Práctica 4

Javier Berdecio Trigueros
Nicolás Alcaine Camilli

1. Función de coste

```
#Cálculo del coste según las fórmulas explicadas en el enunciado

function [J grad] = costeRN(params_rn, num_entradas, num_ocultas,
num_etiquetas, X, y, lambda)
Theta1 = reshape(params_rn(1:num_ocultas * (num_entradas + 1)),
num_ocultas, (num_entradas + 1));
Theta2 = reshape(params_rn((1 + (num_ocultas  * (num_entradas + 1))):end),
num_etiquetas, (num_ocultas + 1));

#Se obtiene el vector y como vector de booleanos
y = y == [1:num_etiquetas];
A1 = [ones(rows(X), 1) X];
Z2 = A1 * Theta1';
A2 = [ones(size(Z2, 1), 1) sigm(Z2)];
Z3 = A2*Theta2';
h = A3 = sigm(Z3);
#Cálculo del coste con regularización

J = sum(sum((-y).*log(h) - (1-y).*log(1-h))) / rows(X) +
(lambda/(2*rows(X)))*(sum(sum(Theta1(:, 2:end).^2, 2)) +
sum(sum(Theta2(:,2:end).^2, 2)));
endfunction
```

Para lambda = 0.1 se obtiene J = 0.39208

2. Cálculo del gradiente

```
#Calculo de la derivada de la función sigmoide
function dg = sigm_derivada(X)
dg = sigm(X) .* (1 - sigm(X));
end

#Calculo de los pesos aleatorios
function W = pesos_aleatorios(L_in, L_out)
epsilon_init = 0.12;
W = rand(L_out, 1 + L_in) * 2 * epsilon_init - epsilon_init;
end
```

2.1. Retro-propagación

```
#Implementación de sigma y delta para el calculo de gradiente sin regularizacion
Sigma3 = A3 - y;
Sigma2 = (Sigma3 * Theta2 .* sigm_derivada([ones(size(Z2, 1), 1) Z2]))(:, 2:end);
Delta_1 = Sigma2' * A1;
Delta_2 = Sigma3' * A2;
Theta1_grad = Delta_1 /rows(X);
Theta2_grad = Delta_2 /rows(X);

#Calculo final del gradiente uniendo ambas thetas
grad = [Theta1_grad(:) ; Theta2_grad(:)];
```

2.2. Chequeo del gradiente

Para chequear el gradiente obtenido se emplea la función checkNNGradients, y como resultado se obtiene:

```
  -9.2783e-03  -9.2783e-03
   8.8991e-03   8.8991e-03
  -8.3601e-03  -8.3601e-03
   7.6281e-03   7.6281e-03
  -6.7480e-03  -6.7480e-03
  -5.6188e-04  -3.0498e-06
   1.3283e-03   1.4287e-05
   1.9528e-03  -2.5938e-05
```

```
   8.6123e-04   3.6988e-05
  -1.1349e-03  -4.6876e-05
  -2.1750e-03  -1.7506e-04
  -8.4000e-04   2.3315e-04
   5.5287e-04  -2.8747e-04
   2.3165e-03   3.3532e-04
   9.2436e-04  -3.7622e-04
  -6.7207e-04  -9.6266e-05
  -1.8048e-03   1.1798e-04
  -1.6391e-03  -1.3715e-04
   4.5300e-04   1.5325e-04
   1.6593e-03  -1.6656e-04
   3.1454e-01   3.1454e-01
   1.1106e-01   1.1106e-01
   9.7401e-02   9.7401e-02
   1.6258e-01   1.6409e-01
   5.5656e-02   5.7574e-02
   4.9899e-02   5.0458e-02
   1.6588e-01   1.6457e-01
   5.9765e-02   5.7787e-02
   5.1577e-02   5.0753e-02
   1.5725e-01   1.5834e-01
   5.3924e-02   5.5924e-02
   4.8089e-02   4.9162e-02
   1.5197e-01   1.5113e-01
   5.5678e-02   5.3697e-02
   4.8446e-02   4.7146e-02
   1.4899e-01   1.4957e-01
   5.1231e-02   5.3154e-02
   4.5058e-02   4.6560e-02
The above two columns you get should be very similar.
(Left-Your Numerical Gradient, Right-Analytical Gradient)

If your backpropagation implementation is correct, then
the relative difference will be small (less than 1e-9).

Relative Difference: 0.00745284
```

2.3. Redes neuronales regularizadas

```
#Ampliacion del calculo de delta implementando la regularizacion
Theta1_grad = Delta_1 / (rows(X)) + (lambda /
(rows(X)))*[zeros(size(Theta1,1), 1) Theta1(:, 2:end)];
Theta2_grad = Delta_2 / (rows(X)) + (lambda /
(rows(X)))*[zeros(size(Theta2,1), 1) Theta2(:, 2:end)];
```

```
grad = [Theta1_grad(:) ; Theta2_grad(:)];
```

Se vuelve a repetir la ejecucion para chequear el gradiente y se obtiene:

```
  -9.2783e-03  -9.2783e-03
   8.8991e-03   8.8991e-03
  -8.3601e-03  -8.3601e-03
   7.6281e-03   7.6281e-03
  -6.7480e-03  -6.7480e-03
  -5.6188e-04  -5.6188e-04
   1.3283e-03   1.3283e-03
   1.9528e-03   1.9528e-03
   8.6123e-04   8.6123e-04
  -1.1349e-03  -1.1349e-03
  -2.1750e-03  -2.1750e-03
  -8.4000e-04  -8.4000e-04
   5.5287e-04   5.5287e-04
   2.3165e-03   2.3165e-03
   9.2436e-04   9.2436e-04
  -6.7207e-04  -6.7207e-04
  -1.8048e-03  -1.8048e-03
  -1.6391e-03  -1.6391e-03
   4.5300e-04   4.5300e-04
   1.6593e-03   1.6593e-03
   3.1454e-01   3.1454e-01
   1.1106e-01   1.1106e-01
   9.7401e-02   9.7401e-02
   1.6258e-01   1.6258e-01
   5.5656e-02   5.5656e-02
   4.9899e-02   4.9899e-02
   1.6588e-01   1.6588e-01
   5.9765e-02   5.9765e-02
   5.1577e-02   5.1577e-02
   1.5725e-01   1.5725e-01
   5.3924e-02   5.3924e-02
   4.8089e-02   4.8089e-02
   1.5197e-01   1.5197e-01
   5.5678e-02   5.5678e-02
   4.8446e-02   4.8446e-02
   1.4899e-01   1.4899e-01
   5.1231e-02   5.1231e-02
   4.5058e-02   4.5058e-02
The above two columns you get should be very similar.
(Left-Your Numerical Gradient, Right-Analytical Gradient)

If your backpropagation implementation is correct, then
```

```
the relative difference will be small (less than 1e-9).

Relative Difference: 2.38423e-11
```

3. Aprendizaje de los parámetros

```
initial_theta = zeros(n + 1, 1);
options = optimset('GradObj', 'on', 'MaxIter', 50);

for c = 1:10
    [theta] = fmincg (@(t)(costeRN(params_rn, 400, 25, 10, X, y, 1)),
initial_theta, options);
    # Se va guardando en cada fila de all_theta el valor optimizado
    all_theta(c, :) = theta';
end

X = [ones(rows(X), 1) X];
pred = sigm(X * all_theta');

# Se emplea la funcion max para conseguir la clase mas optimizada en cada
ejemplo de entrenamiento
[pred_max, index_max] = max(pred, [], 2);
p = index_max;

# Porcentaje de aciertos
porcentaje = (sum(p == y) / m) * 100
```

Usando lambda = 1 y 50 iteraciones, se obtiene porcentaje = 94.640. Esto se debe a la inicializacion aleatoria de theta1 y theta2 en costeRN por lo que calcula un 1% menos.