Matemática Aplicada a Tecnologías de la Información
Curso 2023/24

Práctica 6: Perceptrón

En esta práctica vamos a implementar un perceptrón multicapa y a dar nuestros primeros pasos usando `Tensorflow`.

```
import math
import numpy as np
```

Ejercicio 1 (modelo) El perceptrón está compuesto por dos operaciones, la agregación y la función de activación. Dado un dato $x \in \mathbb{R}^n$, y una función de activación $\sigma : \mathbb{R} \to \mathbb{R}$, el perceptrón se define como:

$$\mathcal{N}_{W,b}(x) = \sigma\left(\sum_{i=1}^{n} x_i \cdot w_i + b\right)$$

Implementa una función `output_perceptron(W,b,x)`, dada una matriz de pesos y un término *bias*, calcule la salida del perceptrón con función de activación sigmoide. La función sigmoide se define como:

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}$$

Ejercicio 2 (función error/coste) Para medir el rendimiento de un perceptrón, hay que disponer de un conjunto de datos $D = (X, Y, f)$. En nuestro caso, X será un array e Y otro array codificando las clases, tal que la clase de cada $x \in X$ es la del $y \in Y$ que ocupa la misma posición en el array. Un ejemplo de función error es el Error Cuadrático Medio (ECM):

$$E(X, Y, \mathcal{N}_{W,b}) = \frac{1}{N}\sum_{i=1}^{N} \left(Y_i - \mathcal{N}(X_i)\right)^2,$$

donde N es el tamaño de X. Implementa una función `error_fun(X,Y,W,b)` que, dado un un conjunto de datos y los parámetros del perceptrón, devuelva el error cometido.

Ejercicio 3 (entrenamiento) Para entrenar, emplearemos el algoritmo de descenso por gradiente. Para cada peso w, la regla de actualización es la siguiente:

$$w \leftarrow w + \eta \cdot \frac{\partial E(X, Y, \mathcal{N}_{W,b})}{\partial w}$$

Implementa la regla de actualización del algoritmo de descenso por gradiente. Para ello, tendrás que calcular previamente la expresión de la derivada. *Nota:* La derivada de la sigmoide es $\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$.

Ejercicio 4 (Aplicación) Emplearemos ahora un conjunto de datos para poner en práctica nuestro perceptrón. Para ello, vamos a trabajar con dos nubes de puntos sintéticas en $\mathbb{R}^2$.

a) Crea dos datasets distintos compuestos por 200 puntos que definan un problema de clasificación binaria empleando las siguientes funciones:

```
from sklearn import datasets

make_classification(*)
make_circles(*)
```

b) Combina todo lo que has implementado para el perceptrón en una sola clase que además, permita entrenar el perceptrón durante un determinado número de iteraciones.

c) Entrena el modelo empleando un 70 % del conjunto de datos como entrenamiento y un 30 % como conjunto test. ¿En qué conjunto de datos funciona mejor?

Ejercicio 5 (`Tensorflow`) Emplearemos ahora una red neuronal algo más compleja empleando una librería para *Deep Learning*. Implementa un perceptrón multicapa con una capa oculta compuesta por tres neuronas. Luego, entrénala igual que el perceptrón del ejercicio anterior.

```
import tensorflow as tf
```

Un ejemplo que podéis adaptar a lo pedido en el ejercicio:

```
inputs = Input(shape=(input_dim,))
x=Dense(3, use_bias=bias,activation=activation_function)(inputs)
x=Dense(2, use_bias=bias,activation=activation_function)(x)
predictions=Dense(input_dim,use_bias=bias, activation='softmax'
            ,name='final_output')(x)
model = Model(inputs=inputs, outputs=predictions)
model.summary()
lr = 0.03
lossfunction = 'mean_squared_error'
model.compile(optimizer="adamax", loss = lossfunction,
            metrics = ['accuracy'] )
model.fit(X,label,epochs = 500, batch_size = 5,verbose=True)
model.evaluate(X,label)
```