REDES DE NEURONAS ARTIFICIALES

PRACTICA 1

Universidad Carlos III Madrid, Ingeniería Informática, 2021

IVAN AGUADO PERULERO – 100405871
JORGE SERRANO PÉREZ – 100405987

Contenido

1.- INTRODUCCIÓN

El objetivo de esta práctica es abordar un problema real de regresión utilizando dos modelos de redes neuronales supervisados: el modelo lineal Adaline y el modelo no-lineal Perceptrón Multicapa. Hemos trabajado sobre el dominio conocido como "*Concrete Compressive Strength* (Resistencia a la compresión del hormigón)". Para abordar el problema se nos ha proporcionado un conjunto de datos que contiene 1030 instancias con 8 variables de entrada. Tendremos que realizar una previa preparación de datos antes de comenzar el aprendizaje de las redes. Además, se nos ha proporcionado también el simulador SNNS bajo el lenguaje de programación R con un script básico para entrenar la red del Perceptrón Multicapa.

El presente documento se divide en un capítulo que explica cómo se ha realizado el preproceso de los datos, otro capítulo para cada uno de los modelos con una descripción de la experimentación realizada, los resultados obtenidos y un análisis de los resultados, y un capítulo con la comparación de los modelos y las conclusiones obtenidas.

2.- PREPARACIÓN DE DATOS

Para la preparación de los datos hemos utilizado el programa Weka, que nos permite aplicar distintos filtros a nuestro fichero de datos.

En primer lugar, convertimos el fichero en formato *xlsx*, pues fue el formato elegido para trabajar en esta práctica. Además, hemos añadido una columna llamada *Threshold* o Umbral asignando a todas las filas el número 1, que solo se hará para los ficheros de Adaline. Esto nos facilitará más tarde en el código el poder calcular sin problemas el valor del umbral como si se tratara de un peso más. De esta manera, en el momento de calcular la salida de la red, al multiplicar el umbral por su entrada (el valor 1 que hemos mencionado) se mantiene el valor ajustado. Sin embargo, en la parte de MLP no es necesario. Además, en ese caso los ficheros serán de tipo *csv* en vez de *xlsx*.

A continuación, una vez abierto el fichero desde Weka, aplicamos el filtro *Normalize*, el cual es un filtro de la categoría atributo de no supervisado. Este aplicará la transformación lineal necesaria para acotar las variables dentro del rango [0, 1]. Tras esto, para aleatorizar los datos utilizamos *Randomize*, filtro en la sección del tipo no supervisado. Esto nos permite desordenar los datos para que el entrenamiento de las redes se realice correctamente.

Por último, hemos dividido el fichero en tres, entrenamiento, validación y test, tal y como se pedía en el enunciado. El de entrenamiento será el que utilizaremos para realizar el aprendizaje de la red y se corresponde con un 70% de las instancias, el de validación se utilizará para elegir los valores óptimos de los hiperparámetros de la red (razón de aprendizaje, número de ciclos y número de neuronas (Perceptrón Multicapa)) y utiliza un 15% de las instancias, y por último el de test sirve para evaluar la capacidad de generalización de la red y es el 15% restante.

3.- ADALINE

Experimentación realizada

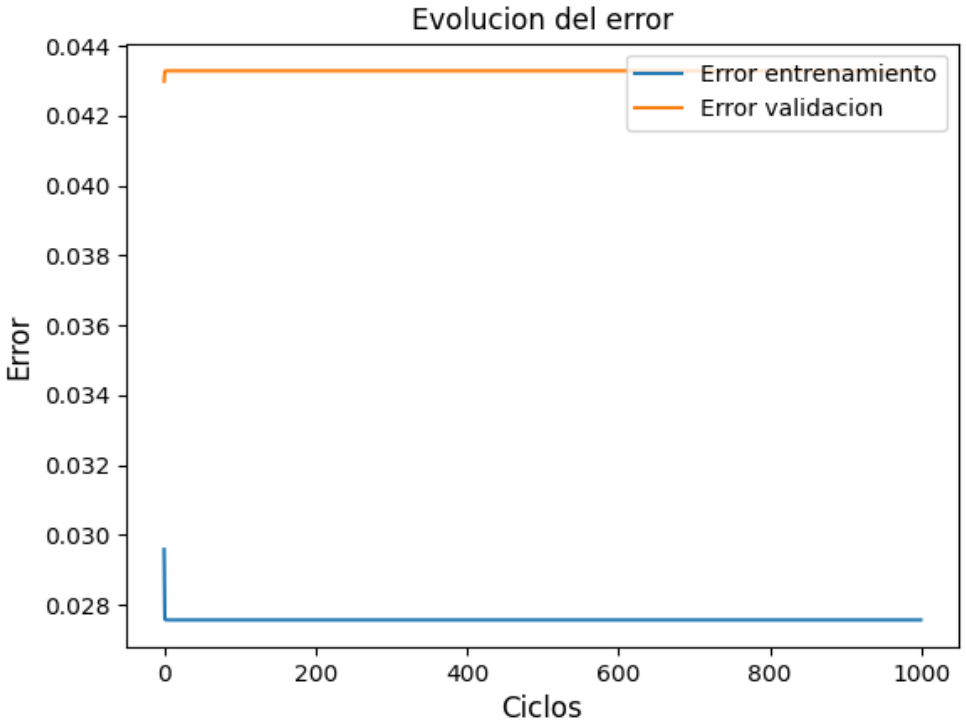
Antes de mostrar los resultados obtenidos en la experimentación nos gustaría destacar lo siguiente acerca de nuestra implementación de Adaline:

- El error de validación determina el número de ciclos óptimo, es decir, a partir del cual dicho error sube. Por lo tanto, los valores de los errores mostrados más abajo son del ciclo con el error de validación mínimo.
- En la columna ciclos de la siguiente tabla se detalla el número de ciclos con el que hemos experimentado y entre paréntesis el ciclo óptimo en el que recogemos los errores y del que sacamos el modelo, en caso de que dicho experimento sea el que posee mejores resultados.

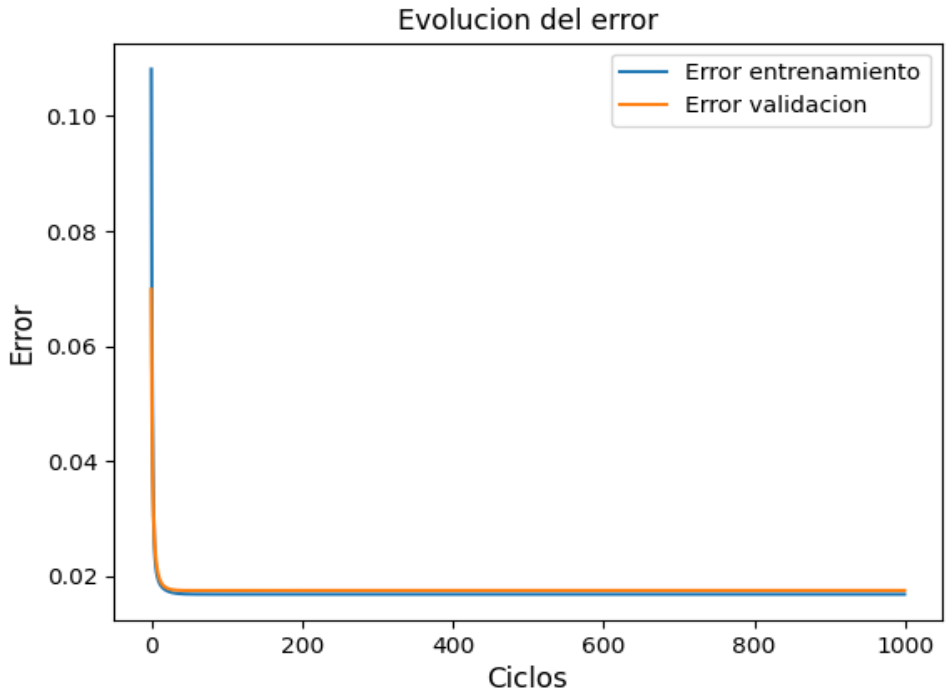
Ciclos	Factor de aprendizaje	Error Entrenamiento	Error Validación	Error test
1000 (1)	0.5	0.03182522147513122	0.043029187835306834	0.03246729169790109
1000 (2)	0.1	0.018539930578721924	0.022146662403725266	0.01693456660664142
1000 (59)	0.01	0.016904720540288545	0.017548953576631044	**0.015804261444080852**
1000 (1000)	0.001	0.016750300402720485	0.016795213978848411	0.016291786806799224
1000 (1000)	0.0001	0.019243750939179172	0.019928248464420165	0.016182439128436758
10000 (5083)	0.001	0.016750197636170774	0.016794707465193325	0.016309656083036136
10000 (10000)	0.0001	0.016729104342913706	0.016748990809183063	0.016366060686600504
30000 (30000)	0.0001	0.016729063378251183	0.016748528872014563	0.016377613594204848
60000 (54422)	0.0001	0.01672906337823963	**0.016748528838533852**	0.016377614489124923

Resultados obtenidos

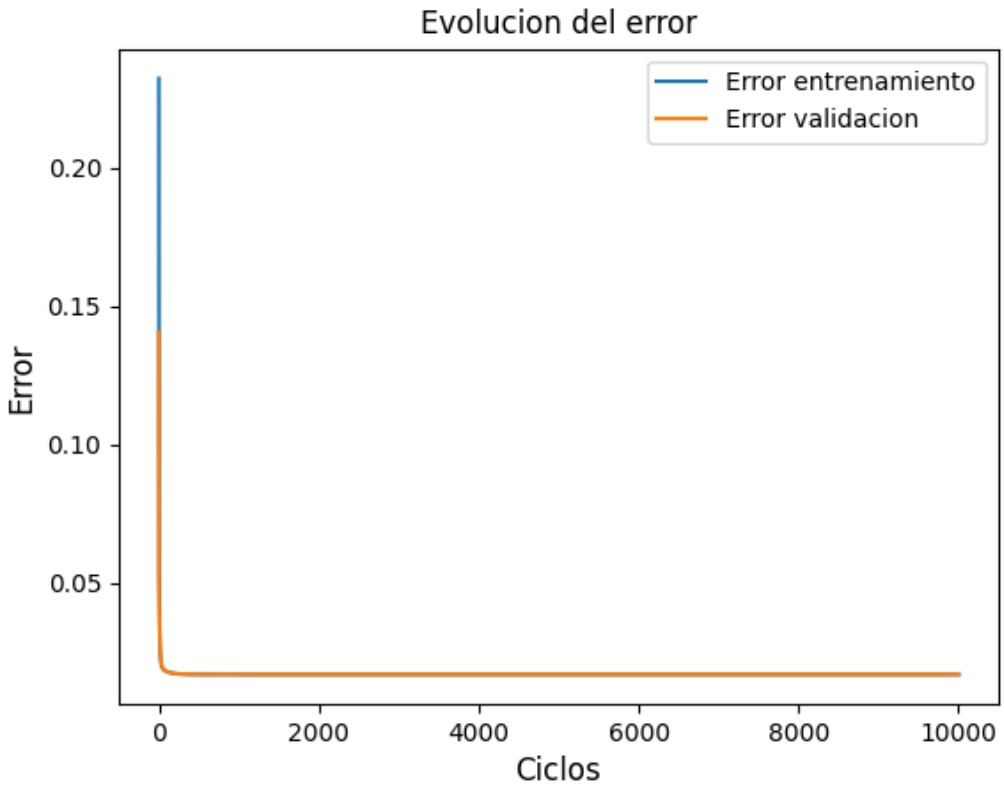
En primer lugar, comenzamos probando con un factor de aprendizaje de 0.5 y 1000 ciclos. Tras la ejecución pudimos observar que este factor era demasiado alto, pues el sobre aprendizaje ya se dejaba ver en el primer ciclo.



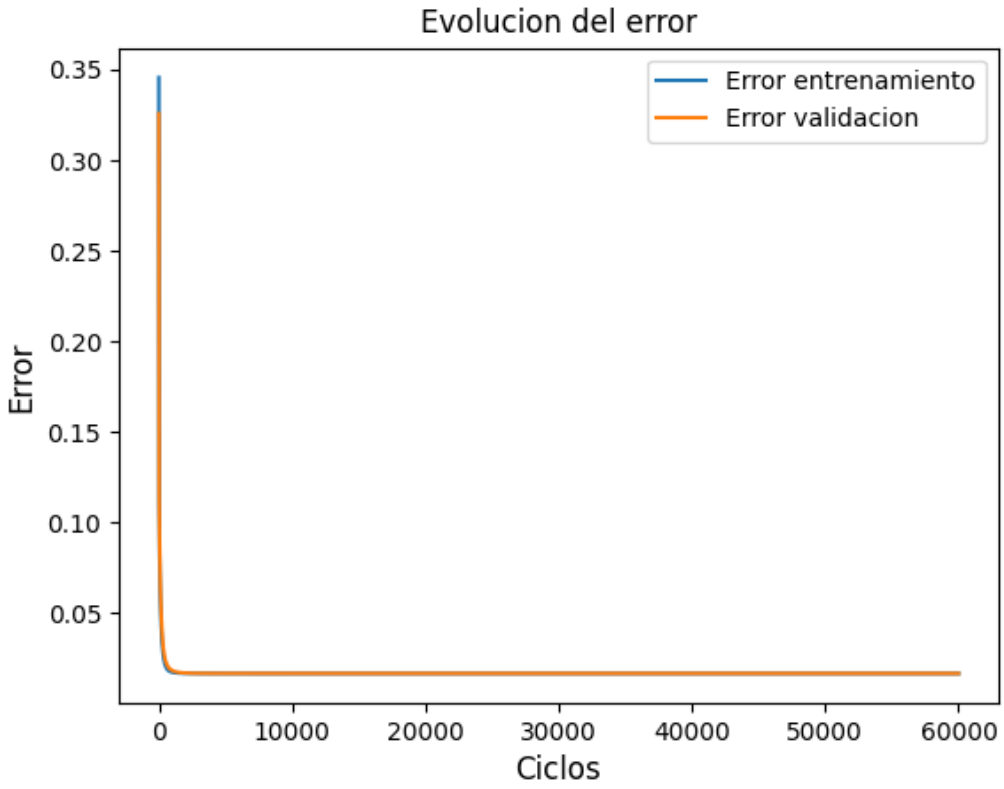
Debido a esto, nuestro siguiente experimento lo realizamos bajando el factor de aprendizaje a 0.1 y utilizamos el mismo número de ciclos, sin embargo, comprobamos que esta bajada no había sido suficiente para que el modelo se entrenase de una forma correcta. Por este motivo bajamos a un factor de aprendizaje de 0.01. Tras este experimento conseguimos mejorar significativamente los tres errores, el de validación alcanzaba su mínimo concretamente en el ciclo 59. Cabe destacar que el error de test alcanzó su mínimo en este experimento (0.015804261444080852), lo que nos indica que este experimento sería el mejor a la hora de generalizar.



El aprendizaje en este momento se realizaba de forma rápida, pero no sabíamos si era de la forma correcta. Por ello, seguimos probando con razones de aprendizaje más bajas aún, lo que ralentizaría este proceso de aprendizaje. Los parámetros seleccionados fueron un factor de aprendizaje de 0.001 y 1000 ciclos. De esta manera conseguimos reducir el error de validación por el cual nos estábamos guiando. Sin embargo, este alcanza su mínimo en el último ciclo, lo que quería decir que aún no habíamos conseguido el número de ciclos óptimo para este factor de aprendizaje. Aumentando los ciclos a 10000 pudimos comprobar que este número de ciclos buscado era 5083.



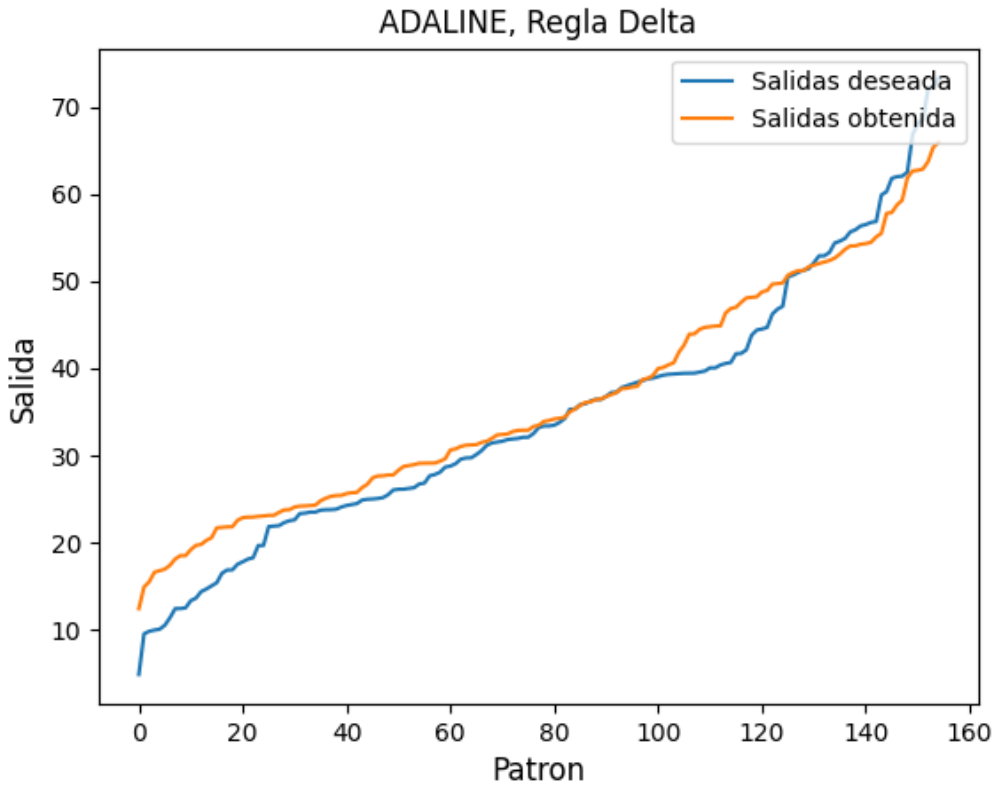
Tras esto bajamos aún más la razón de aprendizaje (0.0001) y fuimos probando con diferentes valores para el número de ciclos (10000, 300000 y 60000). Este último valor nos hizo averiguar el valor óptimo del error de validación (0.016748528838533852) el cual se alcanzaba a los 54422 ciclos. Este era entonces el modelo más adecuado.



A partir de aquí, el bajar aún más el factor de aprendizaje no compensaba demasiado, ya que el error disminuía poquísimo (en el undécimo decimal) y sin embargo la duración del entrenamiento ascendía exponencialmente. Por ello, decidimos terminar con la experimentación.

Análisis de los resultados

En conclusión, el modelo elegido ha sido el de 54422 ciclos y 0.0001 de factor de aprendizaje ,con un error de validación de 0.016748528838533852 y un error de test de 0.016377614489124923. En la siguiente gráfica podemos observar las salidas deseadas y las salidas obtenidas por la red:



Antes que nada, hemos tenido que desnormalizar estas salidas para que el rango de valores fuera el original. Para ello creamos una función que recibía los valores máximo y mínimo y revertía la fórmula de la normalización. Además, ordenamos los valores en orden ascendente para poder observar más claramente la comparación de cada patrón.

En cuanto a la gráfica, podemos observar que las salidas obtenidas por el modelo de la red no se adecuan demasiado bien a las deseadas. Esto se debe simplemente a que se trata de una regresión lineal, que busca un hiperplano para clasificar las salidas.

4.- PERCEPTRÓN MULTICAPA

Experimentación realizada

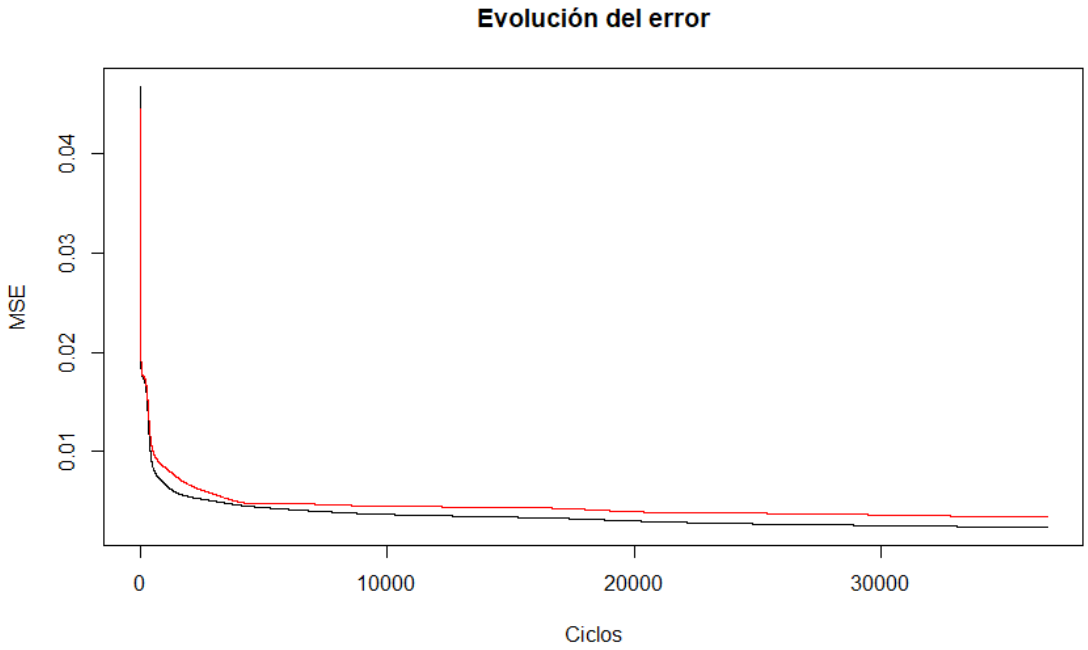
En la parte del perceptrón multicapa, el script que se nos proporciona nos devuelve el error mínimo de validación de cada experimento y el número de ciclos que se necesitan para llegar a ese error. Debido a esto en la tabla en la columna de los ciclos se ha procedido igual que en la experimentación con Adaline, primero el número de ciclos con el que se ha ejecutado el programa y entre paréntesis el óptimo.

Destacar que en las gráficas de los errores que se mostraran a continuación, la línea roja se corresponde con el error de validación y la negra con el error de entrenamiento.

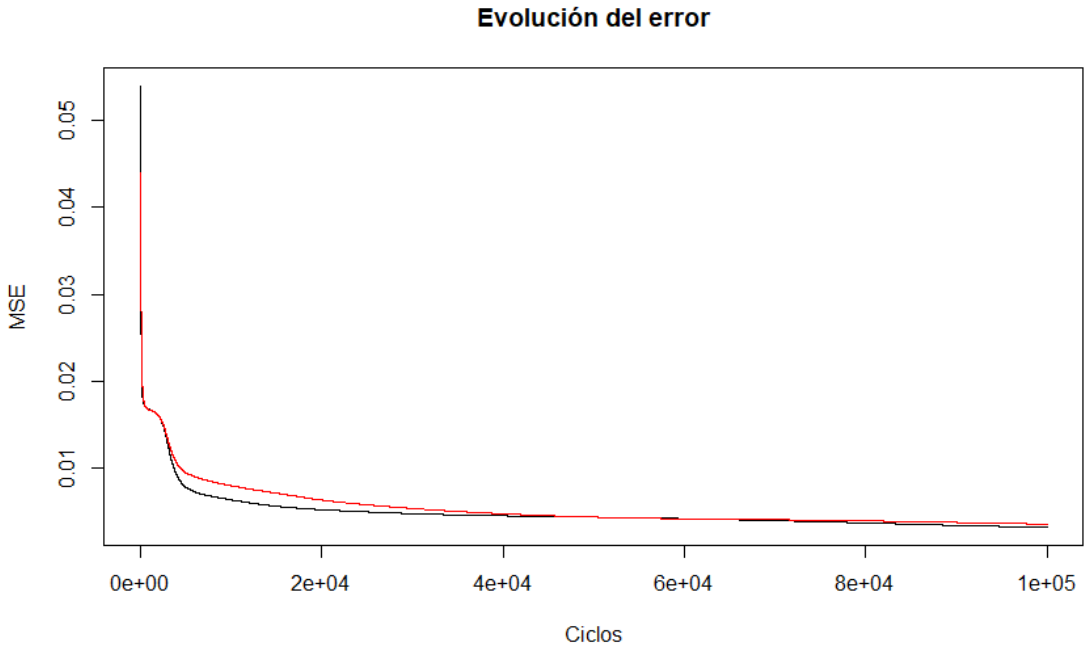
Topología	Ciclos	Razón de aprendizaje	Error Entrenamiento	Error Validación	Error test
60	15000 (5791)	0.5	0,002853656753 74062	0,00389635 532957721	0,00387031 942207551
60	15000 (15000)	0.1	0,003283951416 61695	0,00439398 232765854	0,00421593 529004603
60	50000 (36719)	0.1	0,002280400802 40411	0,00345395 569808086	0,00376433 486578616
60	100000 (100000)	0.01	0,003248803267 65475	0,00363832 957869765	0,00455511 279656763
30	50000 (26364)	0.1	0,002772657619 2521	0,00401894 409094519	0,00441016 785888827
90	50000 (18051)	0.1	0,002820832604 91606	0,00339054 697316133	0,00443186 728230347
120	50000 (50000)	0.1	0,002113397355 66385	0,00248657 419122835	0,00379774 78458158
120	90000 (84136)	0.1	0,001812099963 21542	**0,00222735 837493565**	**0,00342651 8957361**
120	20000 (20000)	0.5	0,001512400076 43222	0,00236753 395335016	0,00399198 38963138
30	5000 (3203)	0.5	0,003481201971 10148	0,00431918 632655609	0,00364911 578249463
30	100000 (100000)	0.01	0,003741003499 75575	0,00444569 256690479	0,00443220 848854825

Resultados obtenidos

En primer lugar, comenzamos experimentando con una topología de 60 neuronas en la capa oculta, una razón de aprendizaje de 0.1 y 15000 ciclos. Sin embargo, durante estos ciclos no se producía sobre aprendizaje, por lo que decidimos aumentar el número de ciclos a 50000 para buscar un error de validación más bajo, este se alcanzaba en el 36719. De esta manera hemos obtenido unos resultados iniciales con los que empezar a comparar los distintos errores de entrenamiento, validación y test.

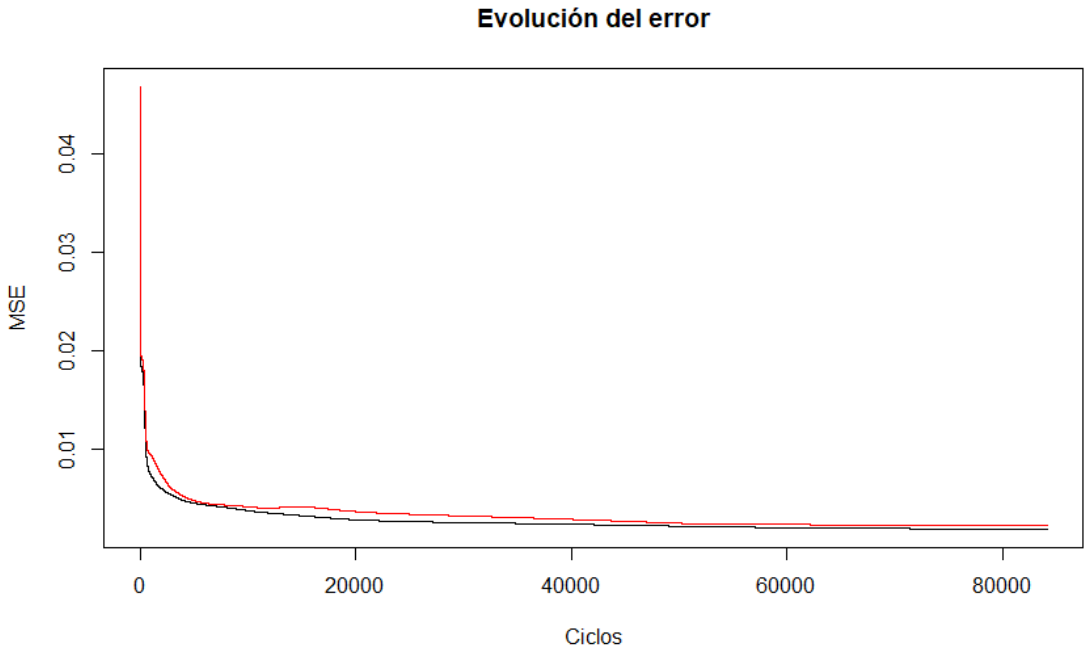


Nuestro siguiente paso fue probar con un factor de aprendizaje diferente: primero más alto (0.5) y luego más bajo (0.01). Para este último, subimos el número de ciclos a 100000 pues el aprendizaje se realizaría más lentamente y el ciclo óptimo sería más alto. Aun así, el número de ciclos probado se quedó corto y los resultados no mejoraron. Debido a este motivo y a la lentitud de ejecución decidimos descartar factores de aprendizaje tan bajos, pues es probable que obtengamos mejores resultados a la larga, pero estos tardarán en conseguirse y no serán muy significativos.



Con estos experimentos, la mejor razón de aprendizaje es 0.1. Por ello, continuamos experimentando, cambiando ahora la topología. Tras realizar algunos experimentos, vemos que el camino más adecuado es aumentando el número de neuronas, ya que de esta manera se

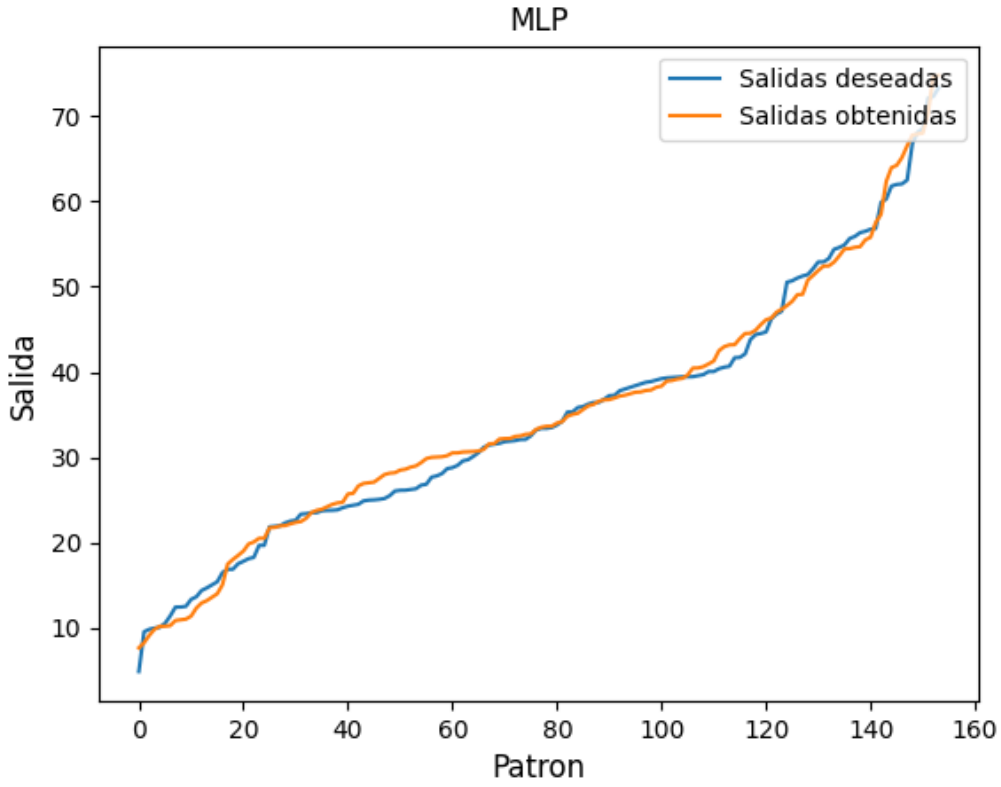
reduce el error de validación (hemos podido observar que al bajarla aumenta). Por lo tanto, probamos con 90 neuronas y después con 120, hasta que encontramos el número de ciclos óptimo para este experimento, 84136. Fue en este dónde obtuvimos el mejor modelo, con un error de validación de 0.00222735837493565.



Tras esto probamos con otros dos experimentos, con el factor de aprendizaje inicial (0.5), subiendo y bajando el número de neuronas de la capa oculta. Observamos que al igual que veníamos comprobando, funcionaba mejor con un número alto de las mismas. Sin embargo, los resultados no eran suficientes para superar los ya obtenidos. Por último, intentamos volver a la razón de aprendizaje de 0.01, que descartamos por entrenar el modelo de manera lenta. Esta vez probamos con un número de neuronas más bajo (30) pues quizá así compensaríamos esa lentitud. Pero tras probarlo, seguía requiriendo de un número muy elevado de ciclos y se seguían sin conseguir errores más bajos.

Análisis de los resultados

Tras la experimentación, el modelo final escogido ha sido el de 120 neuronas en la capa oculta, con un factor de aprendizaje de 0.1 y un número de ciclos de 84136. Con él hemos obtenido los mejores resultados tanto en validación (0,00222735837493565) como en test (0,003426518957361). Esto nos indica que además de tener los mejores hiperparámetros, es también el que mejor generaliza de todos. A continuación, podemos ver una gráfica de las salidas deseadas y las obtenidas.



De nuevo, como con el modelo Adaline, hemos tenido que desnormalizar las salidas a su valor original. Hemos creado un programa en *Python* que realiza esta función, guarda las salidas en un fichero y crea la gráfica previa.

Al observar la gráfica, vemos que el modelo con el MLP es claramente mejor que el de Adaline, ya que las salidas son mucho más parejas que en el otro caso. Esto es porque el MLP es mejor para representar datos no lineales porque se necesitan curvas complejas para clasificarlos correctamente.

5.- COMPARACIÓN Y CONCLUSIONES

Observando los resultados tanto de los errores como de las salidas obtenidas, podemos ver de manera perfectamente clara que el Perceptrón Multicapa es el modelo que obtiene mejores resultados, lo cual no nos sorprende. Esto se debe a que al conectar las neuronas a través de funciones de activación no lineales, podemos crear límites de decisión complejos que nos permiten abordar problemas en los que las clases no son linealmente separables. Sin embargo, el modelo Adaline realiza la separación mediante hiperplanos, lo cual es mucho más difícil de adecuar.

Por ello, el Perceptrón Multicapa obtiene errores tanto de entrenamiento como de validación y de test mucho más bajos, ya que tiene más flexibilidad para adecuarse a las salidas deseadas.