Práctica 1. Problema de Regresión
Redes de Neuronas Artificiales

Bogdan Andrei Baltes - 100406687

Álvaro Marco Pérez - 100383382

1. Introducción

En esta práctica implementamos dos de los modelos aprendidos en la asignatura de Redes de Neuronas Artificiales, ADALINE y Perceptrón Multicapa, con la finalidad de predecir el valor de la resistencia del hormigón, frente a una fuerza de compresión, teniendo en cuenta características del mismo.

Durante el desempeño de la práctica, realizamos un proceso iterativo de comparación y selección de aquellos modelos que mejoran la implementación del modelo para conjunto de datos.

La memoria se estructurará de la siguiente manera:

2. Preproceso

Antes del desarrollo de los modelos de regresión, se nos pedía realizar un preprocesado del conjunto de datos para obtener tres conjuntos de datos aislados, con los que entrenar, validar y, por último, comprobar el desempeño del modelo de salida. Para ello hicimos uso de las librerías de Python numpy, pandas y sklearn, siguiendo el siguiente procedimiento:

1. Normalización de los datos. para evitar que el rango de las unidades de los valores de entrada afectaran en los pesos del modelo. Usando las herramientas implementadas en panda y sklearn, calculamos los valores utilizando el MinMaxScaler de sklearn, con su valor por defecto [0, 1].

2. Aleatorización de los datos: para evitar que se produzca un sesgo en el modelo debido a un determinado orden en el conjunto de datos, se aleatorizan las entradas, haciendo uso de la herramienta split de la librería numpy.

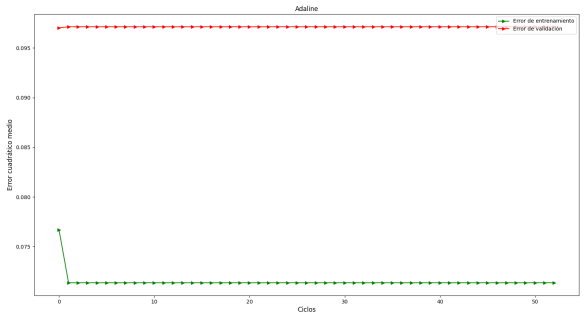
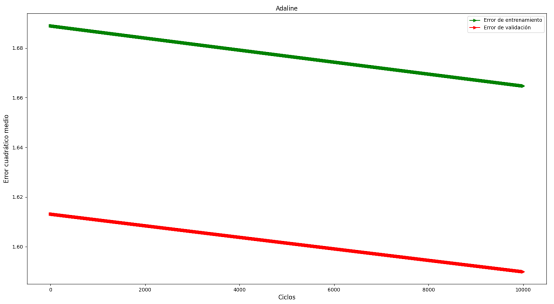
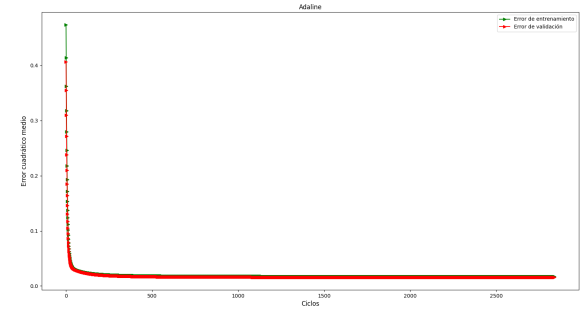
3. Separación de los datos en tres conjuntos: para la realización de la validación y comprobación del modelo, de tal modo que la red no haya usado previamente esos datos para su entrenamiento, es necesario dividirlo en tres partes. Esta acción se hace al mismo tiempo que la aleatorización de los datos, por medio de la herramienta split de numpy:

```
datos = np.split(dataf.sample(frac=1), [int(.7*len(dataf)), int(.85*len(dataf))])
```

3. ADALINE

El primer modelo desarrollado es un modelo de regresión lineal, con ajuste de pesos. El modelo es presentado en 1960 por Bernard Widrow y su alumno Ted Hoff, y para su implementación hemos seguido los pasos explicados en el libro *Redes de Neuronas Artificales, Un Enfoque Práctico* (Pedro Isasi Viñuela, Inés M. Galván León).

Este es un algoritmo de regresión lineal, que iterativamente va ajustando los pesos del modelo, con el objetivo de minimizar el error en la predicción. La naturaleza de este crecimiento depende de los hiperparámetros escogidos, tal y como se puede observar en los siguientes ejemplos:

Razón de Aprendizaje	Gráfica Evolución del Error	Explicación
1		En este caso, podemos observar que el error de validación tiene una gran diferencia con el error de entrenamiento y que este último es mayor que en otros casos. Esto es porque un factor de aprendizaje tan grande no permite granularizar el problema.
0.0000000000001		En este caso podemos observar la cara de la moneda contraria, al ser el factor de aprendizaje tan pequeño, el modelo aprende muy lentamente.
0.0001		En este caso podemos observar que la red aprende con bastante rapidez, y que es capaz de llegar al mínimo de error en el conjunto de datos de validación, con poco coste computacional.

Una vez obtenidos un algoritmo que implementase el modelo computacional, comenzamos a probar la modificación de los hiperparámetros de entrada, con los siguientes resultados:

Ciclos	Factor de Aprendizaje	Error de entrenamiento	Error de validación	Error de test
100, 1000, 4000	1	0.06622643783670	0.094505119969887	0.099466311847245
100, 1000, 4000	0.3	0.02298298086805	0.030979354311210	0.034899241764951

100, 1000, 4000	0.01	0.018576553055131	0.0156650311225741	0.018457308419069
100	0.001	0.01767844653441	0.015978691235033	0.017802845138057
1000, 4000	0.001	0.017091098314727	0.01604924882275	0.01639850000373
100	0.0001	0.026508617129536	0.026508617129536	0.02496403998873
1000	0.0001	0.017474672966663	0.0162630151783105	0.0174114736733520
4000	0.0001	0.017483443585120	0.01564370324924	0.017269152690684
100	0.00001	0.320951791987369	0.298611661726880	0.303868757110948
1000	0.00001	0.027433282710106	0.0258351119147448	0.025797851147292
4000	0.00001	0.021020674317981	0.017733409195614	0.021644527186604

Para la configuración de los hiperparámetros cuyo factor de aprendizaje es mayor que la milésima, el algoritmo se detiene antes de los 100 epochs (iteraciones), debido a que el error de validación empieza a crecer. Por ello, unimos los valores de 100, 1000 y 4000 epochs.

Podemos observar que los hiperparámetros que minimizan el error de entrenamiento y validación son:

Ciclos	Factor de Aprendizaje
4000	0.0001

Estos valores permiten que la red se entrene hasta que el error de validación comienza a crecer y el aprendizaje sea lo más granular posible. El error de test obtenido para esta configuración es 0.017269152690684.

4. Perceptrón Multicapa

Para la segunda parte de la práctica, se ha utilizado el simulador SNNS bajo el lenguaje de programación R, mediante RStudio. Así, usando el script de R proporcionado, se han ido haciendo diversos experimentos, modificando por una parte la razón de aprendizaje (adaptando los ciclos necesarios a cada una) y por otra la topología del sistema (número de capas ocultas y número de neuronas en cada capa). Se indicará en color verde la mejor configuración, y en color amarillo algunas otras configuraciones que pudiesen ser buenas candidatas a proporcionar una buena solución.

Para los primeros experimentos, se ha empleado una topología de dos capas ocultas de cuatro neuronas cada una - representada como (4,4) - y se ha ido modificando la razón de aprendizaje, como se puede ver en la siguiente tabla.

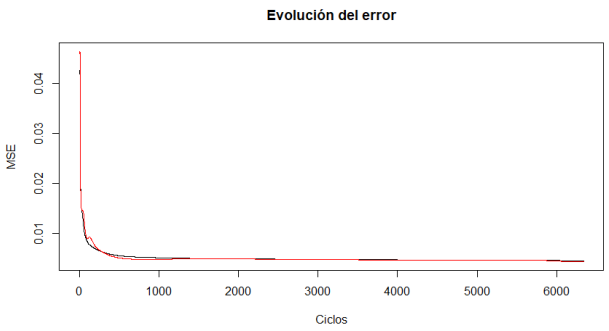
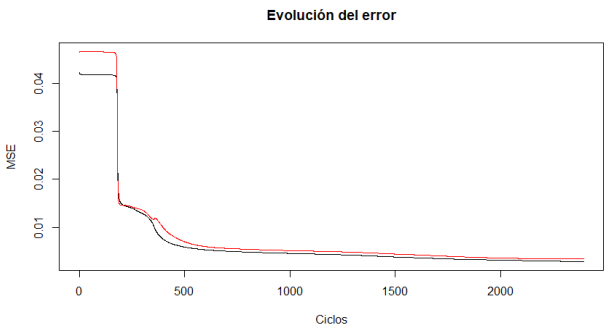
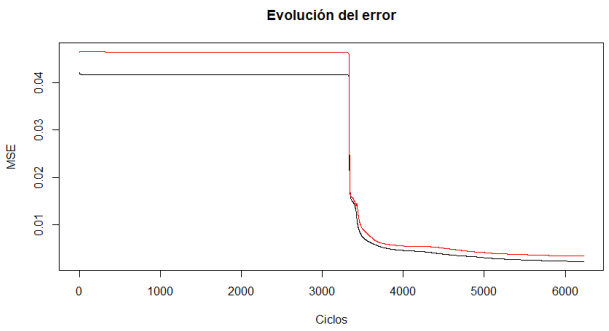
Ciclos	Factor de Aprendizaje	Error de entrenamiento	Error de validación	Error de test
8000 (6337*)	1	0,00417082410676	0,00441954585288	0,00544345362027
8000	0.3	0,004817779322051	0,00518336790990	0,006369232170782
8000	0.01	0,00668201056072	0,00568514505948	0,00653003561030
8000	0.0001	0,041159812963814	0,04318090345007	0,0531312388531846
8000 (4007*)	0.00001	0,041177394563212	0,043187996947117	0,05318354133045

Como se puede observar, la mejor configuración viene dada por el factor de aprendizaje 1, por lo que se mantiene ese factor de aprendizaje para los siguientes experimentos. En estos, se va modificando la topología del PM, partiendo de la de (4,4) de los anteriores experimentos.

Ciclos	Topología	Error de entrenamiento	Error de validación	Error de test
8000 (6337*)	(4, 4)	0.00417082410676	0.00441954585288	0.00544345362027
8000	(3, 3)	0.00497649323894	0.00514689675548	0.00625004659708
8000 (831*)	(5, 5)	0.00468129000076	0.005689954773141	0.00646125965440
8000	(4, 4, 4)	0.00463422164839	0.00525041805435	0.00530559359820
8000	(6, 4, 2)	0.00382854653309	0.00551309673752	0.005074974733311
25000	(6, 4, 2)	0.003460189891011	0.00539281405556	0.00492034187086

8000 (2833*)	(10,10)	0.00263246472438	0.00408071825766	0.00407085186964
8000 (2392*)	(10,5,2)	0.00260206120130	0.003473212651980	0.0039684132521511
8000 (6224*)	(10,8,5,2)	0.0020493430420	0.00343028009362	0.003312886159440

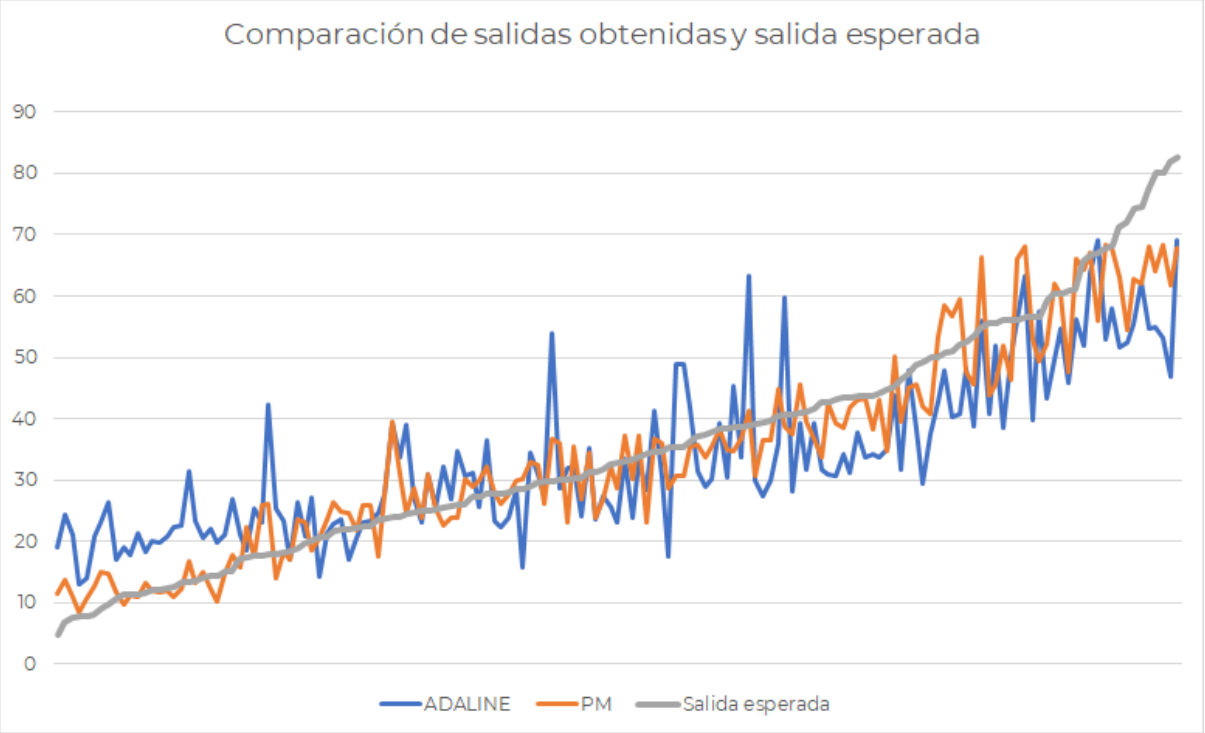
Como se puede observar en la tabla anterior, la configuración de (4,4) se mantiene como una de las mejores. Sin embargo, la topología que minimiza el error de validación es la de cuatro capas de 10, 8, 5 y 2 neuronas, respectivamente - (10,8,5,2) - seguida muy de cerca por la topología de (10,5,2). Estas tres configuraciones se pueden observar en la siguiente tabla, junto a sus gráficas de evolución del error.

Topología	Gráfica Evolución del Error	Explicación
(4, 4)		Tal y como se puede observar en las gráficas de la izquierda, a mayor número de neuronas y, sobre todo, a mayor número de capas, el aprendizaje de la red tarda en permeabilizar más ciclos.
(10, 5, 2)		
(10, 8, 5, 2)		

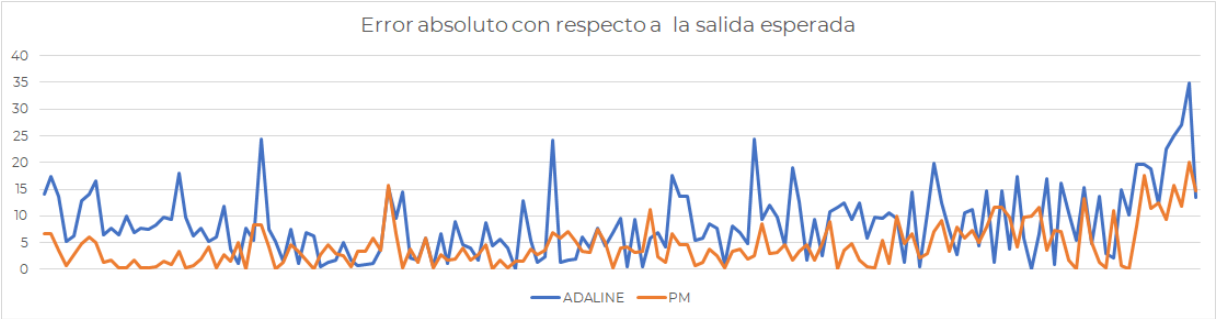
5. Comparación de ambos modelos y conclusiones

Después de ambos estudios utilizando los distintos algoritmos, es importante comparar los funcionamientos de ambos.

Para ello, se pueden apreciar los comportamientos de ambos algoritmos (ADALINE y PM) en la siguiente gráfica. Para obtenerla, fueron utilizadas las salidas para los datos de test utilizando los resultados correspondientes al mejor experimento en cada caso. Para una mejor visualización, los resultados esperados fueron ordenados de menor a mayor en la unidad original, desnormalizada: megapascales (MPa).



En la anterior gráfica podemos observar que el recorrido de los resultados del Perceptrón Multicapa son, por lo general, mucho más cercanos a la salida esperada que los obtenidos mediante ADALINE. También se puede comprobar esto en la siguiente gráfica, donde aparece el error absoluto de cada uno de los algoritmos (en megapascales) con respecto a la salida esperada.



Se puede apreciar que en la mayoría de los casos, el Perceptrón Multicapa devuelve valores más cercanos a 0 que el ADALINE. Solo en un 28% de los casos (42 de 155), ADALINE devuelve una predicción más cercana al valor real que el Perceptrón Multicapa.

Esto es debido a que el modelo generado por ADALINE es lineal, mientras que el modelo generado por el Perceptrón Multicapa es no lineal. Esto permite que el modelo generado se ajuste mucho más precisamente a los datos de entrenamiento que los del modelo lineal.

Al inicio de la práctica se asumía que las predicciones del Perceptrón Multicapa llegarían a ser más precisas que las del ADALINE, ya que al tener múltiples capas, tiene la capacidad de resolver problemas que no son linealmente separables. En este caso, por ejemplo, la mejor solución del Perceptrón Multicapa está formada por cuatro capas. Sin embargo, el ADALINE resuelve el problema con un error considerablemente bajo en muchos de los casos observados, por lo que sería una buena alternativa al PM si este último no se pudiese usar.

Consideramos que este trabajo ha sido muy interesante para poner en práctica los conocimientos teóricos adquiridos en clase sobre los distintos modelos existentes, que ha servido para asentar unas bases sobre su funcionamiento y familiarizarnos con conceptos propios de cada uno de ellos. Además, nos ha servido para ser más conscientes de cómo funcionan por dentro cada uno de estos modelos, así como de cuál es la carga de trabajo necesaria para su implementación. Vemos de vital importancia esto último también de cara a futuros trabajos en los que nos veamos en la necesidad de implementar redes de neuronas, tanto en la vida universitaria como en la laboral.