Pr?ctica 4 aprendizaje de redes neuronales

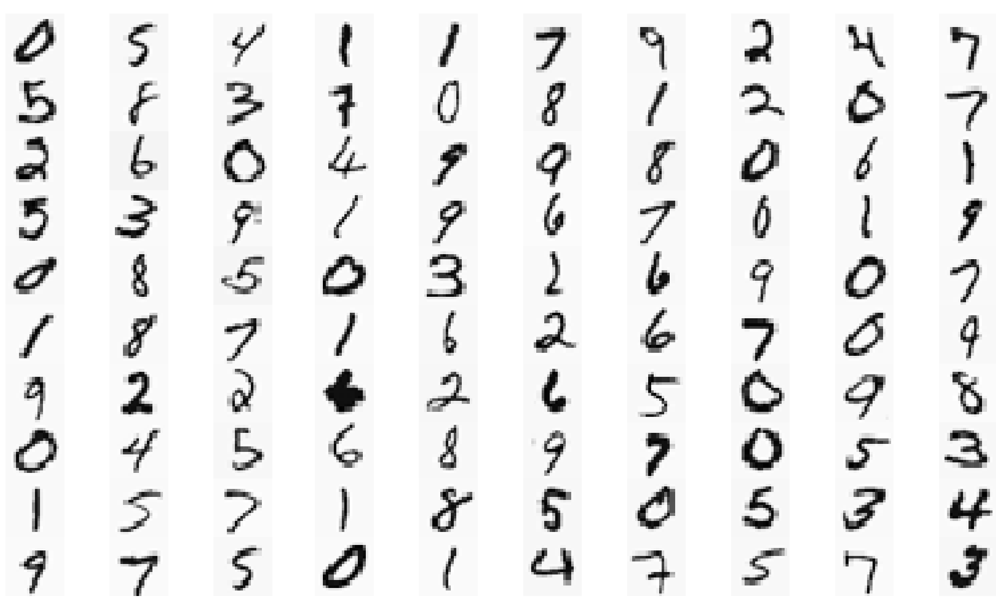
June 20, 2019

```
In [ ]: #José Antonio Garrido Sualdea

In [1]: import matplotlib.pyplot as plt
        import numpy as np
        import displayData
        import checkNNGradients
        from scipy.io import loadmat

In [2]: data = loadmat ('ex4data1.mat')
        y = data['y']
        X = data['X']
        #matrices 1 y 2 con el resultado de haber entrenado la red neuronal
        weights = loadmat ('ex4weights.mat')
        theta1 = weights['Theta1'] # Theta1 es de dimensión 25 x 401
        theta2 = weights['Theta2'] # Theta2 es de dimensión 10 x 26

In [3]: sample = np.random.choice(X.shape[0] ,100)
        (fig, ax) = displayData.displayData(X[sample])
```



In [4]:
```
num_entradas = 400
num_ocultas = 25
num_etiquetas = 10
```

In [5]:
```
params_rn = np.hstack((theta1.flatten(), theta2.flatten()))
```

In [6]:
```
def get_y_n(y, n):
    y_n = np.zeros([len(y), n])
    for i, v in enumerate(y):
        y_n[i, (v - 1) % n] = 1
    return y_n
```

In [7]:
```
def pesosAleatorios(shape, epsilon=0.12):
    return np.random.uniform(-epsilon, epsilon, shape)
```

In [8]:
```
#función de activación para cada neurona
def g(z):
    return 1 / (1+np.exp(-z))
```

In [9]:
```
def g_prima(z):
    return g(z) * (1 - g(z))
```

In [10]:
```
def ho(X, theta1, theta2, intermediate_data=False):
    (m, n) = X.shape
    A1 = []
    Z2 = []
    A2 = []
    Z3 = []
    HO = []
    for i in range(m):
        a1 = np.insert(X[i,:], 0, 1.)
        A1.append(a1)
        z2 = np.insert(theta1.dot(a1), 0, 1.)
        Z2.append(z2)
        a2 = g(z2)
        A2.append(a2)
        z3 = theta2.dot(a2)
        Z3.append(z3)
        a3 = g(z3)
        HO.append(a3)
    if intermediate_data:
        return A1, Z2, A2, Z3, HO
    else:
        return np.array(HO)
```

In [11]:
```
# reconstruimos las matrices de parámetros a partir del vector params_rn
def params_rn2thetas(params_rn, num_entradas, num_ocultas, num_etiquetas):
```

```
    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))
    )
    return theta1, theta2
```

In [28]:
```
#cálculo del coste
def coste(params_rn, X, y_n, num_entradas, num_ocultas, num_etiquetas, reg):
    theta1, theta2 = params_rn2thetas(params_rn, num_entradas, num_ocultas,
                                      num_etiquetas)
    (m, n) = X.shape
    HO = ho(X, theta1, theta2)
    return -(1/m)*(
        sum(
            y_n[i].dot(np.log(HO[i].transpose()))
            + (1 - y_n[i]).dot(np.log(1 - HO[i].transpose()))
            for i in range(m)
        )
    ) + (reg/(2*m)*(
        (theta1.dot(theta1.transpose())).sum()
        + (theta2.dot(theta2.transpose())).sum())
    )
```

In [29]:
```
coste(params_rn, X, get_y_n(y, num_etiquetas),
      num_entradas, num_ocultas,
      num_etiquetas,
      1
      )
```

Out[29]: 0.35629858469417897

In [30]:
```
#cálculo del gradiente
def gradiente(params_rn, X, y_n, num_entradas, num_ocultas, num_etiquetas, reg):
    theta1, theta2 = params_rn2thetas(params_rn, num_entradas, num_ocultas,
                                      num_etiquetas)
    (m, n) = X.shape
    A1, Z2, A2, Z3, HO = ho(X, theta1, theta2, intermediate_data=True)
    acumulado1 = np.zeros(theta1.shape)
    acumulado2 = np.zeros(theta2.shape)
    for i in range(m):
        sigma3 = HO[i] - y_n[i]
        sigma2 = (theta2.transpose().dot(sigma3))*(g_prima(Z2[i]))

        acumulado1 += np.outer(sigma2[1:], A1[i])
        acumulado2 += np.outer(sigma3, A2[i])
```

```
        D1 = acumulado1/m
        D1[:,1:] += (reg/m)*theta1[:,1:]
        D2 = acumulado2/m + reg*theta2
        D2[:,1:] += (reg/m)*theta2[:,1:]
        gradiente_ = np.hstack((D1.flatten(), D2.flatten()))
        return gradiente_
```

In [31]:
```
gradiente_ = gradiente(params_rn, X, get_y_n(y, num_etiquetas), num_entradas,
                       num_ocultas, num_etiquetas, 1)
print('gradiente_.shape: {}'.format(gradiente_.shape))
```

```
gradiente_.shape: (10285,)
```

In [33]:
```
# backprop devuelve el coste y el gradiente de una red neuronal de dos capas.
def backprop (params_rn, num_entradas, num_ocultas, num_etiquetas, X, y, reg):
    y_n = get_y_n(y, num_etiquetas)
    coste_ = coste(params_rn, X, y_n, num_entradas, num_ocultas, num_etiquetas, reg)
    gradiente_ = gradiente(params_rn, X, y_n, num_entradas, num_ocultas,
                           num_etiquetas, reg)
    return (coste_, gradiente_)
```

In [17]:
```
(c, G) = backprop(params_rn, num_entradas, num_ocultas, num_etiquetas, X, y, 1)
print('coste: {}'.format(c))
print('gradiente.shape: {}'.format(G.shape))
```

```
coste: 0.35629858469417897
gradiente.shape: (10285,)
```

In [18]:
```
#2.2
from checkNNGradients import checkNNGradients
checkNNGradients(backprop, 0)
```

Out[18]:
```
array([ 4.16009310e-11, -4.13422768e-12,  4.33593786e-12,  2.94241229e-11,
       -4.43665382e-11,  1.60028327e-12, -1.74072909e-11, -3.24411054e-11,
       -7.80325804e-11,  3.07328121e-12, -3.51056545e-11, -1.02649306e-10,
       -2.49061674e-11,  3.85858359e-12, -1.47769991e-12, -2.38393263e-11,
        2.22312724e-11, -3.89777902e-12,  1.63397837e-11,  3.05878205e-11,
        5.81194814e-12,  4.82391904e-12,  2.91321134e-12, -1.18985863e-11,
        9.73257586e-12,  1.01122444e-11,  9.06702491e-12,  4.22165081e-12,
        5.06776565e-12, -8.75671757e-12,  9.49704204e-12,  1.11206044e-11,
        9.57764423e-12,  6.32909003e-12,  4.82935913e-12, -1.54280061e-12,
        9.36231648e-12,  1.01169489e-11])
```

In [19]:
```
import scipy.optimize as opt

init_params_rn = pesosAleatorios(params_rn.shape)
```

```
    result = opt.fmin_tnc(
        func=coste,
        x0=init_params_rn,
        fprime=gradiente,
        args=(X, get_y_n(y, num_etiquetas), num_entradas, num_ocultas, num_etiquetas, 1)
    )
    theta_opt = result[0]
```

```
In [22]: #3
         prediccion = ho(X, theta1, theta2)
         (n, m) = X.shape
         nAciertos = 0
         nTotal = 0
         for i in range(n):
             nTotal += 1
             if y[i] == np.argmax(prediccion[i]) + 1:
                 nAciertos += 1
         print(100*nAciertos/nTotal)
```

```
97.52
```