Práctica 4: Entrenamiento de redes neuronales

Esta práctica consiste en aplicar el cálculo de la función de coste de una red neuronal para un conjunto de ejemplos de entrenamiento, que en este caso son los mismos que hemos empleado en la práctica 3.

Después implementaremos el algoritmo de retro-propagación, añadiendo el cálculo del gradiente a la función de coste.

Por último, entrenaremos la red neuronal y obtendremos los valores tetha1 y tetha2, pudiendo ver cómo cambia la precisión en función del valor de landa y el número de iteraciones.

Función de coste

Al igual que en la práctica 3, empelamos los ejemplos de entrenamiento que son imágenes que representan números reales del 0 al 9 escritos a mano.

Eligiremos 100 ejemplos aleatoriamente, que serán los que pintaremos mediante la función displayData.

Luego almacenaremos en Theta1 y Theta2 los valores de los pesos.

Para implementar la función de coste empezamos desplegando los parámetros, que utilizaremos en el cálculo del coste de una red neuronal sin regularizar. Y al que luego añadiremos el término de regularización.

Podemos comprobar como tanto el valor de coste y el valor del coste regularizado para los pesos de “ex4weights . mat” son los que deberían:

```
J =  0.28763
J =  0.38377
```

Cálculo del gradiente

Debemos devolver en la variable “grad” el cálculo del gradiente. Para ello ejecutamos un bucle que procese todos los ejemplos de entrenamiento, ejecutando una pasada hacia delante, calculando el valor de la capa de salida, y una pasada hacia atrás acumulando el gradiente empleando la fórmula indicada. Finalmente se divide por m los valores acumulados para calcular el gradiente.

Una vez realizada la función de coste, tenemos que realizar varias pruebas para comprobar que los valores que devuelve son correctos.

```
-9.2783e-003  -9.2783e-003
8.8991e-003  8.8991e-003
-8.3601e-003  -8.3601e-003
7.6281e-003  7.6281e-003
-6.7480e-003  -6.7480e-003
-3.0498e-006  -3.0498e-006
1.4287e-005  1.4287e-005
-2.5938e-005  -2.5938e-005
3.6988e-005  3.6988e-005
-4.6876e-005  -4.6876e-005
-1.7506e-004  -1.7506e-004
2.3315e-004  2.3315e-004
-2.8747e-004  -2.8747e-004
3.3532e-004  3.3532e-004
-3.7622e-004  -3.7622e-004
-9.6266e-005  -9.6266e-005
1.1798e-004  1.1798e-004
-1.3715e-004  -1.3715e-004
1.5325e-004  1.5325e-004
-1.6656e-004  -1.6656e-004
3.1454e-001  3.1454e-001
1.1106e-001  1.1106e-001
9.7401e-002  9.7401e-002
1.6409e-001  1.6409e-001
5.7574e-002  5.7574e-002
5.0458e-002  5.0458e-002
1.6457e-001  1.6457e-001
5.7787e-002  5.7787e-002
5.0753e-002  5.0753e-002
1.5834e-001  1.5834e-001
5.5924e-002  5.5924e-002
4.9162e-002  4.9162e-002
1.5113e-001  1.5113e-001
5.3697e-002  5.3697e-002
4.7146e-002  4.7146e-002
1.4957e-001  1.4957e-001
5.3154e-002  5.3154e-002
4.6560e-002  4.6560e-002
```

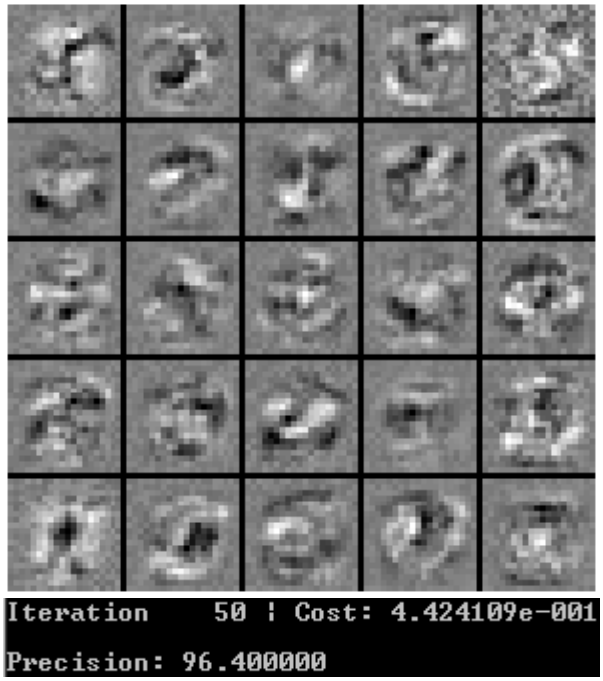
* Como podemos ver, los resultados son correctos.

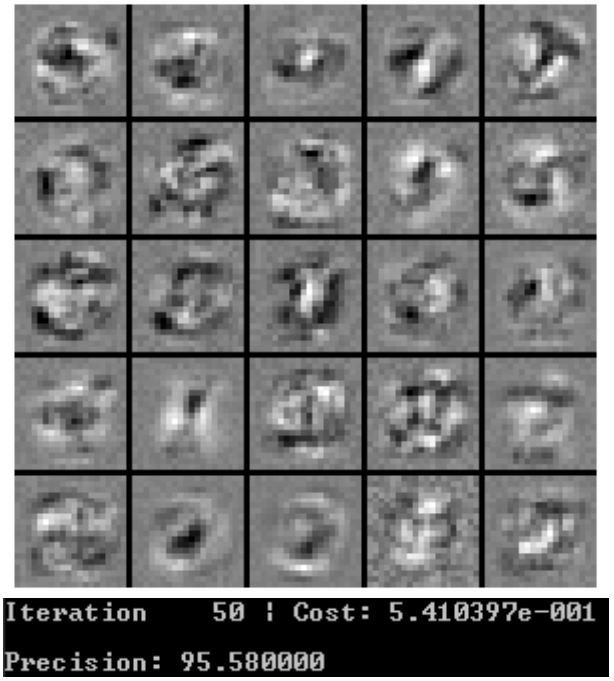
```
If your backpropagation implementation is correct, then
the relative difference will be small (less than 1e-9).

Relative Difference: 2.17385e-011
```

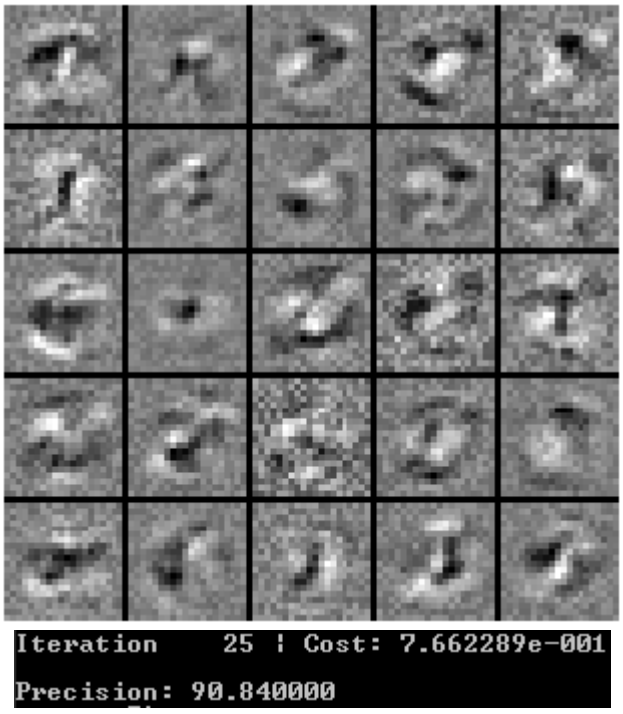
Aprendizaje de los parámetros

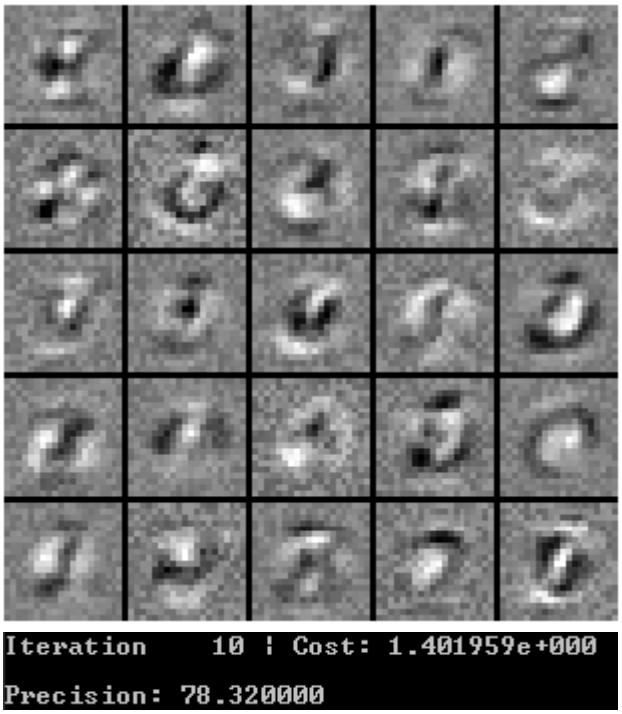
Para entrenar la red neuronal empezamos inicializando aleatoriamente los pesos, con una función que hemos creado. Entonces es cuando empleamos la función fmincg para obtener los valores de Theta1 y Theta2.

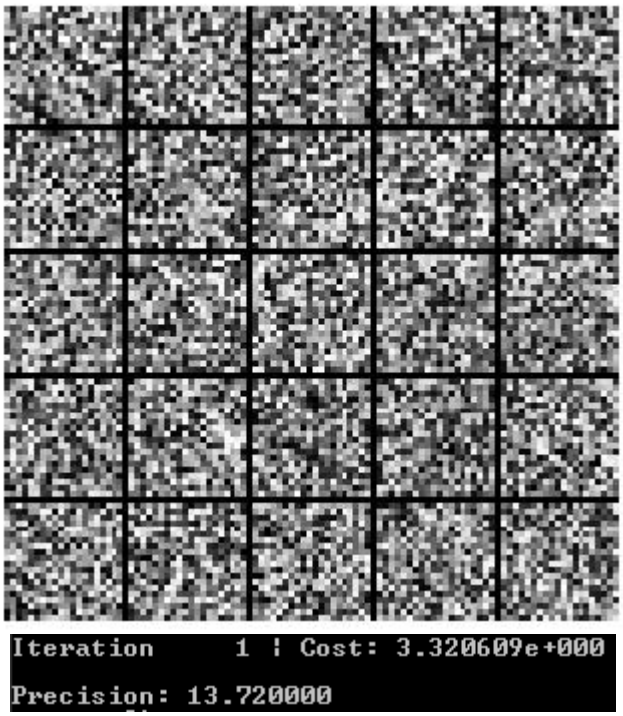
Por último, podemos visualizar el resultado de la red neuronal, y la precisión que se consigue en cada caso. Aquí mostramos el "resultado" de la red neuronal, y la precisión conseguida para distintos valores de número de repeticiones:

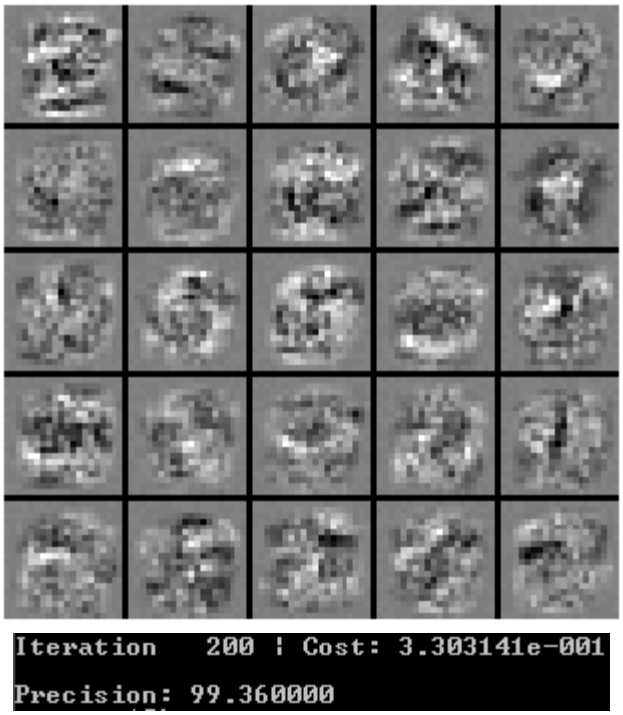












Como se puede comprobar, a mayor número de iteraciones mayor precisión.