Práctica 4: Entrenamiento de redes neuronales

Gabriel Sellés Salvà

Manuel Antonio Fernández Alonso

Aprendizaje Automático UCM 2018

1-Función de coste & retro-propagación

Para la resolución de este apartado hemos implementado una función que calcula el coste y el gradiente asociado a una red neuronal. La función se llama **costeRN**. Para calcular el gradiente, hemos utilizado el algoritmo de retro-propagación.

La implementación de esta función se encuentra a continuación:

```
#Implementa la función de coste de una red neuronal (con regularización).

function [J,grad] = costeRN (params_rn, num_entradas, num_ocultas, num_etiquetas, X,y,lambda)
  #Añadimos la columna de unos a X
  X = [ones(rows(X),1),X];

  #Obtenemos Thetal y Theta2.
  Thetal= 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));

  #Modificamos y para que cumpla los requisitos que se explican en el enunciado.
  Y=zeros(rows(y),num_etiquetas);
  for i=1: rows(y)
    Y(i,y(i))= 1;
  endfor

  #Primero calculamos el coste de la red neuronal.
  J=J_REG(X,Y,Thetal,Theta2,1);

  #Después, el gradiente.
  #Debe tener la misma forma que params_rn
  grad=retro_propagacion(X,Y,Thetal,Theta2,lambda);

endfunction
```

Esta función utiliza una serie de funciones (también implementadas por nosotros) para realizar dicha tarea:

- **sigmoide:** implementa la función sigmoide.

```
#FUNCIÓN SIGMOIDE.
#REUTILIZADA DE LA PRÁCTICA 2.
function result = sigmoide(Z)
  result = (1+ exp(Z*(-1))).^(-1);
endfunction
```

- **derivadaSigmoide:** calcula la derivada de la función sigmoide.

```
function result= derivadaSigmoide(Z)
  result=sigmoide(Z).*(1-sigmoide(Z));
endfunction
```

- **hipothesisRN:** calcula (h_0(X)), el resultado de los cálculos de la red neuronal.

```
#Reutilizada de la práctica 3.
function result= hipothesisRN(X,Thetal,Theta2)

  a_1= X ;     #Add a_0(1). Ya tiene añadidos los unos.

  a_2=sigmoide(a_1*Thetal');  #Matriz resultante de dimensiones 5000x25
  a_2=[ ones(rows(a_2),1) a_2 ];  #add a_0(2). Matriz resultante de dimensiones 5000x26

  a_3=sigmoide(a_2*Theta2'); # Matriz resultante de dimensiones 5000x10

  result=a_3;


endfunction
```

- **J_NR**: realiza el cálculo del coste, sin regularización.

```
#Cálculo del coste, sin añadir el término de regularización.

function res= J_NR(X,y,Thetal,Theta2)
  res=  sum(sum((-y).*log(hipothesisRN(X,Thetal,Theta2)) - (1-y).*(log(1-hipothesisRN(X,Thetal,Theta2))))/rows(y);
endfunction
```

- **J_REG:** realiza el cálculo del coste, con regularización.

```
#Cálculo del coste, añadiendo el término de regularización.

function result= J_REG(X,y,Thetal,Theta2,lambda)
  result=  sum(sum((-y).*log(hipothesisRN(X,Thetal,Theta2)) - (1-y).*(log(1-hipothesisRN(X,Thetal,Theta2))))/rows(X);
  result=result +  (sum(sum((Thetal(:,2:end)).^2)) +sum(sum((Theta2(:,2:end)).^2)))*(lambda/(rows(y)*2));
endfunction
```

- **retro_propagacion:** calcula el gradiente utilizando el algoritmo de retro-propagación.

```
function result= retro_propagacion(X,Y, Theta1,Theta2,lambda)
  #Igual que la función hipothesisRN.
  a_1=X ;    #Add a_0(1).Ya tiene añadidos los unos.

  a_2=sigmoide(a_1*Theta1');  #Matriz resultante de dimensiones 5000x25
  a_2=[ ones(rows(a_2),1) a_2 ];  #add a_0(2). Matriz resultante de dimensiones 5000x26

  a_3=sigmoide(a_2*Theta2'); # Matriz resultante de dimensiones 5000x10

  #Aplicamos las fórmulas del propio enunciado.
  sigma3= a_3-Y; #Ya se ha realizado el arreglo a Y.
  sigma2=(sigma3*Theta2)(:,2:end);
  sigma2= sigma2.*derivadaSigmoide(a_1*Theta1');
  delta1= sigma2' * X;
  delta2= sigma3' * a_2;

  #Calculo del gradiente sin regularizar.
  grad1= delta1/rows(Y);
  grad2= delta2/rows(Y);

  #Regularizamos el gradiente.
  Theta1(:,1)=0;
  Theta2(:,1)=0;
  grad1=grad1+(lambda/rows(Y))*Theta1;
  grad2=grad2+(lambda/rows(Y))*Theta2;

  #El resultado será una sola columna que contendrá ambos gradientes.
  result=[grad1(:); grad2(:)];
endfunction
```

- **pesosAleatorios:** Genera un vector con valores aleatorios entre -e_ini y e_ini (ambos inclusive).

```
function W= pesosAleatorios (L_in, L_out)
  e_ini=0.12;
  W=rand(L_out, L_in+1); #Generamos una matriz de vectores con numeros entre 0 y 1.

  #https://octave.sourceforge.io/octave/function/mod.html
  #https://stackoverflow.com/questions/3680637/generate-a-random-double-in-a-range
  W=mod(W,2*e_ini) - e_ini;
endfunction
```

2-Aprendizaje de los parámetros de la red neuronal

Hechas las funciones del apartado anterior, hemos implementado otra para el entrenamiento de la red neuronal, **aprendizajeRN**. Esta función calcula cuáles son los parámetros para los que nuestra red neuronal tiene un menor coste (respecto a los datos de entrenamiento) y devuelve el porcentaje de acierto que tiene con esos parámetros, utilizando dichos datos de entrenamiento.

La implementación de esta función se encuentra a continuación:

```
function porcentajeAcierto=aprendizajeRN(lambda)

  num_entradas=400;
  num_ocultas=25;
  num_etiquetas=10;

  load('ex4data1.mat'); #Los resultados se almacenan en X e y.

  Theta1=pesosAleatorios(num_entradas, num_ocultas);
  Theta2=pesosAleatorios(num_ocultas,num_etiquetas);

  #Igual que en el propio enunciado.
  options= optimset('MaxIter',50);
  cost = @(p) costeRN(p, num_entradas,num_ocultas,num_etiquetas,X,y,lambda);
  [params_rn, cost]= fmincg(cost, [Theta1(:);Theta2(:)],options);


  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));

  #Añadimos los unos a X.
  X = [ones(rows(X),1),X];
  #Con los siguientes cálculos, calculamos las probabilidades.
  probabilities= hipothesisRN(X,Theta1,Theta2);
  [aux, predictions] = max(probabilities,[],2);
  porcentajeAcierto=mean(predictions==y)*100;
endfunction
```

Con este sistema implementado, los resultados utilizando lambda = 1 son los siguientes:

```
>> aprendizajeRN(1)
Iteration    50 | Cost: 4.736171e-001
ans =  95.620
```

Y utilizando lambda = 0.5

```
>> aprendizajeRN(0.5)
Iteration    50 | Cost: 4.860174e-001
ans =  95.120
```

Por último, usando lambda = 0.1

```
>> aprendizajeRN(0.1)
Iteration    50 | Cost: 5.325567e-001
ans =  94.900
```