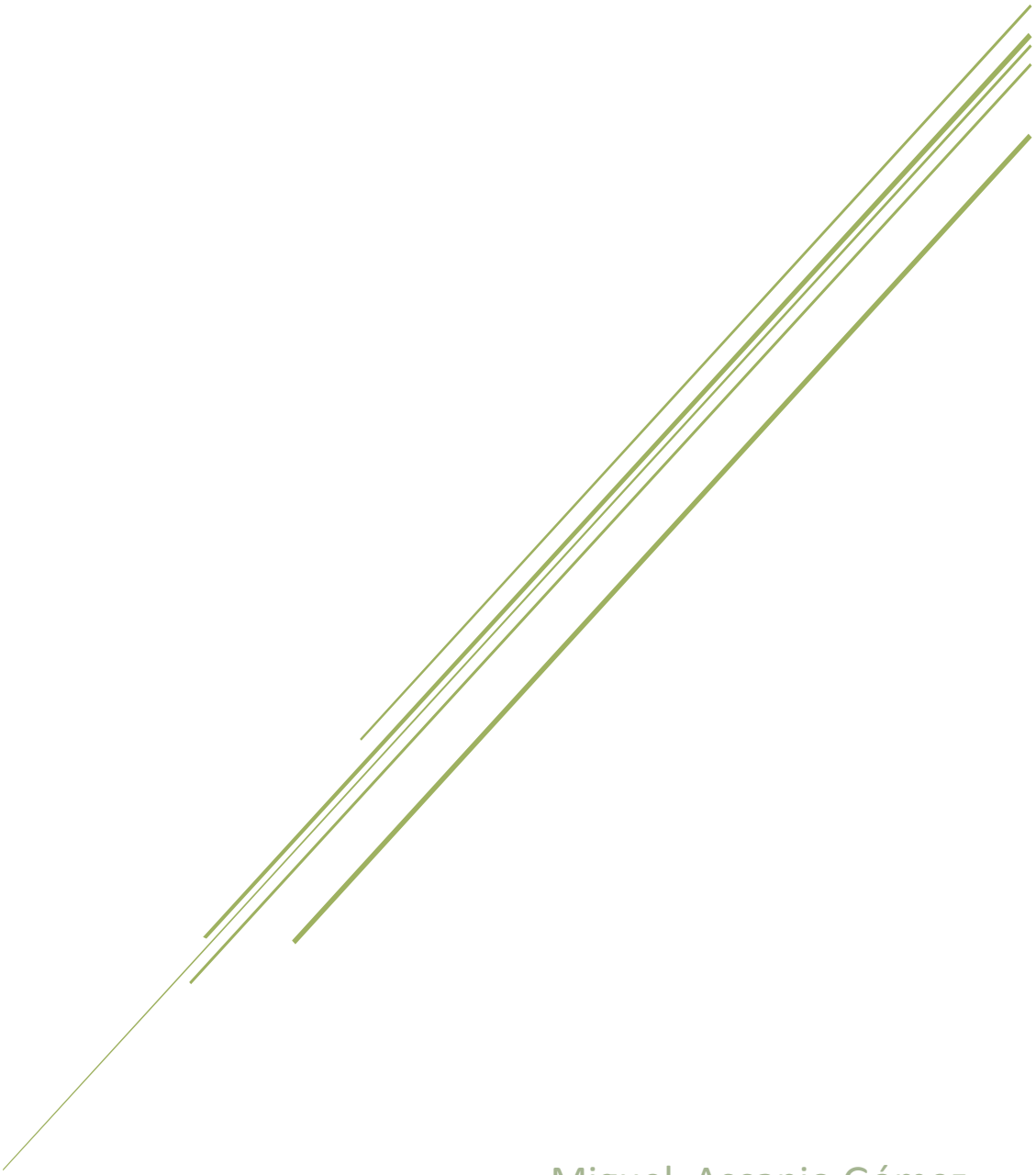
APRENDIZAJE AUTOMÁTICO

Práctica 4



Miguel Ascanio Gómez

Esther Ávila Benito

En esta práctica se ha implementado en Octave el entrenamiento de una red neuronal, así como pruebas de su funcionamiento.

FUNCIÓN DE COSTE

SIN REGULARIZAR

```
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(length (X), 1), X];

y = transformaEtiquetas(y, num_etiquetas)';

[a1, a2, z2, valoresRed] = red(Theta1, Theta2, X);

sumInterno = (0.-y) .* log(valoresRed) - (1.-y) .* log(1 .- valoresRed);
m = length(X(:,1));

Theta12 = Theta1(1:num_ocultas,2:end).^ 2;
Theta22 = Theta2(1:num_etiquetas,2:end) .^ 2;

J = (1 / m) * sum(sum(sumInterno)) ;
```

TÉRMINO DE REGULARIZACIÓN

```
regularizacion = (lambda / (2*m)) * ((sum(sum(Theta12)) + sum(sum(Theta22))));
```

El término anterior se sumaría a J, y obtendríamos el coste regularizado.

IMPLEMENTACIÓN DE TRANSFORMA ETIQUETAS

```
function r = transformaEtiquetas(y, numEtiquetas)

	r = zeros(length(y), numEtiquetas);

	for i = 1:size(y)
		r(i,y(i)) = 1;
	endfor

endfunction
```

Esta función transforma los valores de y, en el rango 1-num etiquetas, para adaptarlas a la regresión logística, devolviendo por cada valor de y un vector con num-etiquetas posiciones, con un uno en la posición i si i=y

IMPLEMENTACIÓN DE RED

```
function [a1, a2, z2, a3] = red(Theta1, Theta2, X)

a1 = X';

z2 = Theta1 * a1;
a2 = sigmoide(z2);
a2 = [ones(1, length (a2)); a2];

z3 = Theta2 * a2;
a3 = sigmoide(z3);

endfunction
```

Esta función devuelve los valores a1, a2, z2 y a3 que se obtienen al pasar la red neuronal caracterizada por las matrices de pesos Theta1 y Theta 2 los valores X

CÁLCULO DEL GRADIENTE

DERIVADA DEL SIGMOIDE

```
function r = derSigmoide(z)
	s = sigmoide(z);
	#s = [ones(1, length (s)); s];
	r = s .* (1 - s);
endfunction
```

Esta función calcula los valores de la derivada de la función sigmoide sobre los valores z

PESOS ALEATORIOS

```
function W = pesosAleatorios(L_in, L_out)

W = rand(L_out, L_in + 1) .* 0.24 .- 0.12;

Endfunction
```

RETRO-PROPAGACIÓN

```
delta3 = valoresRed - y;
grad2 = delta3 * a2';

aux = (Theta2' * delta3);
aux = aux(2:end,:);
delta2 = aux .* derSigmoide(z2);
grad1 = delta2 * a1';

grad1 = grad1 .* (1/m);
grad2 = grad2 .* (1/m);

grad = [grad1(:); grad2(:)];
```

El código anterior realiza la retro-propagación, obteniendo el gradiente para la red, quedano finalmente desplegados en el vector grad.

Al comprobar el funcionamiento del gradiente con la función checkNNGradients.m, se obtiene un resultado de 10^-11, lo cual indica que la implementación es correcta.

REGULARIZACIÓN

Para añadir el término de regularización al gradiente, añadimos el siguiente código a lo anterior:

```
Theta1(:,1) = 0;
Theta2(:,1) = 0;

grad1 = grad1 .* (1/m) .+ (lambda/m) .* Theta1;
grad2 = grad2 .* (1/m) .+ (lambda/m) .* Theta2;

grad = [grad1(:); grad2(:)];
```

Al igual que el anterior, se obtiene un resultado de 10^-11

APRENDIZAJE Y PRUEBAS

Se ha utilizado el siguiente código para el entrenamiento de la red, así como para la prueba de precisión sobre los datos de entrenamiento:

```
load("ex4data1.mat");

num_entradas = 400;
num_ocultas = 25;
num_etiquetas = 10;
iters = 50;
lambda = 1;

ThetaIni1 = pesosAleatorios(num_entradas,num_ocultas);
ThetaIni2 = pesosAleatorios(num_ocultas,num_etiquetas);

thetaIni = [ThetaIni1(:); ThetaIni2(:)];

opciones = optimset ('Gradobj', 'on', 'MaxIter', iters);
func = @(p)(costeRN(p, num_entradas, num_ocultas, num_etiquetas, X, y, lambda));

[theta, cost] = fmincg (func, thetaIni, opciones);

Theta1 = reshape(theta(1:num_ocultas * (num_entradas + 1)), num_ocultas,
(num_entradas + 1));
Theta2 = reshape(theta((1 + (num_ocultas * (num_entradas + 1))):end), num_etiquetas,
(num_ocultas + 1));

m = size(X, 1);
X = [ones(length (X), 1), X];
[a1, a2, z2, resultado] = red(Theta1, Theta2, X);

aciertos = 0;
for i = 1:m
	iterActual = resultado(:,i);
	mx = max(iterActual);
	if(find(iterActual == mx) == y(i))
		aciertos++;
	endif
endfor

printf("Porcentaje de aciertos: %f\n", aciertos / m * 100);
```

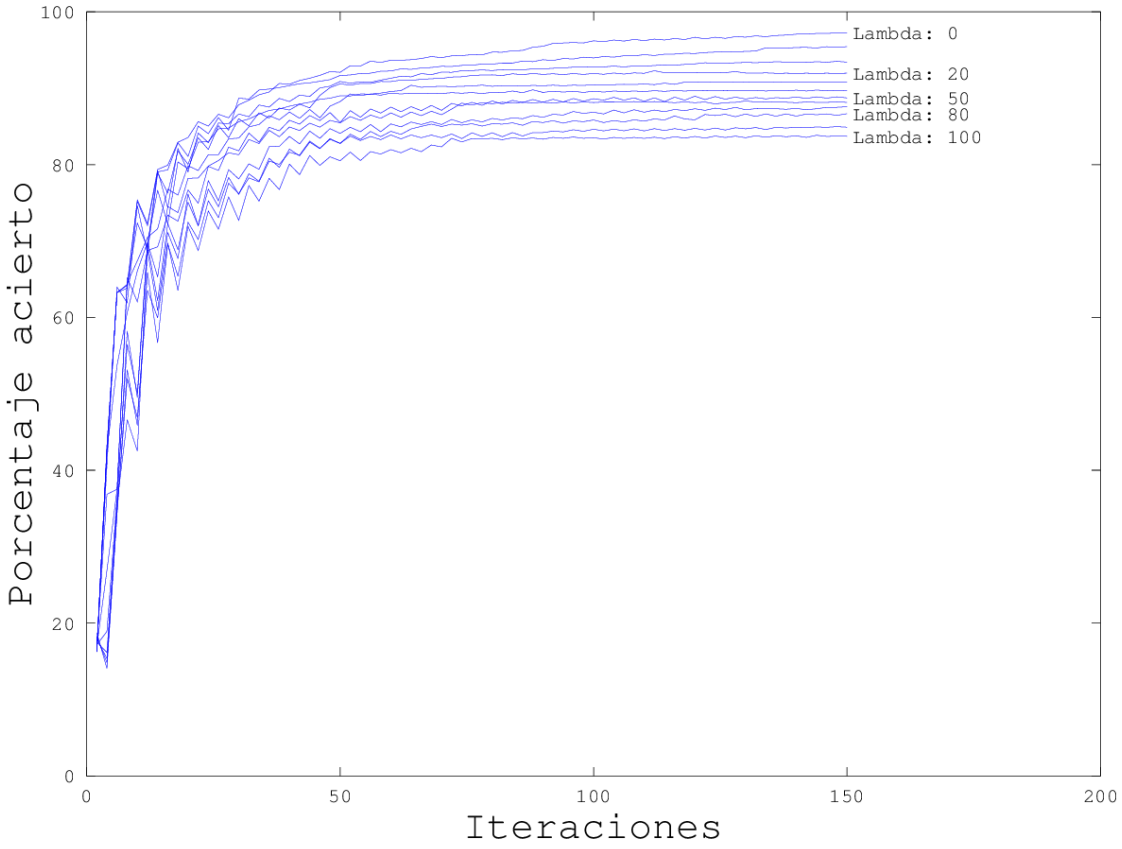
El código anterior muestra una precisión del 96% sobre los ejemplos de entrenamiento.

DISTINTOS VALORES DE LAMBDA E ITERACIONES

Ejecutando el siguiente código realizamos varias pasadas al entrenamiento, cambiando el valor de lambda y las iteraciones de fmincg:

```
load("ex4data1.mat");

num_entradas = 400;
num_ocultas = 25;
num_etiquetas = 10;
itersStep = 2;
numItersMax = 150;
lambda = 1;
m = size(X, 1);
XconUnos = [ones(length (X), 1), X];

ThetaIni1 = pesosAleatorios(num_entradas,num_ocultas);
ThetaIni2 = pesosAleatorios(num_ocultas,num_etiquetas);

thetaIni = [ThetaIni1(:); ThetaIni2(:)];

opciones = optimset ('Gradobj', 'on', 'MaxIter', itersStep);
func = @(p)(costeRN(p, num_entradas, num_ocultas, num_etiquetas, X, y, lambda));

thetaProc = thetaIni;

lambdaPos = 1;
iterPos = 1;


for lambda = sort([1, linspace(0,100,11)]);
        thetaProc = thetaIni;
        func = @(p)(costeRN(p, num_entradas, num_ocultas, num_etiquetas, X, y, lambda));
        iterPos = 1;
        for step = itersStep:itersStep:numItersMax

                [thetaProc, cost] = fmincg (func, thetaProc, opciones);

                ThetaProc1 = reshape(thetaProc(1:num_ocultas * (num_entradas + 1)),
                        num_ocultas, (num_entradas + 1));
                ThetaProc2 = reshape(thetaProc((1 + (num_ocultas * (num_entradas + 1))):end),
                        num_etiquetas, (num_ocultas + 1));

                [a1, a2, z2, resultado] = red(ThetaProc1, ThetaProc2, XconUnos);

                aciertos = 0;
                for i = 1:m
                        iterActual = resultado(:,i);
                        mx = max(iterActual);
                        if(find(iterActual == mx) == y(i))
                                aciertos++;
                        endif
                endfor

                printf("Lmabda: %f Iteraciones: %d Porcentaje de aciertos: %f\n", ...
                        lambda, step, aciertos / m * 100);

                #Preparar para siguiente ietr
                thetaProc = [ThetaProc1(:); ThetaProc2(:)];
                dataSave(lambdaPos, iterPos).lambda = lambda;
                dataSave(lambdaPos, iterPos).step = step;
                dataSave(lambdaPos, iterPos).porcentaje =  aciertos / m * 100;
                [coste, grad] = func(thetaProc);
                dataSave(lambdaPos, iterPos).coste = coste;

                iterPos++;
        endfor
        lambdaPos++;
endfor
save dataSave dataSAve
```

Representando los datos obtenidos:



Se observa que a partir de 50 iteraciones, el porcentaje de aciertos casi no aumenta, independientemente del valor de lambda. Para valores anteriores, se observa entre iteraciones gran diferencia de los resultados, esto es resultado de los ajustes que se van produciendo en los pesos de la red neuronal. Para cualquier lambda, a partir de unas 80 iteraciones es despreciable.

Respecto al valor de lambda, se observa que al aumentarlo disminuye el porcentaje de aciertos, esto es debido a la regularización, ya que al aumentar lambda hacemos que se ajuste menos a los ejemplos de entrenamiento, para evitar el *overfittig*.