# Práctica 4:  Entrenamiento de Redes Neuronales

David Godoy Ruiz
Eva Lucas Leiro

## Función de Coste:

Para el coste sin regularizar es necesario implementar el algoritmo de propagación hacia delante.

```
def propagacion(X, Theta1, Theta2):

    m = X.shape[0]

    a1=np.hstack([np.ones([m, 1]), X])
    z2=np.dot(a1, Theta1.T)
    a2=np.hstack([np.ones([m, 1]), sigmoid(z2)])
    z3=np.dot(a2, Theta2.T)
    H=sigmoid(z3)

    return a1, a2, H
```

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) - (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right]$$

```
def coste_red(theta, X, Y):
    _, _, H = propagacion(X,theta[0],theta[1])
    cost = (-1 / (len(X))) * np.sum((Y * np.log(H)) + (1 - Y) * np.log(1 - H +
1e-9))
    return cost
```

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) - (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\left[\sum_{j=1}^{25}\sum_{k=1}^{400}(\Theta_{j,k}^{(1)})^2 + \sum_{j=1}^{10}\sum_{k=1}^{25}(\Theta_{j,k}^{(2)})^2\right]$$

```
def coste_red_reg(theta, X, Y, lambd):
    a = lambd/(2*(len(X))) * (np.sum(theta[0]**2) + np.sum(theta[1]**2))
    return coste_red(theta, X, Y) + a
```

## Cálculo del Gradiente:

Calculamos el gradiente a través de una función backprop.

```
def backprop ( params_rn , num_entradas , num_ocultas , num_etiquetas , X, y , reg):

    Theta1 = np.reshape ( params_rn [ : num_ocultas * ( num_entradas + 1 ) ] ,( num_ocultas , ( num_entradas + 1 )))
    Theta2 = np.reshape ( params_rn [ num_ocultas * ( num_entradas + 1 ) : ] ,( num_etiquetas , ( num_ocultas + 1 )))

    A1, A2, H = propagacion(X, Theta1, Theta2)
    m = X.shape[0]

    Delta1 = np.zeros_like(Theta1)
    Delta2 = np.zeros_like(Theta2)

    for t in range(m):
        a1t = A1[t, :] # (401,)
        a2t = A2[t, :] # (26,)
        ht = H[t, :] # (10,)
        yt = y[t] # (10,)
        d3t = ht - yt # (10,)
        d2t = np.dot(Theta2.T, d3t) * (a2t * (1 - a2t)) # (26,)
        Delta1 = Delta1 + np.dot(d2t[1:, np.newaxis], a1t[np.newaxis, :])
        Delta2 = Delta2 + np.dot(d3t[:, np.newaxis], a2t[np.newaxis, :])

    G1 = Delta1/m
    G2 = Delta2/m

    G1 = termino_reg(G1, m, reg, Theta1)
    G2 = termino_reg(G2, m, reg, Theta2)

    return coste_red_reg(np.array([Theta1, Theta2]), X, y, reg), np.concatenate([np.ravel(G1), np.ravel(G2)])
```

- Comprobación del gradiente

Comprobamos que el calcula del mismo es correcto

```
    checkNNGradients.checkNNGradients(backprop, 1)
```

## Aprendizaje de los parámetros:

En este apartado entrenamos la red neuronal con 70 iteraciones y un valor de λ = 1.

```
    ini = 0.12
    reg=1
    i=70

    pesos= np.random.uniform(-ini,ini,params_rn.shape[0])
    sol = opt.minimize(fun=backprop, x0=pesos, args=(num_entradas,num_ocultas,
num_etiquetas,X, y ,reg), jac= True,method = 'TNC', options ={'maxiter' :i})

    theta1 = np.reshape ( sol.x [ : num_ocultas * ( num_entradas + 1 ) ] ,(
num_ocultas , ( num_entradas + 1 )))
    theta2 = np.reshape ( sol.x [ num_ocultas * ( num_entradas + 1 ) : ] ,(
num_etiquetas , ( num_ocultas + 1 )))
    m = np.shape(X)[0]

    A1, A2, H = propagacion(X, theta1, theta2)

    maxChance = H.argmax(axis= 1)
    res = y.argmax(axis= 1)
    correctos = np.sum(maxChance == res)
    return correctos/m * 100
```