Práctica 4: Entrenamiento de Redes neuronales

Grupo 13:
David Ortiz Fernández.
Andrés Ortiz Loaiza.

En esta práctica retomamos el guion de la práctica anterior. Comenzamos cargando el fichero de datos, que consiste nuevamente en 5000 imágenes de 20x20 pixeles de tamaño, cada uno de los cuales almacena la intensidad de dicho pixel en la escala de grises.
Se ha desarrollado una función que da pesos iniciales aleatorios cercanos a cero a las thetha iniciales para un correcto funcionamiento.
Posteriormente se ha desarrollado el código de una función destinada a calcular el coste y el gradiente realizando la propagación hacia delante y posteriormente la retropropagación hacia atrás para minimiza esta función de coste (costeRN).
Después de esto, mediante la invocación de la función "checkNNGradients", se ha comparado que el cálculo de gradientes mediante los dos métodos es similar, lo cual nos indica el buen funcionamiento de este.
A continuación, con estos valores se han obtenido los parámetros de la red neuronal óptimos, es decir sus thetas óptimos y de esta manera se puede calcular la probabilidad de pertenencia.
Se adjuntan dichas funciones entre cuyos comentarios se puede apreciar paso a paso:

1. Función de coste y cálculo de gradiente

```
function [J grad] = costeRN(params_rn, num_entradas, num_ocultas,num_etiquetas,
                            X, y, lambda)
%reconstruir las matrices de parámetros a partir del vector params_rn
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));

% variables utililes
J = 0;
Theta1_grad = zeros(size(Theta1));
Theta2_grad = zeros(size(Theta2));
m = size(X, 1);

% para poder entrenar la red neuronal es necesario codificar las etiquetas como
%vectores de 10 componentes con todos sus elementos a 0 salvo uno a 1
I = eye(num_etiquetas);
Y = zeros(m, num_etiquetas);
for i=1:m
  Y(i, :)= I(y(i), :);
end

% feedforward
a1 = [ones(m, 1) X];
z2 = a1*Theta1';
a2 = [ones(size(z2, 1), 1) fsigmoide(z2)];
z3 = a2*Theta2';
a3 = fsigmoide(z3);
```

```
h = a3;

% calculo de la penalizacion
p = sum(sum(Theta1(:, 2:end).^2, 2))+sum(sum(Theta2(:, 2:end).^2, 2));

% calculo de J, si lambda es 0 se devolvera sin el termino regularizado
J = sum(sum((-Y).*log(h) - (1-Y).*log(1-h), 2))/m + lambda*p/(2*m);

% calculo de sigmas
sigma3 = a3.-Y;
sigma2 = (sigma3*Theta2).*fsigmoideGradiente([ones(size(z2, 1), 1) z2]);
sigma2 = sigma2(:, 2:end);

%calculo de deltas acumuladas
delta_2 = (sigma3'*a2);
delta_1 = (sigma2'*a1);

%si lambda es 0 calculo se devolvera el gradiente sin regularizar ya que depende
%de lambda
p1 = (lambda/m)*[zeros(size(Theta1, 1), 1) Theta1(:, 2:end)];
p2 = (lambda/m)*[zeros(size(Theta2, 1), 1) Theta2(:, 2:end)];
Theta1_grad = delta_1./m + p1;
Theta2_grad = delta_2./m + p2;

% el gradiente se devuelve desplegado como un vector columna
grad = [Theta1_grad(:) ; Theta2_grad(:)];
endfunction
```

FsigmoideGradiente:

```
%función que calcula la derivada del sigmoide tanto para un valor concreto,
%como para un vector o una matriz en cuyo caso lo aplicará a cada uno de sus
%componentes.
function F = fsigmoideGradiente(z)

F = zeros(size(z));

F  = fsigmoide(z).*(1-fsigmoide(z));

endfunction
```

Función fsigmoide :

```
%función sigmoide
function g = fsigmoide(z)

g = 1.0 ./ (1.0 + exp(-z));

endfunction
```

```
function [J grad] = costeRN(params_rn, num_entradas, num_ocultas,num_etiquetas,
                            X, y, lambda)
%reconstruir las matrices de parámetros a partir del vector params_rn
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));

% variables utililes
J = 0;
Theta1_grad = zeros(size(Theta1));
Theta2_grad = zeros(size(Theta2));
m = size(X, 1);

% para poder entrenar la red neuronal es necesario codificar las etiquetas como
%vectores de 10 componentes con todos sus elementos a 0 salvo uno a 1
I = eye(num_etiquetas);
Y = zeros(m, num_etiquetas);
for i=1:m
  Y(i, :)= I(y(i), :);
end

% feedforward
a1 = [ones(m, 1) X];
z2 = a1*Theta1';
a2 = [ones(size(z2, 1), 1) fsigmoide(z2)];
z3 = a2*Theta2';
a3 = fsigmoide(z3);
```

Flujo principal de la práctica:

```
clear;
close all;
load('ex4data1.mat');
m = size(X, 1);
load('ex4weights.mat');
% Theta1 es de dimensión 25 x 401
% Theta2 es de dimensión 10 x 26

%desenrrollado de parametros
params_rn = [Theta1(:) ; Theta2(:)];

num_entradas  = 400;  % 20x20
num_ocultas = 25;
num_etiquetas = 10;

%al hacer el termino regularizado dependiente de lambda en la función costeRN
%introduciendo lambda=0 calcularemos el coste sin el termino regularizado
lambda = 0;
J = costeRN(params_rn, num_entradas, num_ocultas,num_etiquetas, X, y, lambda);

printf('Coste sin regularizar: %f\n', J);

%al hacer el termino regularizado dependiente de lambda en la función costeRN
%introduciendo lambda=1 calcularemos el coste con el termino regularizado
lambda = 1;
J = costeRN(params_rn, num_entradas, num_ocultas,num_etiquetas, X, y, lambda);
printf('Coste regularizado: %f\n' , J);
printf('Presione enter para continuar.\n');
pause;


fprintf('Inicializando parametros\n')
Theta1_inicial = pesosAleatorios(num_entradas, num_ocultas );
Theta2_inicial = pesosAleatorios(num_ocultas , num_etiquetas);
% desenrrollado de parametros
params_rn_iniciales = [Theta1_inicial(:) ; Theta2_inicial(:)];
printf('Presione enter para continuar.\n');
pause;

%al hacer el termino regularizado dependiente de lambda en la función costeRN
%introduciendo lambda=0 como parametro de la función checkNNGradients haremos el
%chequeo del gradiente sin el termino regularizado
printf("Chequeo del gradiente sin regularizar. \n")
checkNNGradients(0);
printf('Presione enter para continuar.\n');
pause;
```

```
%al hacer el termino regularizado dependiente de lambda en la función costeRN
%introduciendo lambda>0 como parametro de la función checkNNGradients haremos el
%chequeo del gradiente con el termino regularizado
printf("Chequeo del gradiente regularizado. \n")
checkNNGradients(1);
printf('Presione enter para continuar.\n');
pause;

%Aprendizaje de los parámetros
printf('Entrenando la red con 50 iteraciones y un valor de lambda=1.\n');


%jugar con los valores  de lambda e iteraciones  para ver diferentes resultados
lambda = 1;
opciones = optimset('MaxIter', 50);

funcionCoste = @(p) costeRN(p,num_entradas,num_ocultas ,num_etiquetas,X,y,lambda)
[params_rn, cost] = fmincg(funcionCoste, params_rn_iniciales, opciones);

% obtenemos Thetal y Theta 2 como en la practica anterior
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));

printf('Presione enter para continuar.\n');
pause;

%calculo de la rpecisión de la red una vez entrenada
p = zeros(size(X, 1), 1);

h1 = fsigmoide([ones(m, 1) X] * Thetal');
h2 = fsigmoide([ones(m, 1) h1] * Theta2');
[pmaxi, p] = max(h2, [], 2);

printf('Precisión: %f\n', mean(double(p == y)) * 100);
```

Función PesosAleatorios :

```
%a una función que inicialice una matriz de pesos T(l) con valores aleatorios en
%el rango [-epsilon_ini, epsilon_ini].
function W = pesosAleatorios(L_in, L_out)

W = zeros(L_out, 1 + L_in);
epsilon_ini = 0.12;

%W debe tener tamaño (L_out, 1 + L_in)
W = rand(L_out, 1 + L_in) * 2 * epsilon_ini - epsilon_ini;

endfunction
```

Resultados :

Coste sin regularizar (con lambda 0): 0.287629
Coste regularizado (con lambda 1): 0.383770

Chequeo del gradiente sin regularizar: 2.27415e-011
Chequeo del gradiente regularizado:2.29153e-011

Precisión: 95.36000

Tras el entrenamiento de la red neuronal se ha obtenido así una precisión muy alta, en torno al 95%, lo cual indica la alta probabilidad de predicción correcta de la red neuronal desarrollada. También se ha podido apreciar, gracias a la función “checkNNGradients”, el correcto funcionamiento de la propagación hacia delante y retropropagación, al descender en cada iteración el gradiente de manera correcta, esto se consigue comparando los valores obtenidos al aproximar los cálculos obtenidos al aplicar las derivadas parciales con el método de aproximación de la pendiente.