APRENDIZAJE AUTOMÁTICO Y BIG DATA

Práctica 4

FDI-UCM
Iván Aguilera Calle – Daniel García Moreno

1. Objetivo

El objetivo de esta quinta práctica es entrenar redes neuronales.

2. Implementación

Para entrenar redes neuronales deberemos implementar una función de coste. El coste se calculará siguiendo la siguiente fórmula, la cual contiene al final un término de regularización:

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

Para poder trabajar con esta expresión necesitaremos utilizar un "y codificada" ($y_k^{(i)}$). Para ello, utilizaremos una matriz identidad del tamaño del número de etiquetas. Nuestra y codificada tendrá una dimensión de 5000x10, donde en cada columna habrá un 0 salvo en el número de la columna que exprese la clase a la que pertenece y. La función e hipótesis mantendrá el cuerpo de las anteriores prácticas, tal y como se puede observar en la figura 1:

```
function h = hipotesis(x, theta)
  h = sigmoide(theta' * x);
endfunction
```

Figura 1

Del mismo modo, la función sigmoide es tal y como se puede observar en la figura 2:

```
function s = sigmoide(z)
  s = 1 ./ (1 + exp(-z));
endfunction
```

Figura 2

Por lo tanto, el cálculo del coste dentro de la primera parte de la implementación de la función de coste queda de la siguiente manera, tal y como se puede observar en la figura 3:

```
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));
    m = rows(X);

    yidentidad = eye(num_etiquetas); #generar matriz identidad
    ycodificada = yidentidad(y, :);

    a1 = [ones(rows(X), 1) X];
    z2 = Theta1 * a1';
    a2 = sigmoide(z2);
    a2 = [ones(1, columns(a2)); a2];
    z3 = Theta2 * a2; %aqui
    a3 = sigmoide(z3);
    h = a3;                                Coste sin término de
                                           regularización.

    J = sum(sum((((-ycodificada) .* log(h)') - ((1 - ycodificada) .* log(1 - h)'))))/m;
    J = J + (lambda/(2 * m)) * (sum(sum(Theta1(:, 2:end) .^ 2)) +
sum(sum(Theta2(:, 2:end) .^ 2)));          Coste con término de
    …                                      regularización.

endfunction
```

Figura 3

Para el cálculo del gradiente en la función de coste, necesitaremos, ahora, la derivada de la función sigmoide, que seguirá la siguiente fórmula:

$$g'(z) = \; g(z)(1 - g(z))$$

La función queda implementada en la figura 4:

```
function sd = sigmoideDerivada(z)
  sd = sigmoide(z) .* (1 - sigmoide(z));
endfunction
```

Figura 4

En la función de coste se aplicará el algoritmo de retro-propagación. Para ello realizaremos una pasada "hacia adelante" para calcular la salida de la red y una pasada "hacia atrás" para calcular el error que cada nodo de cada capa ha generado y acumulado. Para ello seguimos una serie de pasos, tal y como se puede observar en la segunda parte de la implementación de la función de coste, en la figura 5:

```
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));
  X = [ones(rows(X), 1) X];
  m = rows(X);

  a1 = [ones(rows(X), 1) X];
  z2 = Theta1 * a1';
  a2 = sigmoide(z2);
  a2 = [ones(1, columns(a2)); a2];
  z3 = Theta2 * a2; %aqui
  a3 = sigmoide(z3);
  h = a3;
```

1º. Pasada hacia delante

```
  J = sum(sum((((-ycodificada) .* log(h)') - ((1 - ycodificada) .* log(1 - h)'))))/m;
  J = J + (lambda/(2 * m)) * (sum(sum(Theta1(:, 2:end) .^ 2)) + sum(sum(Theta2(:,
2:end) .^ 2)));

  sig3 = a3' - ycodificada;
  sig2 = (Theta2' * sig3') .* sigmoideDerivada([ones(1, columns(z2)); z2]);
```

2º. $\delta^{(3)} = (a_k^{(3)} - y_k)$

3º. $\delta^{(2)} = (\theta^{2})^T \delta^{(3)} .* g'(z^{(2)})$

```
  d1 = sig2(2:end, :) * a1;
  d2 = sig3' * a2';
```

4º. $\Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$

```
  grad1 = ((1/m) * d1);
  grad2 = ((1/m) * d2);
```

Cálculo del gradiente sin término de regularización

```
  grad1(:, 2:end) += (lambda/m) * Theta1(:, 2:end);
  grad2(:, 2:end) += (lambda/m) * Theta2(:, 2:end);
  grad = [grad1(:); grad2(:)];

endfunction
```

Cálculo del gradiente con término de regularización

Figura 5

Como se puede observar en la figura 5, una vez seguidos los pasos, ya podemos calcular el par de gradiente utilizando el término de regularización.

Para comprobar la correcta implementación de la función de coste, vamos a ejecutar los siguientes comandos. Para ello, cargamos los datos con los que vamos a trabajar e inicializamos los parámetros de nuestra red neuronal, tal y como se puede observar en la figura 6:

```
load('ex4data1.mat');
load('ex4weights.mat');

num_entradas = 400;
num_ocultas = 25;
num_etiquetas = 10;
params_rn = [Theta1(:) ; Theta2(:)];
```

Figura 6

Ahora, ejecutamos la función de coste obviando el término de regularización, obteniendo los resultados que podemos observar en la figura 7:

```
%Coste sin regularizar (lambda = 0)
>> J = costeRN(params_rn, num_entradas, num_ocultas, num_etiquetas, X, y, 0)
J =  0.28763
```

Figura 7

Volvemos a ejecutar la función de coste, pero ahora, teniendo en cuenta el término de regularización a la hora de calcular J, con un lambda igual a 1, tal y como se puede observar en la figura 8:

```
%Coste con regularización (lambda = 1)
>> J = costeRN(params_rn, num_entradas, num_ocultas, num_etiquetas, X, y, 1)
J =  0.38377
```

Figura 8

Una vez comprobado el correcto funcionamiento de la función de coste, e implementando el cálculo del gradiente en la misma función, quedando tal y como se observa en la figura 5, ejecutamos una función proporcionada, *checkNNGradients*, para comprobar que los gradientes que se obtienen son correctos, tal y como se puede observar en la figura 9:

```
%Gradiente sin regularizar
>> checkNNGradients
 -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
 -3.0498e-06  -3.0498e-06
……………………………
Relative Difference: 2.37276e-11
```

Figura 9

Ralizamos otra vez el chequeo de los gradientes, teniendo en cuenta el término de regularización, pasándole a la función un lambda igual a 1, tal y como se puede observar en la figura 10:

```
%Gradiente regularizado
>> checkNNGradients(1)
 -0.0092783  -0.0092783
  0.0088991   0.0088991
 -0.0083601  -0.0083601
  0.0076281   0.0076281
 -0.0067480  -0.0067480
 -0.0055914  -0.0055914
……………………………….
      Relative Difference: 2.46967e-11
```

Figura 10

Comprobado que los gradiente obtenidos tiene una diferencia mínima respecto a los gradientes obtenidos a través de la función *computeNumericalGradient*, proporcionada en la práctica, procederemos a entrenar a nuestra red neuronal. Para ello, lo primero que debemos hacer es inicializar una matriz aleatoria de pesos. Para ello, se ha implementado la función *inicilizaPesos*, tal y como se puede observar en la figura 11:

```
function  W = inicializaPesos(L_in, L_out)
  eini = sqrt(6)/sqrt(L_in + L_out); %Deberia dar 0.12
  sup = eini;
  inf = -eini;

  W = [inf + (sup - inf) * rand(L_out, 1 + L_in )]; %Reutilizado de la P1
endfunction
```

Figura 11

Ejecutaremos la función de la figura 11 para la inicialización, tal y como se puede observar en la figura 12:

```
>> Theta1_inicial = inicializaPesos(num_entradas, num_ocultas);
>> Theta2_inicial = inicializaPesos(num_ocultas, num_etiquetas);

>> params_rn_inicial = [Theta1_inicial(:) ; Theta2_inicial(:)];
>> options = optimset('MaxIter', 50);
```

Figura 12

Posteriormente, haremos uso de la función *fmincg* para obtener las Thetas que minimizan la función de coste. Para comprobar el comportamiento de los resultados, llamaremos a la función de coste con distintos valores de lambda. Para poder visualizar el porcentaje de éxito de nuestra red neuronal, haremos uso de una función auxilar *damePorcentaje*, cuya implementación podemos observar en la figura 13:

```
function porcentaje = damePorcentaje(Theta1, Theta2, X, y)
  m = rows(X);

  a1 = [ones(rows(X), 1) X];
  z2 = Theta1 * a1';
  a2 = sigmoide(z2);
  a2 = [ones(1, columns(a2)); a2];
  z3 = Theta2 * a2; %aqui
  a3 = sigmoide(z3);
  h = a3; %10x5000

  [maximo clase] = max(h);

  comparacion = (clase' == y);

  bienPredecidos = length(find(comparacion == 1));

  porcentaje = (bienPredecidos/m) * 100;

endfunction
```

Figura 13

Empezamos con la ejecución con un valor de lambda igual a 1 y 50 iteraciones, tal y como se puede observar en la figura 14:

```
>> costFunction = @(t) costeRN(t,num_entradas, num_ocultas, num_etiquetas,
X, y, 1);
>> [params_rn, J] = fmincg(costFunction, params_rn_inicial, options);
>> Theta11 = reshape(params_rn(1:num_ocultas * (num_entradas + 1)),
num_ocultas, (num_entradas + 1));
>> Theta21 = reshape(params_rn((1 + (num_ocultas * (num_entradas +
1))):end), num_etiquetas, (num_ocultas + 1));
>> pred = damePorcentaje(Theta11, Theta21, X, y)
pred =  95.620
```

Figura 14

Ejecución con lambda igual a 1 y 250 iteraciones, tal y como se puede observar en la figura 15:

```
>>costFunction = @(t) costeRN(t,num_entradas, num_ocultas, num_etiquetas,
X, y, 1);
>>[params_rn, J] = fmincg(costFunction, params_rn_inicial, options);
>>Theta11 = reshape(params_rn(1:num_ocultas * (num_entradas + 1)),
num_ocultas, (num_entradas + 1));
>>Theta21 = reshape(params_rn((1 + (num_ocultas * (num_entradas +
1))):end), num_etiquetas, (num_ocultas + 1));
>>pred = damePorcentaje(Theta11, Theta21, X, y)
pred =  99.420
```

Figura 15

Ejecución con lambda igual a 0 y 50 iteraciones, tal y como se puede observar en la figura 16:

```
>> costFunction = @(t) costeRN(t,num_entradas, num_ocultas, num_etiquetas,
X, y, 0);
>> [params_rn, J] = fmincg(costFunction, params_rn_inicial, options);
>> Theta11 = reshape(params_rn(1:num_ocultas * (num_entradas + 1)),
num_ocultas, (num_entradas + 1));
>> Theta21 = reshape(params_rn((1 + (num_ocultas * (num_entradas +
1))):end), num_etiquetas, (num_ocultas + 1));
>> pred = damePorcentaje(Theta11, Theta21, X, y)
pred =  96.280
```

Figura 16

Ejecución con lambda igual a 5 y 50 iteraciones, tal y como se puede observar en la figura 17:

```
>> costFunction = @(t) costeRN(t,num_entradas, num_ocultas, num_etiquetas,
X, y, 5);
>> [params_rn, J] = fmincg(costFunction, params_rn_inicial, options);
>> Theta11 = reshape(params_rn(1:num_ocultas * (num_entradas + 1)),
num_ocultas, (num_entradas + 1));
>> Theta21 = reshape(params_rn((1 + (num_ocultas * (num_entradas +
1))):end), num_etiquetas, (num_ocultas + 1));
>> pred = damePorcentaje(Theta11, Theta21, X, y)
pred =  94.040
```

Figura 17

Tras estas ejecuciones consecutivas, podemos deducir que a igual lambda y mayor número de iteraciones se incrementa el porcentaje de aciertos. Si solo modificamos lambda, mayor lambda, menor porcentaje de aciertos.