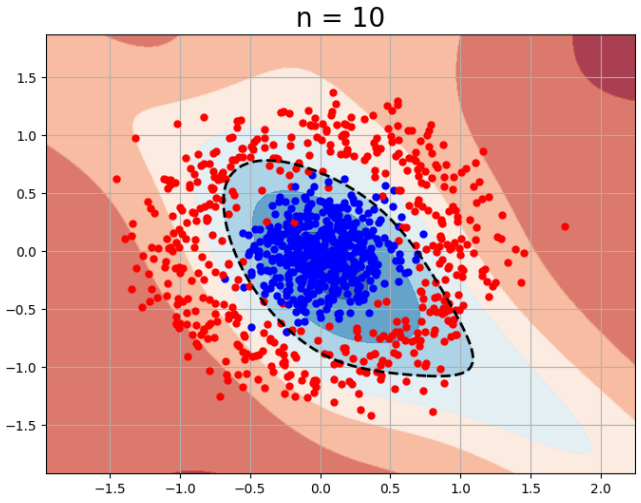
Salidas: 1

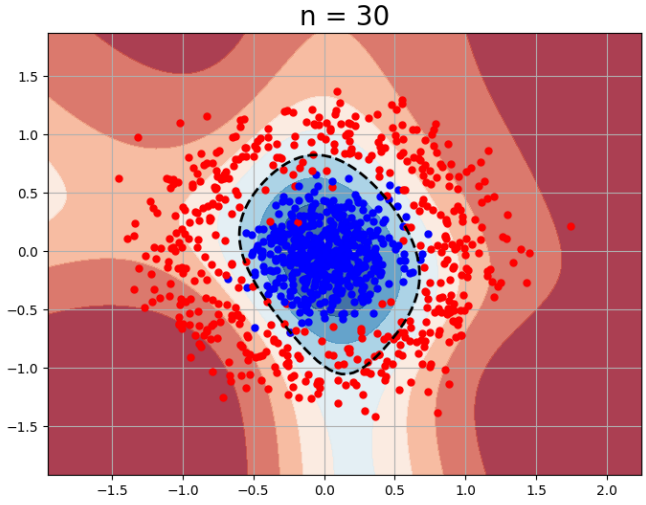
Función de activación: tanh

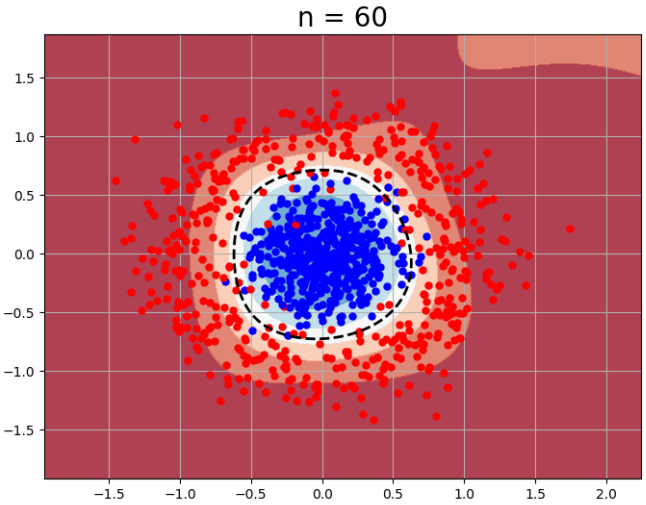
Función de activación salida: logística

Épocas: 300









Moons

Valores de ajuste

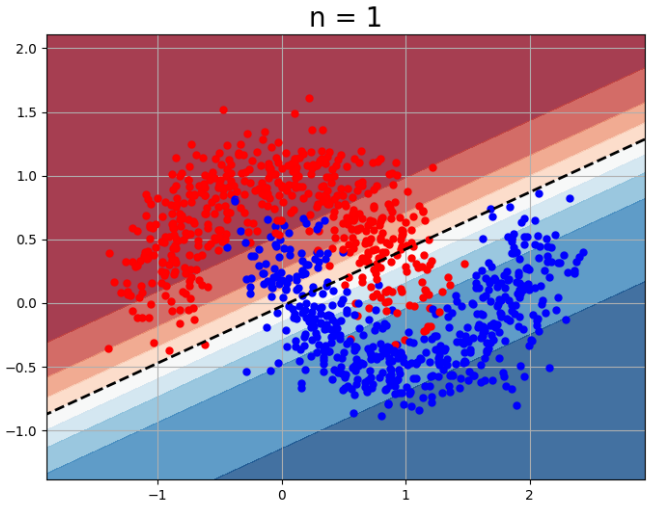
Entradas: 2

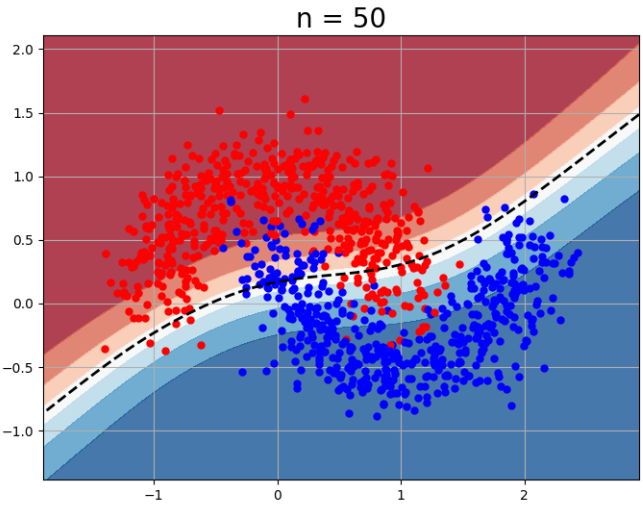
Salidas: 1

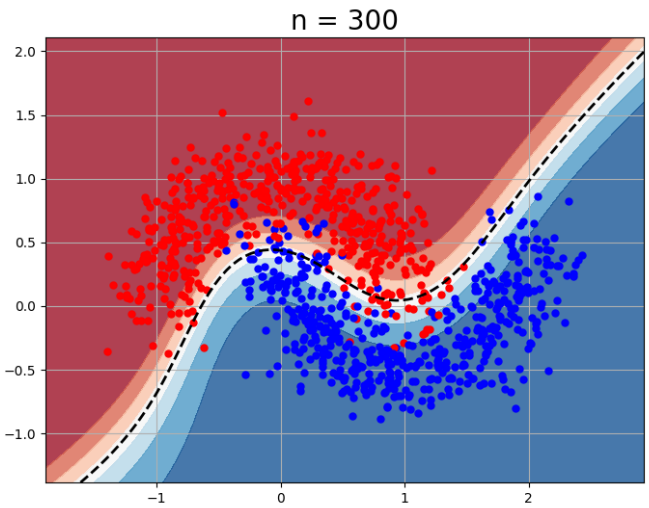
Función de activación: tanh

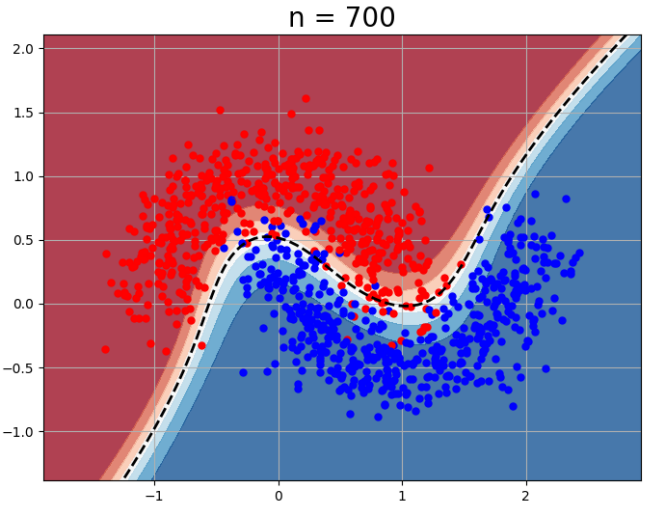
Función de activación salida: logística

Épocas: 300









CÓDIGO FUENTE:

```
#
# By: Dexne
#
# Red Neuronal Multicapa
#
```

```python
# Importamos las librerias necesarias para los calculos matematicos y para graficar
import numpy as np
import matplotlib.pyplot as plt

## Funciones de activacion
def linear(z, derivative=False):
    a = z
    if derivative:
        da = np.ones(z.shape)
        return a, da
    return a

def logistic(z, derivative=False):
    a = 1 / (1+np.exp(-z))
    if derivative:
        da = np.ones(z.shape)
        return a, da
    return a

def logistic_hidden(z, derivative=False):
    a = 1 / (1+np.exp(-z))
    if derivative:
        da = a * (1-a)
        return a, da
    return a

def tanh(z, derivative=False):
    a = np.tanh(z)
    if derivative:
        da = (1+a) * (1-a)
        return a, da
    return a

def relu(z, derivative=False):
    a = z * (z>=0)
```

```
    if derivative:
        da = np.array(z >= 0, dtype=float)
        return a, da
    return a

def softmax(z, derivative=False):
    e_z = np.exp(z - np.max(z, axis=0))
    a = e_z / np.sum(e_z, axis=0)
    if derivative:
        da = np.ones(z.shape)
        return a, da
    return a

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

        # Inicializamos los pesos y los sesgos
        for l in range( 1, self.L + 1):
            self.w[l] = -1 + 2 * np.random.rand(layers_dim[l], layers_dim[l-1])
            self.b[l] = -1 + 2 * np.random.rand(layers_dim[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 activations and gradients
            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]

            # Gradient Descent
            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])

## Funcion auxiliar para graficos
def MLP_binary_classification_2d( subplot, X, Y, net ):
    for i in range(X.shape[1]):
        if Y[i]==0:
            subplot.plot(X[0,i], X[1,i], 'ro', markersize=5)
        else:
```

```
            subplot.plot(X[0,i], X[1,i], 'bo',markersize=5)

    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)
    subplot.contour(xx,yy,zz,[0.5], colors='k',  linestyles='--', linewidths=2)
    subplot.contourf(xx,yy,zz, alpha=0.8, cmap=plt.cm.RdBu)

## Funcion principal
neurons = 10 # Mover el numero de neuronas segun desee
net = DenseNetwork((2, neurons, 1)) # Definimos que tenemos 2 entradas y 1
salida

# Elegir un csv  -->  [ 'moons', 'blobs', 'circles', 'XOR' ]
X = np.genfromtxt("XOR.csv", delimiter=',', skip_header=1, usecols=[0,1]).T
Y = np.genfromtxt("XOR.csv", delimiter=',', skip_header=1, usecols=[2]).T

print(X.shape, Y.shape)

plt.figure(figsize=(8, 6))
plt.title("n = " + str(neurons), fontsize=20) # mostrar el numero de neuronas

# After fit
net.fit(X, Y, epochs=300)
MLP_binary_classification_2d( plt, X, Y, net )

plt.grid()
plt.show()
```