**Práctica 4 (Clasificador)**

¿Cómo he planteado la práctica?

Primero necesitamos los datos, en este caso leemos el .MAT ‘ex4data1’, y separamos los datos en X e Y, como está en un diccionario, accedemos como un array en el que, en lugar de posiciones, tenemos el nombre de las columnas: data[‘X’] y data[‘Y’].

Mostramos 100 ejemplos como se pide en la práctica.

Además, debemos implementar One Hot Encoding para la Y, para así, al operar sobre todos los valores, que ‘aporten’ a su salida los implicados.

También he implementado la función sigmoide y las funciones de coste y gradiente regularizadas, que tienen como parámetros:

* Theta
* X
* Y
* lam

En el siguiente paso, como pide el enunciado, he calculado la función de backpropagation y creado funciones de apoyo (como ‘recolocar’)

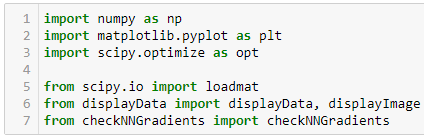
Posteriormente, he comprobado la gradiente y creado una función que permite inicializar las con valores aleatorios.

Finalmente he predicho los resultados, multiplicando cada ejemplo por cada etiqueta para (10 distintas), y he calculado su precisión con .

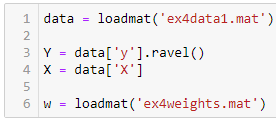
La clasificación ha arrojado unos resultados cercanos al 93%.

Código comentado y gráficas

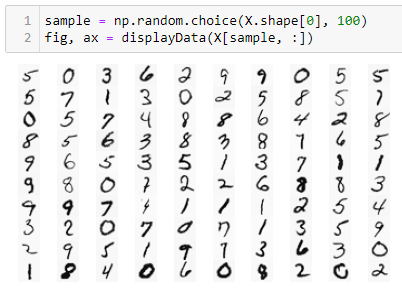
1. Importar las librerías



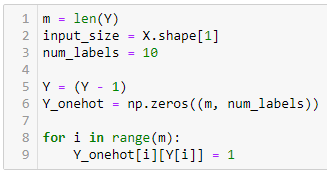
1. Obtener los datos del CSV:



*Es interesante destacar que en la Y necesitamos hacer ravel(), para que la dimensión sea de (5000,) y no de (5000, 1), ya que, si no lo hacemos, la función FMIN\_TNC suele dar problemas.*



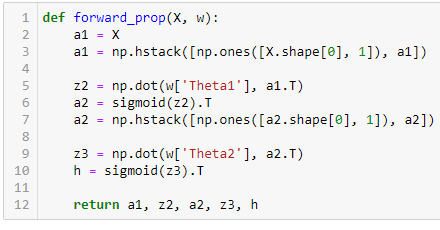
1. Ahora creamos la Y\_onehot para que al operar entre vectores sea más cómodo.



1. Ahora creamos las funciones:
   * Sigmoide



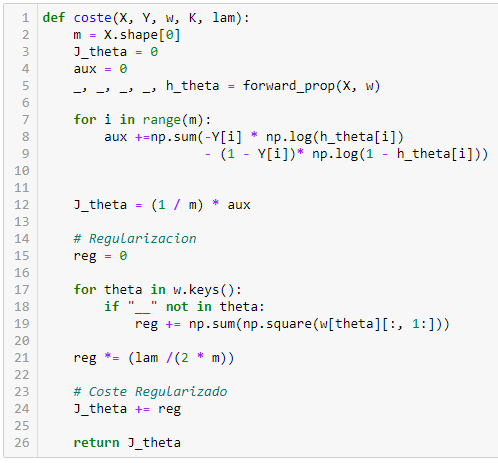
* + Forward Propagation: en la devuelvo a1, z2, a2, z3 y h (=a3)



* + De coste: En la que solo me interesa quedarme con de la Forward Propagation. Como hay que ir fila por fila, lo pongo en un bucle (#7), hago las operaciones y aplico la regularización.

Como el diccionario de s tiene keys que no nos interesan (“\_\_”), solo nos quedamos con las que nos interesan (#18)

Finalmente aplicamos la regularización a la



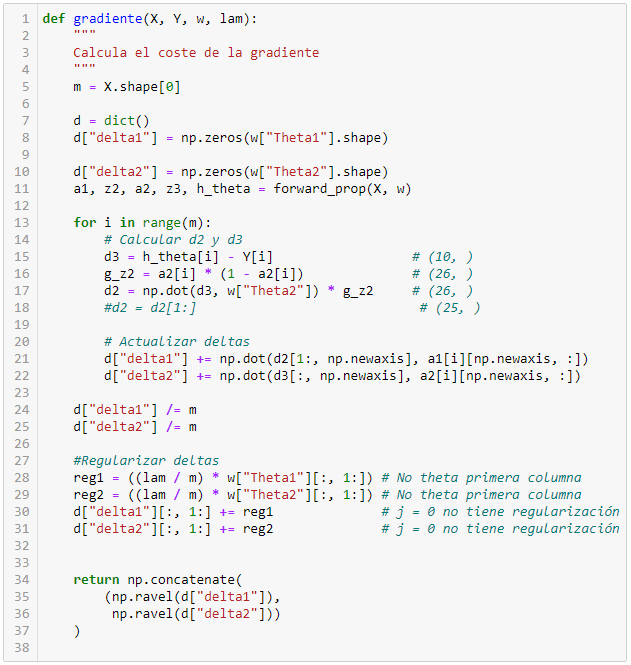
* + De gradiente: Uso diccionarios para las , ahora sí que nos interesa quedarnos con todos los términos que se devuelven en la Forward Propagation (#11).

Se aplican los cálculos dentro del bucle, y se acumulan la suma en los . Al terminar, se divide toda la acumulación y se aplica la regularización

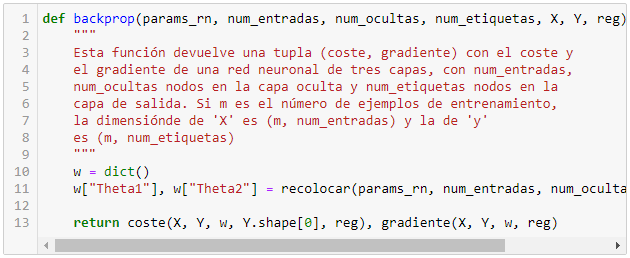
Sin la primera columna (#28 y #29)

Sin contar j=0 (#30 y #31)

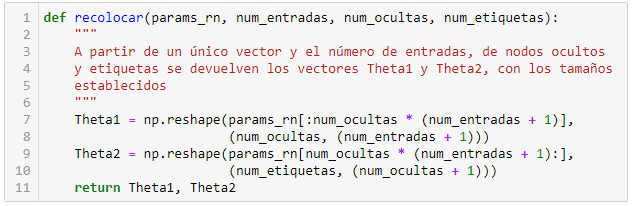
Hay que destacar que es obligatorio el uso de np.newaxis (#21 y #22), que aumenta un eje, para así poder hacer las operaciones.



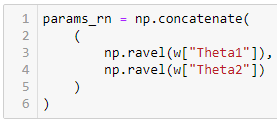
1. Tras ello implementamos la función de Back Propagation, como se pide en el enunciado, en el que devuelve la tupla del coste y la gradiente, ambas regularizadas.



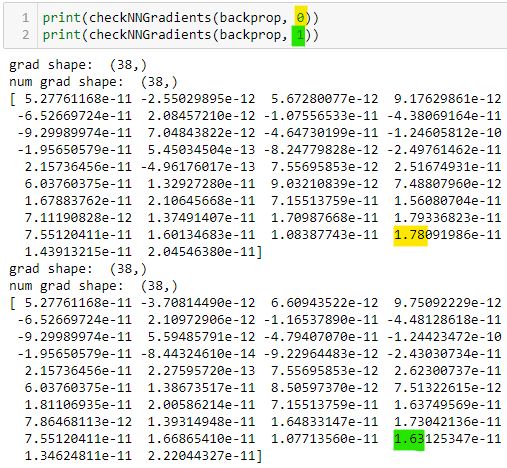
Además, usamos una función de apoyo (‘recolocar’) que nos permitirá partir un vector con los tamaños deseados.



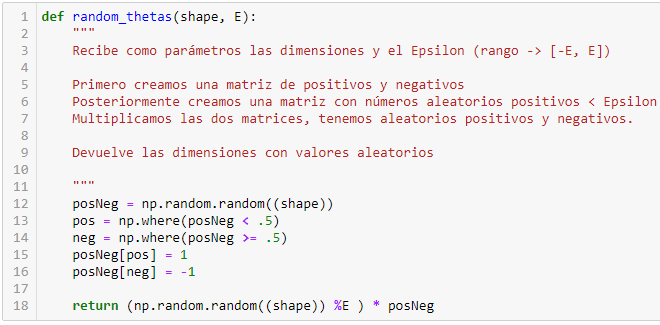
Para poder usar la función ‘backprop’, necesitamos unir todas las s en un vector, para ello concatenamos y usamos la función ‘ravel’



1. Ahora comprobamos la gradiente con la función checkGradients:



1. Para poder aplicar correctamente una red neuronal, necesitamos inicializar los valores de aleatoriamente, tanto en positivo como en negativo.

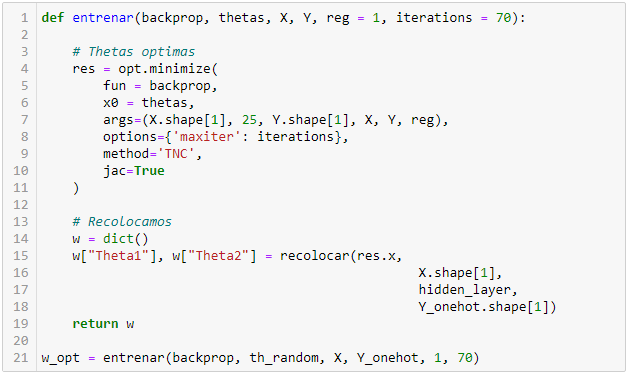


1. Inicializamos los valores.

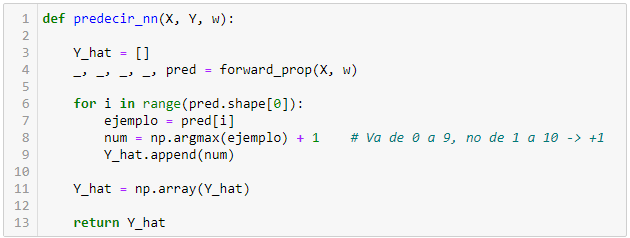
En las líneas #6 y #7 creamos un par con las dimensiones que deberían tener las s, y posteriormente las inicializamos aleatoriamente (#12 y #13)



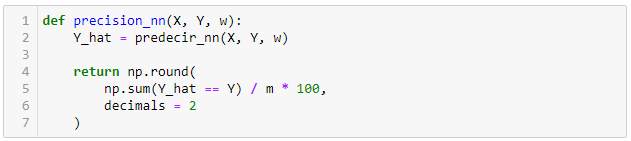
1. Obtenemos las con MINIMIZE y recolocamos el vector que nos devuelve ()

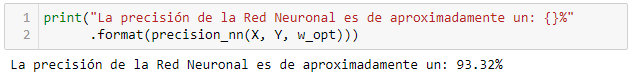


1. Ahora predecimos el resultado

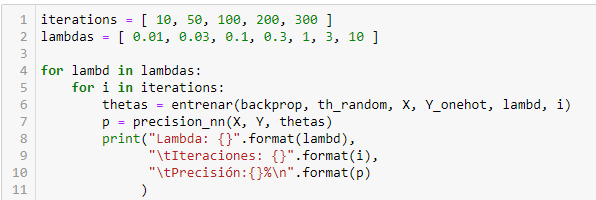


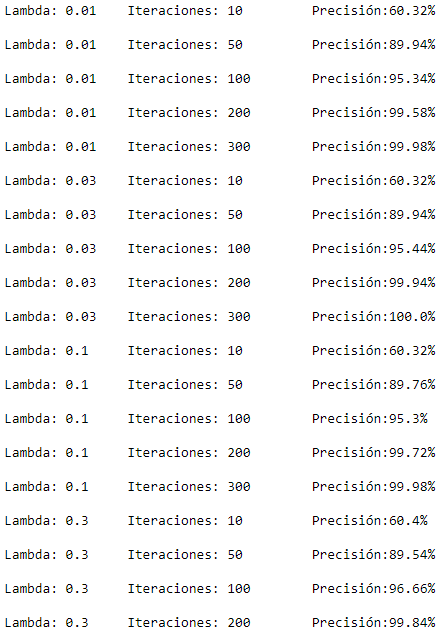
1. Y comprobamos la precisión que tiene nuestra Red Neuronal

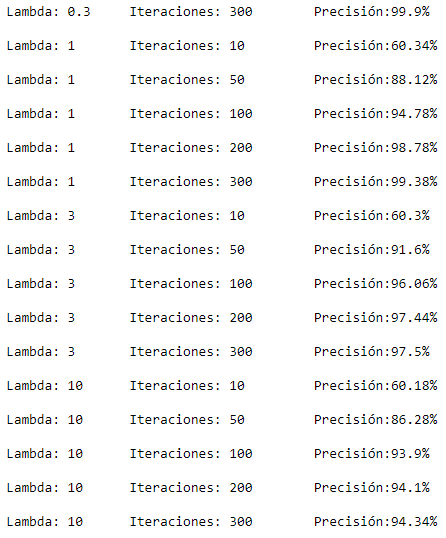




1. Ahora comparamos las iteraciones y las







Como conclusiones sacamos:

* Lo que más aporta son las iteraciones, ya que las distintas tienen pocas diferencias para las mismas iteraciones, por ejemplo, para 100 iteraciones , la precisión es de 95’34%, mientras que para la precisión es de 96’66%.
* Por otro lado, para la misma , en este caso, , la diferencia entre 50 y 100 iteraciones es de aproximadamente un 7%.
* Otro dato a destacar es que, por norma general, **a más iteraciones mejor precisión**, en cambio, y como es lógico, si **la regularización es muy alta, tiende a mostrar resultados peores**, por ejemplo, para y 300 iteraciones, la precisión es del 99’98%, mientras que para y 300 iteraciones la precisión es del 94’34%
* El mejor resultado lo ofrece y 300 iteraciones, con un 100% de precisión.