RED NEURONAL DE UNA CAPA

By: Jared Isaías Monje Flores

20 DE FEBRERO DE 2024
CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENIERÍAS
Inteligencia artificial 2

Red neuronal de una capa

¿Qué es una red neuronal de una capa?

Una red neuronal de una capa, también conocida como perceptrón, es el tipo más simple de red neuronal artificial. Consiste en un conjunto de neuronas organizadas en una única capa, donde cada neurona está conectada a todas las entradas de la red. Cada conexión entre una neurona y una entrada tiene asociado un peso que determina la contribución de esa entrada a la neurona.

En una red neuronal de una capa, la salida de cada neurona se calcula como una combinación lineal de las entradas, ponderadas por los pesos respectivos, seguida de la aplicación de una función de activación. Comúnmente, la función de activación utilizada es la función de paso, que produce una salida binaria basada en si la suma ponderada de las entradas supera un umbral determinado.

Softmax

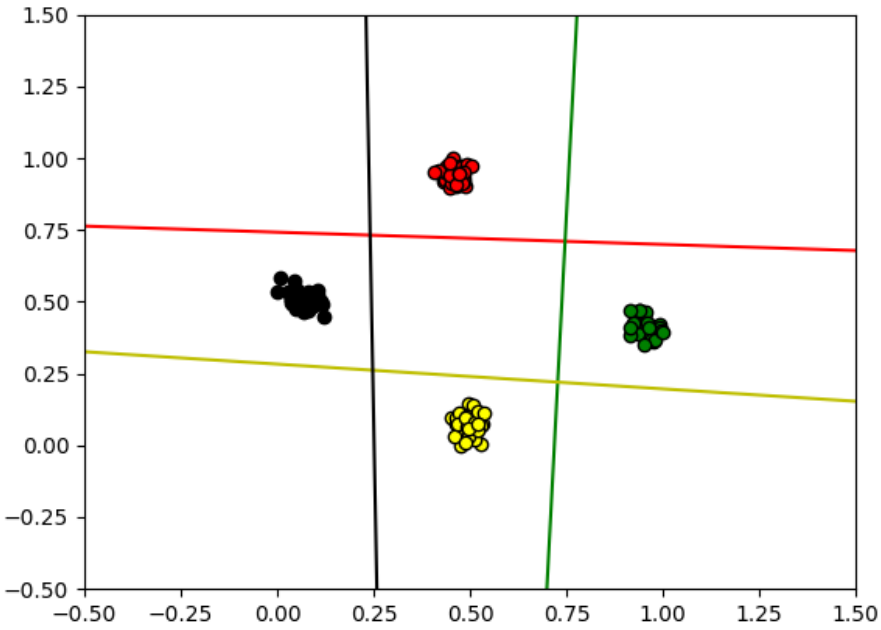
En inteligencia artificial (IA), especialmente en el ámbito del aprendizaje profundo y las redes neuronales, softmax es una función de activación utilizada comúnmente en la capa de salida de una red neuronal para convertir las salidas brutas en una distribución de probabilidad.

La función softmax toma un vector de números reales como entrada y devuelve otro vector de la misma longitud, donde cada elemento del vector de salida representa la probabilidad de que la entrada pertenezca a una de las posibles clases. La función softmax calcula estas probabilidades normalizando exponencialmente los valores de entrada y luego dividiendo cada valor por la suma de todas las exponenciales, asegurando que la suma de las probabilidades resultantes sea igual a uno.

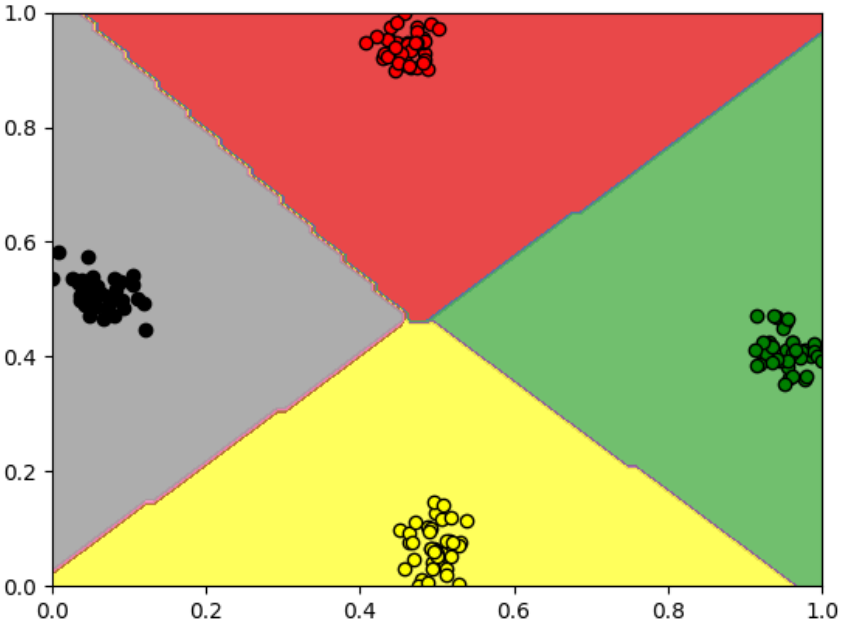
La función softmax es especialmente útil en problemas de clasificación multiclase, donde se busca asignar una instancia de entrada a una de varias clases diferentes. La salida de softmax proporciona una interpretación intuitiva de las salidas de la red neuronal como probabilidades que pueden ser utilizadas para tomar decisiones de clasificación.

Resultados obtenidos

Ejemplo 1



Ejemplo 2



Código:

```
#
# by: Dexne
#
# neurona de una capa
#
# Para mejores resultados podemos editar los valores de las epocas y el learning rate
# Nota: Es importante tener los datos del csv en la misma ruta que este archivo
# De no ser así, se debera de moficiar el path
#


# Importamos las librerias para los calculos matematicos, graficacion y tratado de datos
#


# importamos las librerías necesarias para la graficacion, calculos y tratado de los datos
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd



## neurona lineal
def linear(z, derivate=False):
    a = z
    if derivate:
        da = np.ones(z.shape)
        return a, da
    return a


## neurona logistica
def logistic(z, derivate=False):
    a = 1/(1 + np.exp(-z))
    if derivate:
```

```
        da = np.ones(z.shape, dtype=float)
        return a, da
    return a


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


## One Layer Network
class OLN:
    ## constructor
    def __init__(self, number_inputs, number_outputs, activation_function = linear):
        self.w = -1 + 2*np.random.rand(number_outputs, number_inputs)
        self.b = -1 + 2*np.random.rand(number_outputs, 1)
        self.f = activation_function


    def predict(self, X):
        Z = self.w @ X + self.b # Multiplicacion matricial
        return self.f(Z)


    def fit(self, X, Y, epochs=500, learning_rate=0.1):
        p = X.shape[1]
        for _ in range(epochs):
            # Propagacion --------
            Z = self.w @ X + self.b
            Y_est, dY = self.f(Z, derivate=True)
```

```
            # Calcular gradiente local ( local gradient )
            local_gradient = ( Y-Y_est ) * dY

            # Actualización de parametros
            self.w += (learning_rate/p) * local_gradient @ X.T
            self.b += (learning_rate/p) * np.sum(local_gradient, axis=1).reshape(-1, 1)

def plot_data(X, Y, net):
    dot_c = ('red', 'green', 'yellow', 'black')
    lin_c = ('-r', '-g', '-y', '-k')

    for i in range(X.shape[1]):
        c = np.argmax(Y[:, i])
        plt.scatter(X[0, i], X[1, i], color=dot_c[c], edgecolor='k')

    for i in range(4):
        w1, w2, b = net.w[i, 0], net.w[i,1], net.b[i]
        plt.plot([-0.5, 1.5], [(1/w2)*(-w1*(-0.5)-b), (1/w2)*(-w1*(1.5)-b)], lin_c[i])

    ## limites para el ploteo
    plt.xlim([-0.5,1.5])
    plt.ylim([-0.5,1.5])

## Ejemplo
df = pd.read_csv('Dataset_A05.csv') # leemos los datos del csv

## Tratamos los datos para que conicidan
X = np.asanyarray(df.iloc[:, :2]).T
Y = np.asanyarray(df.iloc[:, 2:]).T

net = OLN(2, 4, logistic) # [ 'logistic', 'linear' ]
net.fit(X, Y, epochs=1000, learning_rate=1)
```

```
plot_data(X, Y, net)


## Parte 2
net = OLN(2, 4, softmax)
net.fit(X, Y, epochs=10000, learning_rate=1)


def plot_data_softmax(X, Y, net):
    dot_c = ('red', 'green', 'yellow', 'black')
    xx, yy = np.meshgrid(np.linspace(0,1,100), np.linspace(0,1,100))

    data = [xx.ravel(), yy.ravel()]
    zz = net.predict(data)
    zz = np.argmax(zz, axis=0)
    zz = zz.reshape(xx.shape)
    plt.contourf(xx, yy, zz, alpha=0.8, cmap=plt.get_cmap('Set1'))

    for i in range(X.shape[1]):
        c = np.argmax(Y[:, i])
        plt.scatter(X[0, i], X[1, i], color=dot_c[c], edgecolor='k')

    plt.xlim([0,1])
    plt.ylim([0,1])

plt.figure()
plot_data_softmax(X, Y, net)
plt.show()
```