uc3m

Campus de Colmenarejo

Escuela Politécnica Superior Campus Colmenarejo
Universidad Carlos III de Madrid

Problema de regresión
Resistencia a la compresión del hormigón

Redes de Neuronas Artificiales

Grado en Ingeniería Informática

Autor: Luis Cabrero García

Profesor: José María Valls Ferrán

Curso 4º - Gr80

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

La presente práctica trata de abordar mediante el punto de vista de las redes de neuronas supervisadas la resolución de un problema real: la predicción de la resistencia del hormigón a la fuerza de compresión. Para ello disponemos de ocho variables de entrada: la edad del hormigón en días y siete variables que representan los componentes que forman el hormigón (cemento, escoria de alto horno, cenizas volantes, agua, superplastificante, agregado grueso y agregado fino). Estos datos han sido tomados del KEEL[1].

1.1 Motivación

La utilidad de los modelos de regresión que vamos a aplicar de forma práctica es reseñable: nos permiten determinar el valor de la resistencia a la compresión del hormigón a partir de sus características, lo que tiene bastante interés a la hora de saber predecir la resistencia del material en construcciones reales. Si somos capaces de mirar un poco más alla somos capaces de ver que estos modelos de regresión lineal (Adaline) y no lineal (Perceptrón multicapa) son capaces de estimar salidas en función de los datos de entrada pero no solo en este caso sino en problemas de otra índole. Esto nos da una primera idea de la potencia de estos modelos.

1.2 Objetivos

Los objetivos de la práctica son: procesar los datos del KEEL[1], implementar el aprendizaje del modelo Adaline en el lenguaje de programación deseado (en mi caso PHP), grabar en distintos ficheros los resultados del aprendizaje de la red, las salidas obtenidas y error cometido para el conjunto de test, y posteriormente realizar experimentación con MLP y el modelo Adaline. Posteriormente se realizará una comparación entre ambos modelos para ver en que casos un modelo puede aproximar mejor que el otro.

1.3 Estructura de la memoria

La memoria dispone de tres índices, uno para consultar cada una de las secciones y otros dos para figuras y tablas. En la memoria se puede encontrar un apartado general para el modelo Adaline y otro para el Perceptrón multicapa. Dentro de los respectivos apartados de cada modelo, se explica la base teórica de cada uno de ellos, se explica la implementación en el caso de Adaline, y en ambos se explica los distintos experimentos realizados y se razona el motivo por el que se consideran dichos experimentos como ne-

cesarios y suficientes. En las conclusiones se explican las diferencias entre ambos modelos y se comparan los resultados obtenidos en la experimentación. Al final de la memoria se indican las referencias bibliográficas consultadas para llevar a cabo el siguiente trabajo.

CAPÍTULO 2
Modelo Adaline

En este capítulo se va a explicar en primer lugar el modelo Adaline, se explicará la implementación que se ha llevado a cabo en PHP y el formato que se ha elegido para devolver las salidas del modelo. También se expermientará con distintos ciclos de aprendizaje y distintas razones de aprendizaje, y se verá el comportamiento de la red en todos los casos posibles, para ello se representará cada posible situación con un experimento para abordar todos los casos posibles.

2.1 ADALINE: ADAptive LInear NEuron

Es un tipo de red de neuronas artificiales desarrollada por Bernard Widrow y Ted Hoff. La red está compuesta por n neuronas(2.1), con m entradas. La salida de cada neurona se representa por la función de activación(2.1) y representa el impacto de cada peso sobre las entradas, sumando el umbral de la neurona (θ).

$$y = \sum_{i=1}^{n} w_i * x_i + \theta \tag{2.1}$$

En el caso que nos ocupa(2.2), teniendo ocho entradas, la salida que se obtiene es la estimación de la resistencia del hormigón a la compresión. Los pesos aplicables a cada una de las entradas, junto con el umbral, se inicializan de manera aleatoria entre *-0.5* y *0.5*.

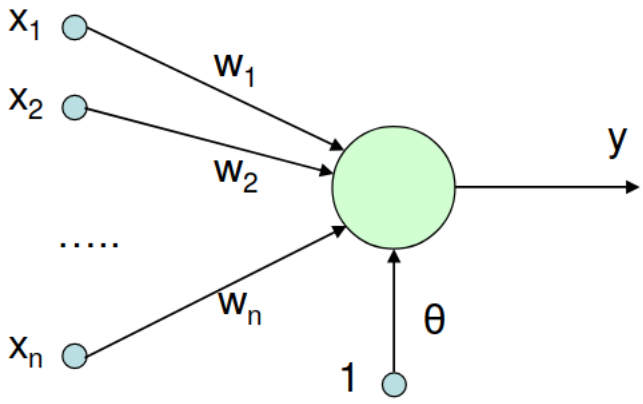

Figura 2.1: Neurona ADALINE [2]

$$ConcreteCompressiveStrengthEst = \sum_{i=1}^{8} w_i * x_i + \theta \tag{2.2}$$

El **aprendizaje** de la red, se realiza teniendo en cuenta la diferencia entre la salida deseada (adjunta como dato en los conjuntos de entrenamiento, test y validación) y la salida obtenida(2.1). Teniendo en cuenta esta diferencia se trata de minimizar el error obtenido mediante la modificación de los pesos y el umbral. Para hallar el error cuadrático medio(2.3) se opera con todos los errores cuadráticos obtenidos en cada uno de los N patrones.

$$E = \frac{1}{N} * \sum_{p=1}^{N} (d^p - y^p)^2 \tag{2.3}$$

Por último, la modificación de los pesos y umbral se realiza siguiendo la Regla Delta, que busca la minimización iterativa de la función de error(2.3), realizando un cambio a cada peso proporcional a la derivada del error, medida en el patrón actual, respecto del peso (2.4). El umbral también se modifica(2.5).

$$\Delta_p w_j = -\gamma \frac{\partial E^p}{\partial w_j} = \gamma * (d^p - y^p) * x_j \tag{2.4}$$

$$\Delta_p \theta = \gamma * (d^p - y^p) \tag{2.5}$$

siendo γ la razón o tasa de aprendizaje.

En este punto es importante resaltar que los datos deben **normalizarse**, **aleatorizarse** y **separados** en tres conjuntos de datos: **entrenamiento**, **validación** y **test**.

- **Datos de entrenamiento**: (70 % del total de datos) para realizar el aprendizaje de la red.
- **Datos de validación**: (15 % del total de datos) utilizados para elegir los valores óptimos de los parámetros de la red (razón de aprendizaje, número de ciclos, número de neuronas).
- **Datos de test**: (15 % del total de datos) para evaluar la capacidad de generalización de la red.

2.2 Implementación del modelo

El modelo se ha decidido implementar en PHP por estar el autor familiarizado con dicho lenguaje y por la tremenda facilidad que ofrece a la hora de hacer cambios de tipo. En el fichero adaline.php se define la clase Adaline, que tiene atributos que caracterizan a la red de una sola neurona. Se leen los parámetros pasados por terminal y se ejecuta el aprendizaje de la red. Posteriormente se pasa el conjunto de validación y por último se pasa el conjunto de test.

```
<?php

    //EJECUCION: php entrenamiento.php <tasa_aprendizaje> <num_ciclos>

    include 'adaline.php';
```

```
    $tasa_aprendizaje = $argv[1];
    $num_ciclos = $argv[2];
    $adaline = new Adaline($tasa_aprendizaje);
    $adaline->ejecutaradaline($num_ciclos);

?>
```

Especificación de los métodos implementados

- *construct*. Parámetro: tasa de aprendizaje. Inicializa pesos y umbral de manera aleatoria, inicializa la tasa de aprendizaje, inicializa los arrays necesarios.
- *aprendizaje*. Parámetros: fichero a utilizar y ciclo. Realiza el aprendizaje de la red utilizando los datos de entrenamiento provenientes de un fichero csv.
- *error*. Parámetros: fichero a utilizar y ciclo. Calcula el error producido en un ciclo de entrenamiento sin modificar los pesos ni el umbral.
- *errorvalidacion*. Parámetros: fichero a utilizar y ciclo. Calcula el error producido en un ciclo de validación sin modificar los pesos ni el umbral.
- *errortest*. Parámetro: fichero a utilizar. Calcula el error cuadrático medio sobre el conjunto de los datos de test.
- *resultados*. Parámetros: carpeta, ciclo y número de ciclos. Muestra en un fichero HTML las salidas obtenidas para el conjunto de entrenamiento y validación normalizadas y desnormalizadas.
- *mostrarerrores*. Parámetros: carpeta y número de ciclos. Muestra en un fichero HTML los errores cuadráticos medios por cada ciclo para entrenamiento y validación, el valor del error cuadrático medio para el conjunto de test, los pesos y umbral resultantes tras aprendizaje y las salidas desnormalizadas de test.
- *datostest*. Parámetros: carpeta y número de ciclos. Muestra en un fichero HTML los errores sobre el conjunto de test normalizado y sin normalizar, muestra también una gráfica comparando las salidas obtenidas con las esperadas.

2.3 Experimentación realizada

La experimentación realizada en Adaline se basa en ir variando la razón de aprendizaje (γ) y el número de ciclos de ejecución. Iremos extrayendo ciertas conclusiones en función de los valores posibles de γ, entre 0 y 1 y los ciclos de ejecución, que pueden ser desde 1 al número que se quiera indicar.

Experimentaciones

$\gamma = 0{,}8$ **y** $ciclos = 500$

Con una tasa de 0.8 y 500 ciclos la red "no aprende"(2.2) dado que comienza cometiendo un error y en el ciclo inmediatamente posterior el error aumenta, tendencia que continúa hasta que el error se estabiliza en un valor elevado, por tanto considero de poco interés este experimento. El error obtenido sobre el conjunto de test es de 0.49219936325144.

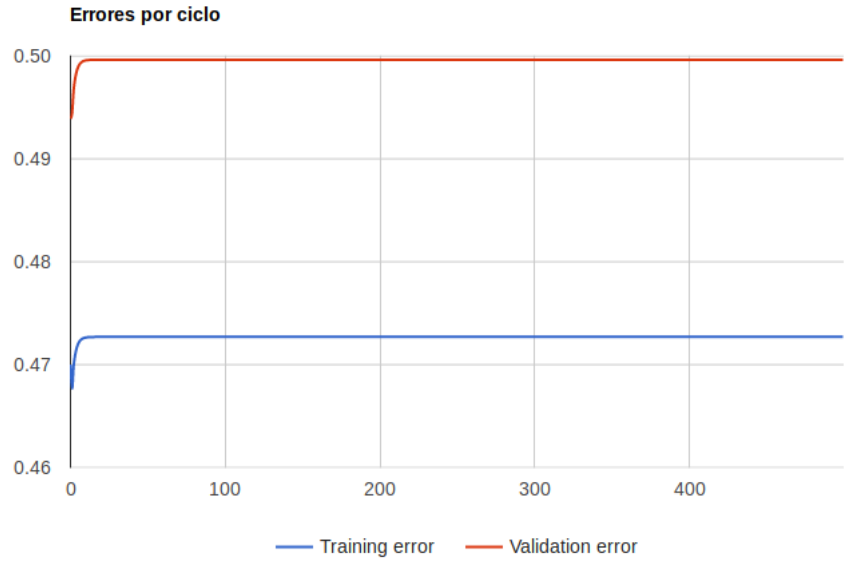

Figura 2.2: Gráfica Adaline 0.8 - 500

$\gamma = 0{,}2$ **y** $ciclos = 500$

Con una tasa de 0.2 y 500 ciclos (2.3) se produce un claro sobreaprendizaje dado que la red es capaz de generalizar el conjunto de entrenamiento con un error cuadrático medio que se estabiliza en torno a 0.021043565092019 mientras que el conjunto de validación se estabiliza en torno a 0.021394445575013. Como vemos la diferencia es muy pequeña pero digamos que en este valor se comienza a producir el sobreaprendizaje, situación que no deseamos porque esto indica que la red no será capaz de generalizar bien datos que no conozca. El error de test es de 0.023142903383433.

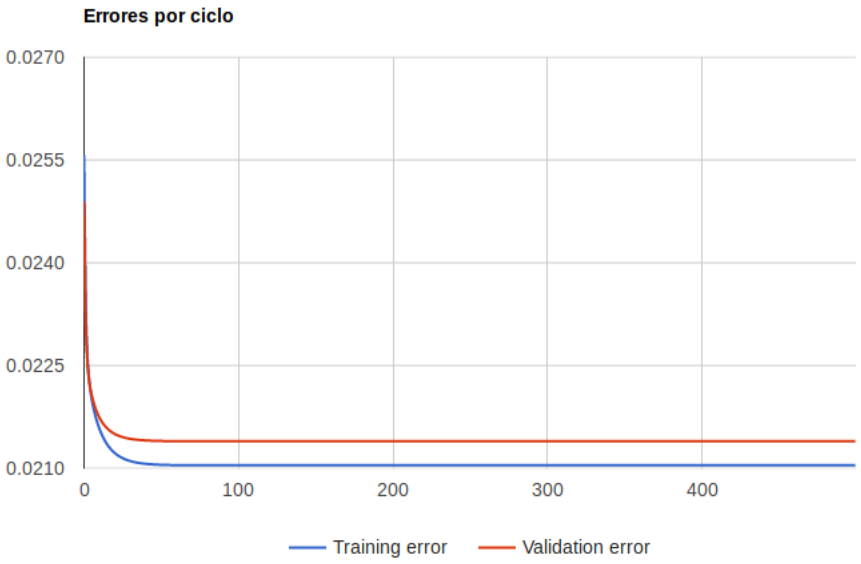

Figura 2.3: Gráfica Adaline 0.2 - 500

$\gamma = 0{,}15$ **y** $ciclos = 500$**,** $\gamma = 0{,}1$ **y** $ciclos = 500$

Con una tasa de 0.15 y 500 ciclos se sigue produciendo sobreaprendizaje pero se puede observar que según se va disminuyendo la tasa de aprendizaje, el valor absoluto de la diferencia entre ambos errores va disminuyendo. También va disminuyendo el error de test que ahora se sitúa en 0.021433296435724. Esta tendencia se puede comprobar con

la experimentación realizada para $\gamma = 0{,}1$ y $ciclos = 500$ (2.4). El error de entrenamiento se estabiliza en 0.017396882217266 y el de validación en 0.018214407942476. El error producido en test en este caso es de 0.020000915138424. En este caso, con $\gamma = 0{,}1$ sigue produciendose sobreaprendizaje.

Figura 2.4: Gráfica Adaline 0.1 - 500

$\gamma = 0{,}0001$ **y** $ciclos = 20000$

Con una tasa muy pequeña ($\gamma = 0{,}0001$) son necesarios cerca de 20000 ciclos (2.5) para conseguir que los errores de validación y entrenamiento se estabilicen. Se obtienen 0.016186448864616 y 0.016999895299418 como errores estabilizados en entrenamiento y validación respectivamente. El error producido sobre el conjunto de test es 0.019147481314295.

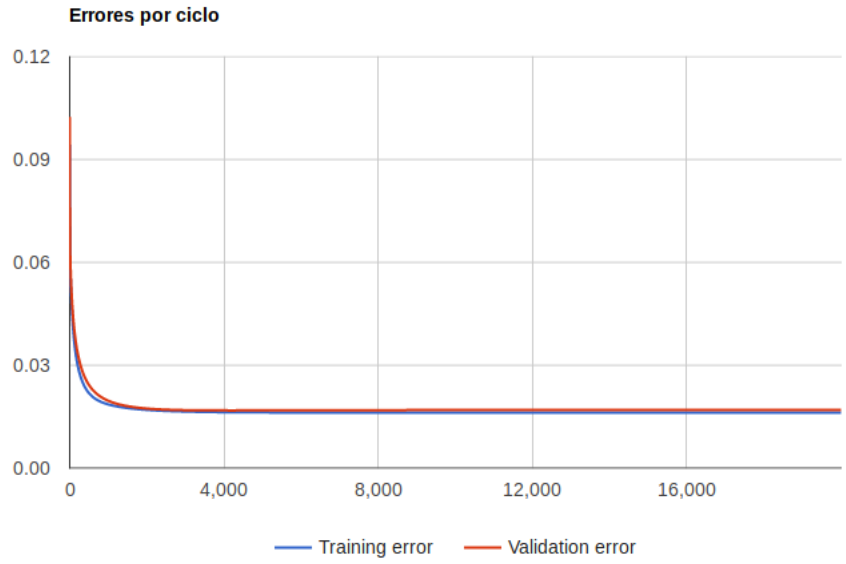

Figura 2.5: Gráfica Adaline 0.0001 - 20000

2.4 Resumen experimentación Adaline

Tabla 2.1: Resumen experimentación Adaline

Tasa de aprendizaje	Error entrenamiento	Error validación	Error test
0.8	0.47269316589935	0.49962652341666	0.49219936325144
0.2	0.021043565092019	0.021394445575013	0.023142903383433
0.1	0.017396882217266	0.018214407942476	0.020000915138424
0.0001	0.016186448864616	0.016999895299418	0.019147481314295

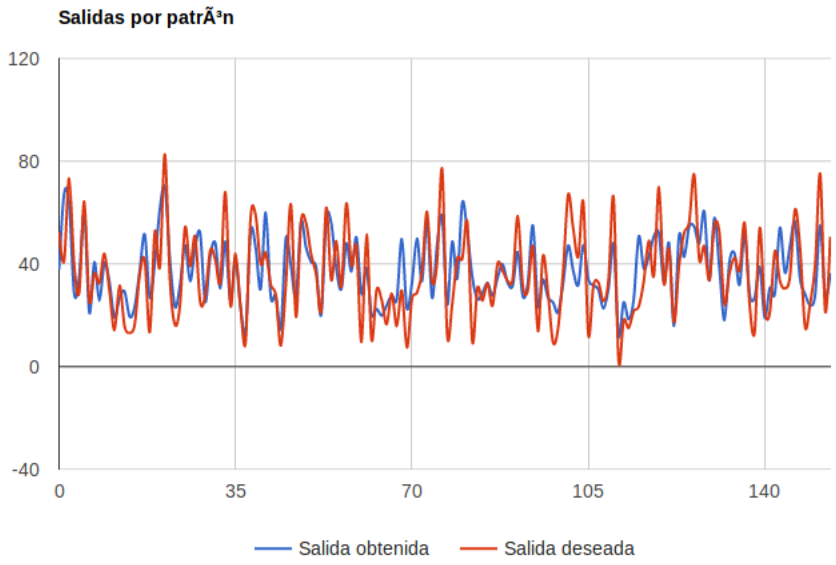

Figura 2.6: Salidas obtenidas vs salidas deseadas para tasa de aprendizaje 0.0001

Como vemos, el mejor experimento realizado es aquel que minimiza el error cuadrático medio, esto es, el realizado con tasa de aprendizaje 0.0001, por lo que se muestran las salidas obtenidas frente a las deseadas en dicho experimento. Podemos ver que para algunas de las salidas el modelo aproxima casi a la perfección pero para otras hay grandes diferencias, en esas diferencias es en las que se genera el error que en este caso se eleva a 123.26694496161, valor desnormalizado. Posteriormente evaluaremos el MLP y veremos si estos valores son mejorables [1].

[1] Para ver los valores en detalle acudir a los resultados adjuntos en formato HTML

CAPÍTULO 3
Modelo Perceptrón Multicapa

En este capítulo se explicará el modelo del Perceptrón multicapa, se explicará la experimentación realizada sobre el script básico proporcionado en lenguaje R, se resumirá la experimentación y se tratará de explicar los resultados. Para realizar la experimentación se irán variando tanto la tasa de aprendizaje como el número de neuronas ocultas.

3.1 Perceptron Multicapa (MLP)

Es una red de neuronas artificiales formada por múltiples capas (3.1). A diferencia del modelo Adaline, resuelve problemas que no son linealmente separables. Además, se ha demostrado que es un aproximador universal, es decir, cualquier función contínua en el espacio $\mathbb{R}^n$ puede aproximarse con este modelo.

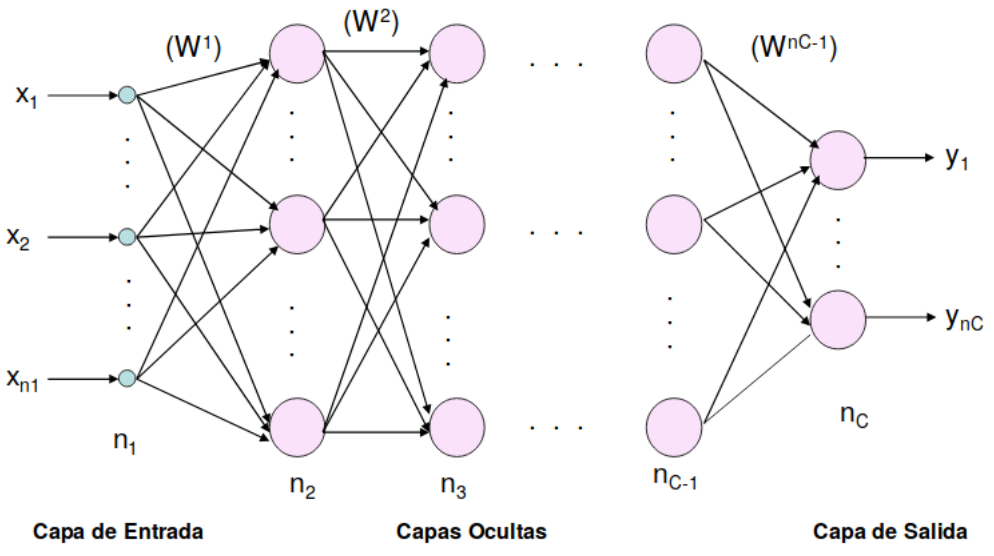

Figura 3.1: Red Perceptrón multicapa [3]

La arquitectura está distribuida en tres tipos de neuronas: neuronas de entrada, que únicamente reciben las entradas y las propagan a la siguiente capa, las neuronas ocultas, que procesan de manera no lineal las entradas, y las neuronas de salida, que devuelven las salidas al exterior.

Teniendo dos neuronas *i* y *j*:

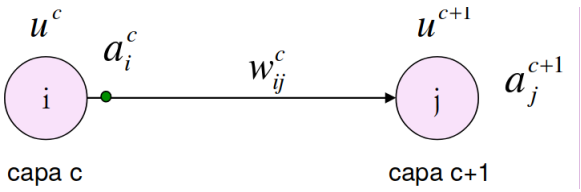

Figura 3.2: Esquema MLP [3]

Los pesos y el umbral (3.1) se expresan en forma matricial, e influyen en la activación de la neurona perteneciente a la capa siguiente. Las neuronas de entrada simplemente tienen su activación equivalente a la entrada que reciben, sin embargo, tanto las neuronas de la capa de salida (3.3) como las neuronas de las capas ocultas (3.2) necesitan computar los pesos y sus entradas sumándoles el umbral.

$$W^c = (w_{ij}^c) = \begin{pmatrix} w_{11}^c & w_{12}^c & \dots & w_{1n_{c+1}}^c \\ w_{21}^c & w_{22}^c & \dots & w_{2n_{c+1}}^c \\ \dots & \dots & & \dots \\ w_{n_c1}^c & w_{n_c2}^c & \dots & w_{n_cn_{c+1}}^c \end{pmatrix}, U^c = (u_i^c) = \begin{pmatrix} u_1^c \\ u_2^c \\ \dots \\ u_{n_c}^c \end{pmatrix} \tag{3.1}$$

$$a_i^c = f(\sum_{j=1}^{n_{c-1}} w_{ji}^{c-1} * a_j^{c-1} + u_i^c) \tag{3.2}$$

$$y_i = a_i^c = f(\sum_{j=1}^{n_{c-1}} w_{ji}^{c-1} * a_j^{c-1} + u_i^c) \tag{3.3}$$

Siendo f la función de activación. Las funciones más utilizadas son la función sigmoidal ($f_1(x) = \frac{1}{1+e^{-x}}$) y la tangente hiperbólica ($f_2(x) = \frac{1-e^{-x}}{1+e^{-x}}$). El aprendizaje de la red se realiza de forma semejante que en Adaline, pero no entraremos en detalle en esta ocasión [3].

3.2 Experimentación realizada

En cuanto a la experimentación realizada se ha prestado especial atención en conseguir una capacidad de generalización adecuada. Esto es: no nos interesa conseguir que la red sea capaz de estimar valores del conjunto de entrenamiento pero que cuando entren datos de un conjunto nuevo, no sea capaz de generalizarlos. Esta situación es conocida como **sobreaprendizaje**. La siguiente figura (3.3) tomada de las transparencias de los profesores de la asignatura ilustra muy bien este comportamiento no deseado. Aun así trataremos de ver una situación de sobreaprendizaje en la experimentación para tratar de explicar el motivo de este comportamiento.

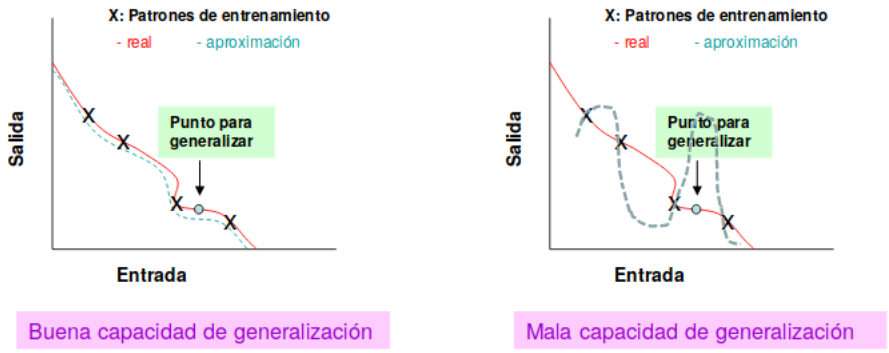

Figura 3.3: Sobreaprendizaje en MLP [3]

Esta situación se puede evitar renunciando a cierta capacidad de generalización sobre el conjunto de entrenamiento, si esto nos va a llevar a una mejor capacidad de generalización sobre otros conjuntos de datos como pudieran ser test o validación. Esto nos aboca a dos posibles situaciones: estabilizar los errores de entrenamiento y test (el aprendizaje ha terminado con éxito) o bien que el error de entrenamiento se estabiliza y el error de test se dispara (se ha producido sobreaprendizaje y el modelo no generaliza bien los datos). Si inicializamos un modelo MLP con demasiadas neuronas ocultas o pesos, o incluso demasiados ciclos, el modelo en su aprendizaje será capaz de generalizar con mucha exactitud el conjunto de entrenamiento. Sin embargo existe la posibilidad de que al presentar al modelo datos de otro conjunto, su capacidad de generalización resulte en fracaso (3.3).

Experimentaciones

$\gamma = 0{,}8$**,** $ciclos = 10000$ **y** $n_{ocultas} = 50$ **(**$f_a(x) = \frac{1}{1+e^{-x}}$**)**

Con una tasa de 0.8, bastante elevada, 50 neuronas en la capa oculta y 10000 ciclos (3.4), lo que tenemos es una clara situación de sobreaprendizaje. El modelo es capaz de generalizar bien el conjunto de entrenamiento con un error de 0.00215141436100276 mientras que en test se produce un error de 0.00305097518407372. Este experimento se ha realizado utilizando la función de activación sigmoidal. Es de suponer que si incrementamos el número de capas ocultas el sobreaprendizaje se acentuaría y la gráfica sería más evidente.

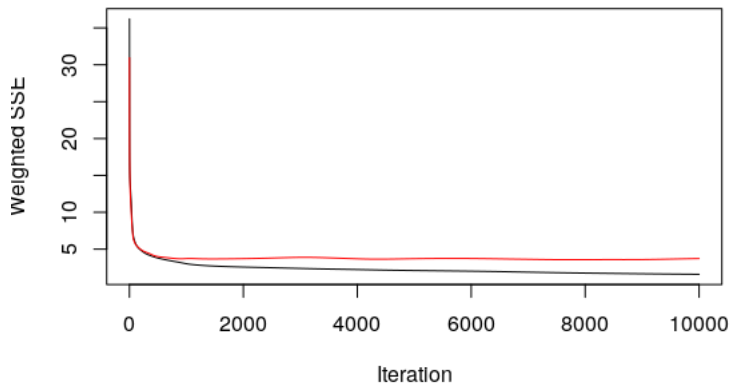

Figura 3.4: Gráfica MLP 0.8 - 50 ocultas - 10000 ciclos ($f_a(x) = \frac{1}{1+e^{-x}}$)

$\gamma = 0{,}8$**,** $ciclos = 10000$ **y** $n_{ocultas} = 50$ **(**$f_a(x) = \frac{1-e^{-x}}{1+e^{-x}}$**)**

Con una tasa de 0.8, bastante elevada, 50 neuronas en la capa oculta y 10000 ciclos (3.7), lo que tenemos es una clara situación de sobreaprendizaje. El modelo es capaz de generalizar bien el conjunto de entrenamiento con un error de 0.00429123031163413

mientras que en test se produce un error de 0.00604587325351324. Este experimento se ha realizado utilizando la función de activación tangente hiperbólica. A simple vista vemos la diferencia entre las dos gráficas (3.7 y 3.4), si nos damos cuenta, a parte de que el error producido es prácticamente el doble utilizando la tangente hiperbólica, esta segunda función presenta un comportamiento bastante inestable, al necesitar un mínimo de 3000 ciclos para comenzar su estabilización, mientras que la sigmoidal entra en esta fase a partir del ciclo 700. Podemos afirmar que en este caso la sigmoidal genera mejor modelo que la tangente hiperbólica.

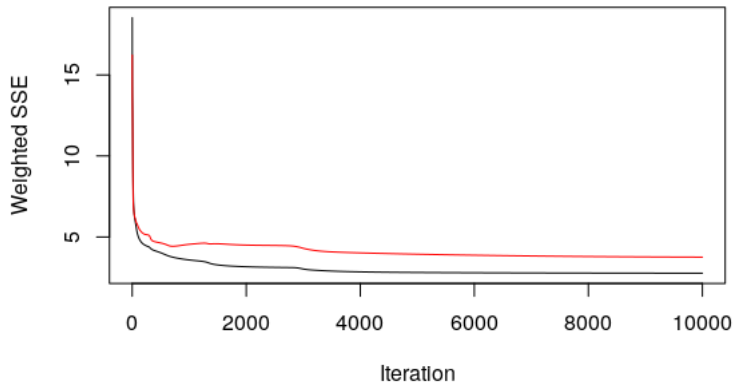

Figura 3.5: Gráfica MLP 0.8 - 50 ocultas - 10000 ciclos ($f_a(x) = \frac{1-e^{-x}}{1+e^{-x}}$)

$\gamma = 0{,}2$**,** $ciclos = 6000$ **y** $n_{ocultas} = 50,50$ **(**$f_a(x) = \frac{1}{1+e^{-x}}$**)**

Con una tasa de 0.2, 50 neuronas en dos capas ocultas y 6000 ciclos (3.6), lo que tenemos es sobreaprendizaje de nuevo. El modelo es capaz de generalizar bien el conjunto de entrenamiento con un error de 0.00283018784738848 mientras que en test se produce un error de 0.00431462177153583. Este experimento se ha realizado utilizando la función de activación sigmoidal.

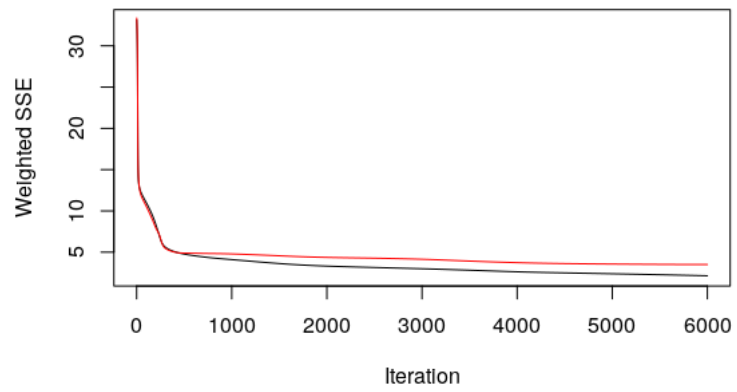

Figura 3.6: Gráfica MLP 0.2 - 50,50 ocultas - 6000 ciclos ($f_a(x) = \frac{1}{1+e^{-x}}$)

$\gamma = 0{,}2$**,** $ciclos = 6000$ **y** $n_{ocultas} = 50,50$ **(**$f_a(x) = \frac{1-e^{-x}}{1+e^{-x}}$**)**

Con una tasa de 0.2, 50 neuronas en dos capas ocultas y 6000 ciclos (3.7), lo que tenemos es sobreaprendizaje más evidente que utilizando la función sigmoidal. El modelo es capaz de generalizar bien el conjunto de entrenamiento con un error de 0.000889548028478484 mientras que en test se produce un error de 0.00346949202753306. Este experimento se ha realizado utilizando la función de activación tangente hiperbólica. Resulta evidente que no es adecuada la utilización de esta función.

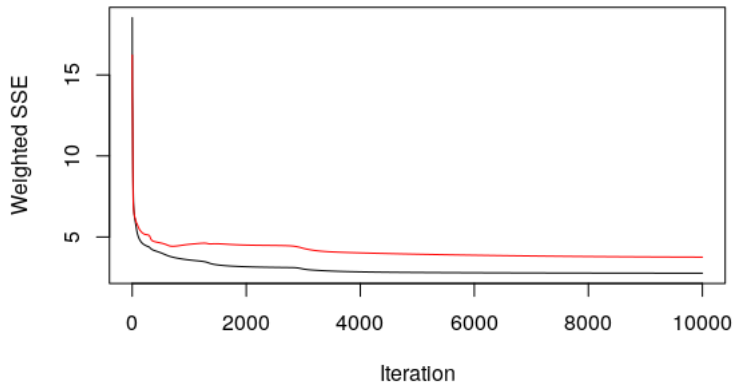

Figura 3.7: Gráfica MLP 0.2 - 50,50 ocultas - 6000 ciclos ($f_a(x) = \frac{1-e^{-x}}{1+e^{-x}}$)

$\gamma = 0{,}1$, $ciclos = 2000$ **y** $n_{ocultas} = 100, 50$ **(**$f_a(x) = \frac{1}{1+e^{-x}}$**)**

Con una tasa de 0.1, 50 y 100 neuronas en dos capas ocultas y 2000 ciclos (3.8), tenemos un buen modelo. El modelo es capaz de generalizar bien el conjunto de entrenamiento con un error de 0.00415175928712051 mientras que en test se produce un error de 0.00448553303982486. Utilizando la función sigmoidal, vemos que hemos renunciado a cierta capacidad de generalización sobre el conjunto de entrenamiento, a cambio de mejorar la generalización sobre otros conjuntos.

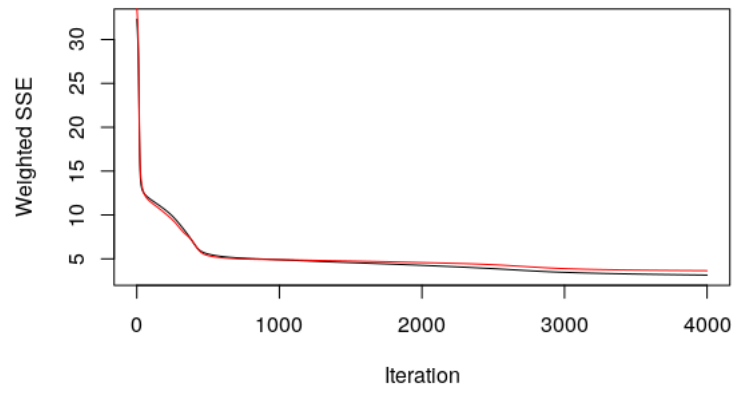

Figura 3.8: Gráfica MLP 0.1 - 100,50 ocultas - 2000 ciclos ($f_a(x) = \frac{1}{1+e^{-x}}$)

$\gamma = 0{,}1$, $ciclos = 2000$ **y** $n_{ocultas} = 100, 50$ **(**$f_a(x) = \frac{1-e^{-x}}{1+e^{-x}}$**)**

Con una tasa de 0.1, 50 y 100 neuronas en dos capas ocultas y 2000 ciclos (3.9), utilizando la función tangente hiperbólica vemos que el error de test se dispara y si aumentásemos los ciclos tendríamos que el error de test diverge. Por tanto no es adecuado este modelo.

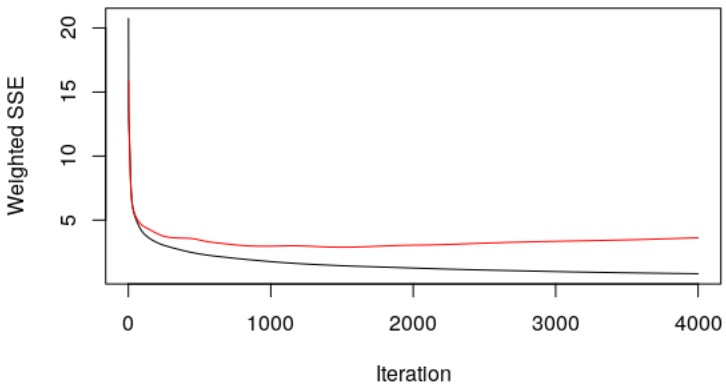

Figura 3.9: Gráfica MLP 0.1 - 100,50 ocultas - 2000 ciclos ($f_a(x) = \frac{1-e^{-x}}{1+e^{-x}}$)

$\gamma = 0{,}0001$**,** $ciclos = 40000$ **y** $n_{ocultas} = 100, 50, 25$ **(**$f_a(x) = \frac{1}{1+e^{-x}}$**)**

Con una tasa de 0.0001, 100, 50 y 25 neuronas en tres capas ocultas y 40000 ciclos (3.10), generaliza el conjunto de entrenamiento con un error de 0.0381546433606143, y el conjunto de test con un error de 0.039593033458852. Se ve que el modelo no es en absoluto apropiado puesto que el error es demasiado elevado y hay demasiada diferencia entre ambos errores.

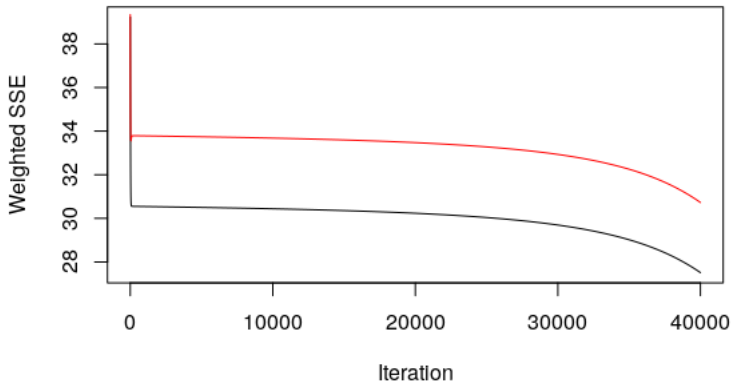

Figura 3.10: Gráfica MLP 0.0001 - 100,50,25 ocultas - 40000 ciclos ($f_a(x) = \frac{1}{1+e^{-x}}$)

$\gamma = 0{,}0001$**,** $ciclos = 40000$ **y** $n_{ocultas} = 100, 50, 25$ **(**$f_a(x) = \frac{1-e^{-x}}{1+e^{-x}}$**)**

Con una tasa de 0.0001, 100, 50 y 25 neuronas en tres capas ocultas y 40000 ciclos (3.11), utilizando la función tangente hiperbólica tenemos un error sobre el conjunto de entrenamiento de 0.00528260390284608 , un error sobre el conjunto de test de 0.00569856426934457. Vemos que los errores son semejantes. Comparando en este caso ambas funciones, vemos que en este caso particular, la función de activación tangente hiperbólica genera mejor modelo que la sigmoidal.

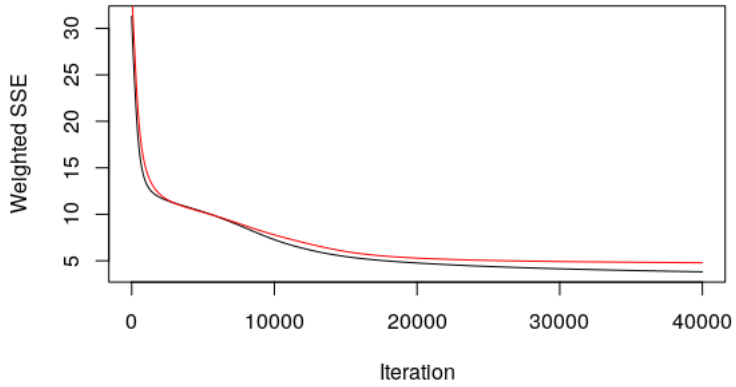

Figura 3.11: Gráfica MLP 0.0001 - 100,50,25 ocultas - 40000 ciclos ($f_a(x) = \frac{1-e^{-x}}{1+e^{-x}}$)

3.3 Resumen experimentación Perceptrón Multicapa

Tabla 3.1: Resumen experimentación MLP

Tasa de aprendizaje	Error entrenamiento	Error validación	Error test
0.8 50 ocultas sigmoidal	0.00215141436100276	0.00517340674087032	0.00305097518407372
0.8 50 ocultas tangente hiperbólica	0.00429123031163413	0.0052114219011533	0.00604587325351324
0.2 50, 50 ocultas sigmoidal	0.00283018784738848	0.00485646757920353	0.00431462177153583
0.2 50, 50 ocultas tangente hiperbólica	0.000889548028478484	0.00553734379737086	0.00346949202753306
0.1 100, 50 ocultas sigmoidal	0.00415175928712051	0.0050566079914164	0.00448553303982486
0.1 100, 50 ocultas tangente hiperbólica	0.00131125349800932	0.00502077763421073	0.00533692446637476
0.0001 100, 50, 25 ocultas sigmoidal	0.0381546433606143	0.0426178103593746	0.039593033458852
0.0001 100, 50, 25 ocultas tangente hiperbólica	0.00528260390284608	0.00663522559427817	0.00569856426934457

En la tabla 3.1 se observa claramente que el mejor modelo es el último considerado, puesto que a pesar de que presenta un error mayor que el primero, es capaz de mantener la diferencia entre los errores muy baja. Esto significa que si siguiésemos presentando conjuntos diferentes de datos sería capaz de generalizarlos correctamente. Por ello se considera que es el mejor modelo. Las salidas obtenidas vs las salidas deseadas las podemos apreciar en la gráfica 3.12.

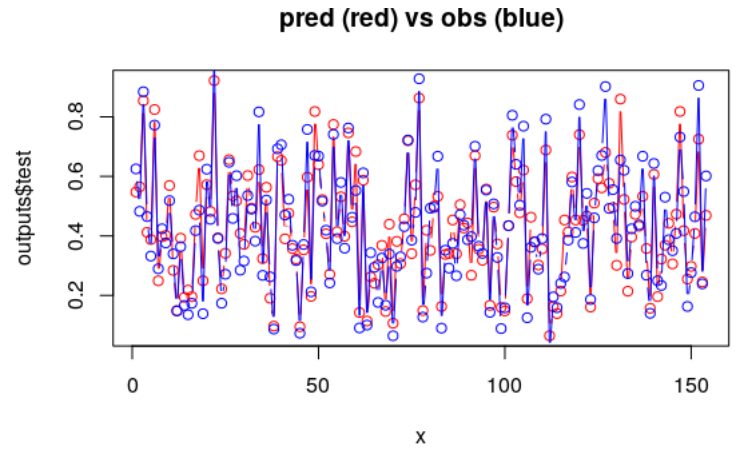

Figura 3.12: Salidas obtenidas vs salidas deseadas para tasa de aprendizaje 0.0001 neuronas: 100, 50, 25

CAPÍTULO 4
Conclusiones

Analizados ambos modelos, estamos en disposición de valorar objetivamente cual de ellos da mejor resultado para el problema que teníamos que resolver: para ello veamos los dos modelos elegidos para cada tipo, y también la tabla de resultados de cada uno de los dos modelos. La diferencia entre ambos modelos es que en Adaline teníamos la posibilidad de modificar ciclos y tasa de aprendizaje mientras que en Perceptrón Multicapa hemos visto que además se pueden añadir capas de neuronas ocultas para hacer el modelo más complejo. El error de test que obtenemos en el mejor modelo escogido por Adaline es de 0.019147481314295 mientras que el error de test que obtenemos en el mejor modelo escogido MLP es de 0.00569856426934457. Como vemos resulta evidente que el error mejora significativamente en un factor de $\frac{1}{4}$ aproximadamente. Consideramos entonces que una red MLP aproxima mejor y es capaz de generalizar mejor en este caso concreto, aunque hay que tener cautela porque es fácil dejarse llevar y añadir las suficientes capas para que MLP aproxime exactamente el conjunto de entrenamiento, que como ya hemos visto en 3.9 no nos lleva más que a una red que posteriormente fallará en la generalización de otros conjuntos de datos.

Es importante resaltar que aunque parezca que la diferencia es mínima en cuanto se "desnormalizan"los datos vemos que algunas salidas tanto en 3.12 como en 2.6 presentan errores de una magnitud bastante importante, y que en estos casos nuestras redes fallan en su generalización.

En cuanto a la práctica en general, pienso que ha habido problemas de rendimiento tanto en el script en R como en mi programa en PHP debido a que el proceso de aprendizaje si los ciclos son muy elevados he llegado a ver ejecuciones de veinte minutos. Lo cual ha provocado tener que invertir más tiempo en la práctica. Sería bueno plantearse mejorar el rendimiento de estos programas aunque se entiende que no es el objetivo principal.

Bibliografía

[1] Conjunto de datos del hormigón, Knowledge Extraction based on Evolutionary Learning. Obtenido de http://sci2s.ugr.es/keel/dataset.php?cod=44.

[2] Primeros modelos computacionales, Inés M. Galván y José Mª Valls. Obtenido de http://ocw.uc3m.es/ingenieria-informatica/redes-de-neuronas-artificiales/transparencias/material-de-clase.-tema-2/view.

[3] Peceptrón Multicapa, Inés M. Galván y José Mª Valls. Obtenido de http://ocw.uc3m.es/ingenieria-informatica/redes-de-neuronas-artificiales/transparencias/material-de-clase.-tema-3/view.