**Práctica 4: Entrenamiento de redes neuronales**

Gabriel Sellés Salvà

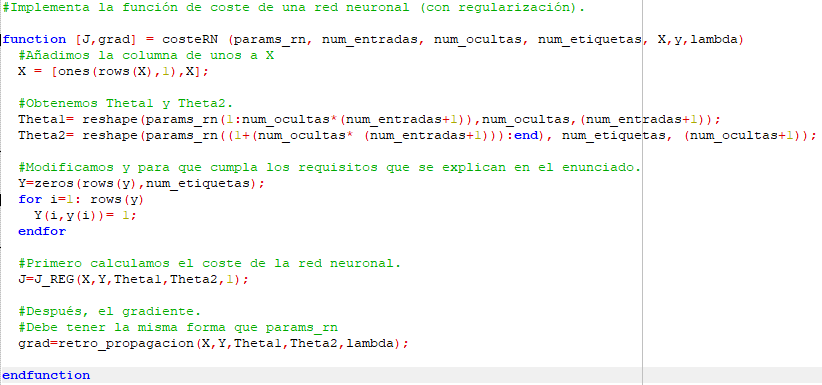
Manuel Antonio Fernández Alonso

Aprendizaje Automático UCM 2018

**1-Función de coste & retro-propagación**

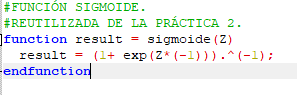
Para la resolución de este apartado hemos implementado una función que calcula el coste y el gradiente asociado a una red neuronal. La función se llama **costeRN**. Para calcular el gradiente, hemos utilizado el algoritmo de retro-propagación.

La implementación de esta función se encuentra a continuación:

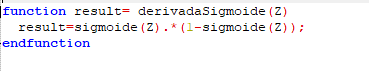


Esta función utiliza una serie de funciones (también implementadas por nosotros) para realizar dicha tarea:

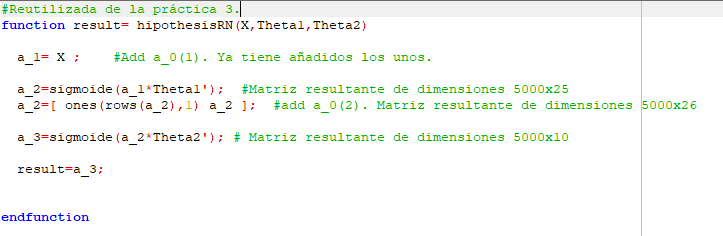
* **sigmoide:** implementa la función sigmoide.



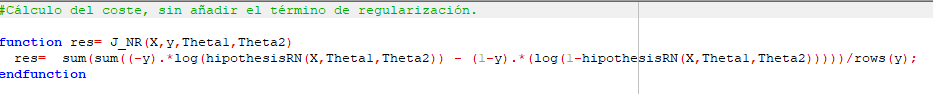
* **derivadaSigmoide:** calcula la derivada de la función sigmoide.



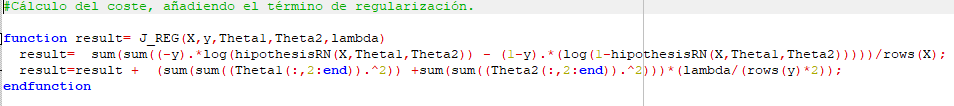
* **hipothesisRN:** calcula (h\_0(X)), el resultado de los cálculos de la red neuronal.



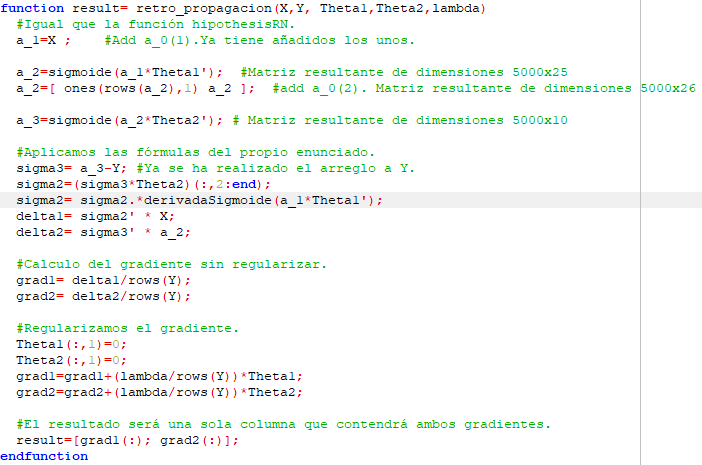
* **J\_NR**: realiza el cálculo del coste, sin regularización.



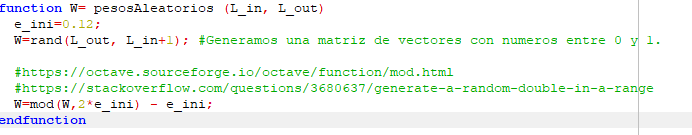
* **J\_REG:** realiza el cálculo del coste, con regularización.



* **retro\_propagacion:** calcula el gradiente utilizando el algoritmo de retro-propagación.



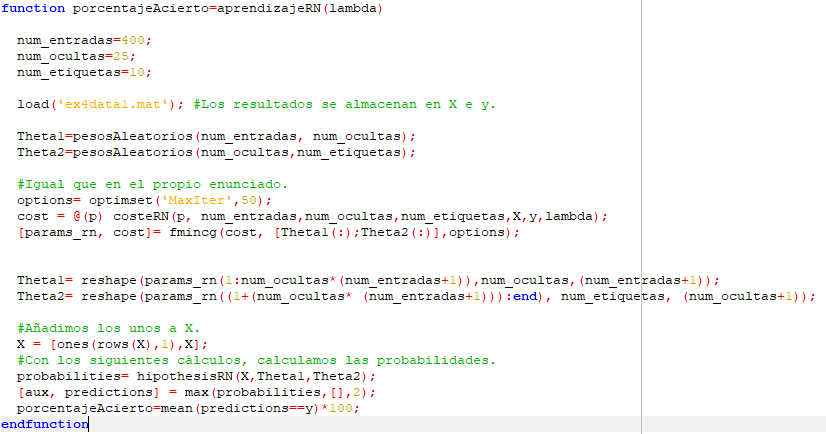
* **pesosAleatorios:** Genera un vector con valores aleatorios entre -e\_ini y e\_ini (ambos inclusive).



**2-Aprendizaje de los parámetros de la red neuronal**

Hechas las funciones del apartado anterior, hemos implementado otra para el entrenamiento de la red neuronal, **aprendizajeRN**. Esta función calcula cuáles son los parámetros para los que nuestra red neuronal tiene un menor coste (respecto a los datos de entrenamiento) y devuelve el porcentaje de acierto que tiene con esos parámetros, utilizando dichos datos de entrenamiento.

La implementación de esta función se encuentra a continuación:



Con este sistema implementado, los resultados utilizando lambda = 1 son los siguientes:



Y utilizando lambda = 0.5



Por último, usando lambda = 0.1

