PRÁCTICA 4: ENTRENAMIENTO DE REDES NEURONALES

Aprendizaje Automático y Big Data

22 DE NOVIEMBRE DE 2018

FELIX VILLAR Y VÍCTOR RAMOS
Universidad Complutense de Madrid

1. ***Código***

```
import numpy as np

import scipy.io

import scipy.optimize as opt



def debugInitializeWeights(fan_in, fan_out):
    """
    Initializes the weights of a layer with fan_in incoming connections and
    fan_out outgoing connections using a fixed set of values.
    """

    # Set W to zero matrix
    W = np.zeros((fan_out, fan_in + 1))

    # Initialize W using "sin". This ensures that W is always of the same
    # values and will be useful in debugging.
    W = np.array([np.sin(w) for w in
                  range(np.size(W))]).reshape((np.size(W, 0), np.size(W, 1)))

    return W



def computeNumericalGradient(J, theta):
    """
    Computes the gradient of J around theta using finite differences and
    yields a numerical estimate of the gradient.
    """

    numgrad = np.zeros_like(theta)
```

```
    perturb = np.zeros_like(theta)
    tol = 1e-4

    for p in range(len(theta)):
        # Set perturbation vector
        perturb[p] = tol
        loss1 = J(theta - perturb)
        loss2 = J(theta + perturb)

        # Compute numerical gradient
        numgrad[p] = (loss2 - loss1) / (2 * tol)
        perturb[p] = 0

    return numgrad


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)


def sigmoide(x):
    return 1/(1+ np.exp(np.negative(x)))

def pesosAleatorios(L_in,L_out):
```

```
    ini =0.12
    pesos = np.random.rand((L_in+1)*L_out)*(2*ini) - ini
    pesos = np.reshape(pesos, (L_out,1+L_in))
    return pesos

def sigmoideDerivada(z):
    sd = sigmoide(z) * (1 - sigmoide(z));
    return sd

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)))
    m = X.shape[0]

    #Propagacion hacia delante
    a1 = np.vstack((np.ones(X.shape[0]),X.T))
    z2=np.matmul(Theta1,a1)
    a2=sigmoide(z2)
    a2 = np.vstack((np.ones(a2.shape[1]),a2))
    z3=np.matmul(Theta2,a2)
    a3=sigmoide(z3)
    h = a3

    etiqueta = np.identity(num_etiquetas)
    aux = np.array(y)-1
    ycod = etiqueta[aux,:]
    J = np.sum((-ycod) *np.log(h).T - (1 - ycod) * np.log(1 - h).T)/m

    #Regularizacion
```

```
    regular = (reg/(2*m))*(np.sum(np.square(Theta1[:,1:]))+np.sum(np.square(Theta2[:,1:])))
    final = J+regular

    #Retro propagacion
    d3 = h.T - ycod
    d2 = np.matmul(Theta2.T,d3.T)[1:,:] *sigmoideDerivada(z2)
    grad1 = np.matmul(d2,a1.T)/m
    grad2 = np.matmul(d3.T,a2.T)/m

    #Regularizacion del gradiente
    reg1= (reg/m) * Theta1[:,1:]
    reg2= (reg/m) * Theta2[:,1:]

    #Regularizacion del gradiente
    fingrad1 = grad1
    fingrad1[:,1:] += reg1
    fingrad2 = grad2
    fingrad2[:,1:] += reg2

    #Fin del gradiente
    aux = np.reshape(fingrad1,fingrad1.shape[0]*fingrad1.shape[1])
    aux2 = np.reshape(fingrad2, fingrad2.shape[0]*fingrad2.shape[1])
    grad =np.concatenate((aux,aux2))

    return final,grad
def main():
    weights = scipy.io.loadmat('ex4weights.mat')
    data = scipy.io.loadmat('ex4data1.mat')
    theta1, theta2 = weights['Theta1'], weights['Theta2']
```

```
    y= data['y']
    y= np.reshape(y,y.shape[0])
    X= data['X']
    num_entradas=400
    num_ocultas=25
    num_etiquetas=10

    aux = np.reshape(theta1,(num_entradas+1)*num_ocultas)
    aux2 = np.reshape(theta2,(num_ocultas+1)*num_etiquetas)
    aux3 = np.concatenate((aux,aux2))
    params_rn=aux3
    print("Coste sin regularizar:(lambda=0)")
    J=backprop(params_rn,num_entradas,num_ocultas,num_etiquetas,X,y,0)
    print(J)
    print("Chequeo del gradiente")
    print(np.sum(np.abs(checkNNGradients(backprop, 0))))
    print("Coste con regularizacion:(lambda=1)")
    J=backprop(params_rn,num_entradas,num_ocultas,num_etiquetas,X,y,1)
    print(J)
    print(np.sum(np.abs(checkNNGradients(backprop,1))))

    #Prueba de minimizacion
    aleatheta1=pesosAleatorios(num_entradas,num_ocultas)
    aleatheta2=pesosAleatorios(num_ocultas,num_etiquetas)
    aleat =
np.concatenate((np.reshape(aleatheta1,(num_entradas+1)*num_ocultas),np.reshape(aleatheta
2,(num_ocultas+1)*num_etiquetas)))
    sol=opt.minimize(backprop,aleat,args=(400,25,10,X,y,1),jac=True)

    print(sol)
main()
```

2. Ejemplo de ejecución

```
Coste sin regularizar:(lambda=0)
(0.2876291651613189, array([ 6.18712766e-05,  0.00000000e+00,
0.00000000e+00, ...,
        9.66104721e-05, -7.57736846e-04,  7.73329872e-04]))
Chequeo del gradiente
1.058905681888822e-09
Coste con regularizacion:(lambda=1)
(0.38376985909092365, array([ 6.18712766e-05, -2.11248326e-12,
4.38829369e-13, ...,
        4.70513145e-05, -5.01718610e-04,  5.07825789e-04]))
1.0641309430847734e-09
```