Práctica 4

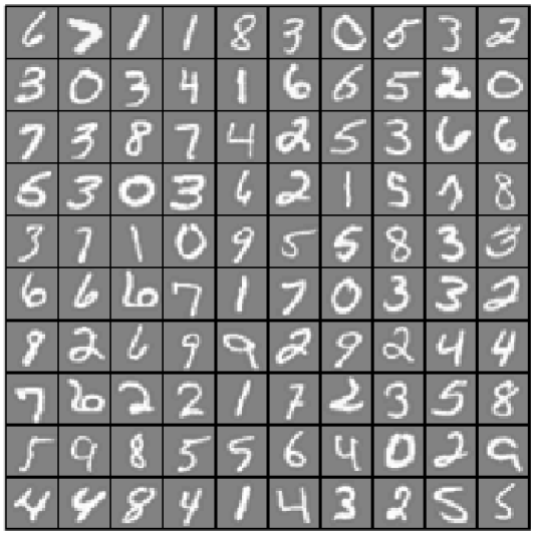
Pablo Arranz Ropero
Juan Alberto Camino Sáez
Grupo 2

Práctica 4: Entrenamiento de redes neuronales

En esta práctica, que se divide en dos apartados, se trata de realizar el entrenamiento de una red neuronal con un conjunto de datos consistente en 5000 imágenes de 20x20 píxeles (que serán nuestros "atributos" donde cada píxel está representado por un número que representa la intensidad de ese píxel en escala de grises. Los datos están en el fichero ex4data1.mat que cargan directamente los datos en X e y, guardándose en y el número al que pertenece ese mapa de píxeles (del 0 al 9, representando el 0 como un 10).

Primeramente, hemos visualizado 100 datos aleatorios en pantalla con el siguiente código.

```
1. m = size(X, 1);
2.
3. rand_indices = randperm(m);
4. sel = X(rand_indices(1:100), :);
5.
6. displayData(sel);
```



Las matrices $\Theta^{(1)}$ y $\Theta^{(2)}$ se almacenan en Theta1 y Theta2 con la instrucción:

```
1. load("ex4weights.mat");
```

Después hemos implementado la función costeRN que aplicando Theta1 y Theta2 comprobamos que da un coste de 0.287 con $\lambda = 0$ y 0.383 con $\lambda = 1$

La función que calcula la función de coste y el valor de los gradientes tendrá el siguiente código:

```
1. function [J grad] = costeRN(params_rn, num_entradas,
   num_ocultas, num_etiquetas, X, y, lambda)
```

```
2.
3.    X(:,2:columns(X)+1) = X;
4.    X(:,1) = ones(rows(X),1);
5.
6.    Theta1 = reshape(params_rn(1:num_ocultas * (num_entradas
   + 1)), num_ocultas, (num_entradas + 1));
7.    Theta2 = reshape(params_rn(1 + (num_ocultas * (num_entradas
   + 1)):end), num_etiquetas, (num_ocultas + 1));
8.    #load("ex4weights.mat");
9.
10.         for i = 1: num_etiquetas
11.           aux(:, i) = y == i;
12.         endfor
13.
14.         #Forward-propagation
15.
16.         y = aux;
17.
18.         a1 = X;
19.         z2 = a1 * Theta1';
20.         a2 = sigmoide(z2);
21.
22.         #Añadimos bias unit
23.         a2(:,2:columns(a2)+1) = a2;
24.         a2(:,1) = ones(rows(a2),1);
25.
26.         z3 = a2 * Theta2';
27.         a3 = sigmoide(z3);
28.
29.         valory0 = (-y .* log(a3));
30.         valory1 = ((1 - y) .* log(1-a3));
31.
32.         J = (1/rows(X)) * sum(sum(valory0 - valory1));
33.         #quitar col
34.
35.         reg
   = (lambda/(2*rows(X))) * (sum(sum(Theta1(:, 2:end) .^ 2)) + s
   um(sum((Theta2(:, 2:end) .^ 2))));
36.         J = J + reg;
37.
38.         #Backpropagation
39.         sigma3 = a3 - y;
40.         sigma2 = Theta2(:, 2:end)'*sigma3' .*
   derivadasig(z2)';
41.         delta2 = sigma3' * a2;
42.         delta1 = sigma2 * a1;
43.
44.         delta1(:,2:end) = delta1(:,2:end) + lambda *
   Theta1(:,2:end);
45.         delta2(:,2:end) = delta2(:,2:end) + lambda *
   Theta2(:, 2:end);
46.
47.         grad = [((delta1 ./ rows(X)) (:)) ; ((delta2 ./
   rows(X))(:))];
48.
49.      endfunction
```

Con los gradientes calculados por esta función hemos llamado a *checkNNGradients* para comprobar que el cálculo de los gradientes con el método de la derivada y su aproximación con el cálculo numérico es similar. La diferencia era de, aproximadamente, 10^{-11} por lo que consideramos nuestros gradientes como correctos ya que la diferencia debería ser menor a 10^{-9}. Además, se han probado distintos lambas, entre los que se encuentran 0, 1 y 0.1, siendo la diferencia menor que 10^{-9}.

Usamos la siguiente función para calcular los thetas óptimos:

```
1. function [all_theta] = getThetas (X, y, lambda)
2.
3.   initial_theta = [pesosAleatorios(401, 25)(:) ;
   pesosAleatorios(26, 10)(:)];
4.   options = optimset ('GradObj', 'on', 'MaxIter', 50);
5.
6.     all_theta = fmincg(@(t)(costeRN(t, 400, 25, 10, X, y,
   lambda)), initial_theta, options);
7.
8. endfunction
```

Con una lambda de 1 y 50 iteraciones, obtenemos un valor mínimo de la función de coste de 4.84×10^{-1}. Para obtener las probabilidades hemos creado esta función que con una entrada y las thetas devueltas por la función anterior devuelve la clase a la que tiene más probabilidades de pertenecer y la probabilidad.

```
1. function [probs, class] = forwardprop(X, theta, num_entradas,
   num_ocultas, num_etiquetas)
2.
3.   Theta1 = reshape(params_rn(1:num_ocultas * (num_entradas
   + 1)), num_ocultas, (num_entradas + 1));
4.   Theta2 = reshape(params_rn(1 + (num_ocultas * (num_entradas
   + 1)):end), num_etiquetas, (num_ocultas + 1));
5.
6.   #Suponemos que viene añadida la bias unit
7.   z2 = Theta1 * X';
8.   a2 = sigmoide(z2);
9.
10.        #añadimos bias unit
11.        a2(2:rows(a2)+1,:) = a2;
12.        a2(1,:) = ones(columns(a2),1);
13.
14.        z3 = Theta2 * a2;
15.        h = sigmoide(z3);
16.
17.        [probs, class] = max(h);
18.
19.     endfunction
```

Esto nos devuelve un vector de m columnas con las probabilidades de cada caso i de ser lo predicho en la columna i del vector class.

Esta función la combinamos con esta otra que aplica la clasificación a todos los ejemplos de entrenamiento para comprobar cuantos clasifica incorrectamente:

```
function [num_wrongs] = checkTrainingCasesNN(X, theta,
num_entradas, num_ocultas, num_etiquetas, y)
    num_wrongs = 0;
    [probs, class] = forwardprop(X, theta, num_entradas,
num_ocultas, num_etiquetas);
    for i = 1:rows(X)
        if (class(1,i) == y(i))
            printf("class: %d , probs: %f \t &#10003; \n",
class(1, i), probs(1, i));
        else
            printf("class: %d , probs: %f , should be: %d
(index: %d) \n", class(1, i), probs(1, i), y(i), i);
            num_wrongs += 1;
             endif
         endfor
     endfunction
```

En nuestro caso, clasifica incorrectamente 231 elementos, es decir, que tiene una precisión de
(5000 – 231) / 5000 ≈ 95.38%

Al ejecutarlo más veces, clasifica incorrectamente en mayor o menor cantidad (debido a la inicialización aleatoria de las thetas iniciales), pero el porcentaje de todas las ejecuciones oscila en torno al 95%.