Práctica 4 – Entrenamiento de redes neuronales

Realizada por Mario Blanco Domínguez y Juan Tecedor Roa

- Objetivo de la práctica

  En esta práctica implementaremos el código para el calculo de la función de coste de una red neuronal. Usaremos el mismo conjunto de datos de la práctica 3: "ex4data1.mat" y "ex4weights.mat".
  También usaremos dos archivos que se nos han dado para comprobar la gradiente y mostrar los datos: "displayData.py" y "checkNNGradients.py".

- Código de la práctica- parte 1
  Comenzamos importando todo lo necesario y implementando la función de coste:

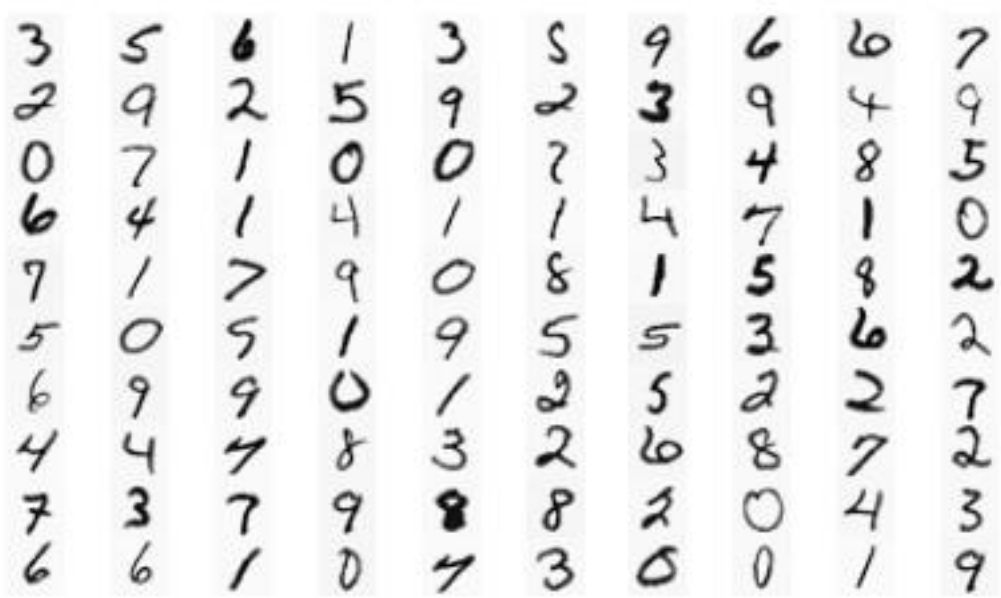
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat
import scipy.optimize as opt
import sklearn.preprocessing

import displayData, checkNNGradients
from displayData import *
from checkNNGradients import *


def sigmoid(X):
    return 1 / (1 + np.exp(-X))


def cost(X, Y, l, T_1, T_2):
    A1, A2, H = forward_propagation(X, T_1, T_2)
    m = X.shape[0]
    l1 = np.transpose(np.log(H))
    l2 = np.transpose(np.log(1 - H))
    ret = ((l1.T * -Y) - ((1 - Y) * l2.T))
    ret = np.sum(ret) / m
    ret += (l / (2 * m)) * (np.sum(np.square(T_1[:, 1:])) +
np.sum(np.square(T_2[:, 1:])))
    return ret


def forward_propagation(X, T1, T2):
    m = X.shape[0]

    A1 = np.hstack([np.ones([m, 1]), X])
    Z2 = np.dot(A1, T1.T)
    A2 = np.hstack([np.ones([m, 1]), sigmoid(Z2)])
    Z3 = np.dot(A2, T2.T)
```

```
        H = sigmoid(Z3)

        return A1, A2, H


    # Devuelve una tupla con coste y gradiente
    def gradient(X, Y, l, theta_1, theta_2):
        m = X.shape[0]
        A1, A2, H = forward_propagation(X, theta_1, theta_2)
        D1, D2 = np.zeros(theta_1.shape), np.zeros(theta_2.shape)

        for t in range(m):
            a1t = A1[t, :]
            a2t = A2[t, :]
            ht = H[t, :]
            yt = Y[t]

            d3t = ht - yt
            d2t = np.dot(theta_2.T, d3t) * (a2t * (1 - a2t))

            D1 = D1 + np.dot(d2t[1:, np.newaxis], a1t[np.newaxis, :])
            D2 = D2 + np.dot(d3t[:, np.newaxis], a2t[np.newaxis, :])

        D1 *= 1 / m
        D2 *= 1 / m
        # Regularizacion de todos menos j=0
        D1[:, 1:] += (l / m * theta_1[:, 1:])
        D2[:, 1:] += (l / m * theta_2[:, 1:])
        grad = np.concatenate((np.ravel(D1), np.ravel(D2)))

        return grad


    def backprop(params_rn, num_entradas, num_ocultas, num_etiquetas, X, Y,
    reg):
        theta_1 = np.reshape(params_rn[:num_ocultas * (num_entradas + 1)],
                             (num_ocultas, (num_entradas + 1)))
        theta_2 = np.reshape(params_rn[num_ocultas * (num_entradas + 1):],
                             (num_etiquetas, (num_ocultas + 1)))

    return (cost(X, Y, reg, theta_1, theta_2), gradient(X, Y, reg, theta_1,
theta_2))

    def main():
        data = loadmat('ex4data1.mat')
        Y = data['y'].ravel()
        X = data['X']
        # X = np.hstack([np.ones([np.shape(X)[0], 1]), X])
        m = len(Y)
        input_size = X.shape[1]
        num_labels = 10

        # Diagonal a unos para poder entrenar
        Y = (Y - 1)
        Y_oneHot = np.zeros((m, num_labels))
        for i in range(m):
```

```
            Y_oneHot[i][Y[i]] = 1

        weights = loadmat('ex4weights.mat')
        theta_1, theta_2 = weights['Theta1'], weights['Theta2']

        sample = np.random.choice(X.shape[0], 100)
        image = displayData(X[sample, :])

        l = 1

        input_layer_size = 400
        hidden_layer_size = 25
        num_labels = 10


        params_rn = np.append(np.ravel(theta_1),(np.ravel(theta_2)))

        print("Coste sin regularizacion:
    ",backprop(params_rn,input_layer_size, hidden_layer_size, num_labels, X,
    Y_oneHot, 0)[0])
    print("Coste con regularizacion: ",backprop(params_rn,input_layer_size,
hidden_layer_size, num_labels, X, Y_oneHot, l)[0])
```

- Resultados de ejecución: parte 1

Tras leer los ejemplos de entrenamiento y elegir una muestra aleatoria, obtenemos la siguiente imagen:



Tras obtener los thetas ya entrenadas, podemos comprobar si funciona correctamente nuestra función de coste. Obtenemos los siguientes resultados de aplicar el coste a los datos:

```
Coste sin regularizacion:  0.2876291651613189
Coste con regularizacion:  0.38376985909092365
```

- Código: segunda parte

```
def calcularAciertos(X, Y, T1, T2):
    aciertos = 0
    j = 0
    tags = len(T2)
    pred = forward_propagation(X, T1, T2)[2]
    for i in range(len(X)):
        maxi = np.argmax(pred[i])
        if Y[i] == maxi:
            aciertos += 1
        j += 1
    return aciertos / len(Y) * 100
……….

def main():
theta_1, theta_2 = np.random.uniform(-.12, .12, theta_1.shape),
np.random.uniform(-.12, .12, theta_2.shape)
    params_rn = np.append(np.ravel(theta_1),(np.ravel(theta_2)))

    result = opt.minimize(fun = backprop, x0= params_rn,
args=(input_layer_size, hidden_layer_size, num_labels, X, Y_oneHot, l),
method = 'TNC', options={'maxiter': 70}, jac=True)

    theta_1= np.reshape(result.x[:25 * (400 + 1)], (25, (400 + 1)))
    theta_2 = np.reshape(result.x[25 * (400 + 1):], (10, (25 + 1)))


    print("El porcentaje de acierto del modelo es: ",
calcularAciertos(X,Y,theta_1,theta_2))
```

- Resultados de ejecución: segunda parte

Tras implementar la función gradiente, utilizamos la función checkNNgradients otorgada por el profesor para ver que está implementada correctamente. Como queríamos ver que ocurría si no estaba bien implementada, sumamos 1 al resultado del gradiente

```
def backprop(params_rn, num_entradas, num_ocultas, num_etiquetas, X, Y, reg):
    theta_1 = np.reshape(params_rn[:num_ocultas * (num_entradas + 1)],
                         (num_ocultas, (num_entradas + 1)))
    theta_2 = np.reshape(params_rn[num_ocultas * (num_entradas + 1):],
                         (num_etiquetas, (num_ocultas + 1)))

    return (cost(X, Y, reg, theta_1, theta_2), gradient(X, Y, reg, theta_1, theta_2) + 1)
```

Obteniendo el siguiente error

```
  File "C:\Users\mario\AppData\Roaming\Python\Python38\site-packages\numpy\testing\_private\utils.py",
line 842, in assert_array_compare
    raise AssertionError(msg)

AssertionError:
Arrays are not almost equal to 7 decimals

Mismatched elements: 38 / 38 (100%)
Max absolute difference: 1.
Max relative difference: 638.5929295
```

Quitando ese ‘+1’ no obtenemos nada por consola, por lo que funciona correctamente. Hicimos la comprobación sin usar el término de regularización y usándolo.

```
checkNNGradients(backprop,0)
checkNNGradients(backprop,1)
```

Por último, realizamos la llamada para entrenar a la red neuronal. Tras el entrenamiento, obtenemos un acierto en torno al 93%.

```
El porcentaje de acierto del modelo es:  93.7
```

Usando un termino de regularización menos (0.1), los resultados varían:

```
El porcentaje de acierto del modelo es:  90.44
```

Con este misma lambda, y bajando las iteraciones a 25, el acierto se ve muy mermado:

```
El porcentaje de acierto del modelo es:  51.580000000000005
```