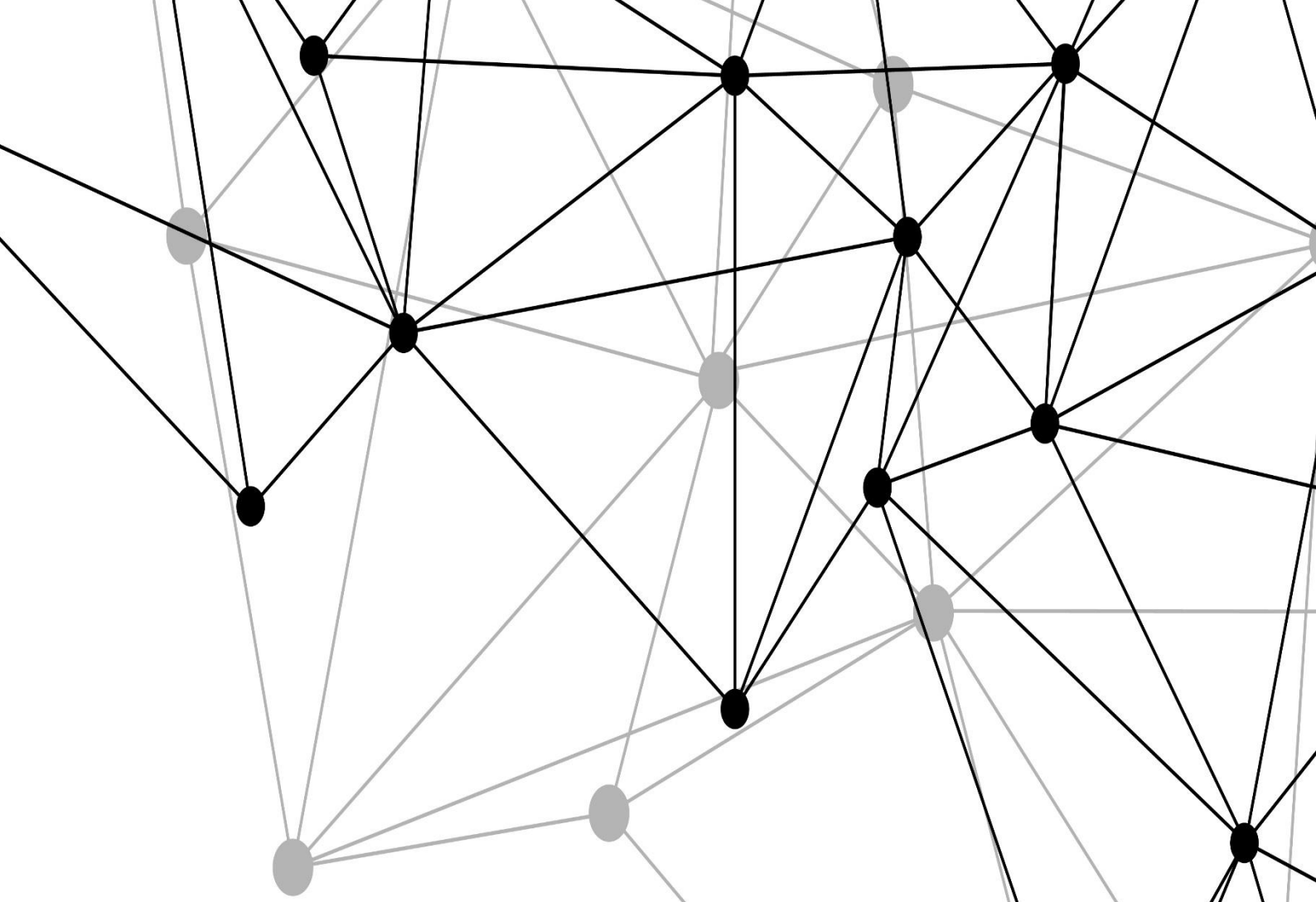


UC3M Campus de Colmenarejo

Adaline y visual studio R

Redes de Neuronas Artificiales

Lucas González de Alba | 100383228 | 100383228@alumnos.uc3m.es
Gonzalo Fernández García| 100383212 | 100383212@alumnos.uc3m.es

Ingeniería Informática, UC3M, Colmenarejo

Índice:

En el documento vamos a encontrar las siguientes secciones:

- **Introducción:** Contextualización del proyecto a realizar
- **Contexto:** Breve explicación de las herramientas
- **Preprocesado de los datos:** en esta parte se explicarán los procesos que experimentan los datos antes de ser usados para entrenar los modelos de redes neuronales.
- **Adaline**: sección que explica el funcionamiento del modelo Adaline y algunas decisiones de diseño que incorpora, además de incluir las diferentes pruebas, experimentos y un análisis de los resultados.
- **Perceptrón multicapa**: en esta sección se explica el funcionamiento del perceptrón multicapa usado además de incluir sus diferentes pruebas, experimentos y un análisis de los resultados.
- **Comparación**: en esta sesión se realizará un análisis comparativo de los resultados y conclusiones obtenidos de los dos modelos propuestos para la resolución del problema.
- **Conclusión**: por último, en esta parte del documento daremos nuestra opinión acerca del trabajo y recapitulamos las distintas fases que hemos pasado para su realización.

Introducción

El presente documento realiza un estudio en el que se comparan los resultados obtenidos para un mismo problema de dos modelos de redes neuronales, Adaline y percepción multicapa. El problema que se va a tratar consiste en predecir el precio medio de la vivienda para diferentes distritos de California a partir de información derivada del censo nacional de 1990. Los atributos que se van a utilizar para ello son:

Longitud media, latitud media, edad media de las casas, cantidad total de habitaciones, cantidad total de camas, cantidad total de grupos familiares, cantidad total de residentes y el ingreso medio de los grupos familiares de un distrito.

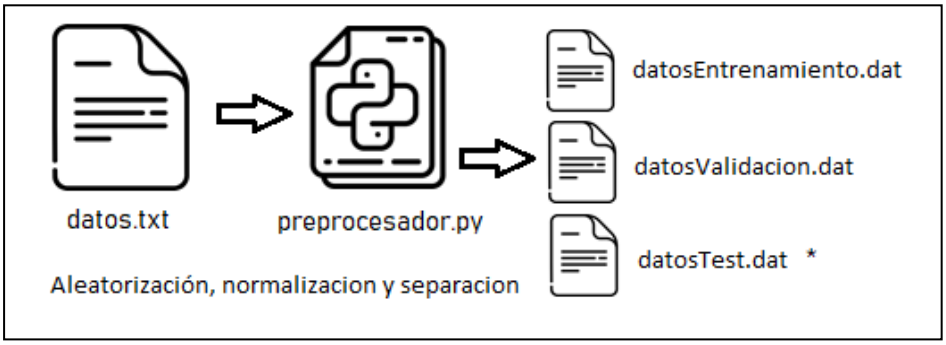
Contexto

El objetivo de esta práctica es afianzar el entendimiento de las redes neuronales a través del estudio de los fundamentos que las rigen. Por tanto, el primer caso que estudiamos es la arquitectura Adaline (ADAptative LINear Element) que es un modelo basado en el trabajo de McCulloch-Pitts sobre neuronas artificiales. Aun siendo algo rudimentario, este diseño permite aproximar de manera eficiente valores que sigan una regresión lineal. Desgraciadamente Adaline no aproxima bien valores resultantes de funciones no lineales (apreciación que hicieron Minsky y Papert en 1969), por lo que son necesarios otros modelos como el perceptrón multicapa.

Preprocesado de datos

Partimos de un archivo con 17,000 instancias de 8 atributos y 1 salida. Ejecutamos el preprocesador. Primero se aleatoriza el orden de los patrones (shuffle), luego se normalizan (buscando el mínimo y el máximo y obteniendo el valor entre 0 y 1) y finalmente se separan en tres conjuntos nuestros datos (60% entrenamiento, 30% validación, 30% test). *Adicionalmente, de manera interna se genera un cuarto archivo temporal con las salidas test sin normalizar para facilitarle el trabajo de desnormalización al programa Adaline.

- python procesadorDatos.py datos.txt ","



Adaline

Hemos implementado el modelo utilizando Python. Las librerías *Numpy* y *matplotlib* de Python datos han resultado clave en el desarrollo del programa puesto que la primera ha servido como base para construir el tratamiento matricial y vectorial de los datos y la segunda como representación gráfica de las salidas. A grandes rasgos el programa principal consta de los siguientes pasos:

1. Recogida de datos mediante parámetro e inicialización de pesos y umbral
    a. leerFicheroDatos(file,separador)
    b. pesos.append(random.uniform(-0.5, 0.5))
    c. iniciarHiperparametros(numeroCiclos,razon,numeroAtributos,pesos)

2. Ronda iterativa por ciclo (0 -> número de patrones)
    a. Entrenamiento: Calculamos las salidas según pesos y los ajustamos
        i. salidaDeseada = patron[numAtributos-1]
        ii. salidaObtenida = calculoSalida(patron,pesos)
        iii. ajustePesosUmbral(patron,pesos,salidaDeseada,salidaObtenida)
        iv. Guardamos error entrenamiento
    b. Validación: Calculamos las salidas según pesos
        i. salidaDeseada = patron[numAtributos-1]
        ii. salidaObtenida = calculoSalida(patron,pesos)
        iii. Guardamos error validación

3. Test: Calculamos las salidas según pesos y desnormalizamos las mismas
    a. salidaDeseada = patron[numAtributos-1]
    b. salidaObtenida = calculoSalida(patron,pesos)
    c. Guardamos error test
    d. desnormalizacionParametrosSalida(SalidasObtenidas,SalidasDeseadas)

4. Generación de gráficas
    a. Grafica para los valores de entrenamiento, validación y test
    b. Grafica para salidas obtenidas, y deseadas desnormalizadas

Para el estudio de Adaline se han establecido tres conjuntos de datos (entrenamiento, evaluación y test). Estos tres ficheros de patrones se han mantenido iguales para todas las pruebas ejecutadas garantizando así que estas se puedan replicar y que puedan ser comparables entre ellas. También nos hemos asegurado de que estos tres conjuntos no estaban sesgados tal y como se ha observado que sucedía en ocasiones.

Una vez que los conjuntos se han definido, hemos comenzado la realización de pruebas, para ello hemos ido variando dos parámetros del algoritmo:

- Numero de ciclos: indica la cantidad de ciclos de entrenamiento a los que se somete la red neuronal.
- Razón de aprendizaje: valor que representa la velocidad con lo que aprende la red neuronal, de tal manera que cuanto mayor sea más rápido entrena, aunque con una precisión menor.

Para los experimentos fijamos un valor de razón de aprendizaje sobre la que probamos diferentes cantidades de ciclos para así encontrar el óptimo. Una vez que los tenemos para ese valor cambiamos la razón de aprendizaje y repetimos el proceso.

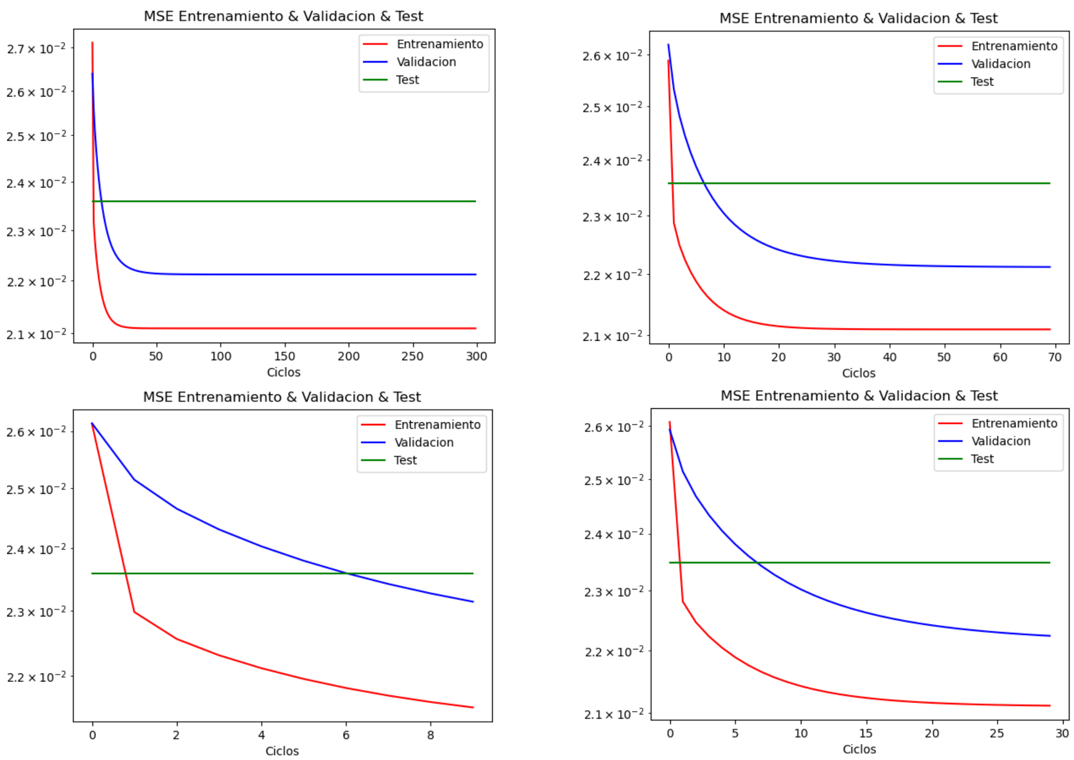
A continuación, mostramos un ejemplo del proceso descrito anteriormente en el que ejecutamos la implementación en Python de Adaline sobre tres conjuntos de datos con una razón de aprendizaje de 0,05 en primer lugar establecemos 300 ciclos y ejecutamos el programa de la siguiente manera:

- python Adaline.py *datosEntrenamiento.dat datosValidacion.dat datosTest.dat salidasDeseadas.dat* "," 300 0.05

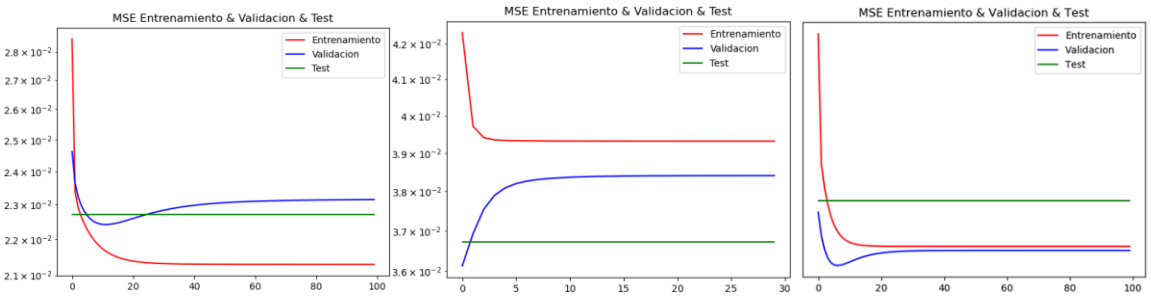
Aquí comprobamos como los errores MSE y MAE se encuentran dentro del límite tolerable. Ahora debemos buscar a partir de que ciclo el descenso del error se estanca.

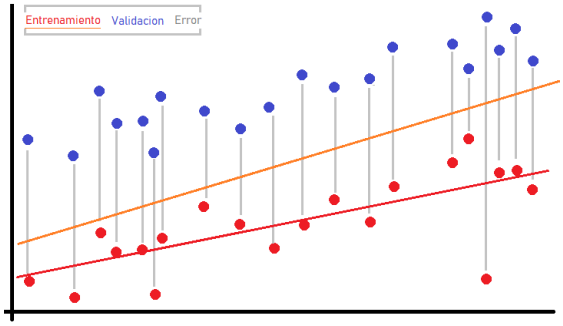
```
Atributos: 9
Numero de ciclos: 300
Razón: 0.050000
Pesos iniciales:
[-0.3259 -0.1349 -0.3745 0.1132 -0.3284 -0.0139 0.3826 -0.3303 0.3055]
-----------------------------------------------------------------------
TEST:  MSE: 0.023592/ MAE: 0.118100
Modelo final
[-0.8796 -0.8403 0.1243 -0.4537 1.3938 -3.2711 0.7011 1.1890 0.7763]
```

Repetimos el proceso usando los mismos conjuntos de datos y cambiando el número de ciclos hasta encontrar el valor optimo, que para una razón de 0.05 resulta ser 30. Estas son las gráficas que corresponden al proceso:



Antes de realizar el análisis hemos hecho numerosas pruebas con diferentes conjuntos de datos y parámetros para ver el funcionamiento de la red en diferentes situaciones y comprobar el funcionamiento correcto del programa.
Gracias a este estudio extenso del problema hemos podido detectar anomalías en el entrenamiento y validación. Nuestro preprocesador de datos aleatoriza el orden de los patrones consiguiendo en promedio una distribución homogénea de los mismos. No obstante, existe la posibilidad de que se generen conjuntos sesgados, y es en estos casos en los que observamos un comportamiento distinto en el aprendizaje. Las siguientes graficas muestran el comportamiento descrito anteriormente:

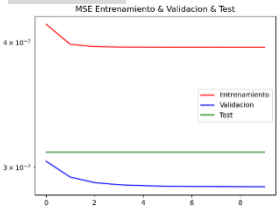


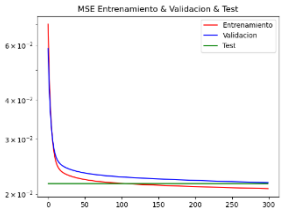


Para entender estas gráficas es útil imaginar un conjunto de datos bidimensional. En este ejemplo los datos de entrenamiento son sesgados y por tanto a pesar de que el error de entrenamiento sea reducido, la validación no lo es. Por ello ocurre que los valores del error en validación oscilan hasta que el modelo va corrigiendo la diferencia.

ADALINE							
Razón	**Ciclos**	**Entrenamiento**		**Validación**		**Test**	
		MSE	**MAE**	**MSE**	**MAE**	**MSE**	**MAE**
0.5	100	0.04107	0.150757	0.02578	0.10980	0.02714	0.11383
	10	0.03942	0.149126	0.02251	0.10548	0.02139	0.10549
0.1	100	0.02268	0.113628	0.02116	0.10968	0.02126	0.11027
	15	0.02268	0.111448	0.02117	0.10955	0.02125	0.11013
0.05	50	0.02141	0.02229	0.11018	0.10772	0.01977	0.10604
	100	0.02158	0.10844	0.02103	0.10795	0.02066	0.10557
0.001	10	0.02483	0.11844	0.02367	0.11587	0.02491	0.11834

	100	0.02180	0.10929	0.02383	0.12873	0.02207	0.10893
	250	0.02131	0.10690	0.02061	0.10776	0.02082	0.10653
	300	0.02177	0.10868	0.02059	0.10620	0.01957	0.10446
	600	0.02403	0.10503	0.03127	0.10690	0.07436	0.10648

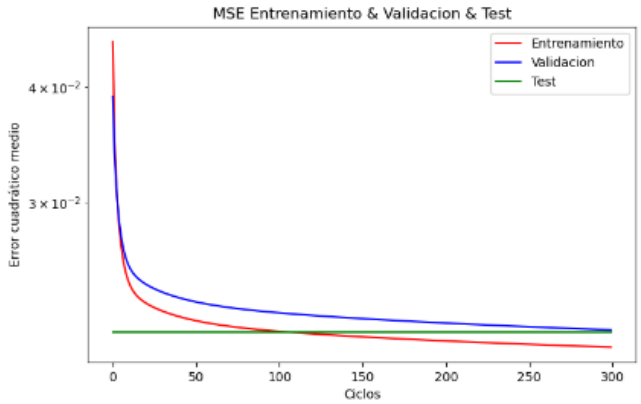


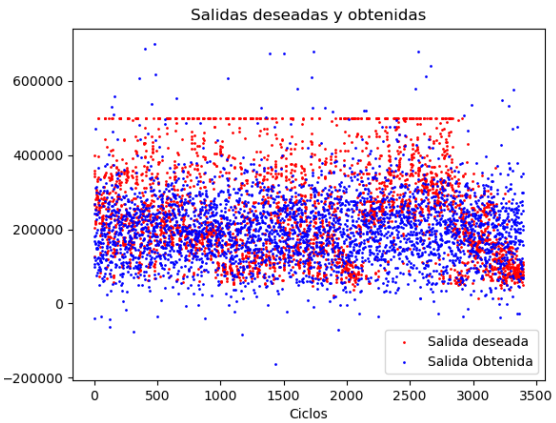


De los resultados obtenidos en estos experimentos se ha podido concluir que con razones de aprendizaje elevadas como la 0,5, la cantidad de ciclos que se necesitan para encontrar los mejores resultados del error son muy bajos en este caso solo con 10 ciclos ha sido suficiente obteniendo así un tiempo de ejecución muy eficiente.

Adicionalmente observamos que razones de aprendizaje pequeñas como es el caso de 0,001 el número de ciclos necesarios para alcanzar el valor mínimo de los errores es mucho mayor que con razones de aprendizaje altas en el caso del 0,001 han sido necesarios un total de 300 ciclos.

Por último, mencionar que los mejores resultados se han obtenido con razones de aprendizaje bajas, pero ciclos muy altos, es decir que los errores son menores cuanto más despacio aprende la red de neuronas. Sin embargo, pese a obtener resultados mucho mejores tienen el inconveniente de que tardan mucho más en llegar a entrenar el algoritmo. Por tanto, también se debe tener en cuenta si este sustancial incremento de tiempo para obtener la solución se compensa o no con la mejoría en los resultados de los errores. Por tanto, consideramos que el modelo optimo es el que se obtiene con una razón de aprendizaje de 0.001 y un total de 300 ciclos de entrenamiento. Con este modelo se obtienen los siguientes resultados:





Perceptrón multicapa

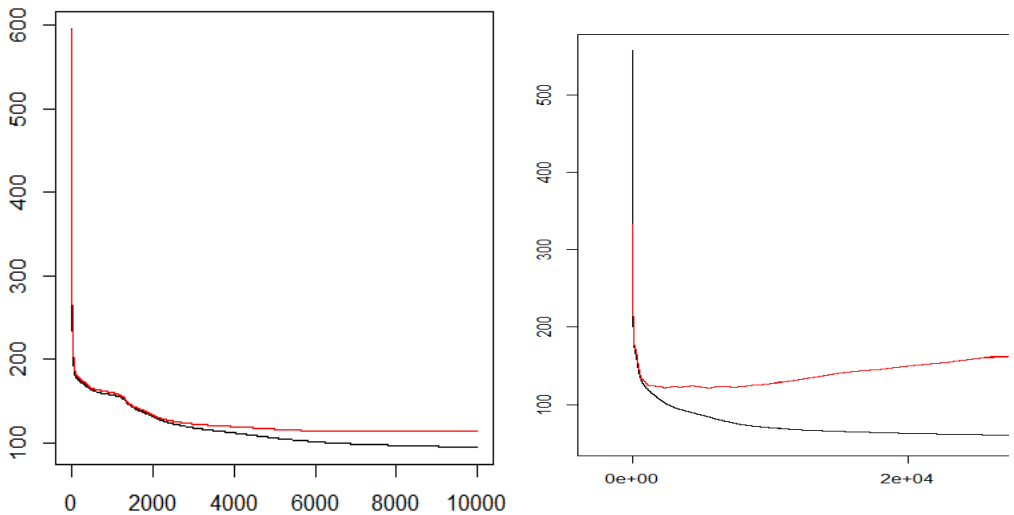
Tal y como hemos visto los modelos neuronales basados en aproximaciones lineales como Adaline tienen ciertas limitaciones. No fue hasta que se descubrió un método de propagación del error generalizado que se pudo estudiar aproximaciones no lineales. Este avance favoreció nuevas redes neuronales como el perceptrón multicapa, que implementa la Regla Delta Generalizada a través de su algoritmo de retropropagación. A continuación, estudiaremos dicha arquitectura aplicada a nuestro problema.

Para la simulación del perceptrón vamos a utilizar la librería RNSS de Rstudio. Hemos utilizado un Script que estructura la lectura de datos y configura los hiperparámetros de la arquitectura (nºcapas y neuronas por capa, ciclos, razón de aprendizaje...etc).
A grandes rasgos el script proporcionado en el lenguaje de programación R que genera el modelo del perceptrón multicapa y lo entrena consta de los siguientes pasos:

1. Lectura de los tres ficheros de datos y establecer hiperparámetros de la red
2. Iniciamos y entrenamos el modelo
3. Mostramos las gráficas de los errores y obtenemos los ciclos óptimos
4. Iniciamos y entrenamos el modelo con la cantidad de ciclos óptimos
5. Mostramos las gráficas de los errores y guardamos el modelo

Respecto al script proporcionado se ha realizado una variación para poder obtener además del MSE el MAE de tal manera que se obtengan ambos errores al igual que en el algoritmo del Adaline.

La realización de las pruebas sobre el perceptrón multicapa se va a realizar modificando los valores de la razón de aprendizaje, el numero de ciclos y la topología de la red tanto el numero de capas como de neuronas por capa. Se van a obtener el MSE y MAE para los tres conjuntos y ademas se obtendrán graficas del error por ciclos como las que se muestran a continuación:

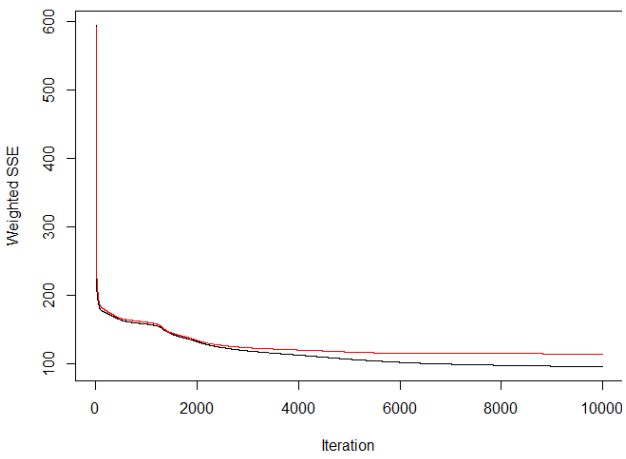


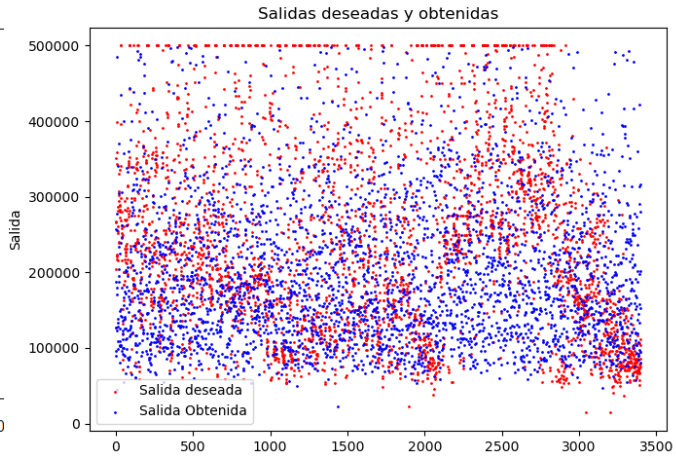
Perceptrón multicapa								
Razón	Ciclos	Neurona por capa [A,B, ...]	Entrenamiento		Validación		Test	
			MSE	MAE	MSE	MAE	MSE	MAE
0.01	3000	[20]	0.01577	0.08876	0.01627	0.08994	0.01648	0.09058
	3000	[50]	0.01616	0.09075	0.01671	0.09196	0.01689	0.09265
	2000	[10]	0.01591	0.08929	0.01632	0.09016	0.01655	0.09099
	2000	[6,6]	0.01569	0.08971	0.01602	0.09055	0.01633	0.09138
	10	[6800]	0.03283	0.14955	0.03333	0.15062	0.03274	0.14847
0.0001	10000	[8]	0.01673	0.09151	0.01744	0.09304	0.01733	0.09284
	10000	[8,4]	0.01669	0.09121	0.01744	0.09301	0.01731	0.09277
	5000	[8,4]	0.01704	0.09220	0.01783	0.09396	0.01761	0.09355
0.1	1000	[16,16]	0.01507	0.08805	0.01547	0.08918	0.01575	0.08999
	10000	[32,32]	0.00927	0.06624	0.01118	0.07245	0.01174	0.07225
	10	[1]	0.01791	0.09797	0.01836	0.099091	0.01820	0.09912
0.5	1000	[16,16,16]	0.01239	0.08022	0.01276	0.08091	0.01356	0.08240
	1500	[16,8,4,2]	0.01294	0.08278	0.01335	0.08429	0.01395	0.08496
	10	[8,8]	0.02230	0.11953	0.02287	0.12084	0.02275	0.12076
0.05	5000	[6]	0.01498	0.08632	0.01531	0.08753	0.01570	0.08830
	2000	[20]	0.01516	0.08768	0.01558	0.08908	0.01586	0.08953
	4000	[3,3]	0.01556	0.08846	0.01588	0.08953	0.01614	0.09031
0.25	100	[10]	0.01687	0.09590	0.01743	0.09708	0.01762	0.09784
	2000	[10,15,20]	0.01213	0.07809	0.01267	0.07935	0.01334	0.08090
	5000	[15]	0.01269	0.07950	0.01290	0.07964	0.01335	0.08033

En primer lugar, establecer que nuestro mejor resultado (MSE = 0.01174 y MAE = 0.07225) se alcanza con r = 0.1, una topología compuesta de 2 capas ocultas con 32 neuronas cada una y 10000 ciclos de entrenamiento. Por el contrario, obtenemos el peor resultado (MSE= 0.03274 y MAE = 0.14847) con r=0.001 y 6800 neuronas en una capa oculta. El resto de resultados oscilan entre estos y si vamos comparando desde el peor al mejor error observamos que la mejoras se produce en la milésima, lo cual indica que a grandes rasgos todas las arquitecturas generan resultados bastante precisos. Ahora bien, para establecer cuál es el mejor modelo debemos poner en valor el esfuerzo computacional que genera (topología y razón de aprendizaje), así como el tiempo que tarda en aprender para el numero de ciclos establecido.

Analizando la tabla de resultados observamos una tendencia similar en cuanto a el esfuerzo recompensa que supone el entrenamiento del modelo con la particularidad de que en este caso también debemos tener en cuenta la carga que supone incrementar el número de neuronas y las capas que las componen. Para razones de aprendizaje elevadas observamos una convergencia rápida mientras que para razones menores el número de ciclos promedio tiende a ser mayor. Ahora bien, no debemos olvidar el papel que juegan las capas ocultas. En este caso no necesariamente un número elevado de capas y neuronas genera un resultado significativamente mejor. Como ejemplo tenemos los resultados con r = 0.01 y n= 2000 ciclos o r = 0.001 y n= 10000 para una o dos capas respectivamente. Para los cuatro casos se mantuvieron constantes la razón y los ciclos, variando solamente la topología. Para el primero se pasó de 10 neuronas en una única capa a 12 neuronas, repartidas en dos capas. Para el segundo se hizo lo mismo pasando de 8 a 12, repartidas en 8 y 4. Si bien es cierto que este aumento de conexiones mediante capas supuso una mejoría esta en comparación al resultado previo es casi despreciable (0.00004 y 0.00002).

Concluyendo con el análisis de topologías cabe resaltar que hemos observado que arquitecturas con dos o más capas ocultas consiguen, en general, buenos resultados y es que creemos que el principal factor a tener en cuenta a la hora de establecer que estructura tiene la red es el número de conexiones que esta va a tener. Las conexiones establecen que potencial de aprendizaje tiene la red y por tanto cuanto conocimiento puede absorber. Si este es muy elevado la red sobreaprende los patrones y no generaliza y si es muy bajo no adapta sus pesos debidamente a las entradas. Las conexiones se producen entre neuronas de capa a capa comenzando por la de entrada hasta la de salida, y es que a pesar de no ser indiferente como se ordenan en nuestros experimentos observamos cierto grado de equivalencia entre configuraciones con el mismo número de conexiones.





Comparación

Después de realizar el experimento y el análisis de los resultados para ambos modelos se puede concluir que en general los resultados obtenido para el MAE y el MSE han sido bastante mejores en el perceptrón multicapa. Obteniéndose 0,01 menos de MSE y 0,02 menos de MAE en los resultados del perceptrón.

Esto se debe a que el perceptrón tiene una mayor cantidad de neuronas y ademas cuenta con capas ocultas que facilitan el proceso de aprendizaje de la red.

Incluso cuando se ha probado una topología de red para el perceptrón en la que solo hay una capa con una única neurona, es decir una configuración muy similar a la del adaline se han obtenido mejores resultados en los experimentos del perceptrón.

Por todo esto se puede concluir que para este problema en concreto la aproximación realizada por el perceptrón multicapa es mejor a la realizada por adaline.

Conclusión

Como conclusión hay que decir que el proyecto se ha dividido en tres secciones.

La primera etapa fue la de diseño o conceptualización en la que nos fuimos enfrentado a la ideación del preprocesador de datos y adaline. Para mantener el problema lo más simple posible desarrollamos dos programas con interfaz vía terminal en python.

La siguiente fase supuso un desafío mayor ya que consistió en llevar a la realidad el diseño. Fue entonces cuando nos enfrentamos a problemas con librerías de python, errores en el tratamiento de los datos etc… Finalmente logramos construir un modelo robusto para cualquier número de patrones y atributos incluyendo además ciertas comprobaciones sobre los parámetros de ejecución (ej: 1 > razón de aprendizaje < 0 o nºCiclos > 0). No obstante, somos conscientes de que nuestra implementación no es la más eficiente posible ya que, a pesar de escoger funciones operativas eficientes, incurrimos en múltiples llamadas a funciones (forzando cambios de contexto de variables) o no aplicamos técnicas de paralelismo cuando podrían utilizarse. Estas restricciones son decisiones en el diseño que siempre podrían mejorarse para acelerar la aplicación. Además, tuvimos que familiarizarnos con RStudio y el script de ejecución para el perceptrón multicapa, aunque a decir verdad este esfuerzo fue menor en comparación al trabajo previo.

La última fase consistió en documentar todo el proceso recogiendo en la memoria todas las decisiones tomadas, lanzando experimentos y estudiando los resultados. Como reseña hay que comentar que una vez compusimos el cuerpo principal de la memoria pudimos contemplar de manera global el panorama y pudimos interpretar mejor todo el proyecto.

Consideramos por tanto que ha sido una práctica muy instructiva con una carga de trabajo bastante bien distribuida.

Referencias

Iconos diseñados por freepik.com

Portada diseñada por rawpixel.com