PRACTICA IC: RED NEURONAL

Manuel Blanco Rienda (manuelbr994@gmail.com)

PRÁCTICA 1 Inteligencia Computacional

Índice

Descripción de la realización de la práctica

Para llevar a cabo la implementación de esta práctica se ha utilizado el lenguaje Java como medio de programación de la red neuronal, a través del IDE Eclipse. No se han usado librerías de terceros ni ningún tipo de implementación externa a Java que tratase las redes neuronales: se ha desarrollado todo el código de esta práctica de forma original, siguiendo el algoritmo de aprendizaje descrito en los apuntes de la asignatura.

Descripción de la implementación realizada

Para realizar la implementación de la red neuronal se ha optado por una estructura de tres capas codificadas en java: no se tienen objetos neurona ni nada parecido, tan solo objetos “CapaNeuronas” que tienen cierta cantidad de ellas y se relacionan con las demás capas. La primera de las capas hace las veces de entrada a la red, y consta de 28*28 neuronas de una sola entrada, las cuales sólo se dedican a propagar su propia entrada (ya normalizada antes de llegar a la red) hacia la siguiente capa: la oculta.

La capa oculta consta de 256 neuronas de tipo “sigmoidal” que propagan la salida calculada hacia la capa final de salida que tiene 10 neuronas. Cada una de las neuronas de la capa de salida (también de tipo sigmoidal) hace referencia a uno de los dígitos clasificados por la red (del 0 al 9). Aquella neurona de la capa de salida que obtiene la salida más alta representa el número que la red detecta a través de las entradas que le han llegado.

Cómo método de aprendizaje se ha implementado “BackPropagation” siguiendo los apuntes y fórmulas matemáticas proporcionados por el profesor. Para mejorar los resultados obtenidos con esta forma de aprendizaje, se han añadido las épocas y los momentos a este proceso.

Por último, para guardar los pesos de las neuronas de todas las capas de cara a poder cargar la red neuronal sin que tenga que volver a aprender, se han implementado dos métodos: “guardaPesos” y “leePesos”, que guardan los mencionados datos en un fichero de texto plano y los recuperan, respectivamente.

Resultados obtenidos

Caso 1:

Se ha utilizado la topología de red explicada anteriormente, pero el entrenamiento se ha ejecutado a lo largo de cuarenta épocas, con una tasa de aprendizaje del 0.01% y un momento del 0.9%. El proceso completo de aprendizaje ha durado 2 horas.

Resultado 2:
- 2.7% de error sobre el conjunto de prueba.
- 0.7% de error sobre el conjunto de entrenamiento.

Caso 2:

Se ha utilizado la topología de red explicada anteriormente, pero el entrenamiento se ha ejecutado a lo largo de treinta épocas, con una tasa de aprendizaje del 0.05% y un momento del 0.9%. El proceso completo de aprendizaje ha durado 1 hora y media.

Resultado 2:
- 2.26% de error sobre el conjunto de prueba.
- 0.38% de error sobre el conjunto de entrenamiento.

Caso 3:

Se ha utilizado la topología de red explicada anteriormente, pero el entrenamiento se ha ejecutado a lo largo de quince épocas, con una tasa de aprendizaje del 0.1% y un momento del 0.9%. El proceso completo de aprendizaje ha durado 45 minutos (ya que se ha usado Java como lenguaje de implementación).

Resultado 1:
- 2.46% de error sobre el conjunto de prueba.
- 0.73% de error sobre el conjunto de entrenamiento.

Caso 4:

Se ha utilizado la topología de red explicada anteriormente, pero el entrenamiento se ha ejecutado a lo largo de treinta y cinco épocas, con una tasa de aprendizaje del 0.1% y un momento del 0.9%. El proceso completo de aprendizaje ha durado 1 hora y media.

Resultado 2:
- 2.02% de error sobre el conjunto de prueba.
- 0.25% de error sobre el conjunto de entrenamiento.

Caso 5:

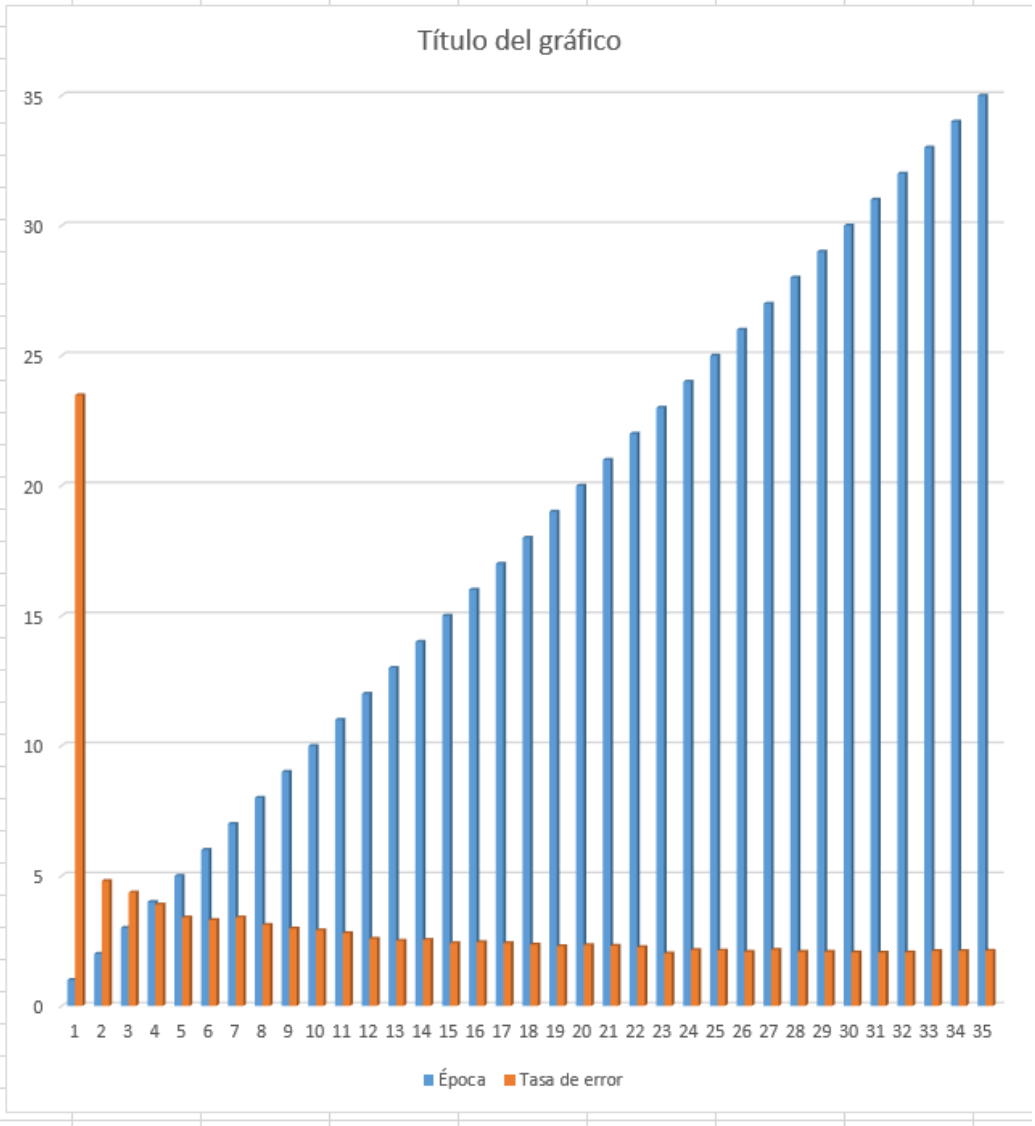
Se ha utilizado la topología de red explicada anteriormente, pero el entrenamiento se ha ejecutado a lo largo de veinticinco épocas, con una tasa de aprendizaje del 0.1% y un momento del 0.9%. El proceso completo de aprendizaje ha durado 1 hora.

Resultado 2:
- 2.08% de error sobre el conjunto de prueba.
- 0.36% de error sobre el conjunto de entrenamiento.

Comparando los casos 5 y 6 se puede observar que se obtienen mejores resultados con menos épocas cuando la tasa de aprendizaje es 0.1, hasta cierto punto, ya que con quince épocas, apenas logra terminar el aprendizaje todo lo profundamente que debería con esta tasa (tal y como puede verse en el caso 4).

Por todo ello, se escogen los parámetros del caso 4 como finales para la práctica y a continuación puede verse la gráfica que representa la oscilación de la tasa de error con dichos números:

Época	Tasa de error
1	23
2	4,8
3	4,36
4	3,9
5	3,4
6	3,3
7	3,4
8	3,12
9	2,98
10	2,91
11	2,8
12	2,58
13	2,51
14	2,54
15	2,41
16	2,45
17	2,41
18	2,36
19	2,3
20	2,34
21	2,31
22	2,26
23	2,02
24	2,15
25	2,13
26	2,08
27	2,16
28	2,08
29	2,08
30	2,06
31	2,05
32	2,06
33	2,11
34	2,11
35	2,12



Se ve claramente como la tasa de error es inversamente proporcional al incremento en el número de época ejecutada. Aunque a partir de la época 25 existe cierto sobre-aprendizaje, dado que se observa un determinado retroceso hacia tasas de error superiores a la mínima ya alcanzada a partir de la época 23. Sin embargo, tal y como se ha dicho, con el algoritmo de "backpropagation" implementado y las técnicas aplicadas es la mejor solución que se puede obtener con la arquitectura definida: tres capas (de entrada, una oculta y una de salida) con 28*28, 256 y 10 neuronas respectivamente.

NOTA: para poder comprobar que el aprendizaje realizado con los datos anteriormente mencionados da los resultados que se han descrito, se adjunta el fichero "pesos_backup" (dentro de la carpeta de los archivos fuente del proyecto) que almacena los pesos que deben cargar las neuronas de esta arquitectura para llegar a estas conclusiones. Para probarlo solo hay que poner la variable booleana "quieroAprender" del método main de la clase "Main.java" a "false", comentar la línea del mismo método que ejecuta la función aprender y des-comentar la llamada al método ejecutar. Para probar el aprendizaje sólo hay que deshacer estos cambios en el main.

NOTA 2: Los archivos fuente están en una arquitectura de proyecto del IDE Eclipse, que puede ser fácilmente importada al IDE Netbeans o al propio Eclipse.

Bibliografía

[1] - Apuntes de Fernando Berzal sobre Redes Neuronales y BackPropagation de la asignatura Inteligencia computacional del Master en Ingeniería Informática de Granada.