PRÁCTICA 1: IMPLEMENTACIÓN DEL PERCEPTRÓN MULTICAPA

Introducción a los modelos computacionales. Asignatura de 4ºCurso

Carlos de la Barrera Pérez. DNI:31010594Q. Email UCO: i12bapec@uco.es

Escuela Politécnica Superior

Índice

ÍNDICE DE GRAFICAS

ÍNDICE DE TALBAS

INDICE DE FIGURAS

1. Descripción de los modelos de red utilizados.

Se han utilizado redes neuronales de tipo perceptrón multicapa. Estas redes tienen la capacidad de resolver problemas que no son linealmente separables. Además, las funciones de cada neurona han de ser derivables, siendo en este caso de tipo sigmoide (Figura 2). Lo que permite la aplicación del algoritmo de retropropagación del error que se ha implementado en esta práctica.

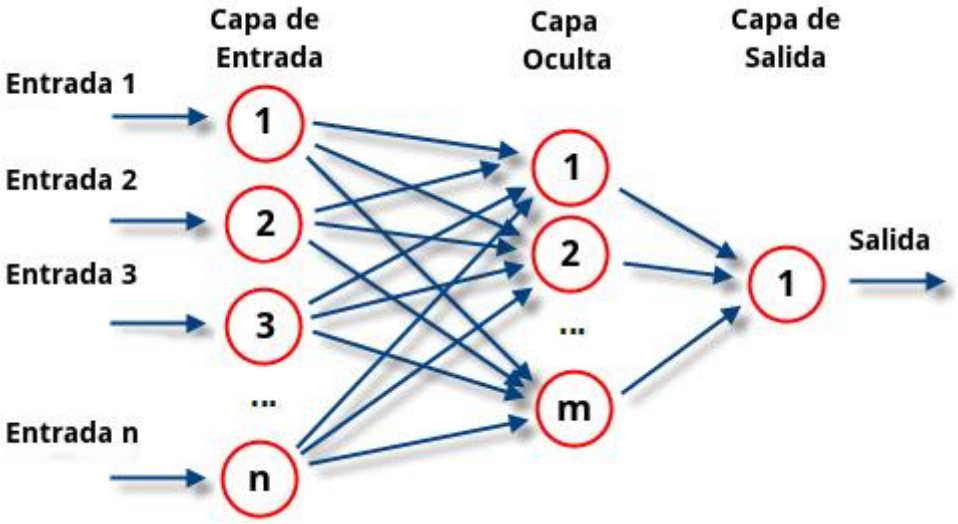


Figura 1 Perceptron multicapa

En el caso que nos ocupa, se ha utilizado una arquitectura con la forma $\{N_0:N_1:\ldots:N_h\}$.

Donde:

- N_0: Capa de entrada cuyo número de neuronas será igual al de un vector de entradas con la forma $(X_1,X_2\ldots.X_K)$.
- El número de capas ocultas viene dado por H-1.
- Nh: Capa de salida cuyo número de neuronas será igual al de un vector de salidas con la forma $(D_1,D_2\ldots D_j)$.

Además, se tiene en cuenta que todas las capas, a excepción de la capa de entrada, dispondrán de un sesgo.

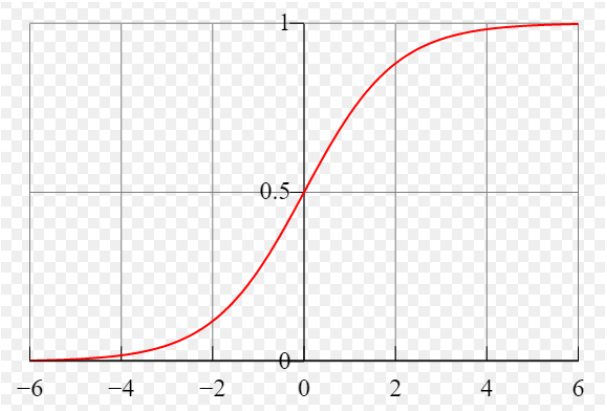


Figura 2 Función de activación Sigmoide

2. Descripción en pseudocódigo del algoritmo de retroprogación.

A continuación, realizaremos una descripción de las funciones más importantes en la implementación del algoritmo.

2.1. Algoritmo de retropropagación

```
INICIO
      iniciarPesosaleatorios()
      REPETIR
            PARA cada uno de los patrones con X entradas e Y
salidas
                  aplicarCambiosPatrones()
                  alimentarEntradas()
                  propagarEntradas()
                  retropropagarError()
                  acumularCambio()
                  ajustarPesos()
            FIN PARA
      HASTA(Condición de parada)
FIN
```

Esta función principal llamaría al resto de funciones que intervienen en el algoritmo de manera secuencial.

El algoritmo terminará una vez se dé la condición de parada. La cual puede ser de varios tipos. En la implementación realizada el algoritmo se detendrá si se alcanza un cierto número de iteraciones. Pero también lo hará si durante 50 iteraciones no se consigue una disminución del error de validación o de entrenamiento.

2.2. Iniciar pesos aleatorios

```
PARA i=1 hasta número de capas
      PARA j=0 hasta número de neuronas de la capa i
            PARA k=0 hasta numero de neuronas +1 de la capa (i-1)
                  n=Numero aleatorio generado entre -1 y 1
                  capa(i).neuronas(j).pesoNeurona(k)=n
            FIN PARA
      FIN PARA
FIN PARA
```

Esta función refleja cómo se introduce un peso aleatorio comprendido entre -1 y 1 a cada neurona cuando comienza el algoritmo.

También refleja la forma habitual de recorrer las estructuras de datos que forman la red neuronal en el código.

Para ello se han utilizado tres bucles donde:

- El primero se encargará de acceder a cada una de las capas.
- El segundo recorrerá las neuronas de la capa en la que se encuentra.
- El tercero accede a la información de cada neurona. Ya que estas se encuentran en un vector que a su vez contiene otros vectores que guardan información sobre sus pesos (peso actual, copia de los pesos, ultimo peso y cambio a aplicar).

Es importante resaltar en este último bucle que termina en (i-1). Esto se hace para que se recorran las capas ocultas. Adicionalmente se suma 1 al número de neuronas para tener en cuenta el sesgo. También es necesario comentar que cada vez que recorramos la red, se comienza desde la capa 1. Esto es porque la capa de entrada no va a disponer de sesgo y por este motivo, dicho sesgo se procesará una vez salgamos del tercer bucle cada vez que sea necesario al recorrer la red.

2.3. Alimentar entradas

```
PARA i=0 hasta número de neuronas de la capa de entrada
      Salida de la neurona i= input(i)
      Derivada de la neurona i= input(i)
FIN PARA
```

Aquí únicamente interesa recorrer el número de neuronas de la capa de entrada. Las cuales se "alimentarán" con un patron "input" recibido por argumento. Concretamente asignaremos ese patrón a la salida producida por dicha neurona y la derivada de la salida.

2.4. Propagar entradas

```
Sumatorio=0
PARA i=1 hasta número de capas
      PARA j=0 hasta número de neuronas de la capa i
            PARA k=1 hasta número de neuronas+1 de la capa (i-1)
                  sumatorio += pCapas[i-1].pNeuronas[k-
1].x*pCapas[i].pNeuronas[j].w[k]
            FIN PARA
      sumatorio += pCapas[i].pNeuronas[j].w[0];
      pCapas[i].pNeuronas[j].x = (1/(1 + exp((-1)*sumatorio)));
      sumatorio = 0.0;
      FIN PARA
```

```
FIN PARA
```

La función realiza una propagación hacia delante de las entradas, realizando las operaciones descritas en la presentación de la práctica.

El sesgo se procesa de manera independiente después de que termine el tercer bucle y depues de eso ya asignaremos a la salida "x" de la neurona el resultado obtenido. Tendremos que reiniciar la variable de sumatorio tras cada una de estas iteraciones.

2.5. Calcular error de la salida (MSE)

```
MSE=0

PARA i=0 hasta numero de neuronas de numero de capas -1

     MSE += pow(target[i] - pCapas[nNumCapas-
1].pNeuronas[i].x,2)

FIN PARA

MSE=errorMSE/pCapas[nNumCapas-1].nNumNeuronas

Devolver MSE
```

Se realiza el cálculo del MSE desde el número de capas -1, ya que se ha de tener en cuenta que nos situamos en la salida. Para calcularlo se realizan las operaciones vistas en la presentación de la práctica.

2.6. Retro propagar el error

```
Sumatorio=0

PARA i=0 hasta número de neuronas de numero de capas -1

     pCapas[nNumCapas-1].pNeuronas[i].dX =  (-1)*(objetivo[i] -
pCapas[nNumCapas-1].pNeuronas[i].x) * pCapas[nNumCapas-
1].pNeuronas[i].x * (1 - pCapas[nNumCapas-1].pNeuronas[i].x);

FIN PARA

PARA i=número de capas -2 hasta i>=0

     PARA j=0 hasta número de neuronas de la capa i

          PARA k=0 hasta número de neuronas de la capa i+1

               sumatorio += pCapas[i+1].pNeuronas[k].w[j+1] *
pCapas[i+1].pNeuronas[k].dX;

          FIN PARA

     pCapas[i].pNeuronas[j].dX = sumatorio *
pCapas[i].pNeuronas[j].x * (1-pCapas[i].pNeuronas[j].x);

     FIN PARA

FIN PARA
```

En esta función se calcula la retropropagación del error. Para ello se realizan las operaciones para el cálculo de las derivadas vistas en la presentación de la práctica.

En primer lugar, hay un bucle que calcula la derivada de la capa de salida (de ahí que el bucle se sitúe en el número de capas -1). A continuación, se utilizarán tres bucles que realizarán las operaciones hacia atrás, así que se comienza a iterar desde el número de capas -2. Después, con el tercer bucle calcularemos el sumatorio de las entradas y su derivada en la siguiente capa.

2.7. Ajustar pesos

```
_deltaW,_ultimodeltaW,etaAux=0

PARA i=1 hasta número de capas

     PARA j=0 hasta número de neuronas de la capa i

     etaAux=this->dEta*pow(this->dDecremento, -(this->nNumCapas-
1-i))

     PARA k=0 hasta número de neuronas+1 de la capa i-1

     _deltaW = pCapas[i].pNeuronas[j].deltaW[k];

_ultimoDeltaW = pCapas[i].pNeuronas[j].ultimoDeltaW[k];

pCapas[i].pNeuronas[j].w[k] -= etaAux * _deltaW + dMu * etaAux *
_ultimoDeltaW;

pCapas[i].pNeuronas[j].ultimoDeltaW[k] =
pCapas[i].pNeuronas[j].deltaW[k];

     FIN PARA

FIN PARA

FIN PARA
```

Realizamos el ajuste de los pesos recorriendo la red neuronal como se ha visto anteriormente y realizando las operaciones vistas en la presentación de la práctica.

Para esta función hemos necesitado variables auxiliares, pero concretamente, la variable “etaAux” ha tenido que hacerse auxiliar para que, a la hora de insertar un parámetro de decremento, el programa funcionase correctamente.

2.8. Acumular cambio

```
PARA i=1 hasta número de capas

     PARA j=0 hasta número de neuronas de la capa i
          PARA k=1 hasta número de neuronas+1 de la capa (i-1)

          pCapas[i].pNeuronas[j].deltaW[k] +=
pCapas[i].pNeuronas[j].dX * pCapas[i-1].pNeuronas[k-1].x;

          FIN PARA

          Capas[i].pNeuronas[j].deltaW[0] +=
pCapas[i].pNeuronas[j].dX;

     FIN PARA

FIN PARA
```

Hacemos el sumatorio de los cambios realizados para guardarlos en la variable correspondiente “deltaW”. Para ello la red se recorre como se ha visto anteriormente y el sesgo se procesa después del tercer bucle.

3. Experimentos y análisis de resultados.

3.1. Descripción de las bases de datos utilizadas.

Para la realización de los experimentos, se han utilizado un total de cuatro bases de datos suministradas específicamente para la práctica. Donde cada una tiene su fichero para “train” y para “test”.

- **Base de datos XOR**:
  - Número de entradas:2
  - Número de salidas:1
  - Número de patrones:4
- **Base de datos Quake**:
  - Número de entradas:3
  - Número de salidas:1
  - Número de patrones:1633
- **Base de datos Sin**:
  - Número de entradas:1
  - Número de salidas:1
  - Número de patrones:120
- **Base de datos Parkinsons**:
  - Número de entradas:19
  - Número de salidas:2
  - Número de patrones:4406

3.2. Descripción de los valores de los parámetros considerados.

A continuación, se muestran los parámetros suministrados al programa y su valor por defecto:

- Eta (e): 0.1
- Mu (m): 0.90
- Tamaño del conjunto de validación (v): 0.0
- Factor de decremento (d): 1
- Numero de Iteraciones (3): 1000
- Numero de capas ocultas (l): 1
- Numero de neuronas por capa oculta (h): 5

Además, el programa recibe un fichero con los datos de “train” y otro con los datos de “test”. Para la realización de esta práctica se han tenido en cuenta el número de capas ocultas y el número de neuronas en cada una de estas capas.

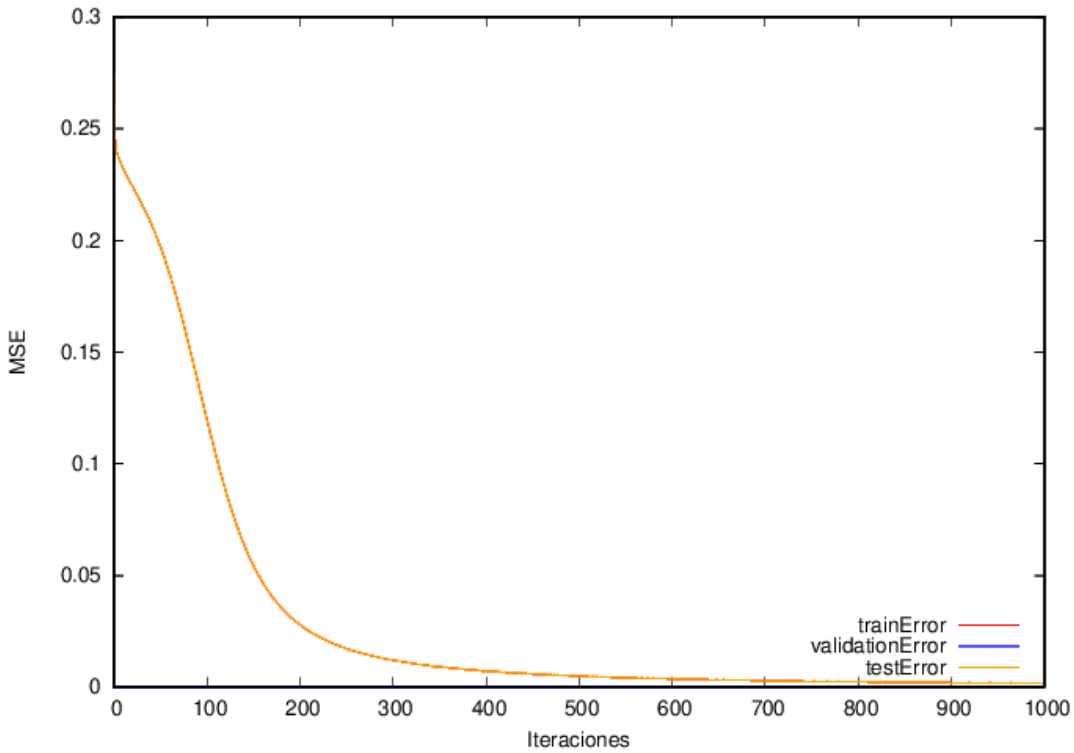
Posteriormente, con los mejores resultados obtenidos con dichos parámetros, se han hecho otras pruebas. Pero modificando esta vez el factor de decremento y el tamaño del conjunto de validación. Los valores que han tomado estos parámetros se indican en las tablas que se muestran a continuación.

3.3. Resultados obtenidos.

3.3.1. Base de datos XOR

UNA CAPA OCULTA						
NODOS EN CAPA OCULTA	2	4	8	32	64	100
MEDIA TRAIN	0,157092	0,0299545	0,0121854	0,00765031	0,00222111	0,101493
DESVIACION TIPICA TRAIN	0,0803387	0,033203	0,00395294	0,00530661	0,00064387	0,136077
MEDIA TEST	0,157092	0,0299545	0,0121854	0,00765031	0,00222111	0,101493
DESVIACION TIPICA TEST	0,0803387	0,033203	0,00395294	0,00530661	0,00064387	0,136077
ERROR TEST FINAL	0,157092	0,0299545	0,0121854	0,00765031	0,00222111	0,101493
DOS CAPAS OCULTAS						
NODOS EN CAPA OCULTA	2	4	8	32	64	100
MEDIA TRAIN	0,203962	0,106817	0,0109428	0,0199848	0,0030811	0,0122472
DESVIACION TIPICA TRAIN	0,0422011	0,106673	0,00991192	0,0281164	0,00432196	0,015782
MEDIA TEST	0,203962	0,106817	0,0109428	0,0199848	0,0030811	0,0122472
DESVIACION TIPICA TEST	0,0422011	0,106673	0,00991192	0,0281164	0,00432196	0,015782
ERROR TEST FINAL	0,203962	0,106817	0,0109428	0,0199848	0,0030811	0,0122472
NOS QUEDAMOS CON LA ARQUITECTURA:	2 capas ocultas, 64 nodos en capa oculta					
Probamos con decremento puesto que en XOR no se considera validacion						
DECREMENTO	1	2				
MEDIA TRAIN	0,0030811	0,00846824				
DESVIACION TIPICA TRAIN	0,00432196	0,0160766				
MEDIA TEST	0,0030811	0,00846824				
DESVIACION TIPICA TEST	0,00432196	0,0160766				
ERROR TEST FINAL	0,0030811	0,00846824				
Mejor arquitectura:	2 Capas ocultas, 64 nodos en capa oculta, 1 Decremento					

Tabla 1 Pruebas Xor

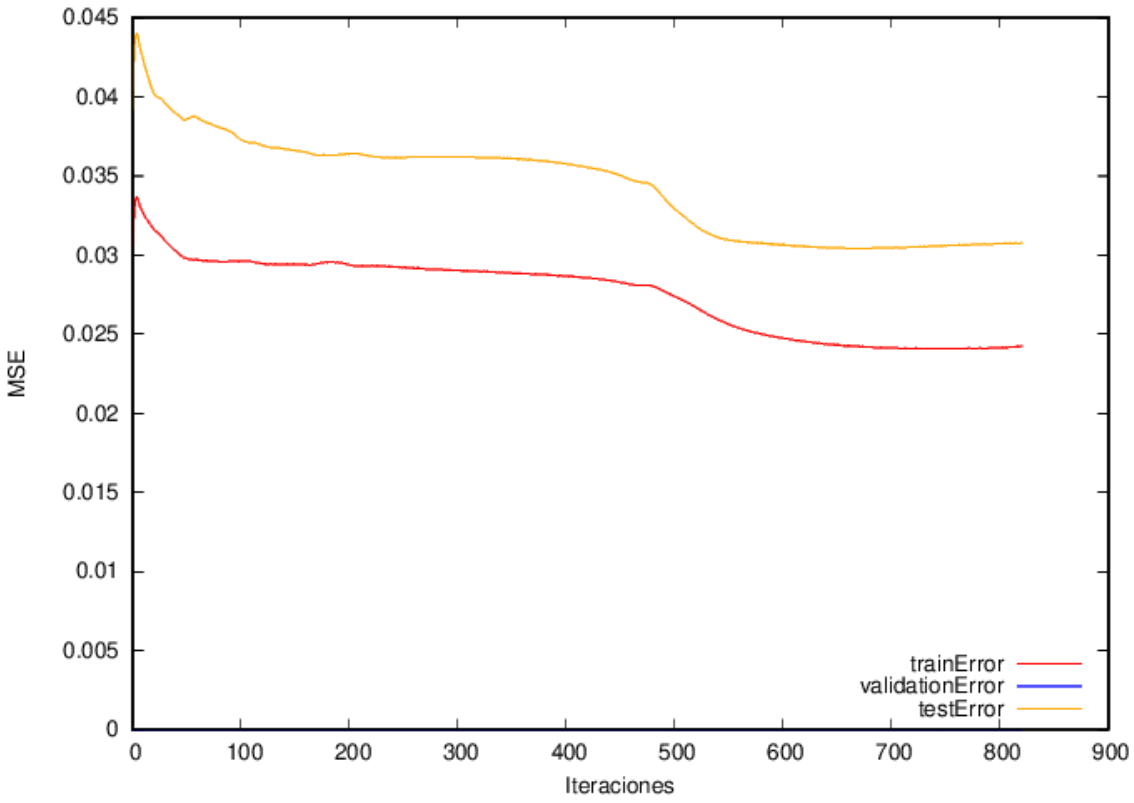


Gráfica 1 XOR l1 h64 d2

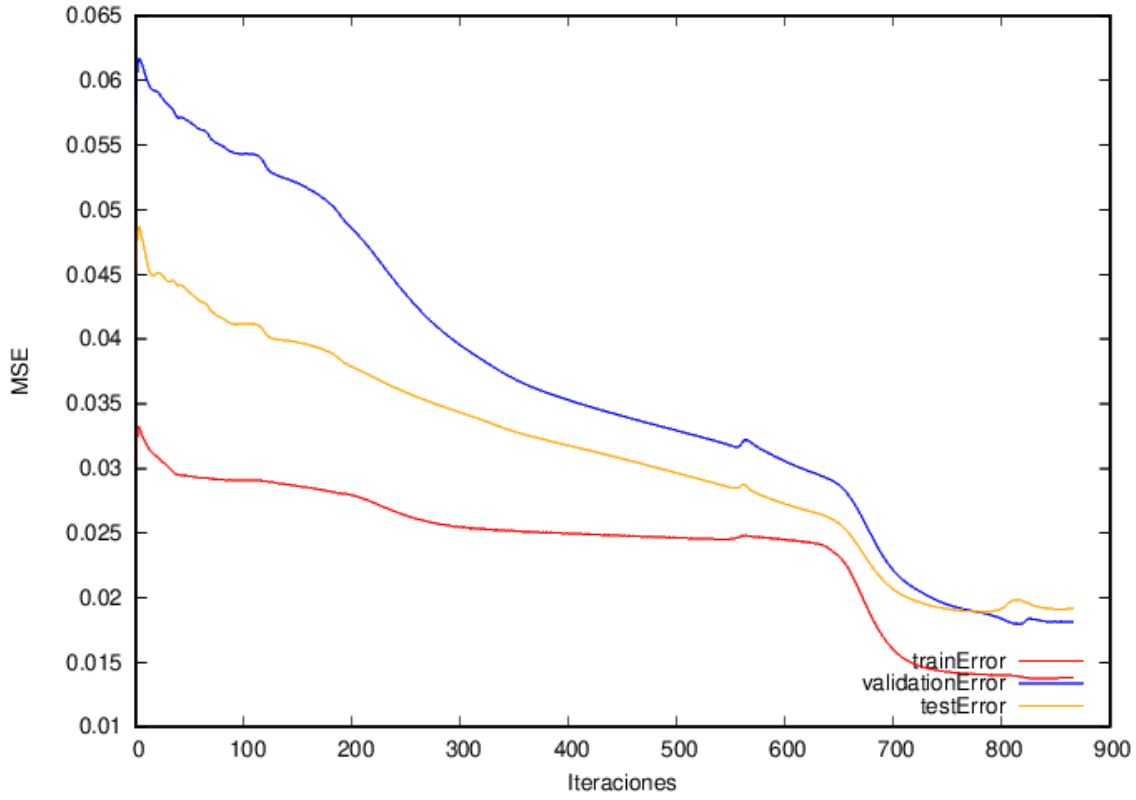
3.3.2. Base de datos Sin

UNA CAPA OCULTA						
NODOS EN CAPA OCULTA	2	4	8	32	64	100
MEDIA TRAIN	0,0298764	0,0293787	0,0282071	0,0291659	0,0244513	0,0257868
DESVIACION TIPICA TRAIN	4,93176E-05	0,00164901	0,000667274	0,00348013	0,0123423	0,00754912
MEDIA TEST	0,0361212	0,0359988	0,0351108	0,0352325	0,0313141	0,0330982
DESVIACION TIPICA TEST	9,66792E-05	0,000659024	0,00181945	0,00473008	0,012953	0,00940531
ERROR TEST FINAL	0,0329988	0,03268875	0,03165895	0,0321992	0,0278827	0,0294425
DOS CAPAS OCULTAS						
NODOS EN CAPA OCULTA	2	4	8	32	64	100
MEDIA TRAIN	0,0298545	0,0298844	0,0301272	0,0283417	0,0213321	0,0161846
DESVIACION TIPICA TRAIN	0,000118367	0,000178368	0,000877849	0,00679496	0,0156707	0,00624033
MEDIA TEST	0,0362816	0,0361163	0,0361861	0,0348718	0,028157	0,0234001
DESVIACION TIPICA TEST	0,000851961	0,000352326	0,00106687	0,00687245	0,0170475	0,00826162
ERROR TEST FINAL	0,03306805	0,03300035	0,03315665	0,03160675	0,02474455	0,01979235
NOS QUEDAMOS CON LA ARQUITECTURA:	2 capas ocultas, 100 nodos					
VALIDACION	0	0	0,15	0,15	0,25	0,25
DECREMENTO	1	2	1	2	1	2
MEDIA TRAIN	0,0161846	0,023937	0,0170902	0,0281256	0,0218569	0,0249573
DESVIACION TIPICA TRAIN	0,00624033	0,0122805	0,00905958	0,00398941	0,0101945	0,0131248
MEDIA TEST	0,0234001	0,0308171	0,0248165	0,036895	0,0290566	0,0326882
DESVIACION TIPICA TEST	0,00826162	0,0138753	0,0115113	0,00650564	0,0114917	0,0158658
ERROR TEST FINAL	0,01979235	0,02737705	0,02095335	0,0325103	0,02545675	0,02882275
Mejor arquitectura:	2 capas oultas, 100 nodos capa oculta, 1 Decremento, 0 validacion					

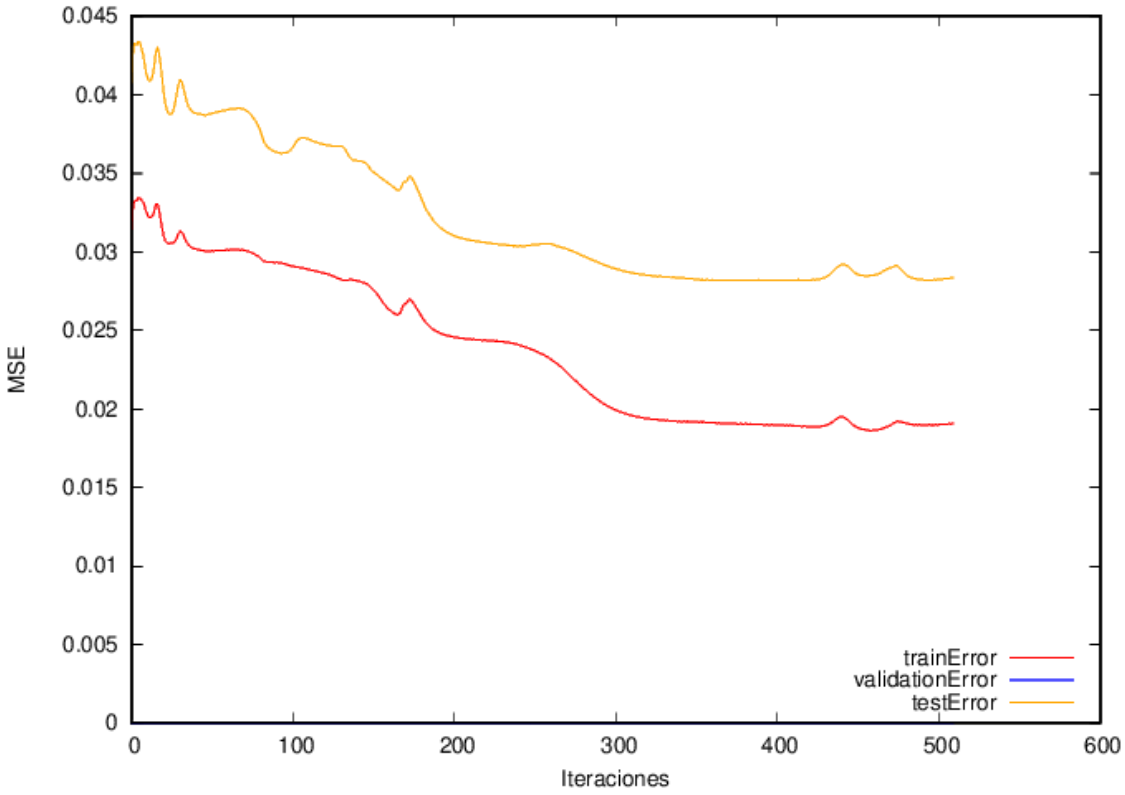
Tabla 2 Pruebas Sin



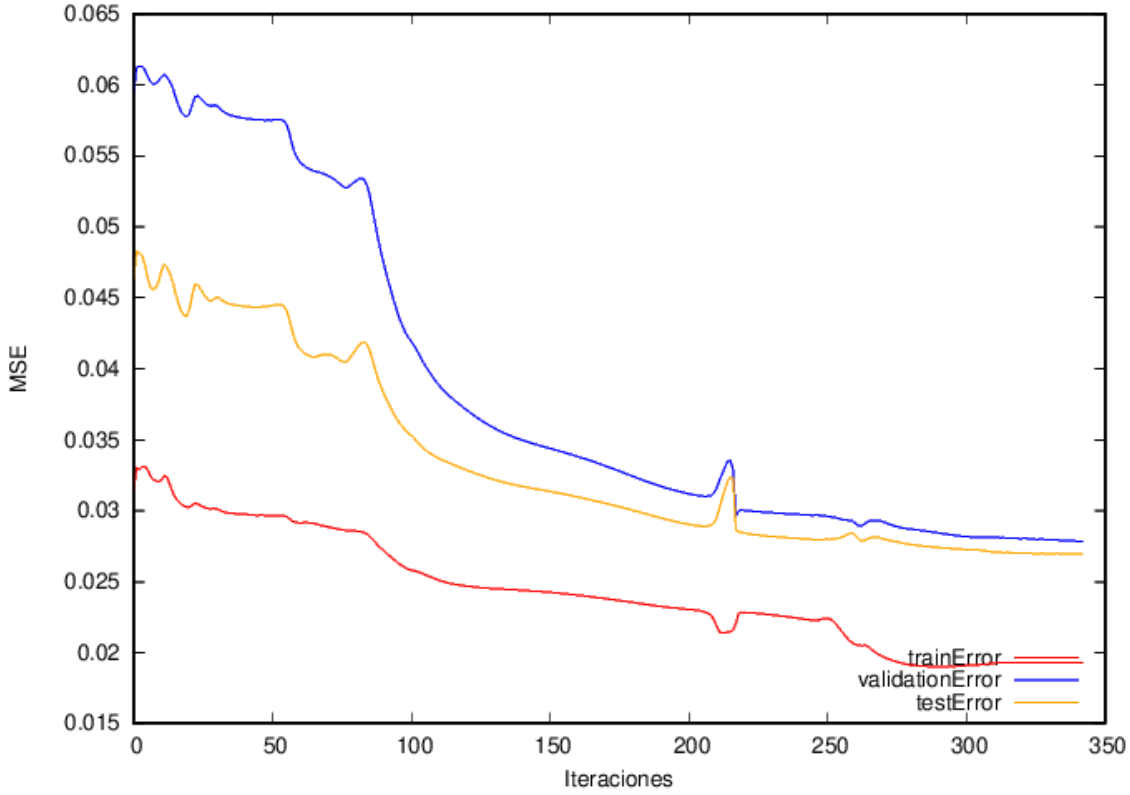
Gráfica 2 Sin l1 h64 d2 v0.0



Gráfica 3 Sin l1 h64 d2 v0.25



Gráfica 4 l1 h64 d1 v0.0

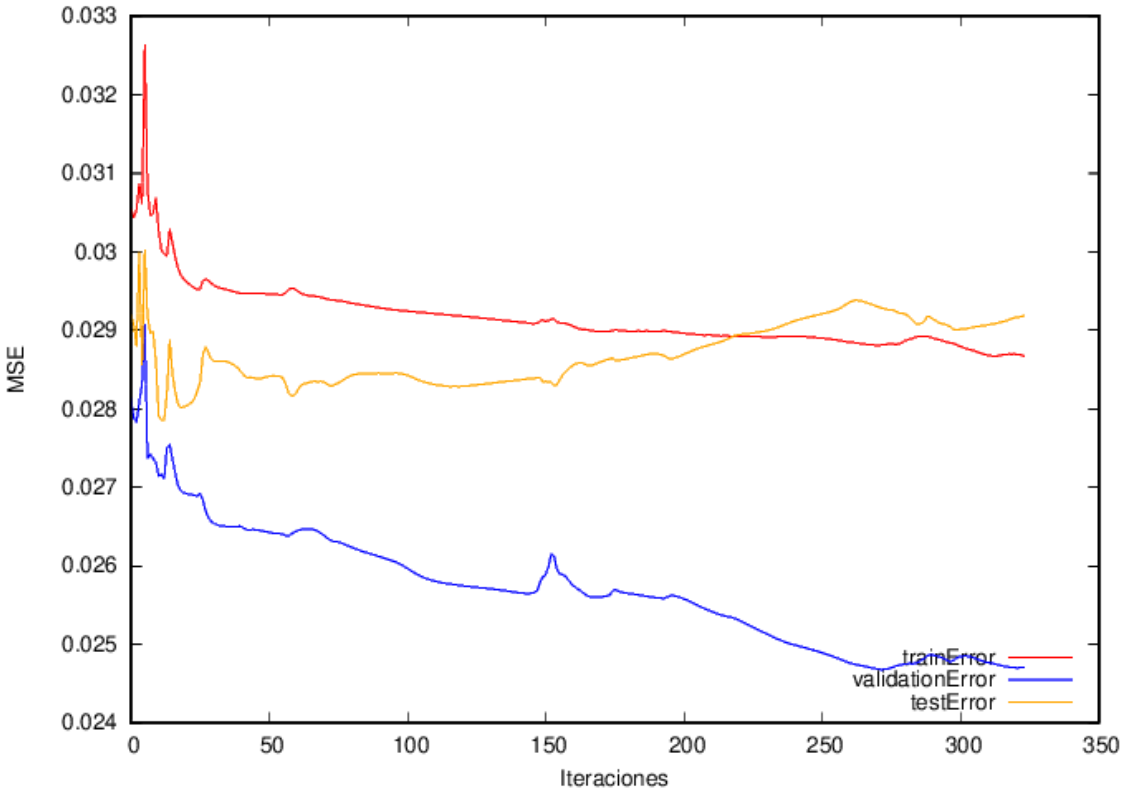


Gráfica 5 Sin l1 h64 d1 v0.25

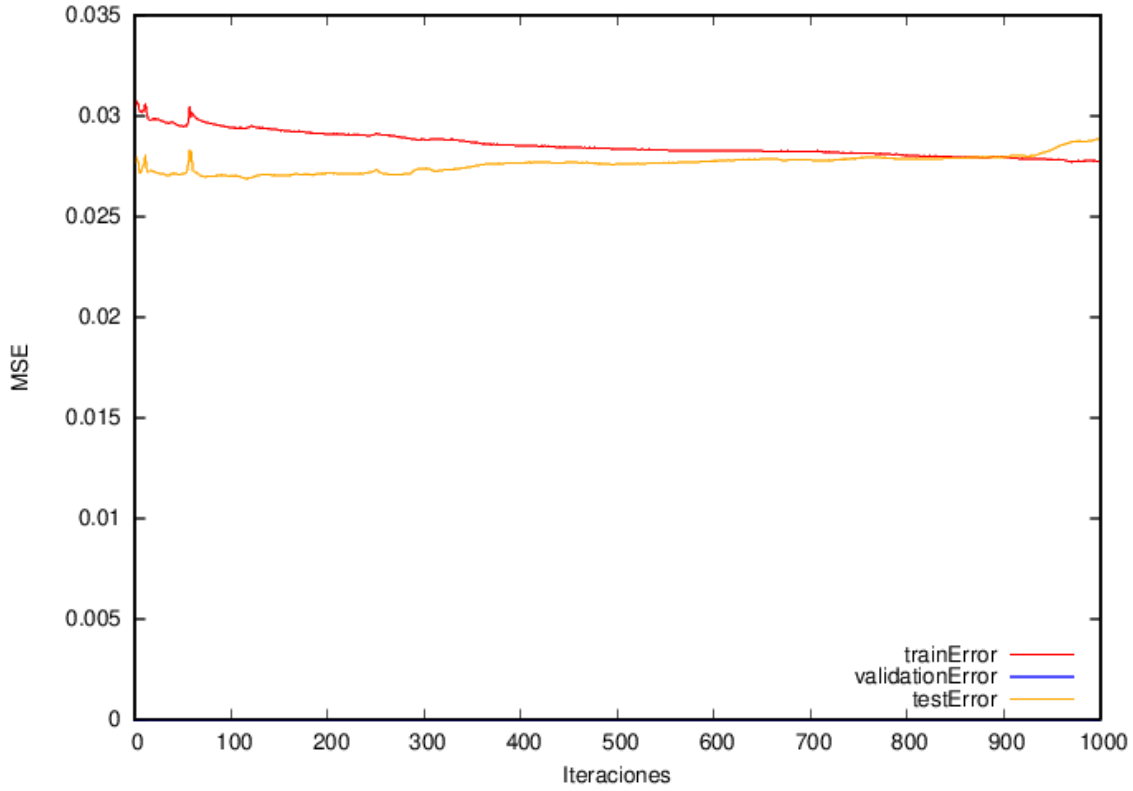
3.3.3. Base de datos Quake

UNA CAPA OCULTA						
NODOS EN CAPA OCULTA	2	4	8	32	64	100
MEDIA TRAIN	0,0300101	0,0299105	0,0295971	0,0295084	0,0295362	0,0294751
DESVIACION TIPICA TRAIN	0,000220102	0,000284939	0,000297777	0,000273825	0,00052874	0,000129328
MEDIA TEST	0,0271114	0,0270395	0,0270706	0,0270786	0,0270594	0,0271162
DESVIACION TIPICA TEST	0,000270713	0,000215776	0,000105097	0,000342819	0,00021607	0,000317384
ERROR TEST FINAL	0,02856075	0,028475	0,02833385	0,0282935	0,0282978	0,02829565
DOS CAPAS OCULTAS						
NODOS EN CAPA OCULTA	2	4	8	32	64	100
MEDIA TRAIN	0,0298723	0,0299251	0,0297571	0,0290166	0,0288671	0,028416
DESVIACION TIPICA TRAIN	0,000473101	0,000576249	0,000552687	0,000940271	0,00104361	0,00102628
MEDIA TEST	0,0269868	0,0270479	0,0270317	0,0270363	0,0272643	0,0273506
DESVIACION TIPICA TEST	0,000553364	0,000565406	0,00039753	0,0003517	0,00061493	0,00167855
ERROR TEST FINAL	0,02842955	0,0284865	0,0283944	0,02802645	0,0280657	0,0278833
NOS QUEDAMOS CON LA ARQUITECTURA:	2 capas ocultas, 100 nodos					
VALIDACION	0	0	0,15	0,15	0,25	0,25
DECREMENTO	1	2	1	2	1	2
MEDIA TRAIN	0,028416	0,0294524	0,0294425	0,0294493	0,0290551	0,0293407
DESVIACION TIPICA TRAIN	0,00102628	0,000917271	0,00025779	0,00124431	0,00093125	0,000585316
MEDIA TEST	0,0273506	0,0271337	0,0289956	0,0290977	0,0283419	0,0275722
DESVIACION TIPICA TEST	0,00167855	0,000234058	0,005841	0,00596121	0,00127817	0,00110551
ERROR TEST FINAL	0,0278833	0,02829305	0,02921905	0,0292735	0,0286985	0,02845645
Mejor arquitectura:	2 capas ocultas, 100 nodos, 0 validacion, 1 decremento					

Tabla 3 Pruebas Quake



Gráfica 6 Quake l2 h100 d1 v0.25

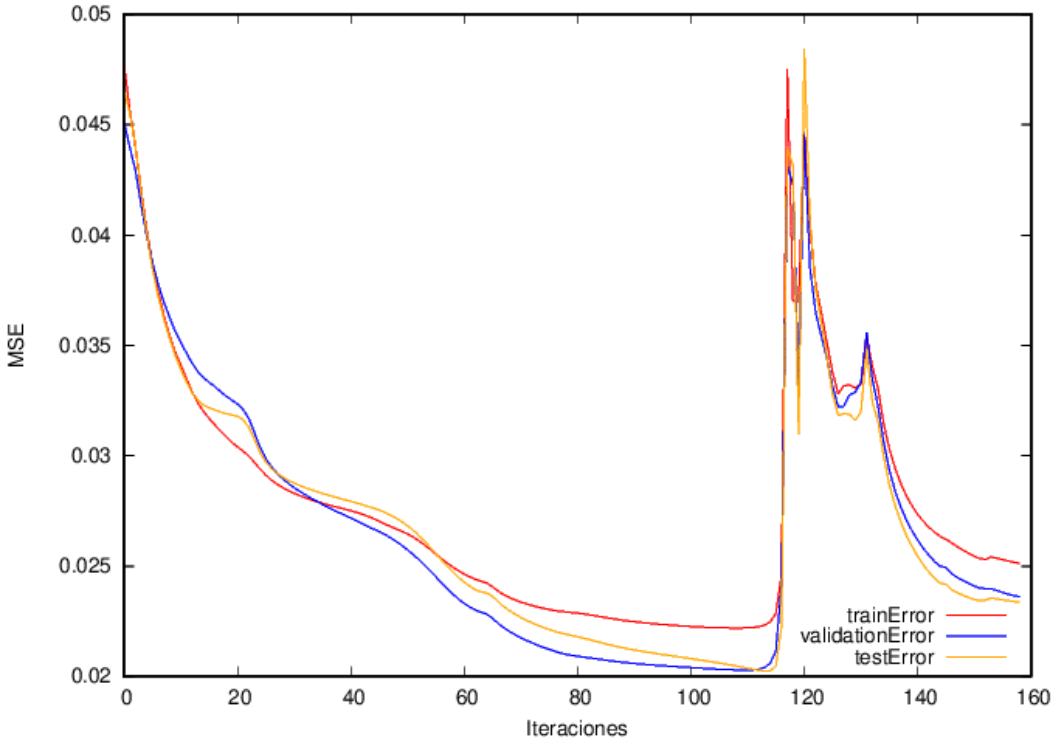


Gráfica 7 Quake l2 h100 d1 v0.0

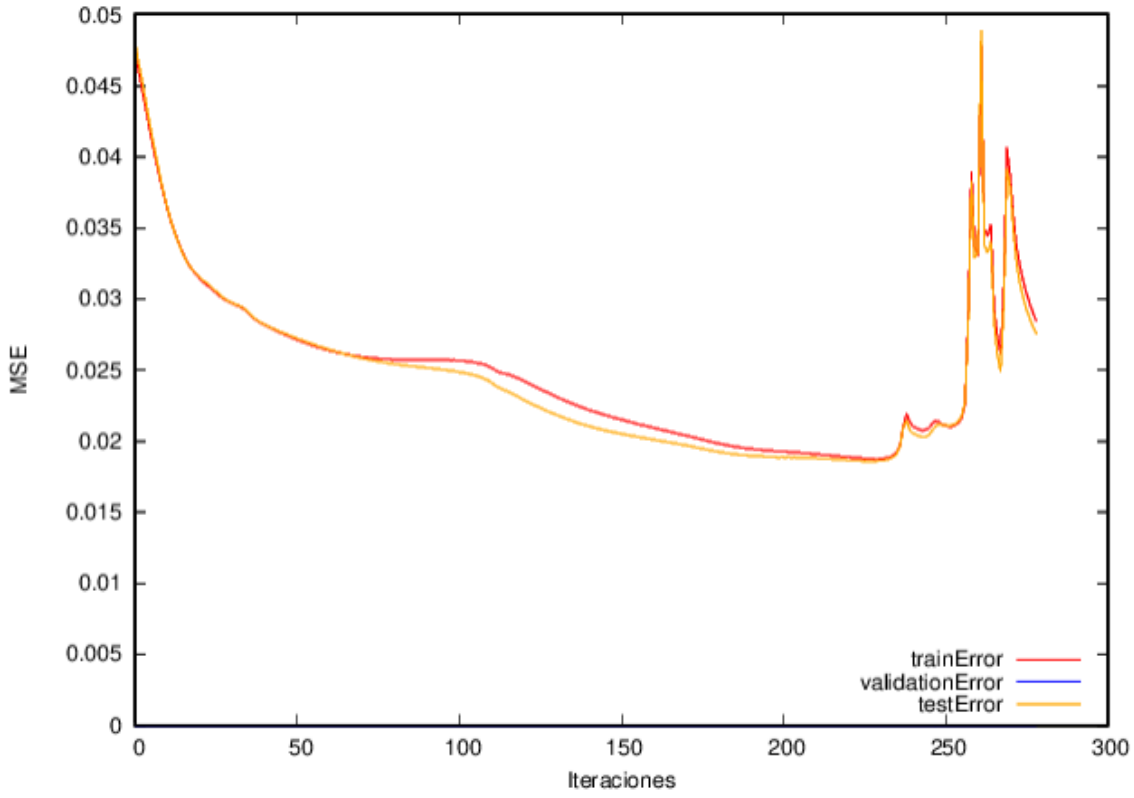
3.3.4. Base de datos Parkinsons

UNA CAPA OCULTA						
NODOS EN CAPA OCULTA	**2**	**4**	**8**	**32**	**64**	**100**
MEDIA TRAIN	0,0347106	0,0268143	0,0256747	0,0127902	0,0206202	0,0203455
DESVIACION TIPICA TRAIN	0,00199264	0,00308442	0,0113761	0,00744157	0,00437914	0,0085575
MEDIA TEST	0,036964	0,0276837	0,0259132	0,0148029	0,0209062	0,0210552
DESVIACION TIPICA TEST	0,000563127	0,00447544	0,0116403	0,00888281	0,00375672	0,00558493
ERROR TEST FINAL	0,0358373	0,027249	0,02579395	0,01379655	0,0207632	0,02070035
DOS CAPAS OCULTAS						
NODOS EN CAPA OCULTA	**2**	**4**	**8**	**32**	**64**	**100**
MEDIA TRAIN	0,0347406	0,0300173	0,0228153	0,0204558	0,0186823	0,0194218
DESVIACION TIPICA TRAIN	0,00197535	0,00460147	0,00226616	0,00912686	0,00486377	0,00929795
MEDIA TEST	0,03667	0,0302805	0,0230824	0,0208521	0,019293	0,0200196
DESVIACION TIPICA TEST	0,00315834	0,00462139	0,00283683	0,00912686	0,00323088	0,00773201
ERROR TEST FINAL	0,0357053	0,0301489	0,02294885	0,02065395	0,01898765	0,0197207
NOS QUEDAMOS CON LA ARQUITECTURA:	1 capa oculta, 32 nodos en capa oculta					
VALIDACION	**0**	**0**	**0,15**	**0,15**	**0,25**	**0,25**
DECREMENTO	**1**	**2**	**1**	**2**	**1**	**2**
MEDIA TRAIN	0,0127902	0,0205502	0,0194979	0,0218155	0,0180894	0,0228012
DESVIACION TIPICA TRAIN	0,00744157	0,00691767	0,00970389	0,00617006	0,00552826	0,00486227
MEDIA TEST	0,0148029	0,0212227	0,0194014	0,0208372	0,0192624	0,0235099
DESVIACION TIPICA TEST	0,00888281	0,00589012	0,0112107	0,00729895	0,00552094	0,00710634
ERROR TEST FINAL	0,01379655	0,01322016	0,01944965	0,02132635	0,0186759	0,02315555
Mejor arquitectura:	1 capa oculta, 32 nodos en capa oculta, 0 validacion, decremento 2					

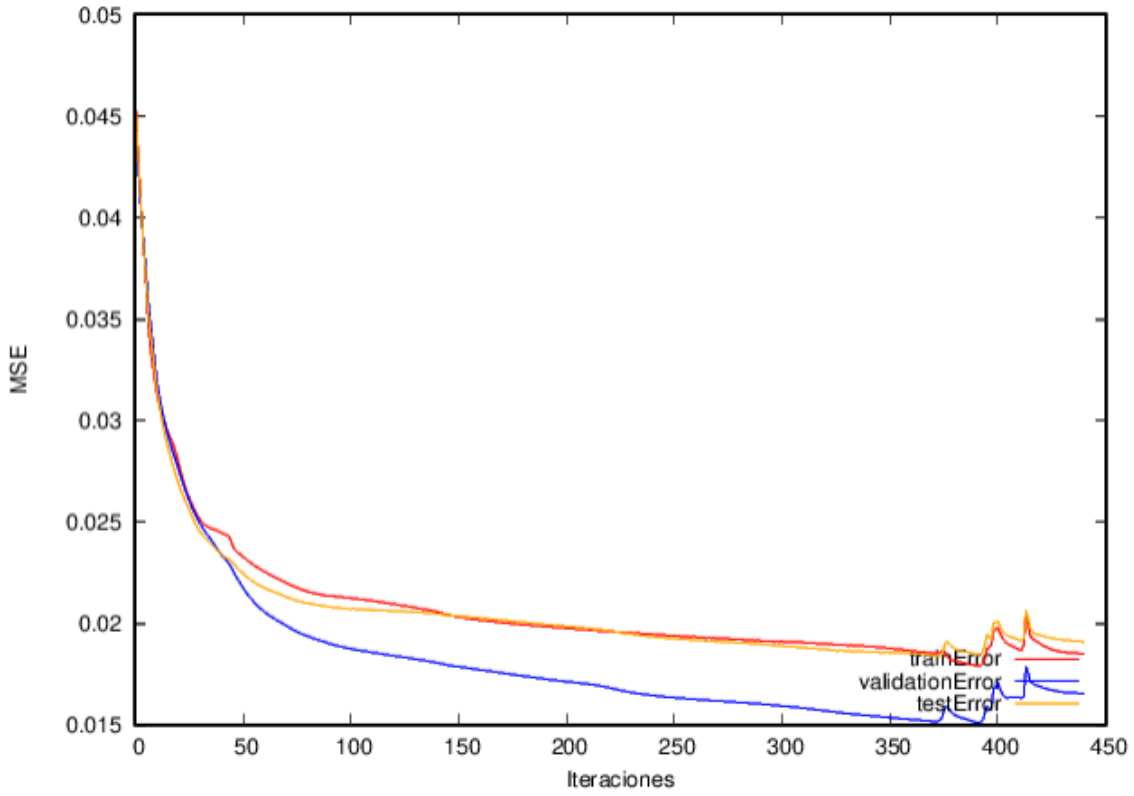
Tabla 4 Pruebas Parkinsons



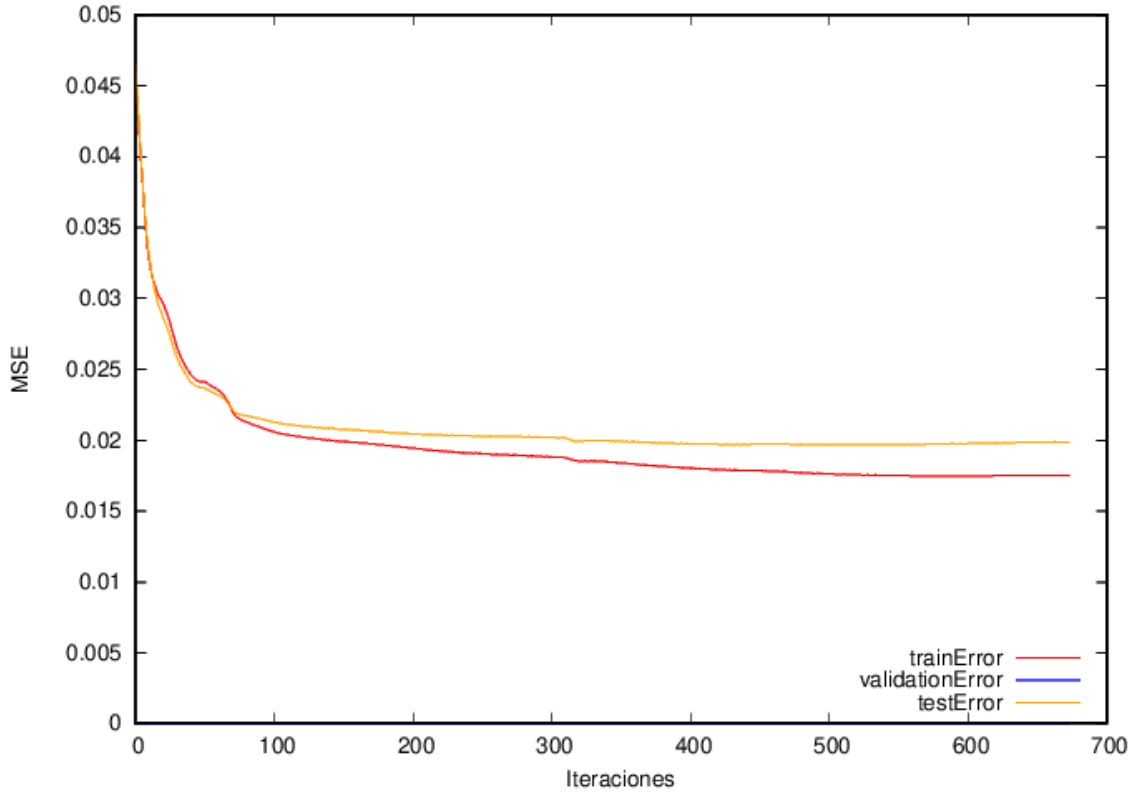
Gráfica 8 Parkinsons l1 h32 d2 v0.25



Gráfica 9 Parkinsons l2 h32 d2 v0.0



Gráfica 10 Parkinsons l2 h32 d1 v0.25



Gráfica 11 Parkinsons l1 h32 d1 v0.0

3.4. Análisis de resultados.

En vista de los resultados obtenidos en las tablas y su posterior ilustración en las gráficas, podemos sacar las siguientes conclusiones.

En primer lugar, se observa que la activación del conjunto de validación reduce, en la mayoría de los casos de forma muy considerable, el número de iteraciones que realiza el algoritmo. Esto es debido a la condición de parada que activa este conjunto, donde si el error de validación no mejora tras 50 iteraciones, el algoritmo se detiene. Lo cual nos lleva a una mayor eficiencia el tiempo de cómputo sobre todo en bases de datos grandes, como es el caso de Parkinsons por ejemplo. Sin embargo, este conjunto no se ha tenido en cuenta en la base de datos Xor. Debido a que tiene tan pocos patrones que no es necesario

Pero también sucede que la media del error no mejora, sino que sufre un ligero aumento. Sin embargo, el error aumenta de forma tan poco significativa que no se puede decir que empeore el algoritmo. Así que sería recomendable utilizar este conjunto de validación pese a ese pequeño incremento del error. Ya que por un lado ahorramos tiempo de cómputo y por otro, evitamos un posible sobre entrenamiento.

Por otro lado, el factor de decremento ha facilitado un mejor ajuste de los pesos a lo largo de la ejecución del algoritmo. Como se puede ver, por ejemplo, en la Gráfica 5 contra la Gráfica 3, donde poner un factor de decremento a 2 provoca menos picos en el gráfico. Aun así, este parámetro no ha provocado mejoras significativas en el error. Tan solo en la base de datos Parkinsons, que es la que más patrones tiene.

Con respecto a las capas ocultas y el número de neuronas que en ellas se encuentra. No se aprecia una diferencia significativa en el error. Dado un mismo número de neuronas en capa oculta, probando con una o dos capas ocultas, la diferencia es mínima, con lo que el parámetro más importante en este caso vendría a ser el número de neuronas en capa oculta. Después ya se observa si el error es menor con una o dos capas. Es importante recalcar que, a mas neuronas y más capas ocultas, evidentemente el tiempo de cómputo aumenta. Con lo que para este caso también hay que considerar obtener, en según que casos, un error ligeramente mayor. Pero a cambio obtendríamos mucho menos tiempo de cómputo.

4. Análisis del modelo de red para el problema Xor

Se ha planteado una red neuronal con una capa oculta y dos nodos, en ella. Siendo este el caso más simple.

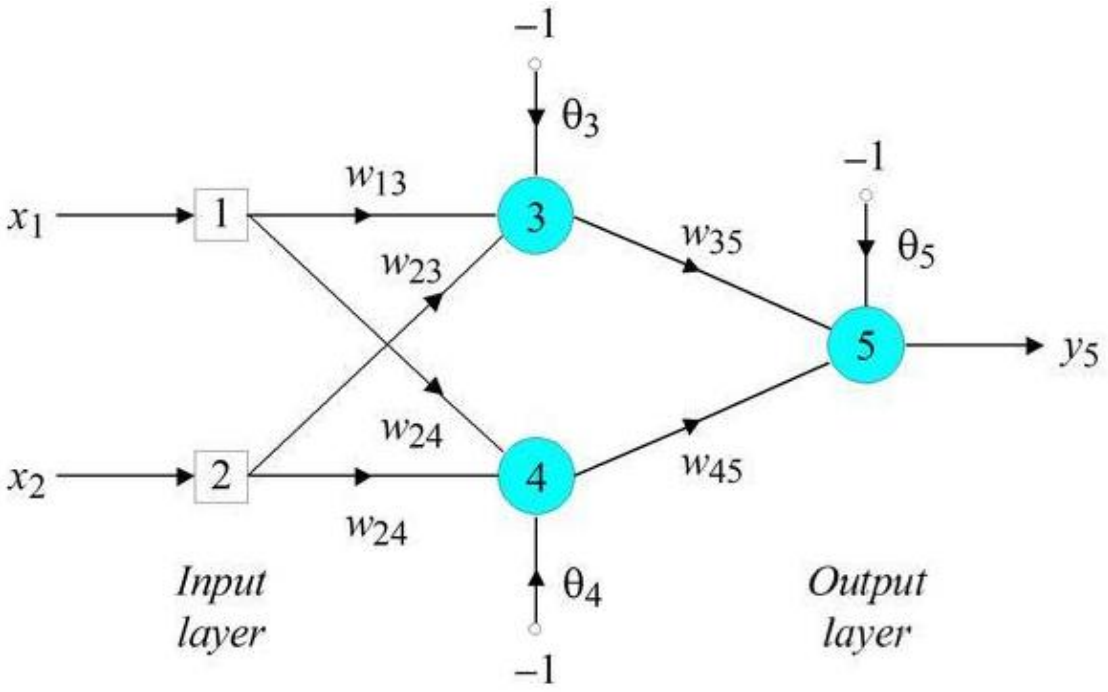


Figura 3 Red neuronal XOR más simple

Los pesos obtenidos han sido los siguientes:

```
PESOS DE LA RED
===============
Capa(1)------
[--2.94902-] [--2.73863-]
[--3.10418-] [-3.05075-]
Capa(2)------
[--1.90892-] [-3.89799-]
```

Figura 4 Pesos de la red Xor

Con esta configuración tan simple, la red es capaz de hacer unos ajustes más o menos decentes. Aunque se quedan lejos de la calidad del ajuste que se produce con más nodos en capa oculta, como se ha demostrado en la Tabla 1. Siendo este un problema con muy pocos datos, el tiempo de cómputo no será elevado aún en caso de poner más capas y nodos.

```
Salida Esperada Vs Salida Obtenida (test)
=========================================
1 -- 0.84035
0 -- 0.174514
1 -- 0.845721
0 -- 0.174507
```

Figura 5 Ajustes Red neuronal XOR