Práctica 2 de Neurocomputación

Introducción a las Redes Neuronales Artificiales

Ares Aguilar Sotos
Daniel Giménez Llorente

Tabla de Contenidos

1. Introducción

En esta práctica nos marcamos como objetivo estudiar el algoritmo de backpropagation en redes neuronales con una capa oculta.

A la hora de definir nuestra solución, hemos continuado con la práctica anterior, añadiéndole las funcionalidades requeridas en esta prácticas. Estas funcionalidades son, por ejemplo, que la capa de salida contenga varias neuronas, que se pueda crear una capa oculta o por último normalizar los datos.

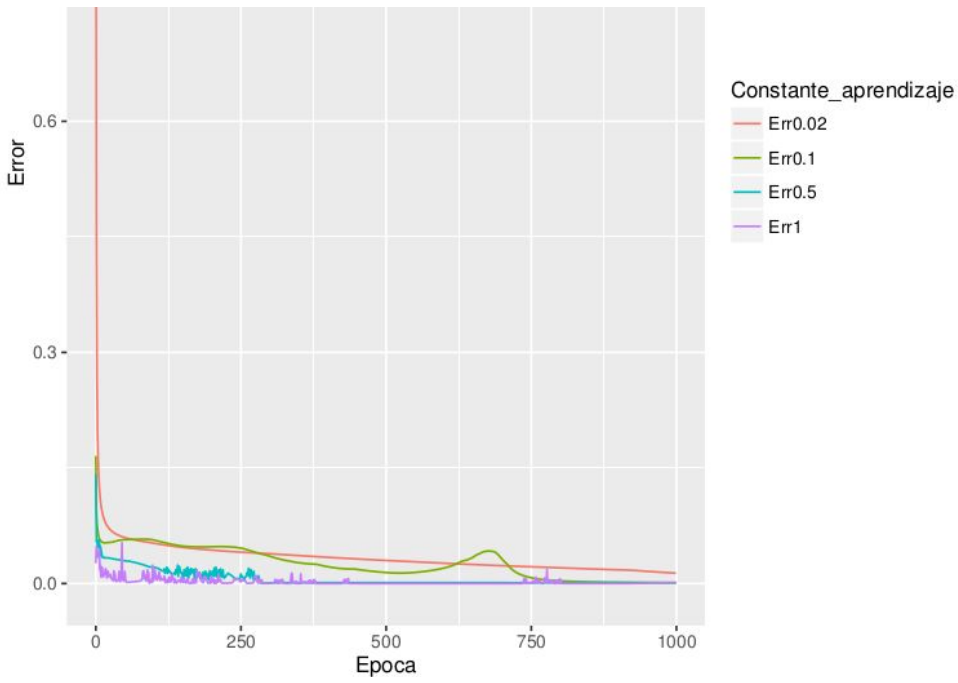
2. Funcionamiento de la red en los ficheros anteriores

Para cada uno de los ficheros de la práctica anterior se han probado diferentes combinaciones del número de neuronas en la capa oculta y de la constante de aprendizaje.

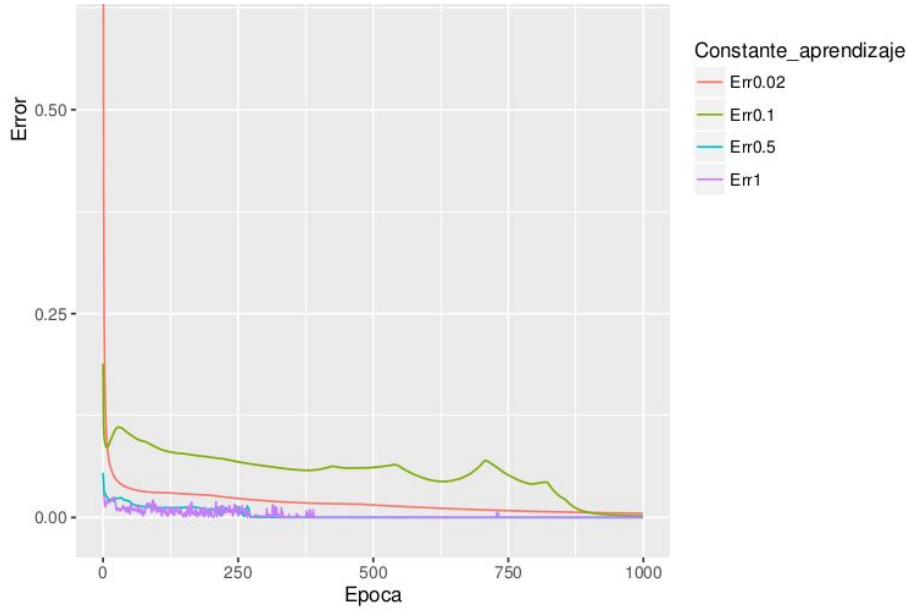
2.1 Problema real 1

En las dos gráficas se muestra el error cuadrático medio para 4 y para 9 neuronas. Cada una de las distintas líneas hacen referencia a la constante de aprendizaje, que puede tomar de valores: 0.02, 0.1, 0.5, o 1.

4 Neuronas en la capa oculta



9 Neuronas en la capa oculta



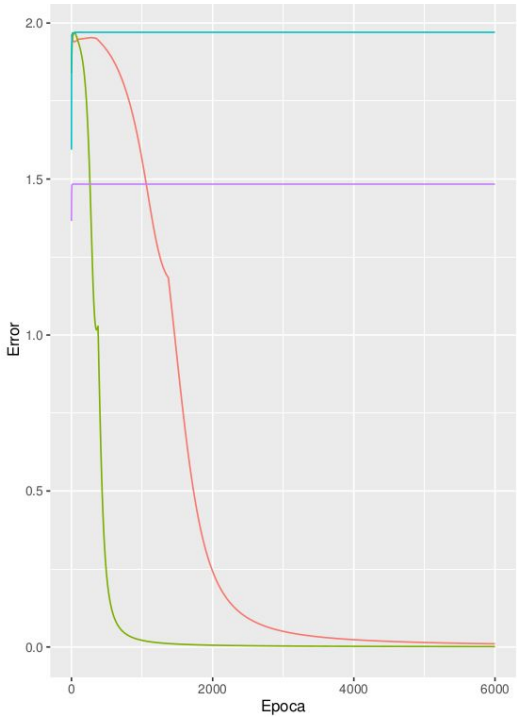
Este problema se resolvía bien usando el Adaline o el Perceptron, (tenían una tasa de un 95%, excepto con tasas de aprendizaje muy altas, donde el Adaline salía perdiendo. Si k comparamos con los resultados en la capa oculta son parecidos, aunque a veces este último consigue un 96 o 97% de acierto.
Si nos fijamos en el número de neuronas no se aprecia mucha diferencia, pero en el error cuadrático si que podemos ver algunas conclusiones. El multicapa puede usar tasas de aprendizaje altas porque clasifica bastante bien este problema, converge relativamente rápido. En el caso del Adaline y el Perceptron, los errores solían ser mayores. También se pueden observar oscilaciones cuando la tasa de aprendizaje es demasiado alta (de 1) que indica que no al principio el error crece y decrece hasta que se estabiliza.
Los objetivos para ejecutar estas simulaciones en el Makefile son, respectivamente, p2.2.1-multilayer4 y p2.2.1-multilayer9.

2.2 Puertas lógicas

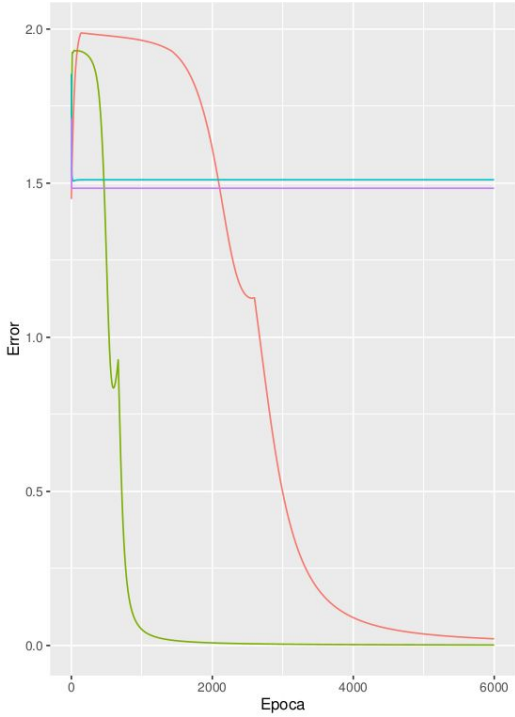
Solo vamos a estudiar el caso de la puerta xor, ya que en el resto de casos los resultados son iguales (son separables). Para este ejercicio no eran suficientes las 1000 épocas, ya que solo contiene 4 muestras cada época. Hemos realizado la prueba para 6000 épocas con 4 y 9 neuronas. En el caso de 9 neuronas, la convergencia con 0.02 y 0.1 está asegurada, mientras que en el caso de 4 neuronas, alguna vez fallaba con la tasa de 0.1.

En el Makefile se debe llamar al objetivo p2.2.2-xor para ejecutar esta simulación.

ECM 4 Neuronas



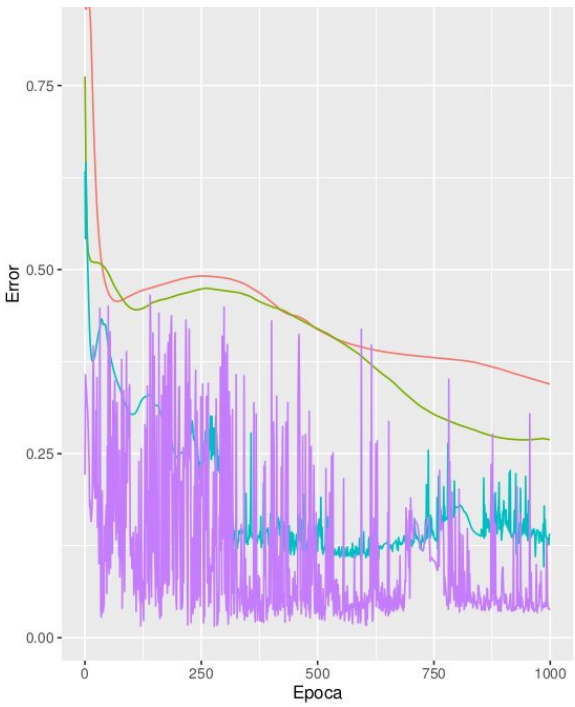
ECM 9 Neuronas



2.3 Problema 2

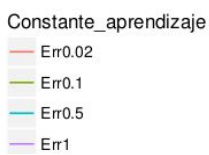
Hemos repetido el análisis para el problema 2 de la práctica anterior. Las gráficas asociadas a este problema son:

ECM 4 Neuronas



ECM 9 neuronas

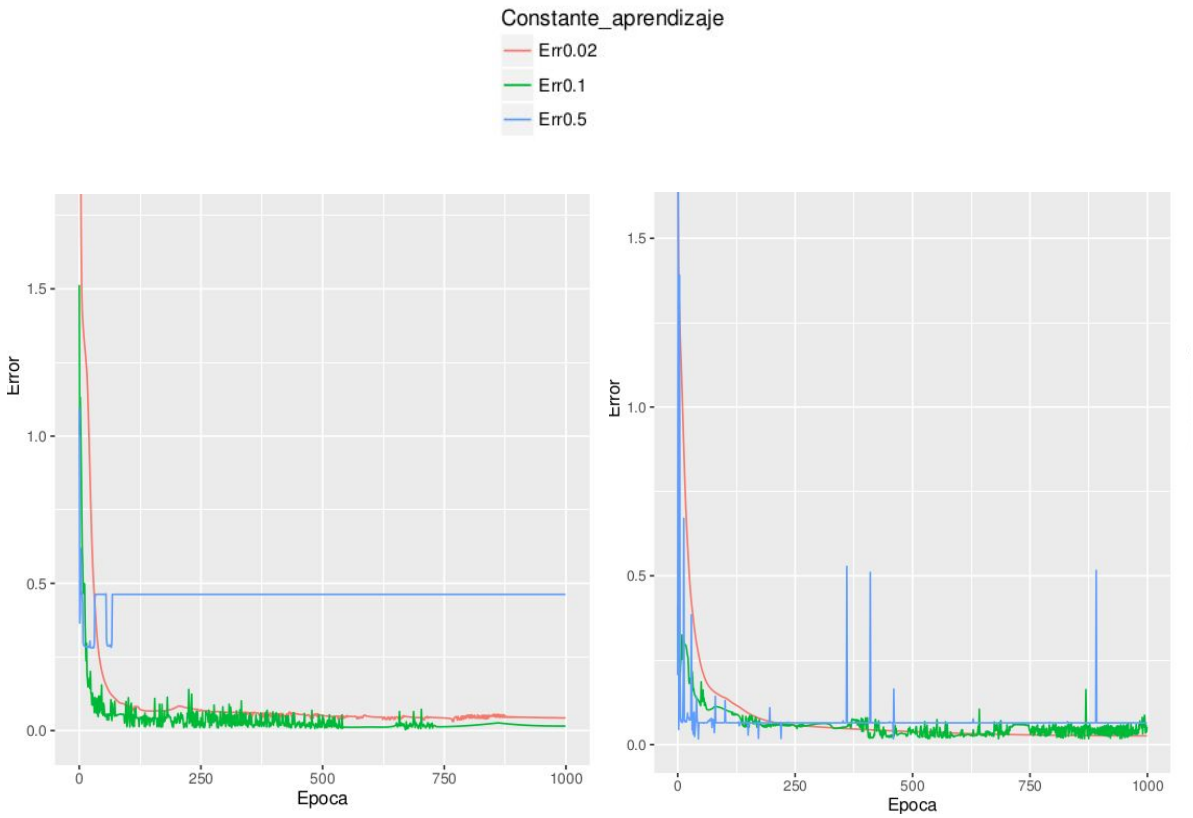
En este caso el Adaline estaba en un 75% mientras que el del Perceptron era un poco más bajo. En el caso del multicapa, con 4



neuronas con la tasa de aprendizaje de 0.02 y 0.1 se consigue llegar a casi un 80 % de acierto (para ejecutar esta simulación se puede llamar al objetivo p2.2.3-problema2 del Makefile). En el caso de 9 neuronas, al ratio de acierto varía entre el 75 y el 80% de acierto. Se puede observar en la gráfica la inestabilidad del modelo cuando la tasa de aprendizaje es demasiada alta. Por ello, cuando poníamos una tasa de aprendizaje de 1, el porcentaje de acierto y de train no pasaba del 65%.

3. Predicción en problemas con más de dos clases

Nuestro programa ya estaba diseñado para poder tener más de dos clases. Tiene dos métodos de funcionar, modo de única neurona(usado por el Perceptron o el Adaline) o crear tantas neuronas de salida como clases haya. Si aplicamos los mismos parámetros que en los ejercicios anteriores



ECM 4 Neuronas

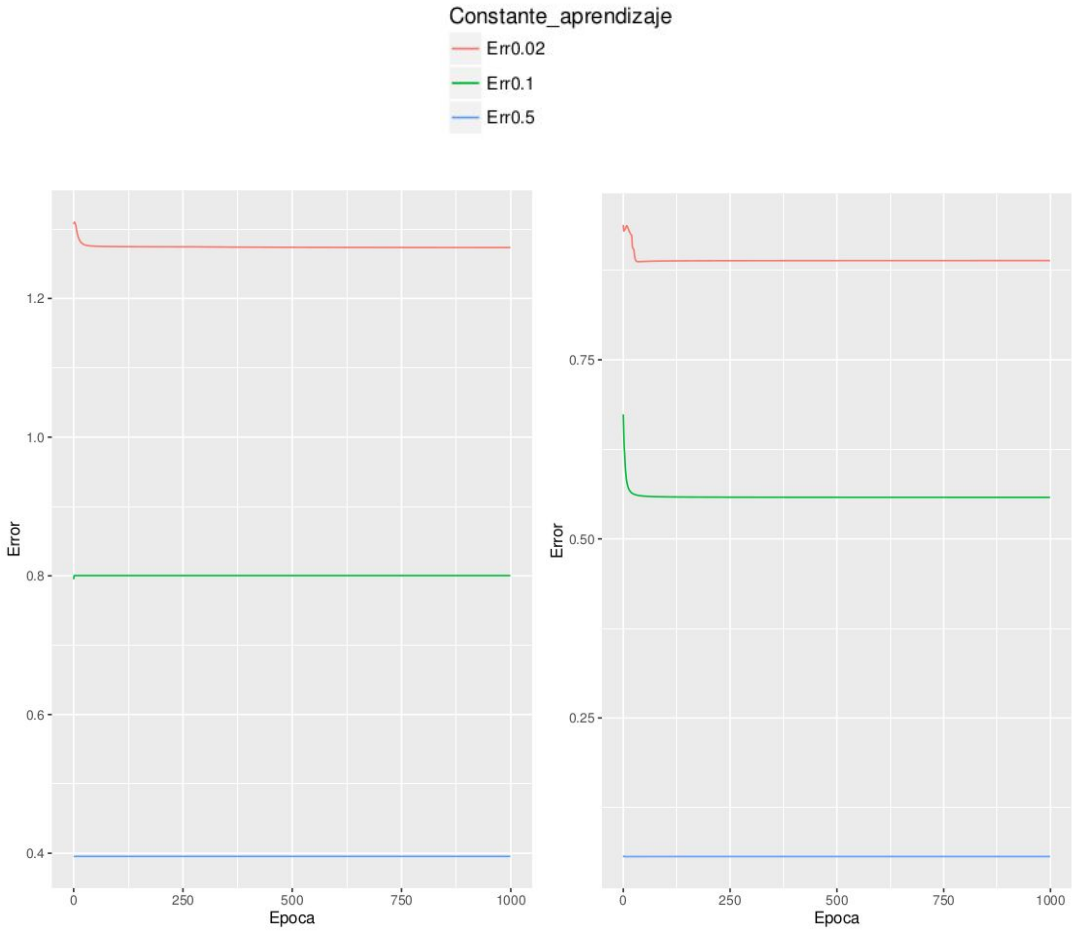
ECM 9 Neuronas

Para este ejemplo no hemos probado con la tasa de aprendizaje de 1, ya que con 0.5 ya daba bastantes malos resultados.
Los mejores resultados se consiguen con 4 neuronas con una tasa de 0.1 donde se llega al 97-98% de acierto y con 9 neuronas con la tasa de 0.02 que se consigue un 97%.

Esta simulación se encuentra como objetivo del Makefile con el nombre p2.3-problema3.

4. Predicción en un problema complejo

Hemos realizado nuestro análisis característica al problema 4, proporcionando los resultados:



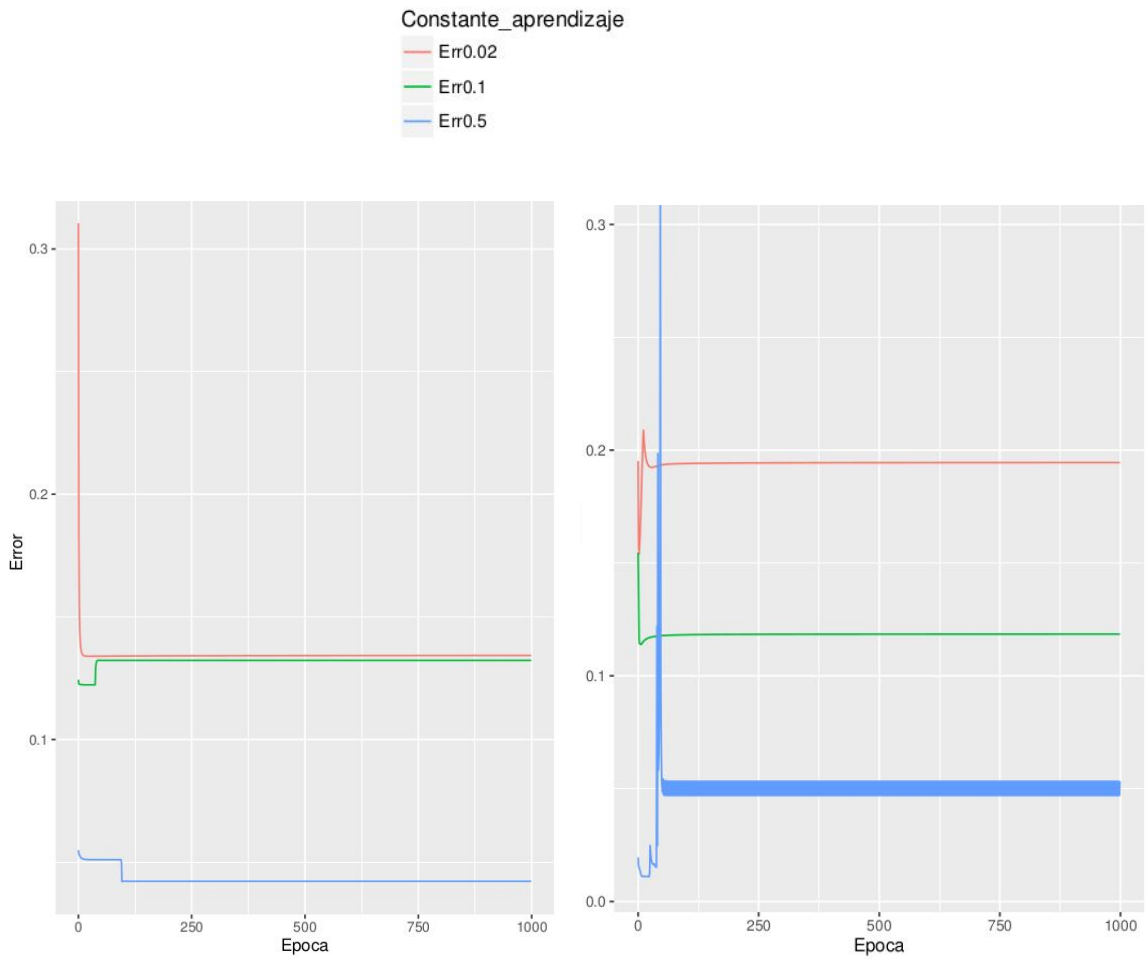
ECM 4 Neuronas ECM 9 Neuronas

Si solo miramos el error cuadrático media es mejor el de 9 neuronas, pero las tasas de acierto son bastantes bajas en los dos, en torno a un 65%, teniendo a veces picos de un 35% de acierto.
Para intentar arreglar esto hemos observado las medias y las desviaciones, tal y como se nos pedía en el enunciado. Hemos observado una gran diferencia en las valores de los atributos. Había dos variables con unos valores demasiado altos con respecto al resto.
Además, estamos usando una función sigmoide bipolar, que tiene valores finales comprendidos entre el -1 y 1. Por lo tanto, es de comprender que los valores de entrada se encuentren cercanos a estos valores para un buen funcionamiento.

Los resultados de esta simulación se generan llamando al objetivo p2.4-problema4 del Makefile.

5. Normalización de los datos

Para solucionar el problema anterior tenemos que normalizar los datos. Para ellos hemos introducido una funcionalidad a nuestro clasificador que, usando las medias y las desviaciones del conjunto de test, se las aplica al conjunto de test y al de train. Además estas medias y desviaciones se pueden guardar dentro de la red neuronal de tal forma que se puede crear entrenar una red con valores normalizados, y luego, en otra ejecución se podrían pasar valores sin normalizar y el programa los normalizaría con la información guardada en la red. Los resultados de la red son los siguientes:



ECM 4 Neuronas ECM 9 Neuronas

En ambos se obtiene un porcentaje de acierto con tasa de 0.02 un 89-90% de acierto. Como se puede observar, se ha aumentado bastante la tasa de acierto con respecto a los datos sin normalizar. Seguramente se podría aumentar la tasa de acierto si normalizásemos todo el conjunto a la vez, (es decir, con la media de todo el conjunto y su desviación) en vez de solo el conjunto de train. Sin embargo hemos decidido esa opción porque es menos específica al conjunto de datos determinado.

El objetivo del Makefile para esta simulación es p2.5-normalizado.

6. Predicción de datos no etiquetados

Para este apartado se ejecuta dos veces nuestro programa, la primera para crear la red neuronal y guardarla, y la segunda para predecir con el otro fichero. Los resultados se han guardado en el fichero predicciones_nnet.txt .

Los dos pasos se ejecutan mediante el objetivo del makefile p2.6-predicciones.