Redes de Neuronas Artificiales

GRADO EN INGENIERÍA INFORMÁTICA

Práctica 1: ADALINE y Perceptrón Multicapa

Curso 2021/2022

Jorge Rodríguez Fraile, 100405951, Grupo 83, 100405951@alumnos.uc3m.es
Carlos Rubio Olivares, 100405834, Grupo 83, 100405834@alumnos.uc3m.es

Índice

Introducción

En esta práctica vamos a desarrollar dos modelos de redes de neuronas. El primero es lo más simple que se puede desarrollar para hacer regresión, que es ADALINE (ADAptative LINear Element), como este modelo es bastante simple, vamos a implementarlo nosotros mismos para conocer mejor su funcionamiento. El otro modelo de red neuronal que emplearemos en este proyecto será un perceptrón multicapa, en este caso al ser más complejo emplearemos el SNNS (Stuttgart Neural Network Simulator) implementado en lenguaje R que se nos ha proporcionado.

Ambos tipos de redes neuronales se utilizarán para la misma tarea, crear un modelo capaz de predecir la resistencia a la compresión del hormigón dada su antigüedad y cantidad de sus 7 componentes (cemento, escoria de alto horno, cenizas volantes, agua, superplastificante, agregado grueso y agregado fino).

ADALINE

Modelo

Para el modelo ADALINE hemos utilizado dos programas de python, processData.py y main.py.
En el primero de los ficheros nos centramos en procesar las instancias proporcionadas, para ello lo primero es aleatorizar la posición de las instancias, normalizar los valores y crear los conjuntos de entrenamiento, validación y test, dividiendo las instancias en 70 %, 15 % y 15 % respectivamente. Para esta finalidad hemos utilizado la librería panda y numpy.
En cuanto al código en sí no hay mucho que mostrar, ya que simplemente son llamadas a funciones de las librerías para poder tener los datos entre valores de 0 y 1, y luego escoger aleatoriamente un porcentaje de los mismos y separarlos en ficheros.

En el fichero main.py es donde cae todo el cálculo del modelo ADALINE.
Empezamos por funciones simples como *desnormalization* que simplemente desnormaliza los datos del fichero para poder mostrar los datos predichos y compararlos. Pero el verdadero algoritmo comienza en *createDataTraining* donde se escogen los datos a analizar y se crean de manera aleatoria los pesos más el umbral.
Una vez tenemos los datos iniciales, pasamos a *calculateOutput* que simplemente hace el producto vectorial de las entradas (en la parte de clase se pone 1) y los pesos con el umbral.
El siguiente paso se realiza en *adjustWeights* donde teniendo en cuenta el factor de aprendizaje y el output de la función anterior para actualizar los pesos.
Por último, *calculateErrors* calcula el error cuadrático medio (mse) y error absoluto medio (mae) del ciclo en el que nos encontramos.
Una vez tenemos codificadas todas las fases del algoritmo, la función *ADALINE* se encarga de llamar a todas las anteriores en el orden correcto. Se crean todos los datos (validación, test y entrenamiento) y en cada ciclo se realiza lo siguiente:

1. Para cada patrón de los datos de entrenamiento
   1.1. Se calcula su salida con *calculateOutput*.
   1.2. Se ajustan los pesos y umbral en función de las salidas que obtenemos con *adjustWeights*.
2. Se obtienen los errores de entrenamiento con *calculateErrors*.
3. Se realizan los pasos 1.1 y 2 para el conjunto de datos de validación, de esta manera probamos el modelo. Si no se ha completado todos los ciclos de entrenamiento se vuelve al paso 1, si no vamos al paso 4.
4. Por último, ya generado el modelo, se realiza el paso 1.1 y 2 para los datos de test, solo que esta vez al obtener la salida desnormalizamos los valores para su posterior comparación. Para terminar se calculan los errores para el test y se termina el programa.

Pruebas

Para las pruebas de nuestro modelo de ADALINE hemos optado por los siguientes valores para el factor de aprendizaje (γ) y número de ciclos:

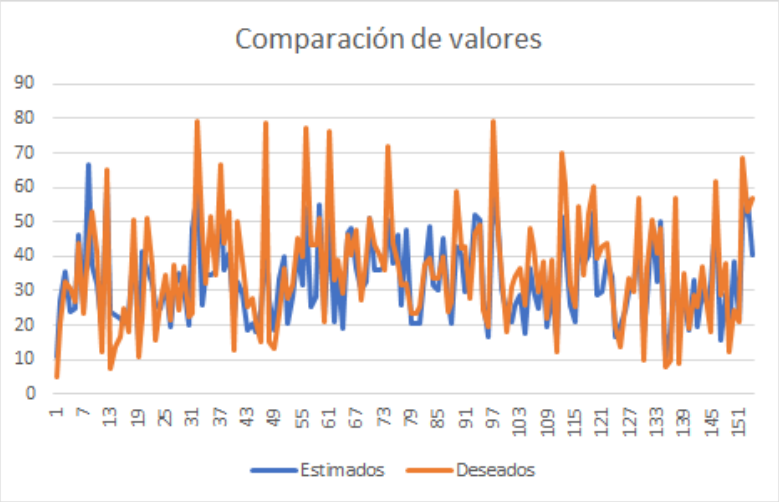
F. APRENDIZAJE	CICLOS	E. ENTREN.	E. VAL.	E. TEST
0,00001	5000	0,020999608743	0,020023868456	0,017685480517
0,0005	2500	0,017485230225	0,015508727709	0,014759172376
0,0005	1000	0,017440106987	0,015510844798	0,014680363775
0,001	1000	0,017590225818	0,015657830604	0,014879563365
0,001	250	0,017809979509	0,015920972657	0,015034440613
0,01	100	0,017744081989	0,016105837200	0,015421134070
0,01	20	0,017861438127	0,016298365566	0,015524721043

Los errores que se muestran en esta tabla son el error cuadrático medio.

Hemos tenido en cuenta estas variaciones, ya que creemos que son las que mejor reflejan la mejora constante de los errores sin excederse en número de ciclos. Además, cuentan con un cambio suficientemente significativo como para tenerlos en cuenta, unos modelos son más progresivos al variar sus pesos en menor medida y otros más bruscos al cambiar más rápido, en la siguiente sección se podrá apreciar.

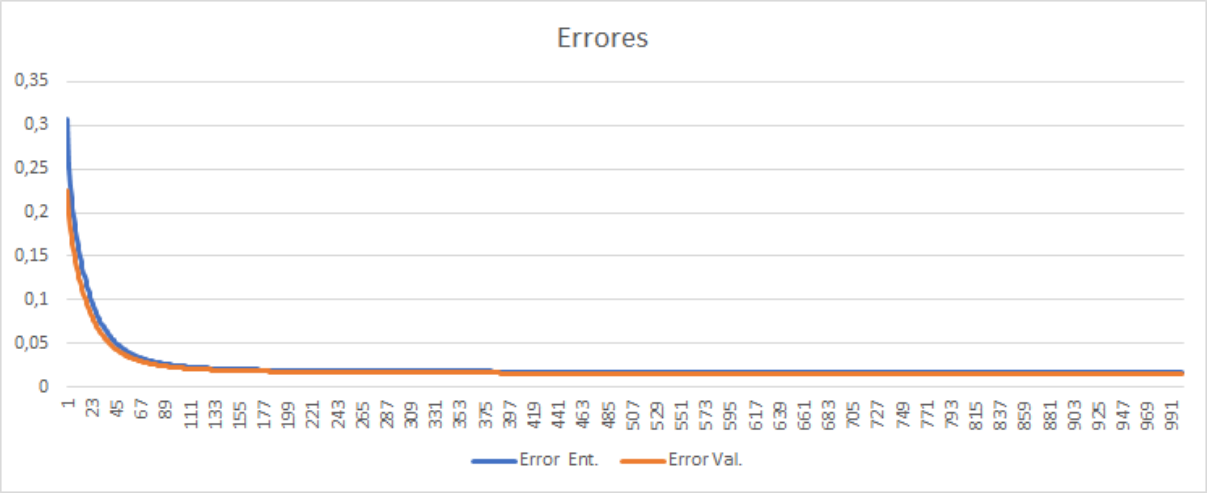
Resultados

El modelo con mejor resultado ha sido el de **factor de aprendizaje 0,0005**, **1000 ciclos** y pesos iniciales [0,62932481 0,44815587 0,21296195 -0,1983607 0,13660222 0,08460019 0,08899163 0,50673283 -0,07228504], que nos ha dado un **error cuadrático medio de test de 0,01475**. Una vez elegido el modelo más eficaz, hemos pasado a comparar los valores predichos con los reales, siendo la gráfica resultante la siguiente:



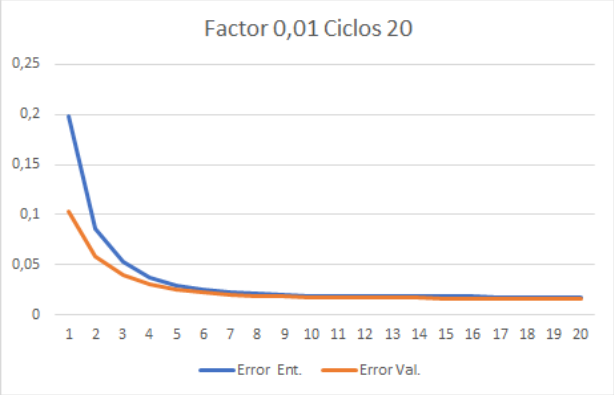
La gráfica azul viene dada por los valores predichos y la naranja por los verdaderos valores. Podemos ver que en algunos puntos se asemejan bastante, pero hay algunos picos como en la instancia 74 en la que hay una diferencia bastante grande, de 21,573 unidades. Aun así creemos que es una diferencia bastante aceptable teniendo en cuenta que estamos utilizando un modelo lineal y en todo lo que se basa para darnos el valor es en un único hiperplano.

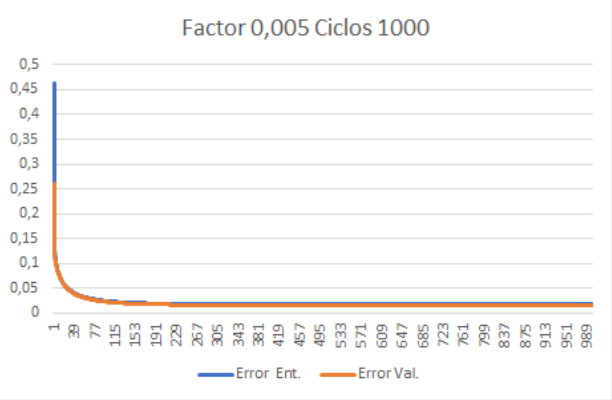
Por otro lado podemos observar en la siguiente gráfica el proceso de aprendizaje del modelo representado por el error cuadrático medio para el conjunto de entrenamiento y validación en cada uno de los ciclos:

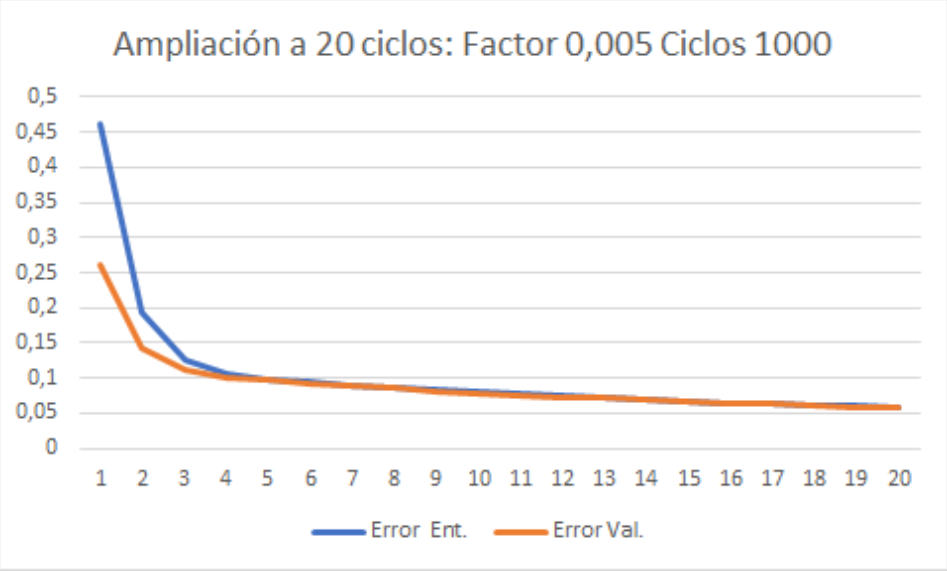


Podemos ver que en las primeras iteraciones el error baja de manera exponencial, dando paso a un ritmo de mejora del error mucho más bajo. Aunque no se aprecia mucho, sigue mejorando hasta aproximadamente los 950 ciclos, a partir de ahí, la mejora es casi inexistente.

Para ampliar nuestras pruebas, hemos optado por elegir dos modelos más para evaluar la diferencia de los errores, el modelo de factor 0,01 y 20 ciclos, y el modelo de 0,0005 de factor de aprendizaje y 1000 ciclos. En las gráficas se observa como el error avanza más rápida y progresivamente cuando se utiliza un factor de aprendizaje mayor, aunque esto puede provocar que al modificar los pesos se produzca oscilación al cambiar más rápido, pero en este caso la mejora del error no es tan significativa para que se aprecie fácilmente:







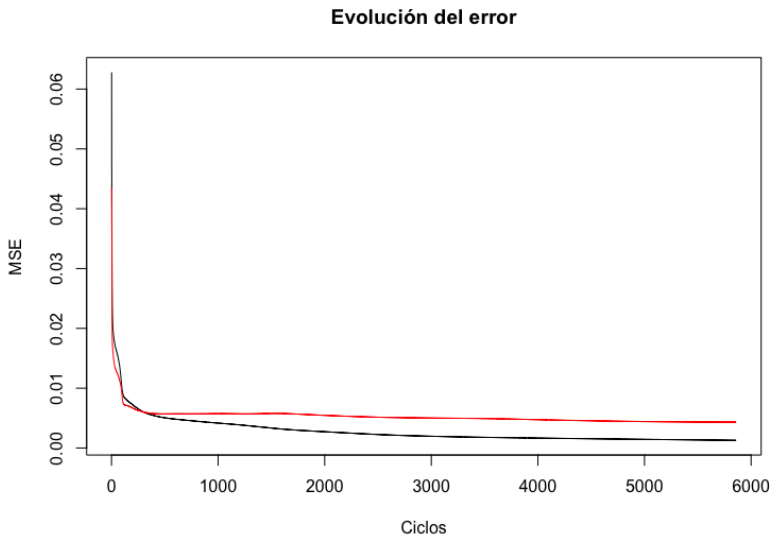
Perceptrón Multicapa

En esta parte del trabajo se procede a utilizar un perceptrón multicapa para evaluar los datos de la primera parte y observar si los resultados obtenidos son mejores. En teoría deberían ser bastante mejores y ofrecer una accesibilidad mayor. Para ello utilizaremos el lenguaje R, que tiene un enfoque de análisis estadístico.

Empezamos con una sola capa, con 60 neuronas y dejamos el mismo factor de aprendizaje que en el apartado de ADALINE, pero esta vez aumentamos los ciclos considerablemente a 100000 debido a que la rapidez del programa es evidente. Con esto obtenemos una media de error de test de 0,006993727 que es muchísimo mejor que los resultados obtenidos con ADALINE. Aun así todavía tenemos más posibilidades que probar y creemos que podemos mejorar estos valores.

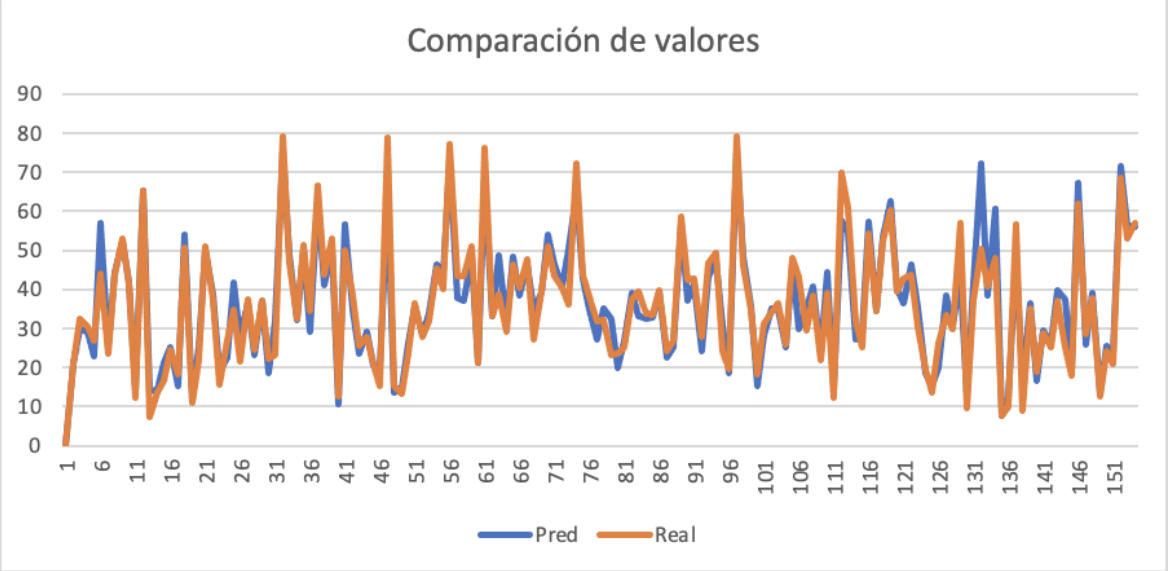
Para ello, aumentamos el factor de aprendizaje en diferentes modelos que no aparecen por exceso de información en la memoria, pero obtenemos mejores resultados con un 0,75 de factor de aprendizaje por lo que lo dejamos fijado a partir de ahora. Una vez ajustado este valor proseguimos y vamos aumentando el n.º de neuronas. Nuestro segundo modelo tendrá 80 neuronas, y un factor de aprendizaje de 0,75; con este modelo obtenemos un error de test de 0,00505. Se ve que vamos por un buen camino, los ciclos y el factor de aprendizaje lo damos por ajustados, por lo que pasamos a probar con más capas.

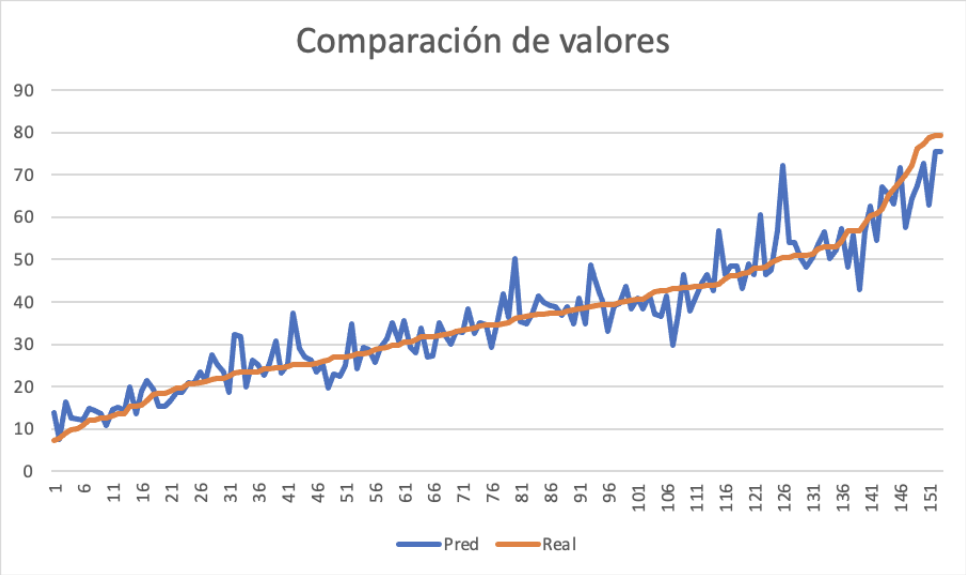
En el último modelo que añadimos dos capas, como el último modelo nos ha dado muy buenos resultados con 80 neuronas hacemos precisamente eso, ‘desdoblamos’ el segundo modelo y obtenemos como resultado dos capas ocultas con 80 neuronas cada una, un factor de aprendizaje de 0,75 y necesitando solo 5860 ciclos de los 100000 ciclos máximos que le damos. El valor de error que obtenemos de este modelo es de 0,003834652 con lo que reducimos casi a la mitad la efectividad de nuestro modelo anterior, y ni que hablar tiene que hemos eclipsado a los modelos de ADALINE.



Estos valores nos parecen los suficientemente buenos para dejar de seguir probando modelos, por lo que dejamos aquí la generación de los mismos y se muestran la tabla de los modelos que hemos creado y la gráfica de comparación de valores del mejor de ellos:

F. APRENDIZAJE	CICLOS	CAPAS	NEURONAS	E. ENTREN.	E. VAL.	E. TEST
0,0005	100000	1	60	0,008338984	0,007791840	0,006993727
0,0005	938379	1	60	0,004015006	0,005854441	0,005314565
0,01	100000	1	60	0,003054421	0,005428635	0,004240335
0,05	86335	1	60	0,001941051	0,004601703	0,004401978
0,75	2987	1	60	0,001782846	0,004505337	0,004582732
0,1	22927	1	60	0,002524447	0,004765384	0,004136684
0,75	4184	1	40	0,002900788	0,004302903	0,003956101
0,75	55718	1	80	0,001177719	0,004136665	0,005059619
0,75	21969	1	100	0,001624175	0,003730599	0,004080071
0,75	3963	2	40, 40	0,001693033	0,004952612	0,004301941
0,75	15323	2	20, 60	0,000935623	0,004044325	0,006112885
0,75	5860	2	80, 80	0,001294568	0,004331397	0,003834652
0,75	5953	2	100, 100	0,001443807	0,004544292	0,005852750





Teniendo en cuenta estos resultados, podemos ver que obviamente, cuanto menor sea el factor de aprendizaje usado, más ciclos harán falta para llegar a una medida de error óptima. También es recalcable que al principio se pensaba que nuestros modelos bajaban indefinidamente al tan solo mirar los gráficos y en las primeras pruebas, pero se puede ver que en algunos modelos el error mínimo se obtiene antes de que terminen los ciclos.

Comparación de ADALINE y Perceptrón multicapa

Hemos visto en el apartado del perceptrón multicapa que los valores obtenidos son muchísimo mejores que los de ADALINE, además, la manera en la que hemos conseguido los resultados con el perceptrón multicapa son bastante más sencillos que todo el código que hemos generado en ADALINE. En definitiva, el perceptrón multicapa nos ofrece una mayor variedad de posibles modelos, una ejecución más sencilla y sobre todo unos resultados más satisfactorios.

Conclusiones

Esta práctica nos ha enseñado a iniciarnos en el mundo de las redes neuronales, además de poder indagar en el proceso que conlleva mediante la creación de código. La creación de modelos y saber cómo manejar los atributos y sus variables también cómo ajustarlos. Incluso hemos podido tener una toma de contacto con el lenguaje de programación R y ver como funciona en sus capas más superficiales. En definitiva creemos que hemos hecho un buen trabajo en cuanto a la generación de modelos y pruebas se refiere y estamos satisfechos con los resultados obtenidos.