Redes de Neuronas – Práctica 1

Jorge Hevia Moreno – 100317565

Luis Víctor Hevia Moreno – 100317546

Contents

[Introducción 3](#_Toc497670070)

[Parte 1 – Adaline 4](#_Toc497670071)

[Introducción al programa 4](#_Toc497670072)

[Estructura del programa 4](#_Toc497670073)

[Ejecución del programa 4](#_Toc497670074)

[Análisis de los resultados 5](#_Toc497670075)

[En función de los ciclos 6](#_Toc497670076)

[En función del factor de aprendizaje 9](#_Toc497670077)

[Análisis de los pesos obtenidos 11](#_Toc497670078)

[Análisis de la salida 12](#_Toc497670079)

# Introducción

En esta práctica se ponía como objetivo abordar un problema de regresión lineal utilizando el modelo lineal Adaline y el modelo no lineal de Perceptron Multicapa.

Este problema de regresión se aplicaría sobre un conjunto de datos acerca del hormigón para determinar su resistencia a partir de distintos datos como la edad de éste, sus componentes y sus cantidades.

La primera parte de la práctica consiste en programar Adaline y realizar una serie de pruebas con los datos del hormigón proporcionados para un posterior análisis.

La segunda parte consiste en utilizar la librería RSNNS en el lenguaje R para realizar una serie de pruebas con los datos y analizar dichas pruebas.

# Parte 1 – Adaline

## Introducción al programa

En esta parte se nos pedía programar Adaline en un lenguaje a nuestra elección. Nosotros escogimos Python por su sencillez, potencia y sus librerías de matemáticas y plotting.

Para poder ejecutar el programa, es necesario tener los siguientes paquetes instalados:

* Python 2.7
* Tkinter
  + En debian y derivados se instala mediante: *apt install python-tk*
* Las librerías de Python:
  + Numpy
  + Pandas
  + Matplotlib
  + Que pueden ser instaladas mediante pip
    - *pip install pandas numpy matplotlib*

### Estructura del programa

El programa está escrito en un único archivo de Python (.py) con las siguientes secciones:

1. La primera sección importa las librerías necesarias y prepara los archivos y variables iniciales para poder ejecutar adaline. Además, esta sección contiene la prevenciones de errores (líneas 1 a 73).
2. La segunda sección contiene todas las *funciones* que se ejecutarán, con comentarios sobre qué hace cada función, su entrada y su salida (líneas 74 a 182).
3. La tercera sección contiene la sección principal del programa, donde se harán las llamadas a las funciones con los datos procesador en la primera sección (líneas 182 a 238).

### Ejecución del programa

Para ejecutar el programa se hará desde una terminal con el siguiente comando:

|  |
| --- |
| python adaline.py <training-file> <validation-file> <test-file> <number of inputs/weights> <number of cycles> <learnfactor> |

En el caso de que no se introduzcan los suficientes argumentos o los archivos sean incorrectos, se notificará y se pondrá un ejemplo de caso de uso.

## Análisis de los resultados

El programa se ha ejecutado nueve veces con los siguientes argumentos:

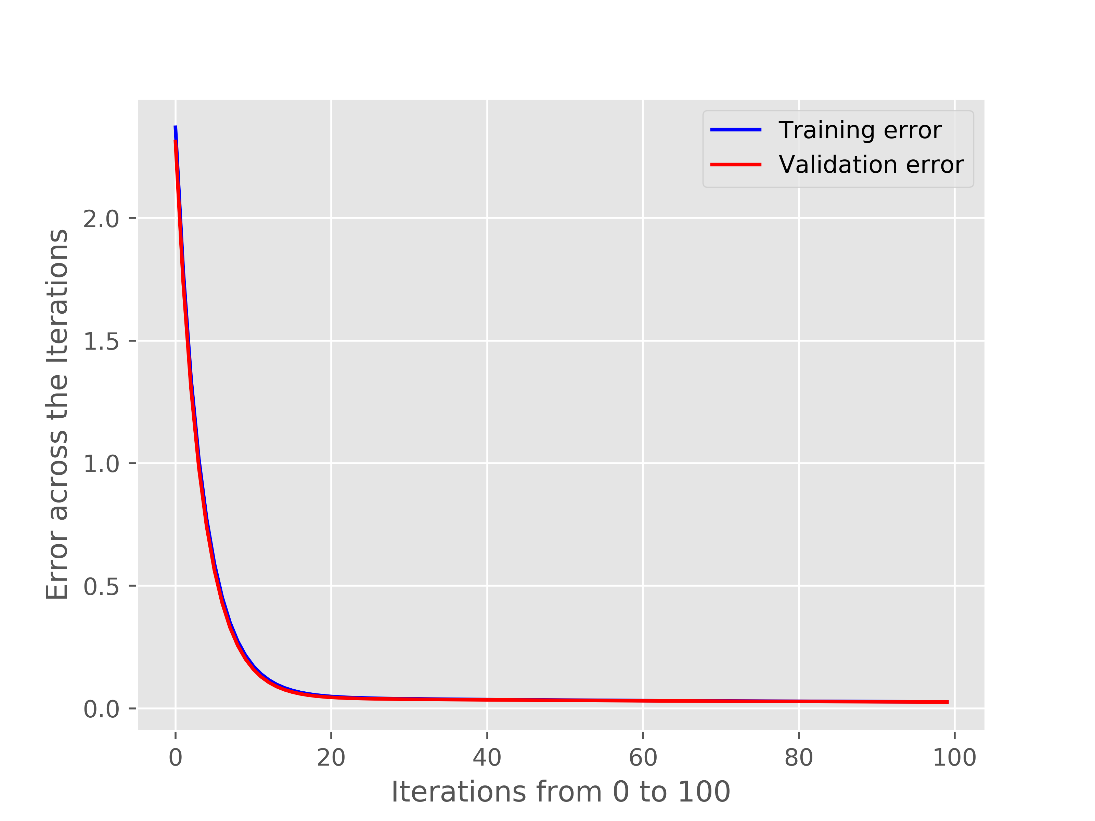
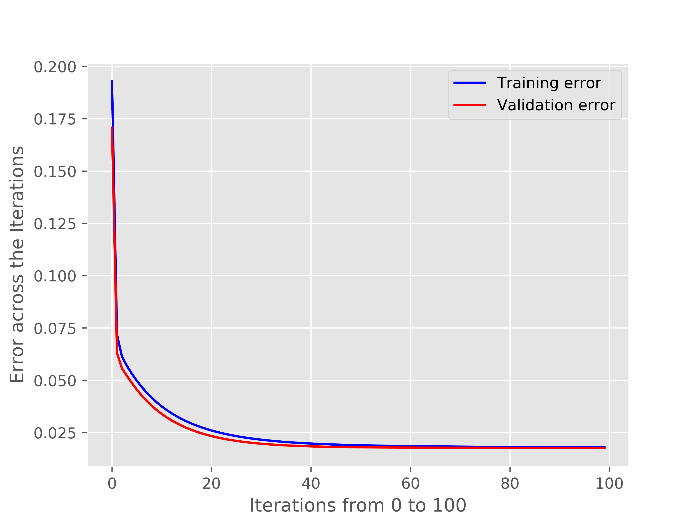
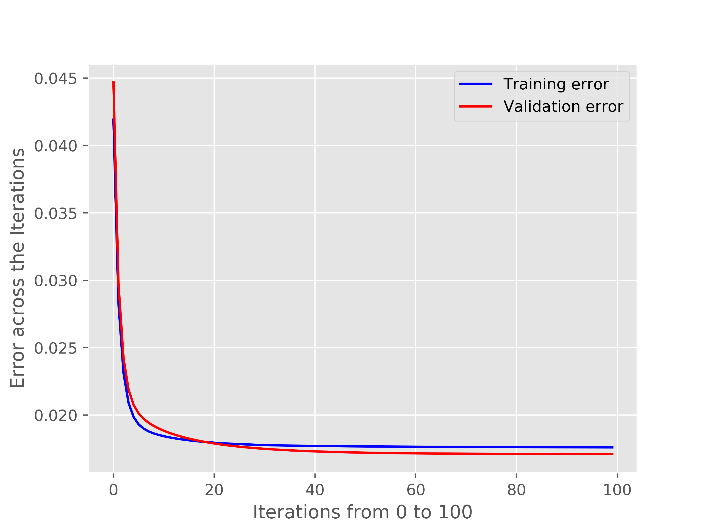
|  |  |
| --- | --- |
| Número de ciclos | Factor de Aprendizaje |
| 100 | 0.01 |
| 100 | 0.001 |
| 100 | 0.0001 |
| 1000 | 0.01 |
| 1000 | 0.001 |
| 1000 | 0.0001 |
| 2000 | 0.01 |
| 2000 | 0.001 |
| 2000 | 0.0001 |

A continuación, se desgranarán los resultados obtenidos por cada una de las ejecuciones y después se realizará un análisis global de los resultados obtenidos.

### En función de los ciclos

#### 100 ciclos

Para 100 ciclos se puede comprobar que no son suficientes para que se estabilice adaline hasta que disminuimos el factor de aprendizaje a 0.0001, y aun así todavía se podría estabilizar más.

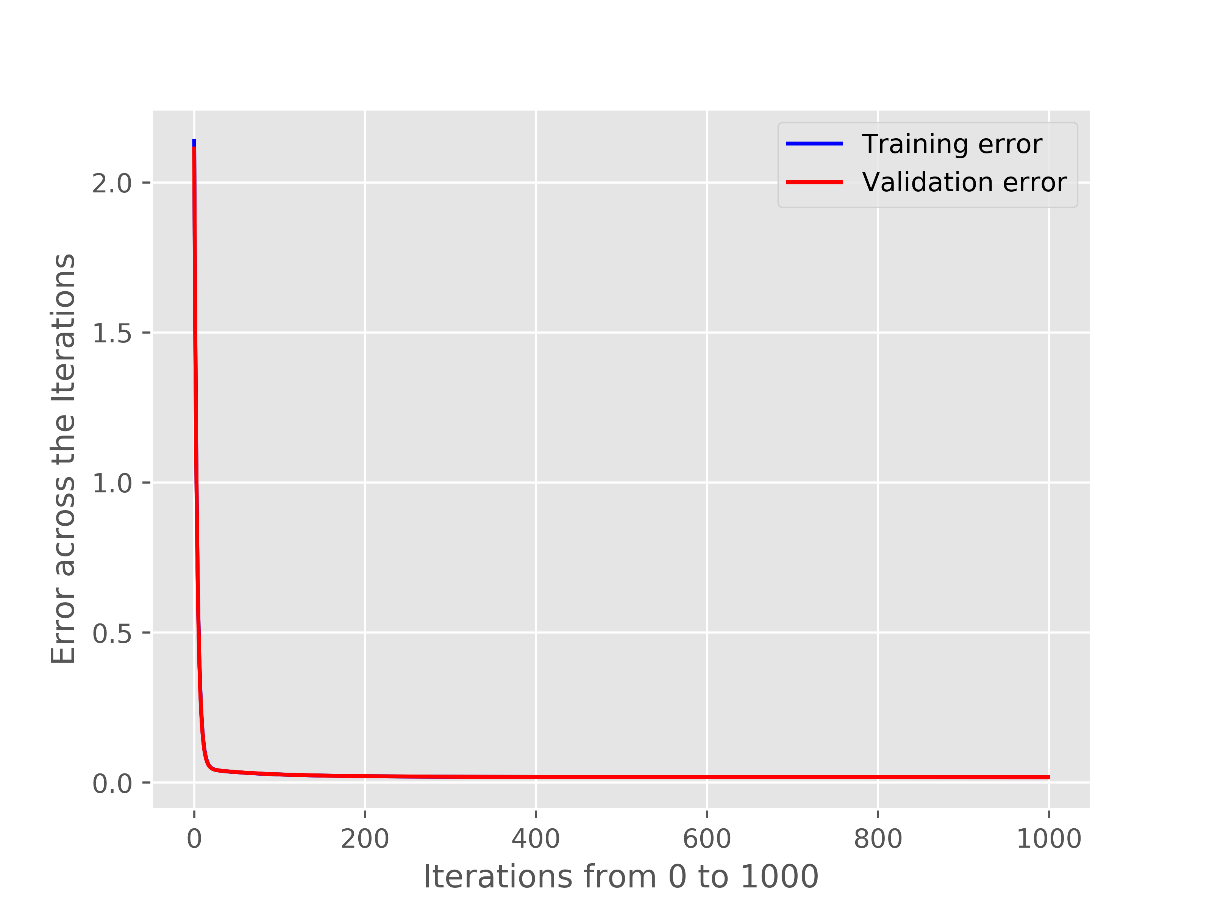
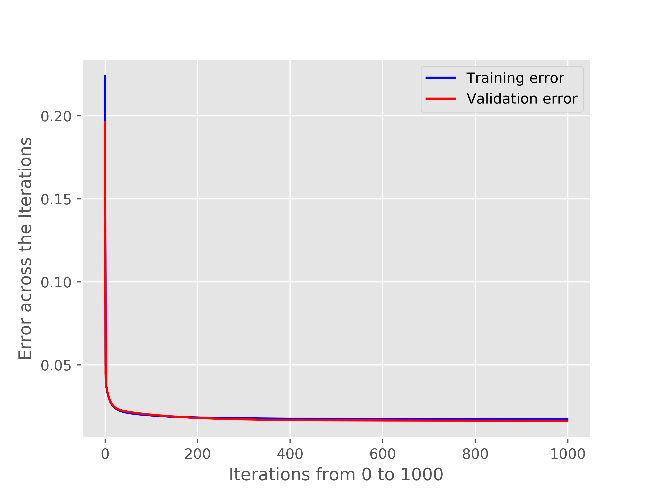
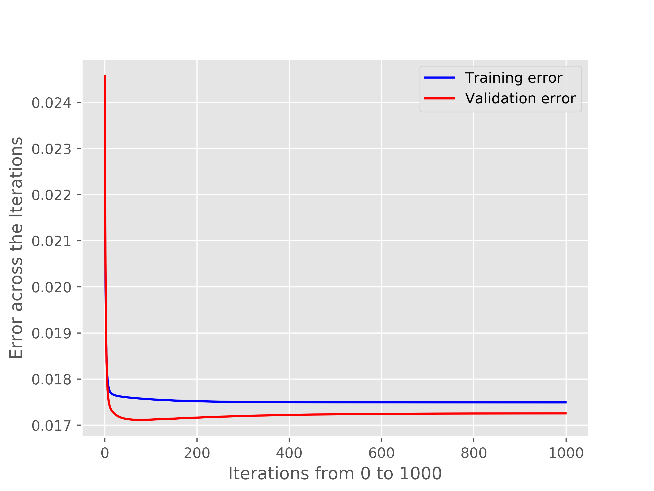


En orden:

* Factor de aprendizaje 0.01
* Factor de aprendizaje 0.001
* Factor de aprendizaje 0.0001

#### 1000 ciclos

Con 1000 ciclos se aprecia mejor que la línea se estabiliza, y que los valores del factor de aprendizaje se ajustan más cuanto más pequeño es éste.

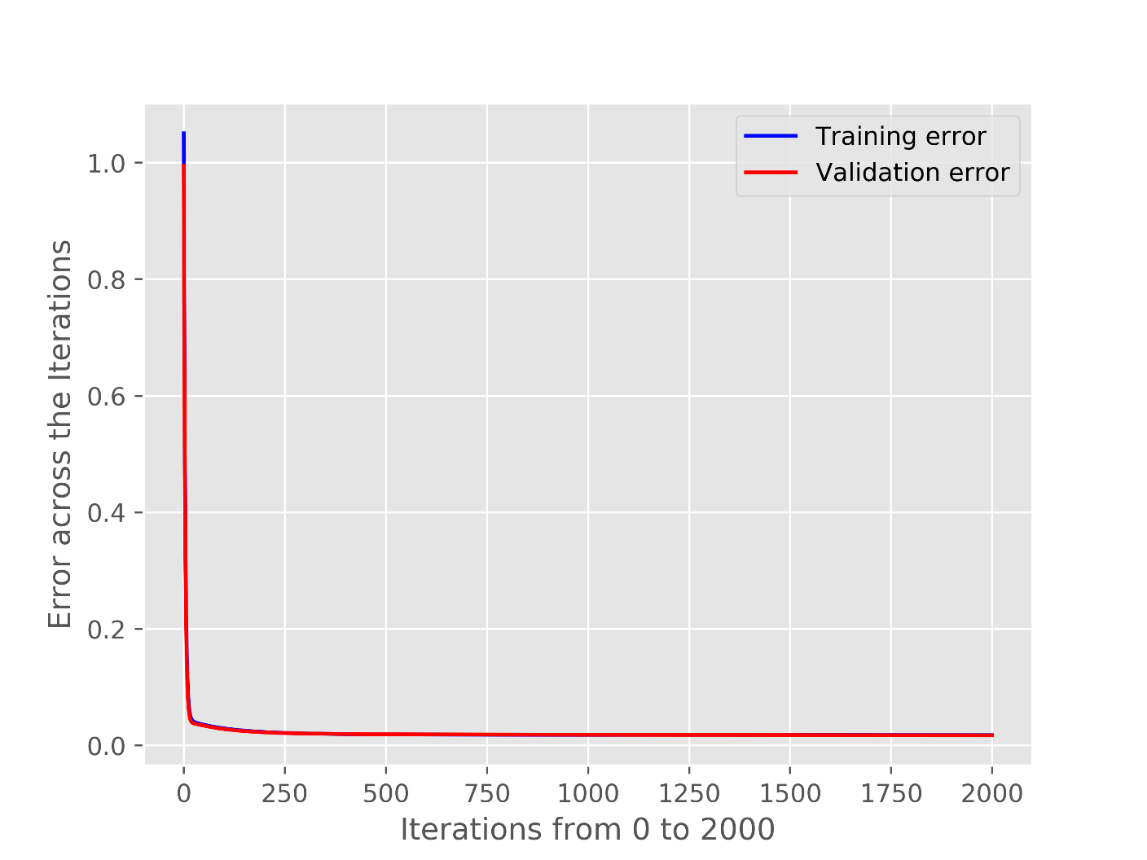
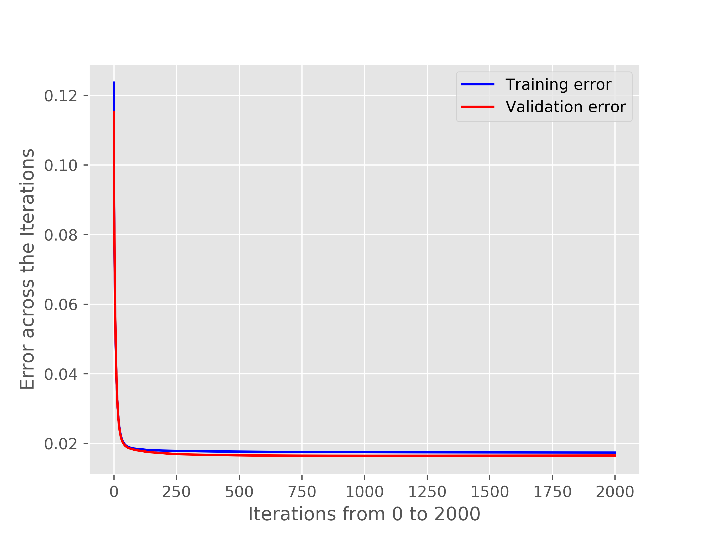
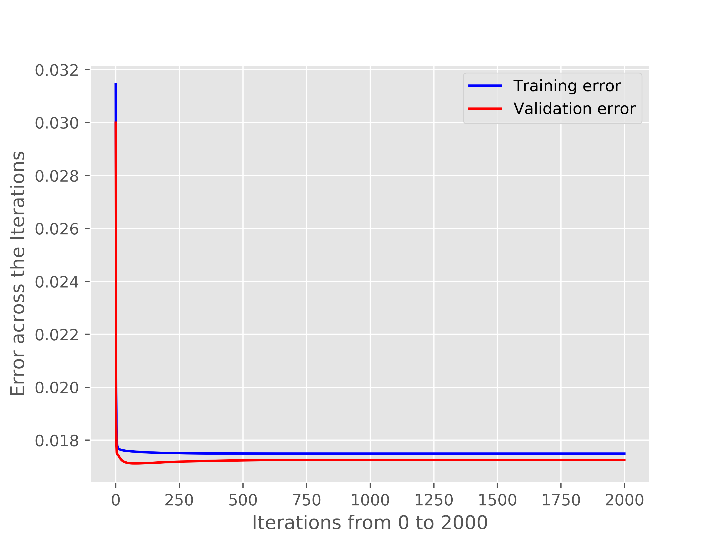


En orden:

* Factor de aprendizaje 0.01
* Factor de aprendizaje 0.001
* Factor de aprendizaje 0.0001

#### 2000 ciclos

Al hacer 2000 ciclos ya podemos asegurar que la función se ha estabilizado por completo (o casi), lo que nos indica que independientemente del factor de aprendizaje que utilicemos, deberíamos correr adaline durante al menos 2000 ciclos para obtener unos resultados más estables.

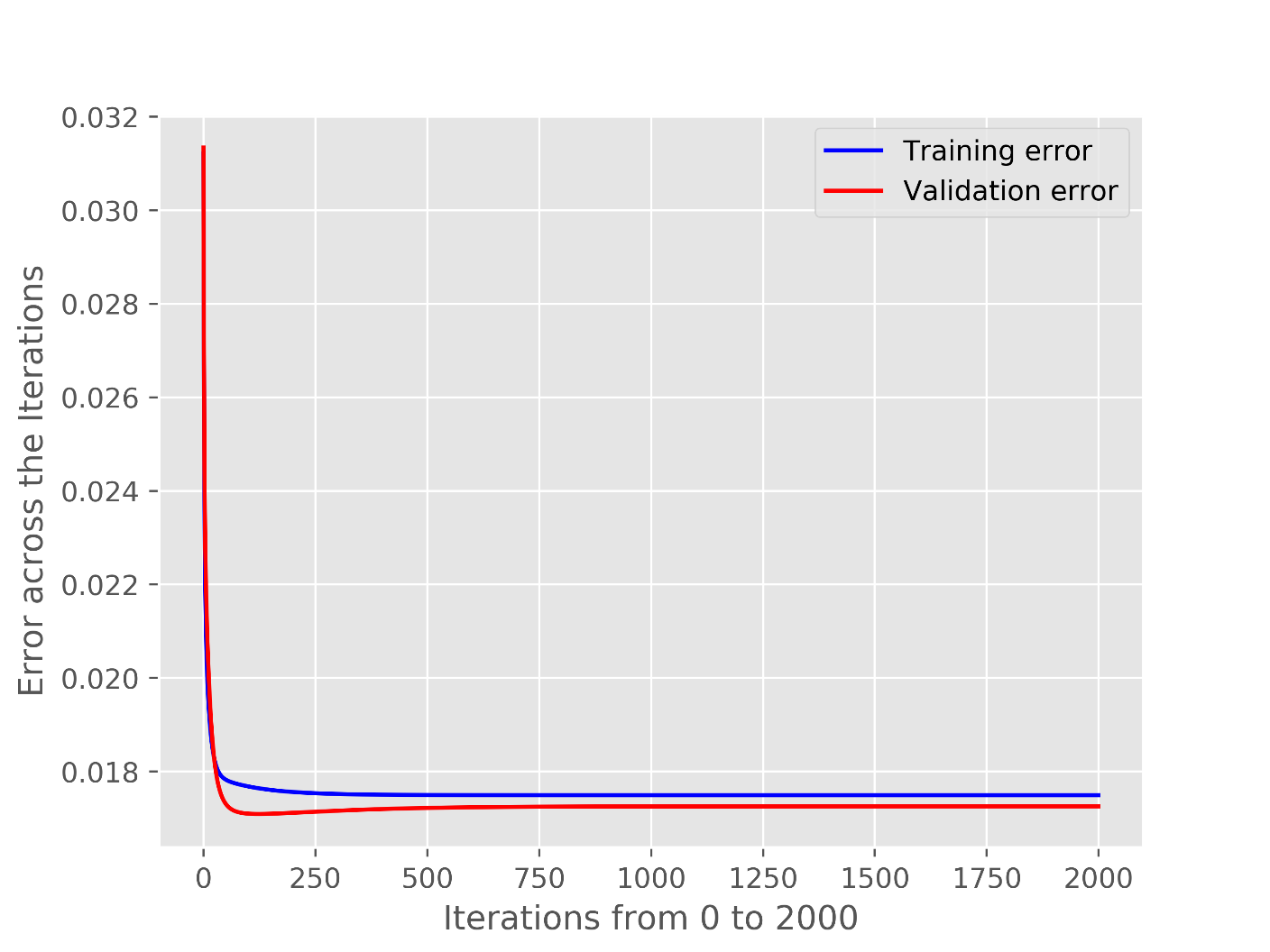


En orden:

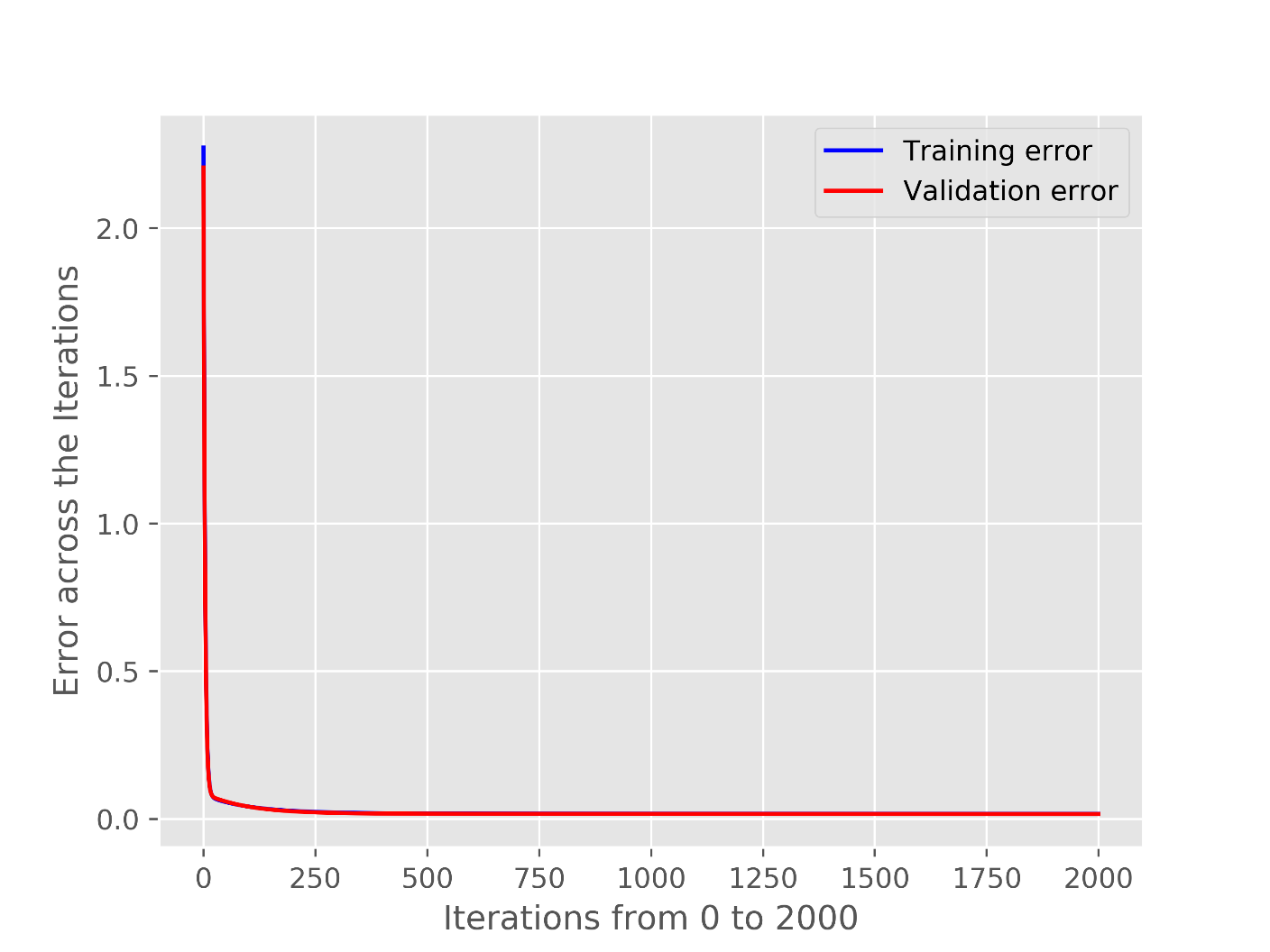
* Factor de aprendizaje 0.01
* Factor de aprendizaje 0.001
* Factor de aprendizaje 0.0001

### En función del factor de aprendizaje

Como se ha podido observar anteriormente, al disminuir el factor de aprendizaje hacemos que la red sea más precisa entre los datos de entrenamiento y los de validación. A continuación, se muestra (para 2000 ciclos) la diferencia entre un factor de aprendizaje de 0.01 y 0.0001:



1Factor de aprendizaje 0.01 (2000 ciclos)



2Factor de aprendizaje 0.0001 (2000 ciclos)

Como se puede ver, al utilizar una menor tasa de aprendizaje la función tarda un poco más en estabilizarse (en torno al ciclo 250) mientras que con una mayor tasa de aprendizaje se estabiliza un poco antes de los 250 ciclos aunque de forma un poco más *brusca*.

### Análisis de los pesos obtenidos

A continuación, se muestra una tabla unificando todos los pesos y umbrales obtenidos.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| W1 | W2 | W3 | W4 | W5 | W6 | W7 | W8 | THR | TEST |
| 0.718123 | 0.516074 | 0.25216 | -0.1592 | 0.156007 | 0.13505 | 0.165057 | 0.510513 | -0.22691 | **100-0.01** |
| 0.665657 | 0.469704 | 0.197693 | 0.020148 | 0.359626 | 0.190021 | 0.17609 | 0.522435 | -0.32576 | **100-0.001** |
| 0.438981 | 0.249344 | 0.212225 | -0.13209 | 0.509024 | -0.05923 | 0.125492 | 0.792802 | -0.03483 | **100-0.0001** |
| 0.625368 | 0.42494 | 0.194188 | -0.28665 | 0.130976 | 0.05177 | 0.051241 | 0.504997 | 0.006009 | **1000-0.01** |
| 0.752225 | 0.550112 | 0.269647 | -0.10015 | 0.170528 | 0.172658 | 0.212566 | 0.515382 | -0.30966 | **1000-0.001** |
| 0.649595 | 0.455538 | 0.180803 | -0.01104 | 0.363671 | 0.17913 | 0.151108 | 0.504029 | -0.27837 | **1000-0.0001** |
| 0.624856 | 0.424436 | 0.193865 | -0.28734 | 0.130852 | 0.051316 | 0.050616 | 0.504965 | 0.007282 | **2000-0.01** |
| 0.656972 | 0.456811 | 0.209948 | -0.23242 | 0.14307 | 0.086171 | 0.095208 | 0.509745 | -0.06971 | **2000-0.001** |
| 0.552975 | 0.356234 | 0.135106 | -0.28672 | 0.197654 | 0.029492 | -0.00776 | 0.503074 | 0.104371 | **2000-0.0001** |

*Nota: en los ficheros cvs, el noveno valor es el threshold*

Como se puede observar, tanto los pesos como el umbral son aleatorios.

### Análisis de la salida

A continuación se mostrarán unas gráficas comparando la salida obtenida con la salida deseada en función de los ciclos (2000) y el factor de entrenamiento, que variará entre 0.01 y 0.0001, donde la línea azul representa la salida esperada y la naranja la obtenida:

1 Ciclos: 2000, Aprendizaje: 0.01

2 Ciclos: 2000, Aprendizaje: 0.001

3 Ciclos: 2000, Aprendizaje: 0.0001

Como se puede ver, aunque la función de error a 2000 ciclos ya esté normalizada, la salida obtenida no llega a ser igual que la deseada en la mayoría de casos, aunque cuanto mayor es el factor de aprendizaje, los resultados varían menos de un resultado a otro.