2 DE OCTUBRE DE 2019

PRÁCTICA 1: IMPLEMENTACIÓN DEL PERCEPTRÓN MULTICAPA

INTRODUCCIÓN A LOS MODELOS COMPUTACIONALES

ALBERTO LUQUE RIVAS
31018546X
I62LURIA@UCO.ES

GRADO EN INGENIERÍA INFORMÁTICA
COMPUTACIÓN
4º CURSO

UNIVERSIDAD DE CÓRDOBA

ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA
Universidad de Córdoba

EPSC

Contenido

1. Introducción

El trabajo que se va a realizar consiste en implementar el algoritmo de retro propagación para entrenar un perceptrón multicapa para un problema concreto. En este caso tenemos acceso a unas muestras de diferentes dataset usados, entre ellos:

- Dataset SIN
- Dataset Quake
- Dataset Parkinson
- Dataset XOR

Para ello, se desarrollará un programa capaz de realizar este entrenamiento, con distintas posibilidades en cuanto a la parametrización de este.

Este programa se utilizará para entrenar modelos que predigan, de la forma más correcta posible, la(s) variable(s) objetivo(s) del conjunto de bases de datos mencionadas anteriormente.

2. Perceptrón multicapa

La arquitectura de la red neuronal perceptrón multicapa se caracteriza porque sus neuronas están agrupadas en diferentes niveles o capas, donde cada una de ellas tienen sus propias neuronas. Este tipo de redes tiene tres niveles diferenciados, un nivel de entrada, los niveles intermedios que forman las capas ocultas y el nivel de salida.

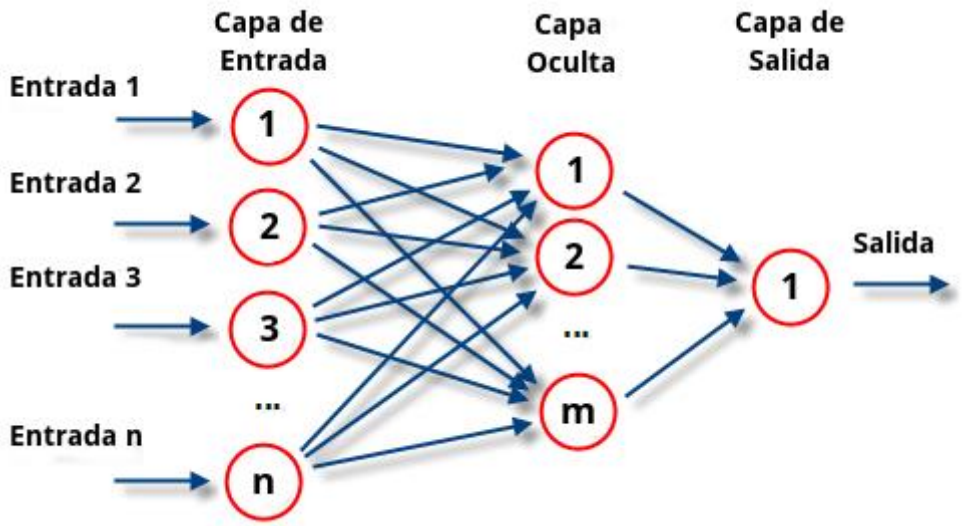

Ilustración 1 Ejemplo perceptrón multicapa con solo una capa oculta

Las neuronas de la entrada no funcionan exactamente como propias neuronas, estas únicamente se encargan de recibir una señal de activación y propagarla a las capas posteriores a través de los propios enlaces neuronales.

En esta red usada para la práctica, se utiliza un enlace completo entre neuronas y se establece un sesgo en las capas ocultas del modelo.

La función de activación usada en las neuronas ha sido la función sigmoide que se define con la siguiente notación matemática:

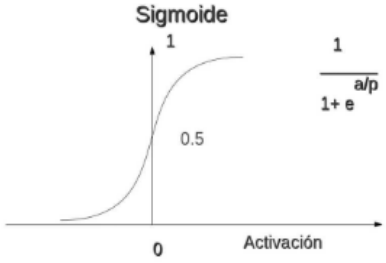

Ilustración 2 Función sigmoide de activación

Donde a/p se considera a toda la suma de enlaces entrantes a la neurona.

3. Pseudocódigo

Descripción en pseudocódigo de los pasos del algoritmo de retro propagación y de todas aquellas operaciones relevantes. El pseudocódigo deberá forzosamente reflejar la implementación y el desarrollo empleados.

Este pseudocódigo descrito en la **Ilustración 3** detalla el algoritmo on-line de la retopropagación en un perceptrón multicapa basados en bases de activación sigmoidal.

```
-----Algoritmo de retropropagacion-----
INICIO
    iniciarPesosAleatorios()
        REPETIR
            PARA cada patron con entradas x, y salidas d
                aplicarCambiosPatrones()
                alimentarEntradas()
                propagarEntradas()
                retropropagarError()
                acumularCambio()
                ajustarPesos()
            FIN PARA
        HASTA(CondicionParada)
FIN
```

Ilustración 3 Pseudocódigo algoritmo de retro propagación del error

Este pseudocódigo descrito en la **Ilustración 4** se encarga de recorrer todas las capas con vectores de peso e inicializarlas con valores aleatorios [-1,1], la primera capa no se tiene en cuenta puesto que es la de entrada y en ella no existen vectores de pesos asociados por no tener conexiones previas con ninguna neurona.

```
----- function iniciarPesosAleatorios() -----

INICIO

    PARA i = 1 hasta numeroCapas HACER
        PARA j = 0 hasta NeuronasCapa[i] HACER
            PARA k = 0 hasta neuronasCapa[i-1]+Sesgo HACER

                CAPA[i].NEURONAS[j].w[k] = aleatorio[-1,1];
                                CAPA[i].NEURONAS[j].wCopia[k] = aleatorio[-1,1];

            FIN PARA
        FIN PARA
    FIN PARA

FIN
```

Ilustración 4 Pseudocódigo iniciar pesos aleatorios

Este pseudocódigo descrito en la **Ilustración 5** es el que se encarga de alimentar las entradas de la capa cero con un vector pasado por argumentos.

```
----- function alimentarEntradas() -----

INICIO

    input[]
        PARA i = 0 hasta neuronasCapa[0] HACER
            CAPA[0].NEURONA[i].x[k] = input[i]
            CAPA[0].NEURONA[i].dX[k] input[i]
        FIN PARA

FIN
```

Ilustración 5 Pseudocódigo alimentar entradas

Este pseudocódigo descrito en la **Ilustración 6** es el encargado de calcular el valor de X de cada neurona, su funcionamiento es recorrer el vector de pesos propio de la neurona y multiplicar por el valor de la X de la neurona, y esto repetir por cada neurona de cada capo. Hay que tener en cuenta que las neuronas de entrada no tienen vector de pesos asociados y por ello se empieza en la capa oculta número uno.

```
----- function propagarEntradas() -----

INICIO
    ACUMULADOR = 0.0

    PARA i = 1 hasta numeroCapas HACER
        PARA j = 0 hasta NeuronasCapa[i] HACER
            PARA k = 1 hasta neuronasCapa[i-1]+Sesgo HACER

            ACUMULADOR = ACUMULADOR + (CAPA[i].NEURONAS[j].w[k] * (CAPA[i-1].NEURONAS[k-1].x)
                        CAPA[i].NEURONAS[j].wCopia[k] = aleatorio[-1,1];

            FIN PARA

            ACUMULADOR = ACUMULADOR + CAPA[i].NEURONAS[j].w[0]
            CAPA[i].NEURONAS[j].x = (1/(1+exponente((-1)*ACUMULADOR)))
            ACUMULADOR = 0.0

        FIN PARA
    FIN PARA

FIN
```

Ilustración 6 Pseudocódigo propagar entradas

Este pseudocódigo descrito en la **Ilustración 7** es el que se encarga de calcular la derivada de cada neurona en cada una de las capas. Recorre desde la penúltima capa hasta la primera.

```
----- function retropropagarError() -----

INICIO

ACUMULADOR

    PARA i = numeroCapas-2 hasta 0 HACER
        PARA j = 0 hasta NeuronasCapa[i] HACER
            PARA k = 0 hasta NeuronasCapa[i+1] HACER

            ACUMULADOR = ACUMULADOR + ( CAPA[i+1].NEURONA[k].w[j+1] * CAPA[i+1].NEURONA[k].dX )

            FIN PARA

            CAPA[i].NEURONA[j].dX = ACUMULADOR * CAPA[i].NEURONA[j].x * (1-CAPA[i].NEURONA[j].x)

        FIN PARA
    FIN PARA


FIN
```

Ilustración 7 Pseudocódigo retro propagar error

Este pseudocódigo descrito en la **Ilustración 8** es el que se encarga de acumular el cambio que hay que aplicar en cada una de las neuronas para el ajuste de pesos nuevos.

```
----- function acumularCambio() -----

INICIO

    PARA i = 1 hasta numeroCapas HACER
            PARA j = 0 hasta NeuronasCapa[i] HACER
                PARA k = 1 hasta NeuronasCapa[i-1]+1 HACER

                    CAPA[i].NEURONA[j].deltaW[k] +=  CAPA[i].NEURONA[j].dX * CAPA[i-1].NEURONA[k-1].x;

                FIN PARA

                CAPA[i].NEURONA[j].deltaW[0] += CAPA[i].NEURONA[j].dX;
            FIN PARA
        FIN PARA

FIN
```

Ilustración 8 Pseudocódigo acumular cambio

Este pseudocódigo descrito en la **Ilustración 9** es el que se encarga de aplicar los cambios en bases a los factores pasados por argumento (MOMENTO, ETA) y el cambio acumulado realizado en la **Ilustración 8.** Este algoritmo va recorriendo todas las capas y todas las neuronas aplicando el cambio en cada uno de sus pesos basados en el calculo del cambio a aplicar.

```
1
2
3  ----- function ajustarPesos() -----
4
5  INICIO
6
7      DELTA_W,ULTIMODELTA_W,ETA
8      PARA i = 1 hasta numeroCapas HACER
9              PARA j = 0 hasta NeuronasCapa[i] HACER
10
11                 ETA = ETA_CLASE * POTENCIA(DECREMENTO_CLAE, -(numeroCAPAS-1-i));
12
13                 PARA k = 1 hasta NeuronasCapa[i-1]+1 HACER
14
15                     DELTA_W = CAPA[i].NEURONA[j].deltaW[k];
16
17                                     ULTIMODELTA_W = CAPA[i].NEURONA[j].ultimoDeltaW[k]
18
19                                     CAPA[i].NEURONA[j].w[k] = CAPA[i].NEURONA[j].w[k] - ( ETA * DELTA_W + FACTOMOMENTO_CLASE * ETA * ULTIMODELTA_W )
20
21                                     CAPA[i].NEURONA[j].ultimoDeltaW[k] = CAPA[i].NEURONA[j].deltaW[k];
22
23                 FIN PARA
24             FIN PARA
25         FIN PARA
26
   FIN
```

Ilustración 9 Pseudocódigo ajustar pesos

4. Experimentos y análisis de resultados

En los experimentos usados hemos tenido acceso a cuatro bases de datos cada una de las con su archivo de test y de train correspondiente al dataset. Las bases de datos usadas han sido:

- **Base de datos XOR:** esta base de datos representa el problema de clasificación no lineal del XOR. Mismo fichero para train y para test.

- **Base de datos SIN:** esta base de datos está compuesta por 120 patrones de train y 41 patrones de test. Ha sido obtenida añadiendo cierto ruido aleatorio a la función seno.

- **Base de datos Parkinson:** esta base de datos está compuesta por 4406 patrones de train y 1469 patrones de test. Contiene, como entradas o variables independientes, una serie de datos clínicos de pacientes con la enfermedad de Parkinson y datos de medidas biométricas de la voz, y, como salidas o variables dependientes, el valor motor y total del UPDRS.

- **Bade de datos Quake:** esta base de datos está compuesta por 1633 patrones de train y 546 patrones de test. Se corresponde con una base de datos en la que el objetivo es averiguar la fuerza de un terremoto (medida en escala sismológica de Richter). Como variables de ´ entrada, utilizamos la profundidad focal, la latitud en la que se produce y la longitud.

Para cada unas de los datases mencionados previamente las pruebas realizadas han sido con una y dos capas ocultas, donde el numero de nodos en cada una de las capas usadas han sido dos, cuatro, ocho, dieciséis, treinta y dos, sesenta y cuatro, cien.

Como base de experimentación el valor elegido para el numero de iteraciones ha sido mil, donde más adelante se explicará que con algunas configuraciones del perceptrón este es capaz de llegar a una buena solución con muchas menos iteraciones.

Para la prueba más fructífera de cada dataset se han realizado pruebas adicionales variando los factores de decremento y de validación, los valores de decremento usados han sido uno y dos, los factores de validación han sido cero coma quince y cero coma veinte cinco.

El total de pruebas de cada dataset han sido dieciocho pruebas excepto para el dataset de XOR que han sido doce, por lo que en su totalidad se han realizado sesenta y seis pruebas.

Para cada una de las pruebas de cada uno de los dataset se han recogido los valores de:

MEDIA TRAIN
DESVIACION TRAIN
MEDIA TEST
DESVIACION TEST
ERROR TEST FINAL

Donde el ERROR TEST FINAL simboliza el error final de la prueba realizada y no del conjunto de test. Este error final junto a los otros parámetros han sido observados para decidir que personalización del perceptrón multicapa ha sido la mas eficiente a la hora de entrenar y generalizar correctamente, la configuración elegida ha sido sometida a variantes del factor de validación y del factor del decremento para ver si mejora respecto a las pruebas normales sin estos parámetros.

Se han analizado graficas de la mejorar configuración por cada de cada dataset y también de esa mejor configuración con los mejores parámetros de validación y decremento.

4.1 Análisis dataset XOR

Los resultados obtenidos son los mostrados en la **Ilustración 10**, esta prueba la única que no se ha usado conjunto de validación y también en la cual se ha usado el mismo fichero de entrada para test y para train.

XOR

	NODOS EN CAPA OCULTA	2	4	8	32	64	100
Una capa oculta	MEDIA TRAIN	0,0759759	0,000657814	0,000456481	0,000494539	0,00222111	0,100185
	DESVIACION TRAIN	0,111313	0,000403475	0,000104936	0,000655406	0,00128773	0,136785
	MEDIA TEST	0,0759759	0,000657814	0,000456481	0,000494539	0,00222111	0,100185
	DESVIACION TEST	0,111313	0,000403475	0,000104936	0,000655406	0,00128773	0,136785
	ERROR TEST FINAL	0,0759759	0,000657814	0,000456481	0,000494539	0,00222111	0,100185
				BEST			
	NODOS EN CADA CAPA OCULTA	2	4	8	32	64	100
	MEDIA TRAIN	0,10908	0,0254003	0,000225694	0,000907581	0,00202127	0,10009
	DESVIACION TRAIN	0,0632923	0,0561263	0,000145121	0,00130988	0,00436372	0,223032
Dos capas oculta	MEDIA TEST	0,10908	0,0254003	0,000225694	0,000907581	0,00202127	0,10009
	DESVIACION TEST	0,0632923	0,0561263	0,000145121	0,00130988	0,00436372	0,223032
	ERROR TEST FINAL	0,10908	0,0254003	0,000225694	0,000907581	0,00202127	0,10009
				BEST			
Eleccion Mejor Configuracion		Una Capa	Una Capa	Dos Capas	Una Capa	Dos Capas	Dos Capas

Mejor Arquitectura 8 nodos y dos capas ocultas

Columna1	Columna2	Columna3
DECREMENTO	1	2
VALIDACION	0	0
MEDIA TRAIN	0,000225694	0,00095889
DESVIACION TRAIN	0,000145121	0,00127658
MEDIA TEST	0,000225694	0,00095889
DESVIACION TEST	0,000145121	0,00127658
ERROR TEST FINAL	0,000225694	0,00095889
	BEST	

Ilustración 10 Datos del dataset XOR

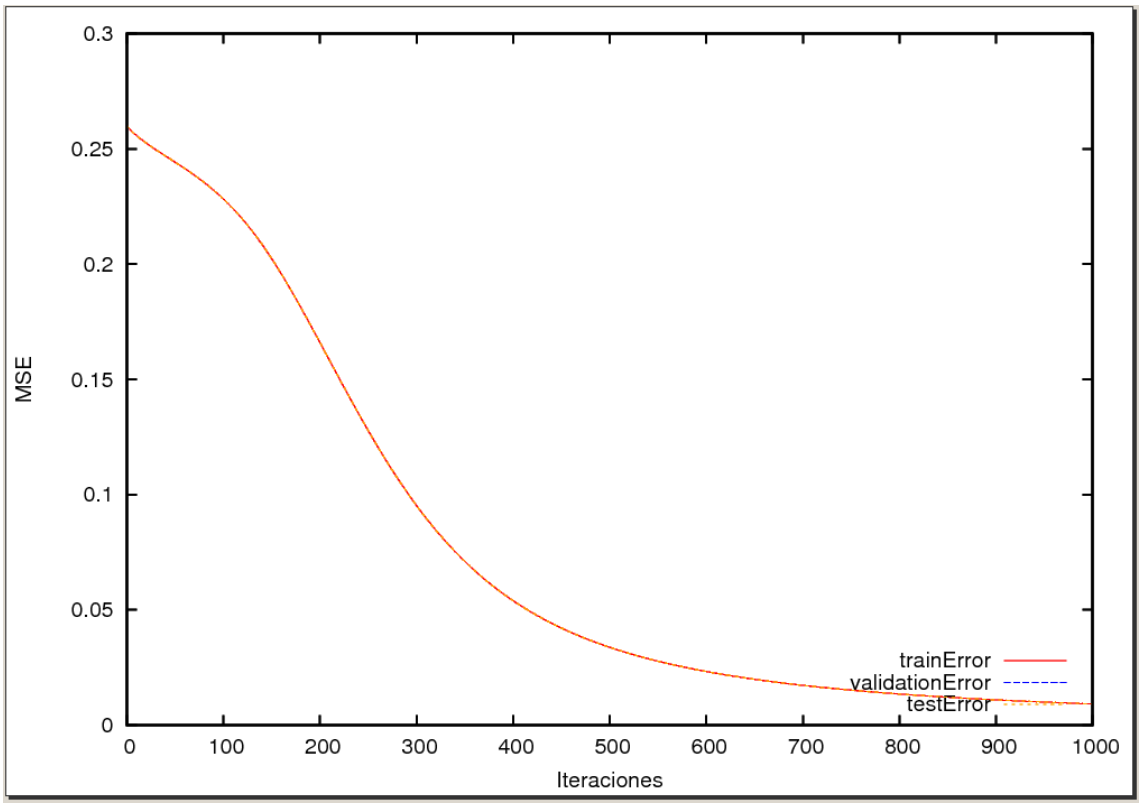

Ilustración 11 Dataset XOR con 1 capa oculta de 8 nodos en 1000 iteraciones.

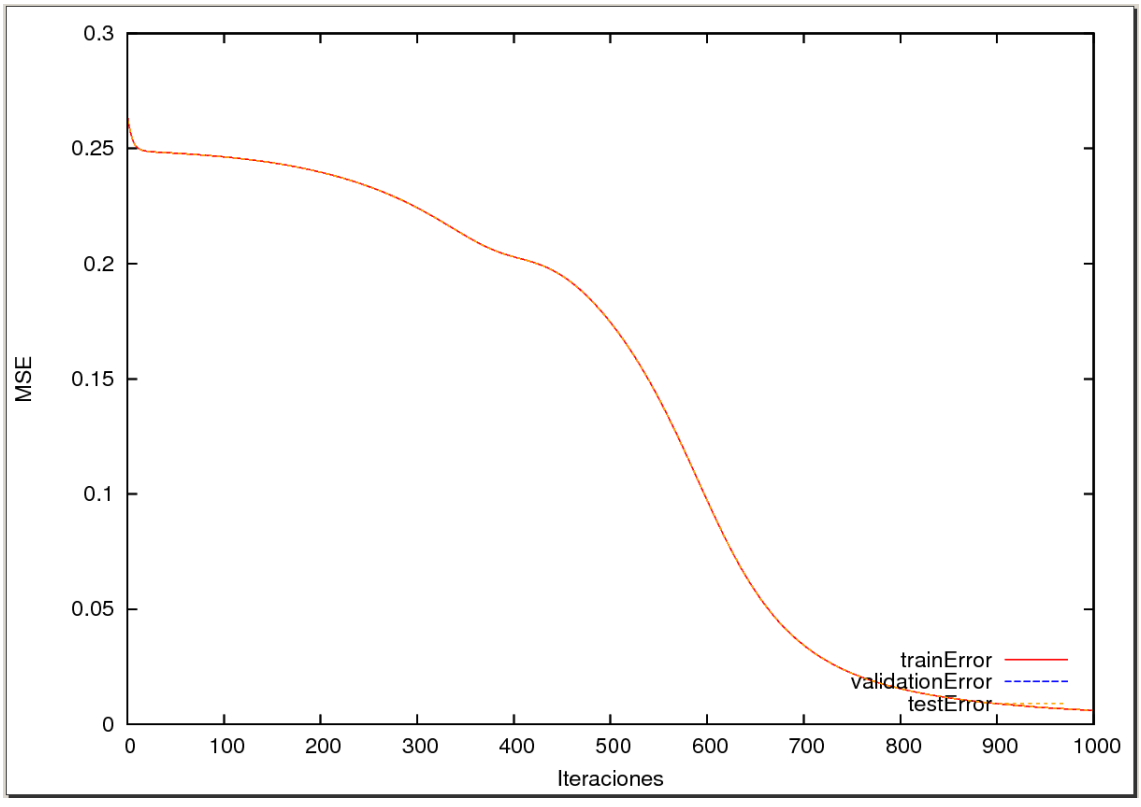

Ilustración 12 Dataset XOR con 2 capas oculta de 8 nodos en 1000 iteraciones.

Como se observa en la **Ilustración 11** el error empieza a decrecer muy rápidamente a partir de las aproximadamente 150 iteraciones mientras que en la **Ilustración 12** este efecto se observa más tardío alrededor de las 550 iteraciones, por lo que se demuestra una mejor configuración en este problema con una sola capa y ocho neuronas.

4.2 Análisis dataset PARKINSON

Los resultados obtenidos son los mostrados en la **Ilustración 13**, en esta prueba si se han usado ficheros independientes del dataset para test y para train, también hemos usado los parámetros de decremento y validación para el estudio.

PARKINSON

	NODOS EN CAPA OCULTA	2	4	8	32	64	100
Una capa oculta	MEDIA TRAIN	0,0393115	0,0325492	0,0308147	0,0276876	0,0279675	0,0315576
	DESVIACION TRAIN	0,00283775	0,00166972	0,00145575	0,00121222	0,00196581	0,0033685
	MEDIA TEST	0,0400353	0,0319507	0,0306483	0,0283678	0,0282454	0,0314655
	DESVIACION TEST	0,00203385	0,00180165	0,00230891	0,000779106	0,00169639	0,00373695
	MEDIA ERROR FINAL	0,0396734	0,03224995	0,0307315	0,0280277	0,02810645	0,03151155
					BEST		
	NODOS EN CADA CAPA OCULTA	2	4	8	32	64	100
Dos capas oculta	MEDIA TRAIN	0,0423804	0,0420782	0,0296265	0,0299916	0,0340655	0,0333846
	DESVIACION TRAIN	0,0072624	0,0123791	0,00281244	0,0035791	0,00309537	0,0035137
	MEDIA TEST	0,0427163	0,0412104	0,0290766	0,0303737	0,0337089	0,0332808
	DESVIACION TEST	0,00625219	0,0124519	0,00289033	0,00394686	0,00339587	0,00478824
	MEDIA ERROR FINAL	0,04254835	0,0416443	0,02935155	0,03018265	0,0338872	0,0333327
				BEST			
Eleccion Mejor Configuracion		Una Capa	Una Capa	Dos Capas	Una Capa	Una Capa	Una Capa

Mejor Arquitectura 32 nodos y 1 capas ocultas	Columna1	Columna2	Columna3	Columna4	Columna5	Columna6	Columna7
	DECREMENTO	1	2	1	2	1	2
	VALIDACION	0	0	0,15	0,15	0,25	0,25
	MEDIA TRAIN	0,0276876	0,0205064	0,0192067	0,0228308	0,0192002	0,0228012
	DESVIACION TRAIN	0,00121222	0,007034	0,00756376	0,00366642	0,00935658	0,00486227
	MEDIA TEST	0,0283678	0,0212115	0,0192414	0,0213077	0,0207065	0,0235099
	DESVIACION TEST	0,000779106	0,00591349	0,00708014	0,0060788	0,00940481	0,00710633
	MEDIA ERROR FINAL	0,0280277	0,02085895	0,01922405	0,02206925	0,01995335	0,02315555
	Corta a a pocas Iteraciones	No	No	Si	Si	Si	Si
				BEST			

Ilustración 13 Datos del dataset PARKINSON

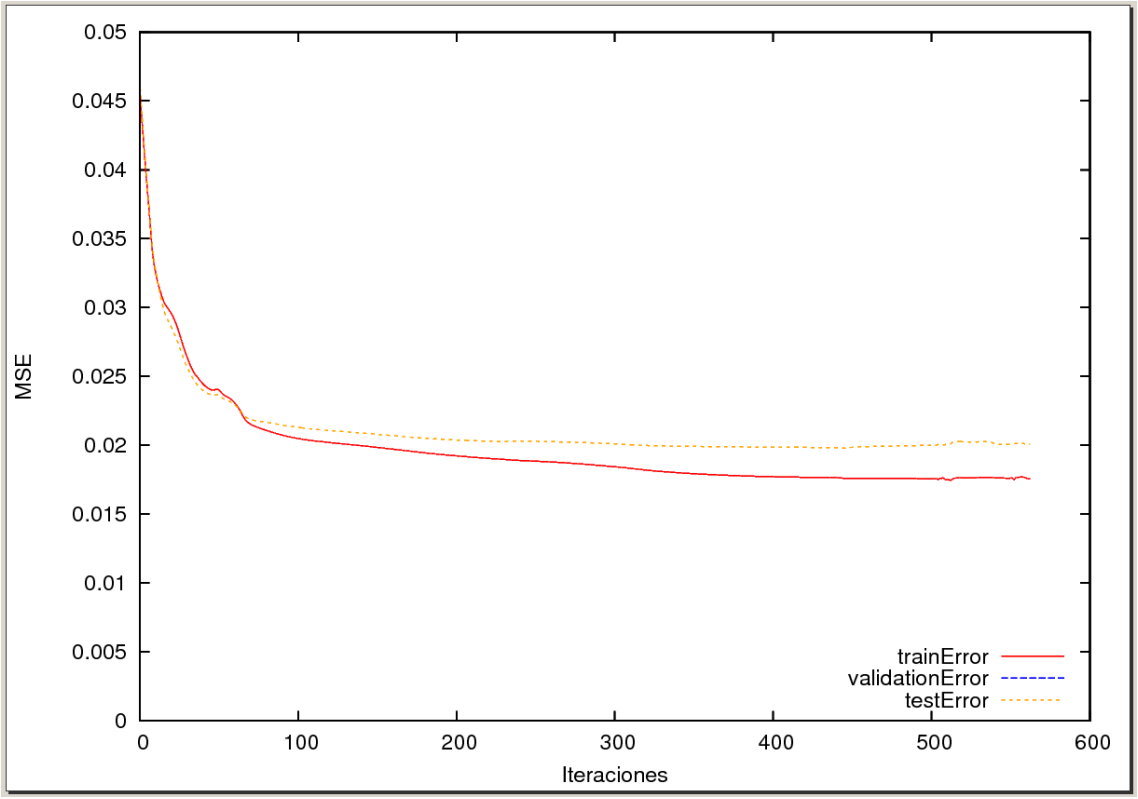

Ilustración 14 Dataset PARKINSON con 1 capa oculta de 32 nodos en 1000 iteraciones.

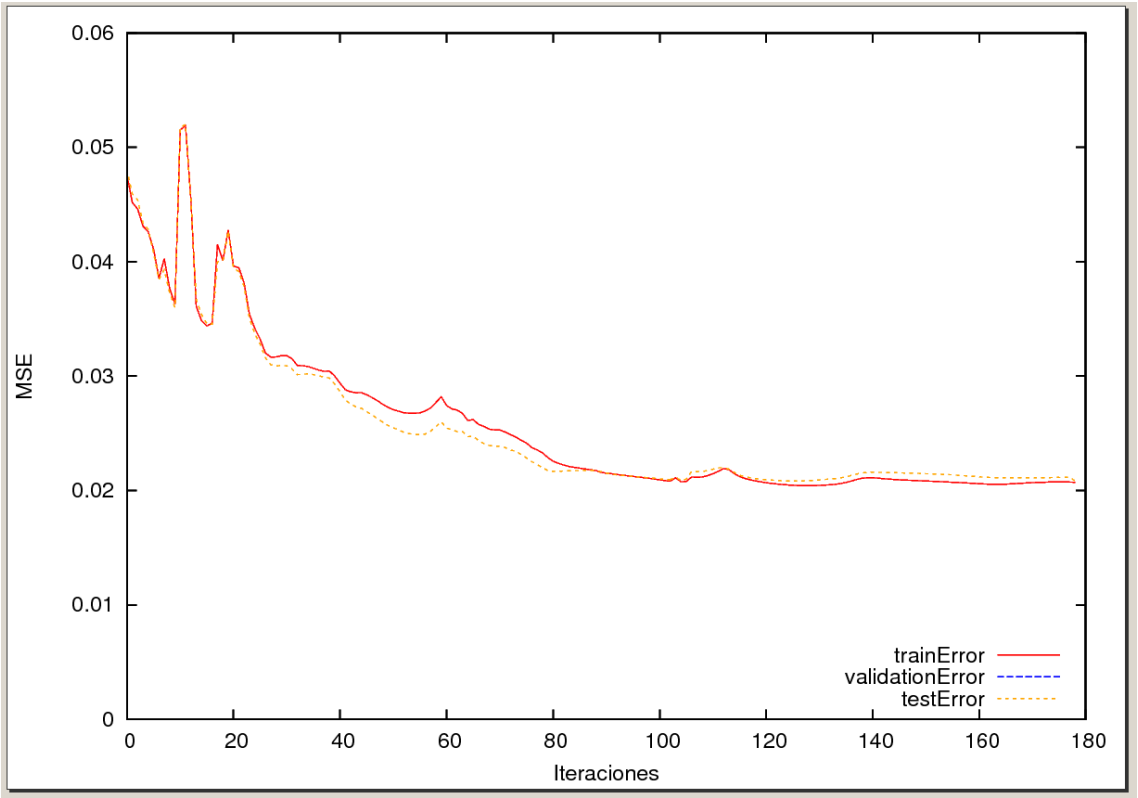

Ilustración 15 Dataset PARKINSON con 2 capas ocultas de 8 nodos en 1000 iteraciones.

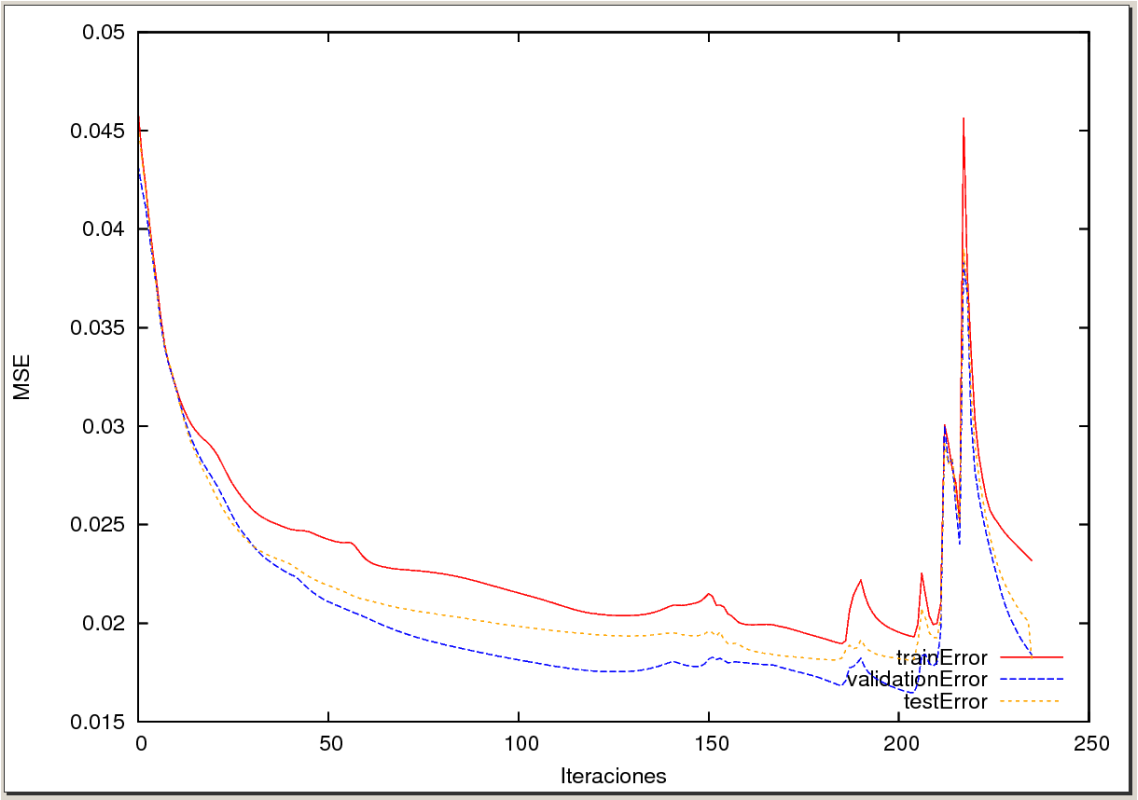

Ilustración 16 Dataset PARKINSON con 1 capa oculta de 32 nodos con 0.15 de validación en 1000 iteraciones

El resultado viendo las **Ilustraciones 15 y 14** son parejos en cuanto a convergencia, pero no en iteraciones en la **Ilustración 14** se observa que alrededor de 70 iteraciones ya encontró un punto de convergencia al igual que en la **Ilustración 15**, excepto que en la Ilustración 15 el algoritmo ha sido capaz de detenerse antes de seguir iterando puesto que las futuras iteraciones no estaban dando mejores resultados y a consecuencia de ello el coste computacional es mas bajo.

En la **Ilustración 16** en la cual se ha usado como valor de validación 0.15 se observa que en torno a las 100 iteraciones se encuentra el mejor valor del algoritmo, sobre las 200 iteraciones el valor de validación y su no mejora hace que el algoritmo no supere las 240 iteraciones. Los parámetros usados en la **Ilustración 16** han mejorado el error final del conjunto como se observa en la **Ilustración 13**, no es una mejora muy apreciable y habría que ver si es conveniente en otros problemas elegir una u otra configuración pues puede que calcular iteraciones de algún otro problema de entrada sea muy costoso en unidades de tiempo y no sea rentable esas iteraciones de mas para una mejora inapreciable.

4.3 Análisis dataset QUAKE

Los resultados obtenidos son los mostrados en la **Ilustración 17**, en esta prueba si se han usado ficheros independientes del dataset para test y para train, también hemos usado los parámetros de decremento y validación para el estudio.

QUAKE

Una capa oculta	NODOS EN CAPA OCULTA	2	4	8	32	64	100
	MEDIA TRAIN	0,0299558	0,0298907	0,0299147	0,0294302	0,0295436	0,0294689
	DESVIACION TRAIN	0,000216136	0,000198524	0,000217434	0,000135856	0,000425707	0,000140382
	MEDIA TEST	0,027185	0,0271783	0,027222	0,0270387	0,0271939	0,0271657
	DESVIACION TEST	0,000106355	0,000191848	0,000174278	0,000467	0,000247674	0,000256305
	MEDIA ERROR FINAL	0,0285704	0,0285345	0,02856835	0,02823445	0,02836875	0,0283173

Dos capas oculta	NODOS EN CADA CAPA OCULTA	2	4	8	32	64	100
	MEDIA TRAIN	0,0300731	0,0299902	0,0296344	0,029436	0,0292259	0,0294786
	DESVIACION TRAIN	0,0300731	0,000257476	0,000194453	0,000231798	0,000234353	0,000261432
	MEDIA TEST	0,0272749	0,0272494	0,0268811	0,0272405	0,027093	0,0270813
	DESVIACION TEST	0,000318629	0,000323371	0,0000732	0,000375453	0,00024037	0,000304669
	MEDIA ERROR FINAL	0,028674	0,0286198	0,02825775	0,02833825	0,02815945	0,02827995

Eleccion Mejor Configuracion						
	Una Capa	Una Capa	Dos Capas	Una Capa	Dos Capas	Dos Capas

Mejor Arquitectura 64 nodos y dos capas ocultas

Columna1	Columna2	Columna3	Columna4	Columna5	Columna6	Columna7
DECREMENTO	1	2	1	2	1	2
VALIDACION	0	0	0,15	0,15	0,25	0,25
MEDIA TRAIN	0,0292259	0,0294055	0,0301202	0,0300088	0,0301225	0,0300461
DESVIACION TRAIN	0,000234353	0,000139902	5,93E-05	0,000121772	7,11E+00	9,08E-05
MEDIA TEST	0,027093	0,0271459	0,0272472	0,0271357	0,0272607	0,027197
DESVIACION TEST	0,00024037	0,000176809	0,000105882	0,000195162	0,000117004	0,000122529
MEDIA ERROR FINAL	0,02815945	0,0282757	0,0286837	0,02857225	0,0286916	0,02862155
Corta a a pocas Iteraciones	No	No	Si	Si	Si	Si
	BEST					

Ilustración 17 Datos del dataset QUAKE

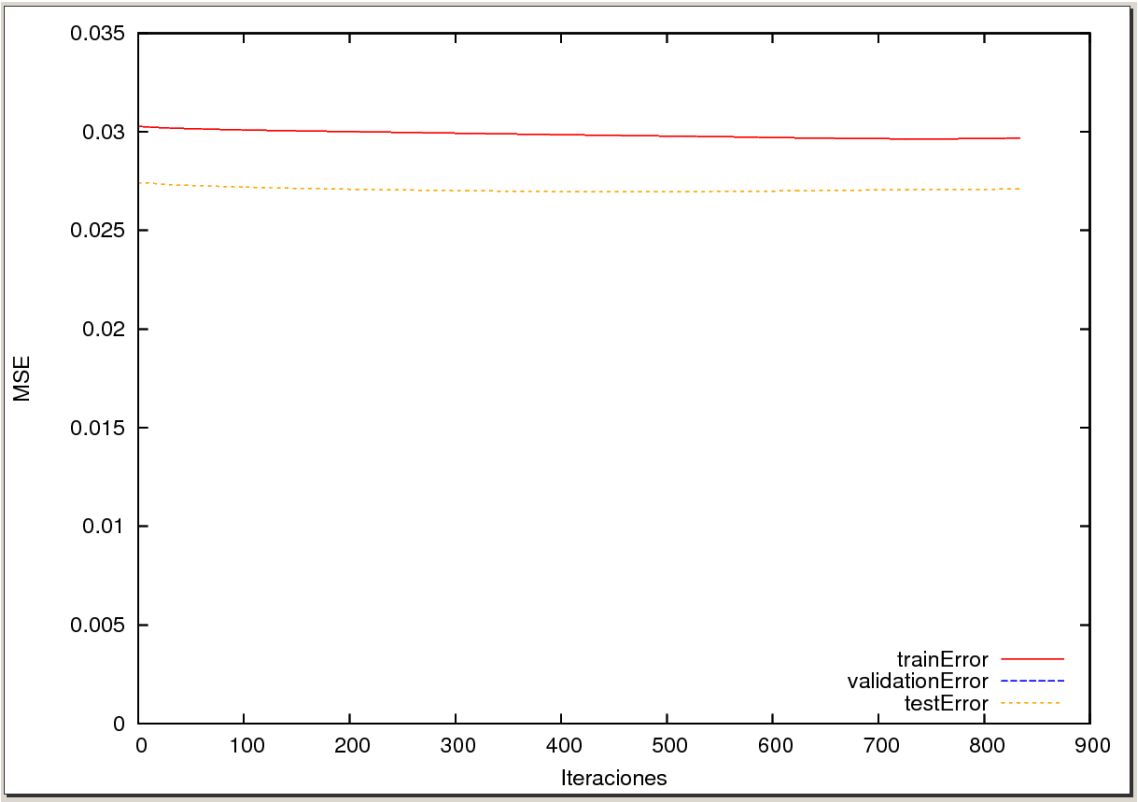

Ilustración 18 Dataset QUAKE con 1 capa oculta de 32 nodos en 1000 iteraciones.

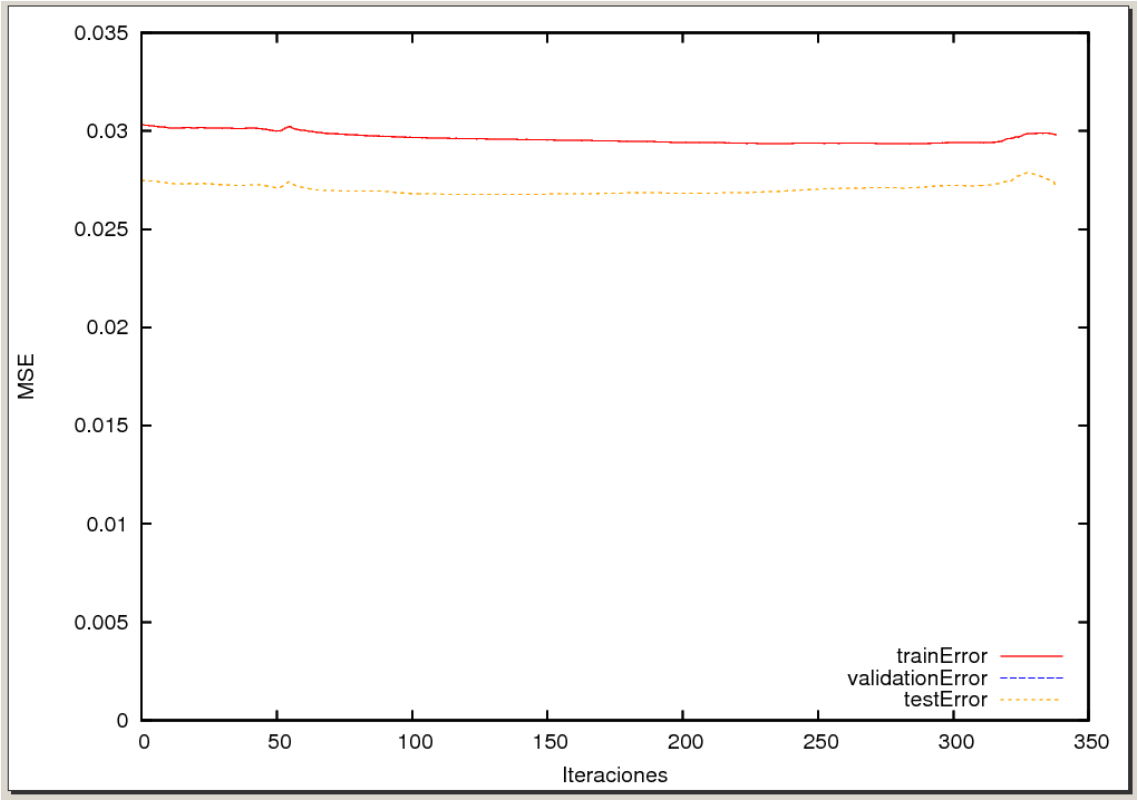

Ilustración 19 Dataset QUAKE con 2 capas ocultas de 64 nodos en 1000 iteraciones.

Se observa las **Ilustraciones 18 y 19** que la convergencia es inmediata pero el error mínimo que acepta el algoritmo para parar no es conseguido hasta las 900 iteraciones en el caso de la **Ilustración 18** y las 350 iteraciones en el caso de la **Ilustración 19**. Hay que valorar que configuración es mejor usar porque, aunque en la **Ilustración 19** el algoritmo para en 250 iteraciones se han usado 2 capas ocultas de 64 nodos lo cual el coste en tiempo y recursos se ha disparado comparado con la ejecución de la Ilustración 18.

En este dataset no se han conseguido mejorar los resultados finales mediante los valores de decremento ni de validación, aunque con algunas de las configuraciones si ha sido posible bajar hasta las doscientas iteraciones en el caso de 2 capas ocultas y 64 nodos.

En ninguna configuración de los valores de decremento y validación se han bajado de las 350 iteraciones.

Entre las **Ilustraciones 18 y 19** no es una mejora muy apreciable y habría que ver si es conveniente en otros problemas elegir una u otra configuración pues puede que calcular iteraciones de algún otro problema de entrada sea muy costoso en unidades de tiempo y no sea rentable esas iteraciones de más para una mejora inapreciable.

4.4 Análisis dataset SIN

os resultados obtenidos son los mostrados en la **Ilustración 20**, en esta prueba si se han usado ficheros independientes del dataset para test y para train, también hemos usado los parámetros de decremento y validación para el estudio.

SIN

NODOS EN CAPA OCULTA	2	4	8	32	64	100
MEDIA TRAIN	0,0296689	0,0286838	0,0289589	0,026555	0,0253582	0,0276844
DESVIACION TRAIN	0,0000235	0,00218898	0,00150444	0,00689316	0,00234843	0,00175766
MEDIA TEST	0,0365665	0,036462	0,0366519	0,0334969	0,0338754	0,0360419
DESVIACION TEST	0,000188859	0,000267259	0,000437483	0,00750741	0,00339288	0,000904961
MEDIA ERROR FINAL	0,0331177	0,0325729	0,0328054	0,03002595	0,0296168	0,03186315
					BEST	

Una capa oculta

NODOS EN CADA CAPA OCULTA	2	4	8	32	64	100
MEDIA TRAIN	0,0286657	0,0294179	0,0268424	0,0154342	0,0227473	0,0212404
DESVIACION TRAIN	0,00226392	0,000534104	0,00402598	0,00298559	0,00688056	0,00906179
MEDIA TEST	0,0361419	0,0367761	0,0343361	0,0220148	0,00688056	0,0294236
DESVIACION TEST	0,000571112	0,000651749	0,00321609	0,00395507	0,00625529	0,00903105
MEDIA ERROR FINAL	0,0324038	0,033097	0,03058925	0,0187245	0,01481393	0,025332
					BEST	

Dos capas oculta

Eleccion Mejor Configuracion	Dos Capas	Una Capa	Dos Capas	Dos Capas	Dos Capas	Dos Capas

Mejor Arquitectura 64 nodos y dos capas ocultas

Columna1	Columna2	Columna3	Columna4	Columna5	Columna6	Columna7
DECREMENTO	1	2	1	2	1	2
VALIDACION	0	0	0,15	0,15	0,25	0,25
MEDIA TRAIN	0,0227473	0,0276747	0,0296349	0,0294568	0,0295992	0,0292895
DESVIACION TRAIN	0,00688056	0,00275338	0,000106876	0,000513246	0,000180736	0,00086103
MEDIA TEST	0,00688056	0,0353066	0,0362461	0,0364677	0,0365001	0,0363541
DESVIACION TEST	0,00625529	0,00146765	0,000325705	0,000878271	0,000779899	0,0005062
MEDIA ERROR FINAL	0,01481393	0,0629813	0,065881	0,0659245	0,0660993	0,0656436
Corta a a pocas Iteraciones	No	No	Si	Si	Si	Si
	BEST					

Ilustración 20 Datos del dataset SIN

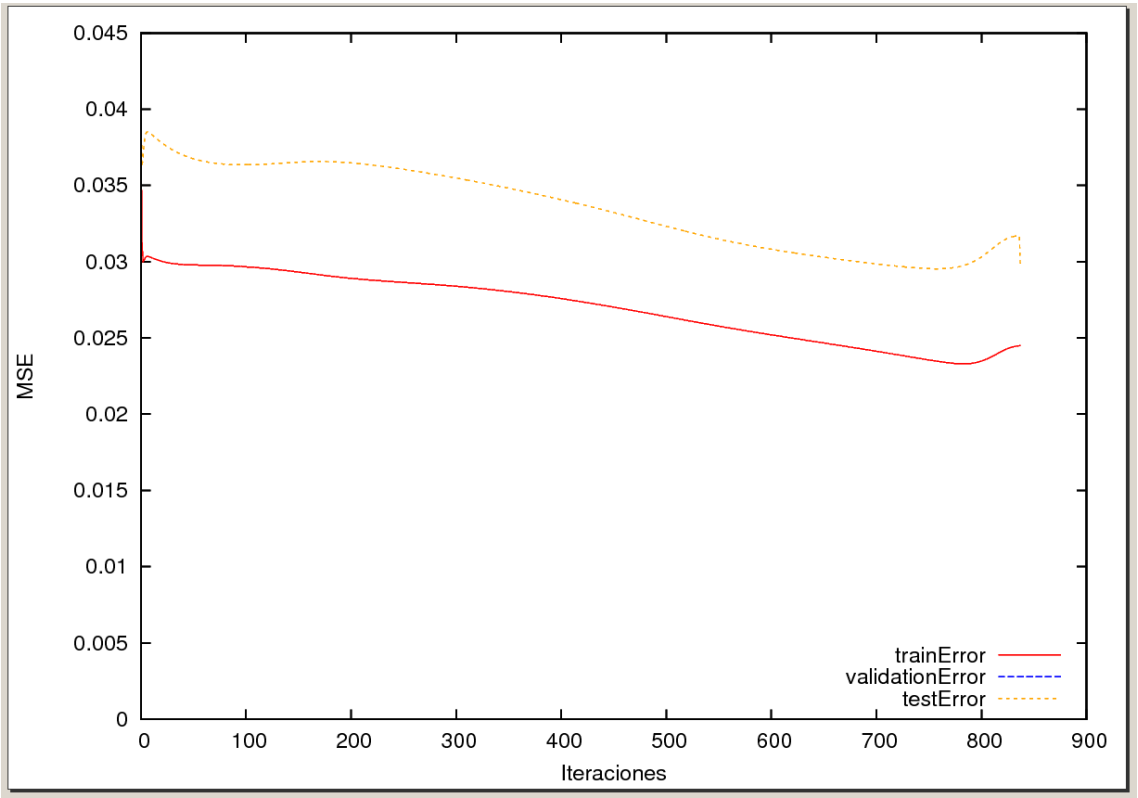

Ilustración 21 Dataset SIN con 1 capa oculta de 64 nodos en 1000 iteraciones.

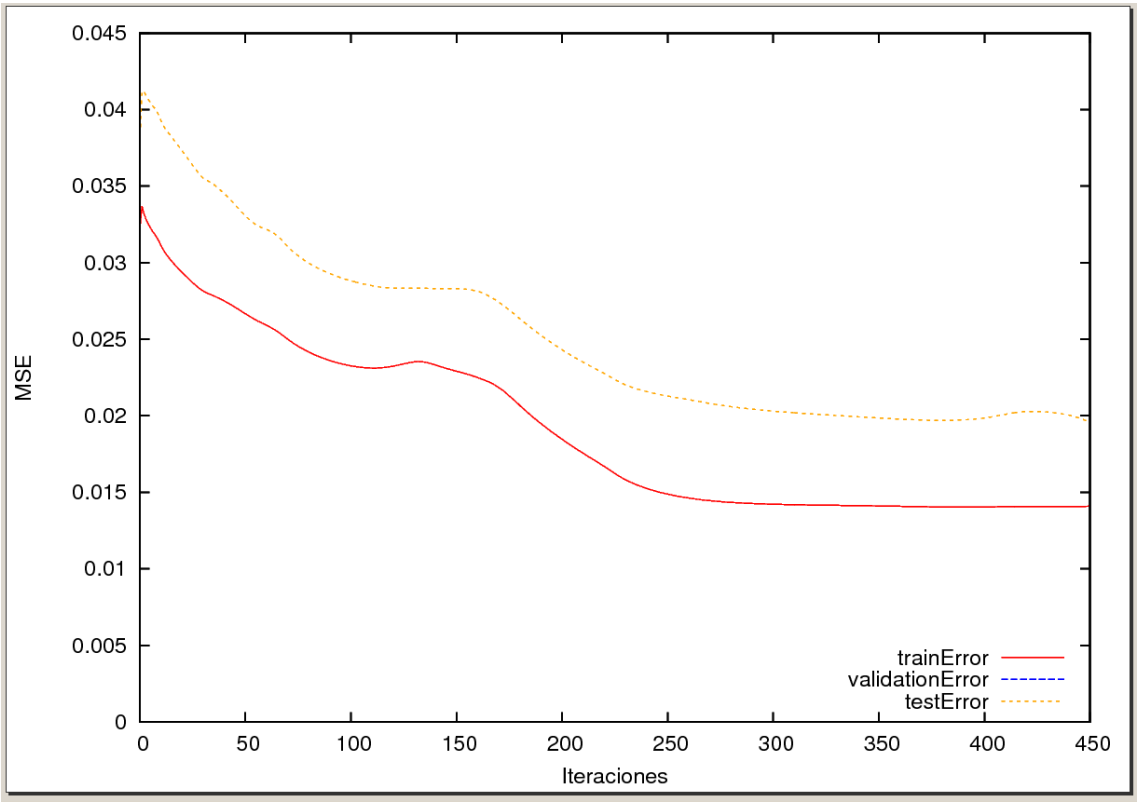

Ilustración 22 Dataset SIN con 2 capas ocultas de 64 nodos en 1000 iteraciones.

Se observa en la **Ilustración 21** que durante todo el segmento de iteraciones los valores de errores van disminuyendo muy lentamente donde se ve que en unos 780 encuentra el mínimo y poco después el algoritmo corta la ejecución por no permitir peores resultados. Sin embargo, en la **Ilustración 22** la convergencia se encuentra prematuramente a unas 300 iteraciones, los resultados obtenidos en este algoritmo con la ejecución de la **Ilustración 22** son algo mejores que los de la **Ilustración 21.**

En este dataset no se han conseguido mejorar los resultados finales mediante los valores de decremento ni de validación, aunque con algunas de las configuraciones si ha sido posible bajar hasta las 270 iteraciones en el caso de 2 capas ocultas y 32 nodos.

Entre las **Ilustraciones 21 y 22** no es una mejora muy apreciable y habría que ver si es conveniente en otros problemas elegir una u otra configuración pues puede que calcular iteraciones de algún otro problema de entrada sea muy costoso en unidades de tiempo y no sea rentable esas iteraciones de más para una mejora inapreciable.

5. Análisis del modelo de red neuronal obtenido para el problema del XOR

Este problema se puede representar con una red neuronal perceptrón multicapa como el de la **Ilustración 23**, donde se han usado dos entradas una capa oculta con 3 neuronas y una capa de salida.

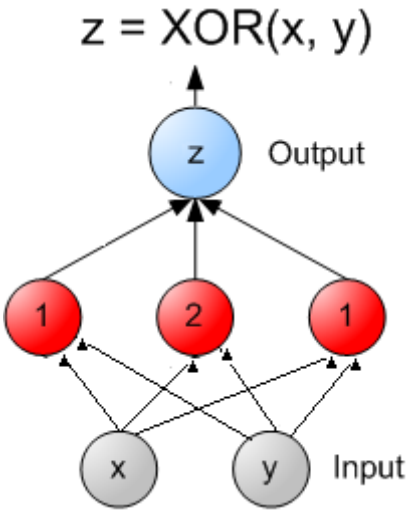

Ilustración 23 Perceptrón Multicapa para XOR

Valor de los pesos:

```
PESOS DE LA RED
===============
------Capa(1)------
Neurona 1: [0.722832] [-0.55786]
Neurona 2: [-3.27833] [-3.03185]
Neurona 3: [-3.42398] [3.45565]
------Capa(2)------
Neurona 1: [2.40079] [-0.258265] [-4.59155]
```

Ilustración 24 Valores de los pesos de la red neuronal XOR de la Ilustración 23

```
ENTRADAS DE LA RED
==================
1--> 1 -1 1
2-->-1 -1 0
3-->-1 1 1
4--> 1 1 0
```

Ilustración 25 Valores de las entradas de la red neuronal XOR de la Ilustración 23

```
Salida Esperada Vs Salida Obtenida (test)
=========================================
1 -- 0.880756
0 -- 0.111779
1 -- 0.869005
0 -- 0.117383
```

Ilustración 26 Valores de las salidas esperadas y obtenidas de la red neuronal XOR de la Ilustración 23

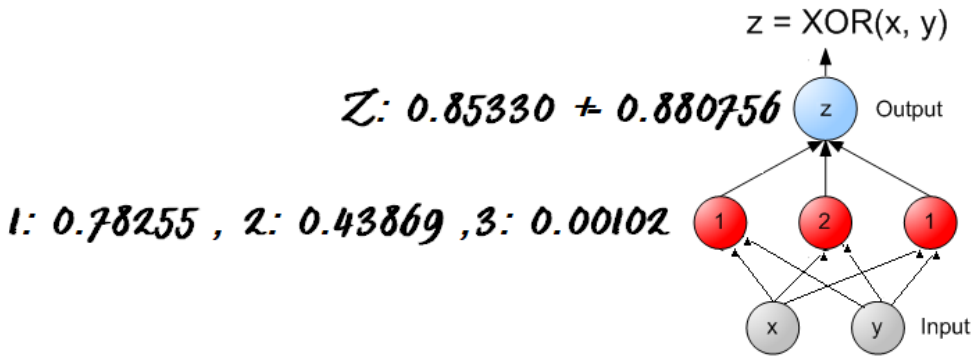

Ilustración 27 Valores calculados de la red neuronal XOR de la Ilustración 23 para el patrón de entrada XOR(x, y) = XOR (1-1) =0.85330

Los valores de la **Ilustración 27** a la hora de calcular la propagación de las entradas se han tenido en cuanta 4 decimales por eso es la variación de **0.85330** a **0.880756.**

6. Conclusiones

El Perceptrón multicapa es una red neuronal artificial que tiene conexiones con las siguientes capas de la red, donde este tipo de redes realiza aproximaciones que son combinaciones lineales de múltiples funciones no lineales.

Estas redes se pueden implementar en el día a día en ámbitos como análisis de imágenes, reconocimiento de la voz, diagnósticos médicos, regresión.

Aunque el Perceptrón multicapa es muy polivalente y una de las mas usadas en la actualidad no es necesariamente la mas potente ni la mejor ya que tiene algunas limitaciones como suele ser el proceso de aprendizaje para ciertos problemas complejos.

En la practica hemos visto que el ajuste a los problemas dado ha sido mejor de lo esperado y que incluso en algunos de usar una red simple hemos usado algún parámetro que ha logrado hacer converger más rápido el algoritmo o mejorar la puntuación final de la función objetivo, que es la del mejor ajuste para todos los patrones de entrada de cada uno de los dataset usados.

Es realmente relevante saber realizar estudios de diferentes valores para las capas, nodos y parámetros para los diferentes problemas ya que esto es determinante para el resultado final del problema y sobre todo para el coste computacional que es necesario para ciertos parámetros.

7. Índice de ilustraciones