Práctica 4: Entrenamiento de redes neuronales

Nuria Bango Iglesias (nubango@ucm.es)

Álvar Julián de Diego López (alvarded@ucm.es)

1. Función de coste

El objetivo de esta primera parte de la práctica es implementar el cálculo de la función de coste de una red neuronal para un conjunto de ejemplos de entrenamiento

```python
def cost(X, Y, Theta1, Theta2, reg):

    a = -Y*(np.log(X))
    b = (1-Y)*(np.log(1-X))
    c = a - b
    d = (reg/(2*X.shape[0]))* ((Theta1[:,1:]**2).sum() + (Theta2[:,1:]**2).sum())
    return ((c.sum())/X.shape[0]) + d
```

2. Cálculo del gradiente

```python
def forPropagation(X1, Theta1, Theta2):
    m = X1.shape[0]
    a1 = np.hstack([np.ones([m, 1]), X1])
    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, z2, a2, z3, h

def backPropAlgorithm(X, Y, Theta1, Theta2, num_etiquetas, reg):
    G1 = np.zeros(Theta1.shape)
    G2 = np.zeros(Theta2.shape)

    m = X.shape[0]
    a1, z2, a2, z3, h = forPropagation(X, Theta1, Theta2)

    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(Theta2.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 = Theta2
    AuxO2[:, 0] = 0

    G1 = G1/m
    G2 = G2/m + (reg/m)*AuxO2

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

2.1. Comprobación del gradiente

```python
def checkNNGradients(costNN, reg_param):
    """
    Creates a small neural network to check the back propogation gradients.
    Outputs the analytical gradients produced by the back prop code and the
    numerical gradients computed using the computeNumericalGradient function.
    These should result in very similar values.
    """
    # Set up small NN
    input_layer_size = 3
    hidden_layer_size = 5
    num_labels = 3
    m = 5

    # Generate some random test data
    Theta1 = debugInitializeWeights(hidden_layer_size, input_layer_size)
    Theta2 = debugInitializeWeights(num_labels, hidden_layer_size)

    # Reusing debugInitializeWeights to get random X
    X = debugInitializeWeights(input_layer_size - 1, m)

    # Set each element of y to be in [0,num_labels]
    y = [(i % num_labels) for i in range(m)]

    # Unroll parameters
    nn_params = np.append(Theta1, Theta2).reshape(-1)

    # Compute Cost
    cost, grad = costNN(nn_params,
                        input_layer_size,
                        hidden_layer_size,
                        num_labels,
                        X, y, reg_param)

    def reduced_cost_func(p):
        """ Cheaply decorated nnCostFunction """
        return costNN(p, input_layer_size, hidden_layer_size, num_labels,
                      X, y, reg_param)[0]

    numgrad = computeNumericalGradient(reduced_cost_func, nn_params)

    # Check two gradients
    np.testing.assert_almost_equal(grad, numgrad)
    return (grad - numgrad)
```

3. Aprendizaje de los parámetros

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

thetaVec = np.append(Theta1, Theta2).reshape(-1)

result = opt.minimize(fun = backPropagation, x0 = thetaVec,
 args = (n, 25, num_etiquetas, X, AuxY, 1), method = 'TNC', jac = True, options = {'maxiter':70})

Theta1 = np.reshape(result.x[:25*(n + 1)], (25, (n+1)))
Theta2 = np.reshape(result.x[25*(n+1):], (num_etiquetas, (25+1)))

success = neuronalSuccessPercentage(forPropagation(X, Theta1, Theta2)[4], Y)
print("Precisión de la red neuronal: " + str(success) + " %")
```

Entrenando a la red con 70 iteraciones y un valor de$\lambda = 1$deberías obtener una precisión entorno al 93 % (puede variar hasta un 1 % debido a la inicialización aleatoria de los parámetros)

```
(base) C:\Users\alvar\Documents\GitHub\AprendizajeAutomatico\Prácticas\Práctica 4>python Practica4.py
Precisión de la red neuronal: 97.54 %
```