Aprendizaje automático y minería de datos: Práctica 4

Jorge Rodríguez García y Gonzalo Sanz Lastra

Código de la práctica:

```python
import numpy as np
import scipy.optimize as opt            # para la funcion de gradiente
import displayData
from matplotlib import pyplot as plt    # para dibujar las graficas
from scipy.io import loadmat
import checkNNGradients as check

def load_mat(file_name):
    """carga el fichero mat especificado y lo devuelve en una matriz data"""
    return loadmat(file_name)

def graphics(X):
    """Selecciona aleatoriamente 10 ejemplos y los pinta"""
    sample = np.random.choice(X.shape[0], 100)
    displayData.displayData(X[sample, :])
    plt.show()

# g(X*Ot) = h(x)
def sigmoid(Z):
    return 1/(1+np.exp(-Z))

def dSigmoid(Z):
    return sigmoid(Z)*(1-sigmoid(Z))

def pesosAleatorios(L_in, L_out, rango):
    O = np.random.uniform(-rango, rango, (L_out, 1+L_in))
    return O

def cost(X, Y, O1, O2, reg):
    """devuelve un valor de coste"""

    a = -Y*(np.log(X))
    b = (1-Y)*(np.log(1-X))
    c = a - b
    d = (reg/(2*X.shape[0]))* ((O1[:,1:]**2).sum() + (O2[:,1:]**2).sum())
    return ((c.sum())/X.shape[0]) + d

def neuronalSuccessPercentage(results, Y):
    """determina el porcentaje de aciertos de la red neuronal comparando los resultados estimados con los resultados reales"""
    numAciertos = 0
```

```
    for i in range(results.shape[0]):
        result = np.argmax(results[i])
        if result == Y[i]: numAciertos += 1

    return (numAciertos/(results.shape[0]))*100

def forPropagation(X1, O1, O2):
    """propaga la red neuronal a traves de sus dos capas"""
    m = X1.shape[0]
    a1 = np.hstack([np.ones([m, 1]), X1])
    z2 = np.dot(a1, O1.T)
    a2 = np.hstack([np.ones([m, 1]), sigmoid(z2)])
    z3 = np.dot(a2, O2.T)
    h = sigmoid(z3)

    return a1, z2, a2, z3, h

def backPropAlgorithm(X, Y, O1, O2, num_etiquetas, reg):
    G1 = np.zeros(O1.shape)
    G2 = np.zeros(O2.shape)

    m = X.shape[0]
    a1, z2, a2, z3, h = forPropagation(X, O1, O2)

    for t in range(X.shape[0]):
        a1t = a1[t, :] # (1, 401)
        a2t = a2[t, :] # (1, 26)
        ht = h[t, :] # (1, 10)
        yt = Y[t] # (1, 10)
        d3t = ht - yt # (1, 10)
        d2t = np.dot(O2.T, d3t) * (a2t * (1 - a2t)) # (1, 26)

        G1 = G1 + np.dot(d2t[1:, np.newaxis], a1t[np.newaxis, :])
        G2 = G2 + np.dot(d3t[:, np.newaxis], a2t[np.newaxis, :])

    AuxO2 = O2
    AuxO2[:, 0] = 0

    G1 = G1/m
```

```
    G2 = G2/m + (reg/m)*AuxO2

    return np.concatenate((np.ravel(G1), np.ravel(G2)))


def backPropagation(params_rn, num_entradas, num_ocultas, num_etiquetas, X, Y, reg):
    O1 = np.reshape(params_rn[:num_ocultas*(num_entradas + 1)], (num_ocultas, (num_entradas+1)))
    O2 = np.reshape(params_rn[num_ocultas*(num_entradas+1):], (num_etiquetas, (num_ocultas+1)))

    c = cost(forPropagation(X, O1, O2)[4], Y, O1, O2, reg)
    gradient = backPropAlgorithm(X,Y, O1, O2, num_etiquetas, reg)

    return c, gradient

def main():
    # REGRESION LOGISTICA MULTICAPA
    valores = load_mat("ex4data1.mat")

    X = valores['X'] # matriz X, con todas las filas y todas las columnas menos la ultima (ys)
    Y = valores['y'].ravel() # matriz Y, con todas las filas y la ultima columna

    m = X.shape[0]      # numero de muestras de entrenamiento
    n = X.shape[1]      # numero de variables x que influyen en el resultado y, mas la columna de 1s
    num_etiquetas = 10
    l = 1

    Y = (Y-1)

    AuxY = np.zeros((m, num_etiquetas))

    for i in range(m):
        AuxY[i][Y[i]] = 1

    # REDES NEURONALES
    weights = load_mat('ex4weights.mat')
    O1, O2 = weights['Theta1'], weights['Theta2']

    thetaVec = np.append(O1, O2).reshape(-1)

    result = opt.minimize(fun = backPropagation, x0 = thetaVec,
```

```
    result = opt.minimize(fun = backPropagation, x0 = thetaVec,
     args = (n, 25, num_etiquetas, X, AuxY, l), method = 'TNC', jac = True, options = {'maxiter':70})

    O1 = np.reshape(result.x[:25*(n + 1)], (25, (n+1)))
    O2 = np.reshape(result.x[25*(n+1):], (num_etiquetas, (25+1)))

    success = neuronalSuccessPercentage(forPropagation(X, O1, O2)[4], Y)
    print("Neuronal network success: " + str(success) + " %")

    #backPropagation(thetaVec, n, 25, num_etiquetas, X, AuxY, l)
    #print(check.checkNNGradients(backPropagation, 0))

    #success = neuronalSuccessPercentage(forPropagation(X, O1, O2), Y)
    #print("Neuronal network success: " + str(success) + " %")

    #GRAFICAS
    #graphics(X[:, 1:])

main()
```

There's an updat