Universidad de Córdoba
Escuela Politécnica Superior de Córdoba

Ingeniería Informática
Especialidad: Computación
Cuarto curso. Primer cuatrimestre

Introducción a los modelos computacionales.

Práctica 1: Implementación del perceptrón multicapa.

José Manuel Cuevas Muñoz
31013019A
i62cumuj@uco.es

Curso académico 2019-2020
Córdoba, 7 de octubre de 2019

Índice

Índice de figuras II

Índice de tablas III

Índice de algoritmos IV

1. Introducción a los modelos de redes neuronales 1

2. Implementación 1
- 2.1. Algoritmo de entrenamiento en linea 1
- 2.2. Alimentar Entradas y Recoger Salidas 2
- 2.3. Propagar entradas 2
- 2.4. Calcular error de salida 2
- 2.5. Retropropagar error 3
- 2.6. Acumular cambio 3
- 2.7. Ajustar pesos 4

3. Resultados prácticos 4
- 3.1. Ejemplo de salida de red neuronal 5
- 3.2. Prueba de las base de datos con distintas arquitecturas 6
- 3.3. Pruebas variando el conjunto de validación y el factor de decremento 9
- 3.4. Gráficos de convergencia 10

Índice de figuras

1. Salida del entrenamiento en nuestra red neuronal 5
2. Red neuronal para el XOR 6
3. Errores en la base de datos del seno 11
4. Sobreentrenamiento en la base de datos del seno 12
5. Sobreentrenamiento en la base de datos del parkinson 13
6. Entrenamiento en la base de datos del seno con eta alta 14
7. Entrenamiento en la base de datos del seno con eta baja . . . 15
8. Entrenamiento en la base de datos quake 16
9. Entrenamiento en la base de datos quake sin mu 17
10. Entrenamiento en la base de datos parkinson. 18

Índice de tablas

1. Tabla de errores de la base de datos xor con distintas arquitecturas 7
2. Tabla de errores de la base de datos seno con distintas arquitecturas 7
3. Tabla de errores de la base de datos quake con distintas arquitecturas 8
4. Tabla de errores de la base de datos parkinsons con distintas arquitecturas 8
5. Tabla de errores de la base de datos seno variando el conjunto de validación y decremento 9
6. Tabla de errores de la base de datos quake variando el conjunto de validación y decremento 9
7. Tabla de errores de la base de datos parkinson variando el conjunto de validación y decremento 10

Índice de algoritmos

1. Entrenamiento en linea de la red 1
2. Alimentar las neuronas de entrada 2
3. Recoger los datos de salidas 2
4. Propagar entradas 2
5. Calcular error de salida 2
6. Algoritmo de retropropagación del error 3
7. Acumular cambio 3
8. Ajustar pesos 4

1. Introducción a los modelos de redes neuronales

Las redes neuronales son un modelo computacional basado en el funcionamiento de las neuronas. En nuestro caso haremos uso del perceptrón multicapa, un modelo en el que además de la capa de entrada y de salida, existen capas ocultas. La función de activación de nuestra red neuronal será una sigmoide.

Las redes neuronales tienen una fase de aprendizaje donde se entrenan para poder adecuarse a nuestro modelo de entrenamiento. Esta fase se compone de la propagar hacia delante los patrones, calcular el error de salida, propagarlo hacia atrás y corregir los pesos. En nuestro caso, lo haremos por cada patrón ya que estamos realizando un entrenamiento en linea.

2. Implementación

En esta sección hablaré de los distintos algoritmos utilizados en el entrenamiento de las redes neuronales:

2.1. Algoritmo de entrenamiento en linea

Algorithm 1 Entrenamiento en linea de la red

```
1:  procedure ENTRENARONLINE
2:      pesosAleatorios()
3:      for all patrón en test con entrada e y salida s do
4:          for all neurona j de la capa i do
5:              for all neurona k de la capa i − 1 do
6:                  Δw_jk ← 0
7:          AlimentarEntradas()
8:          propagarEntradas()
9:          retropropagarError()
10:         acumularCambio()
11:         ajustarPesos()
```

2.2. Alimentar Entradas y Recoger Salidas

Algorithm 2 Alimentar las neuronas de entrada

1: **procedure** ALIMENTARENTRADAS(datosDeEntrada)
2: **for all** neurona j en la capa 0 **do**
3: $w_j^0 \leftarrow datosDeEntrada[j]$

Algorithm 3 Recoger los datos de salidas

1: **procedure** RECOGERSALIDAS(vectorSalidas)
2: **for all** neurona j en la capa de salida(H) **do**
3: $vectorSalidas[j] \leftarrow w_j^H$

2.3. Propagar entradas

Algorithm 4 Propagar entradas

1: **procedure** PROPAGARENTRADAS
2: **for all** capa i desde 1 hasta H **do**
3: **for all** neurona j de la capa i **do**
4: $SigmoidSum \leftarrow w_{j0}^i$
5: **for all** neurona k de la capa $i-1$ $+1$ **do**
6: $SigmoidSum \leftarrow SigmoidSum + x_k^{i-1} \times w_{j,k-1}^i$
7: $x_j^i \leftarrow \frac{1}{1+e^{-SigmoidSum}}$

2.4. Calcular error de salida

Algorithm 5 Calcular error de salida

1: **procedure** CALCULARERRORSALIDA(objetivo)
2: $error \leftarrow 0$
3: **for all** neurona i en la capa de salida(H) **do**
4: $error \leftarrow error + (objetivo[i] - x_i^H)^2$
5: $error \leftarrow \frac{error}{size(H)}$
return $error$

2.5. Retropropagar error

Algorithm 6 Algoritmo de retropropagación del error

procedure RETROPROPAGARERROR(objetivo)
  for all neurona i en la capa de salida(H) **do**
    $out \leftarrow x_i^H$
    $dX_i^H \leftarrow -(objetivo[i] - out) * out * (1 - out)$
  for all capa i desde la ultima capa oculta hasta la capa de entrada **do**
    for all neurona j de la capa i **do**
      $sum \leftarrow 0$
      for all neurona k en la capa $i + 1$ **do**
        $sum \leftarrow sum + dX_k^{i+1} * w_{k,j+1}^{i+1}$
      $out \leftarrow x_j^i$
      $dX_j^i \leftarrow sum * out * (1 - out)$

2.6. Acumular cambio

Algorithm 7 Acumular cambio

procedure ACUMULARCAMBIO
  for all capa i desde 1 hasta H **do**
    for all neurona j de la capa i **do**
      for all neurona k de la capa $i - 1{+}1$ **do**
        $ultimo\Delta w_{jk}^i \leftarrow \Delta w_{jk}^i$
        $\Delta w_{jk}^i \leftarrow \Delta w_{jk}^i + dX_j^i * x_{k-1}^{i-1}$
      $ultimo\Delta w_{j0}^i \leftarrow \Delta w_{j0}^i$
      $\Delta w_{j0}^i \leftarrow \Delta w_{j0}^i + dX_j^i$

2.7. Ajustar pesos

Algorithm 8 Ajustar pesos

```
1: procedure AJUSTARPESOS
2:     η ← dEta
3:     for all capa i desde 1 hasta H do
4:         for all neurona j de la capa i do
5:             for all neurona k de la capa i − 1 +1 do
6:                 w^i_jk ← w^i_jk − η ∗ Δw^i_jk − μ ∗ η ∗ ultimoΔw^i_jk
7:             w^i_j0 ← w^i_j0 − η ∗ Δw^i_j0 − μ ∗ η ∗ ultimoΔw^i_j0
           η ← η ∗ dDecremento^−(numeroCapas−i)
```

3. Resultados prácticos

Para los resultados, tenemos cuatro bases de datos con las cuales haremos diversas pruebas y sacaremos conclusiones a partir del error y desviacion típica del error obtenido:

- **Problema XOR**: En esta base de datos donde el train y el test es el mismo. Ambos están conformados por cuatro datos que representan la salida de la puerta lógica del xor.
- **Función seno**: En esta base de datos conformada por 120 patrones de train y 41 patrones de test se intenta obtener la función seno a partir de unos datos con algo de ruido.
- **Base 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.
- **Base de datos parkinsons**: 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 (de las siglas en inglés Unified Parkinson's Disease Rating Scale).[1]

3.1. Ejemplo de salida de red neuronal

Las salidas de la red neuronal se muestran en un formato parecido al que se muestra en la figura. En este se muestran los pesos de la red, el error, la salida esperada y la obtenida por la red neuronal tras el entrenamiento.

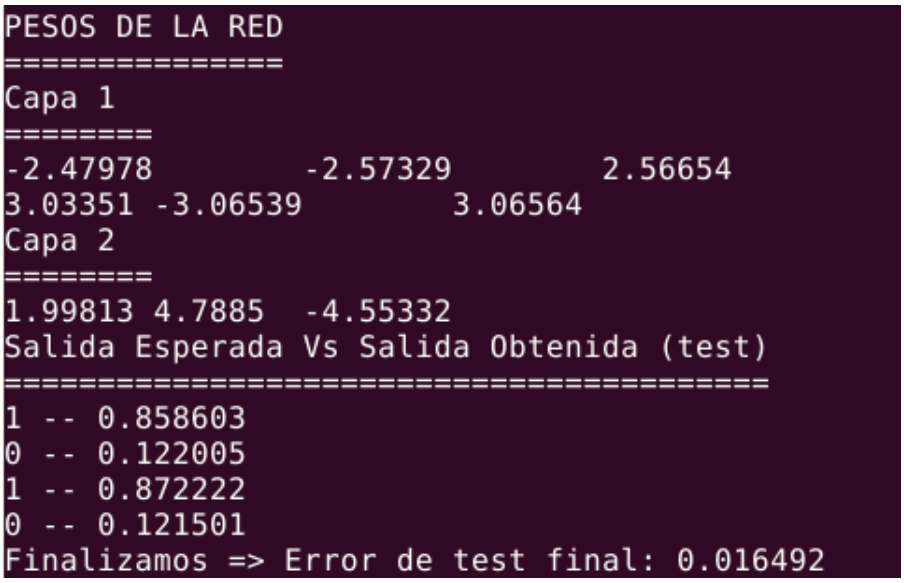

Figura 1: Salida del entrenamiento en nuestra red neuronal

Y esta sería la red neuronal que representa.

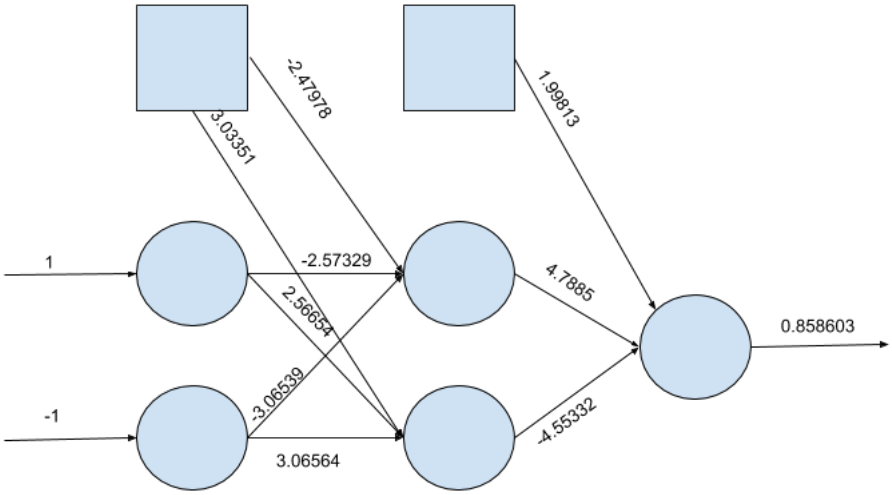

Figura 2: Red neuronal para el XOR

3.2. Prueba de las base de datos con distintas arquitecturas

La primera prueba que vamos a hacer va a ser ver el error de entrenamiento y de test con distintas arquitecturas de red. Para ello hemos usado los valores por defecto; osea, F=1, $\eta = 0{,}1$, $\mu = 0{,}9$, v=0.0 e i=1000 y hemos ido variando el numero de capas y de neuronas en capa oculta.

Como vimos anteriormente, el train y el test en la base de datos XOR es el mismo por lo que el error no varía. En este caso la mejor arquitectura es la formada por dos capas ocultas de 100 neuronas cada una.

Arquitectura	error-train	σ(error-train)	error-test	σ(error-test)
{n:2:k}	0.237245	0.024927	0.237245	0.024927
{n:4:k}	0.17623	0.0624282	0.17623	0.0624282
{n:8:k}	0.0632625	0.0348099	0.0632625	0.0348099
{n:32:k}	0.0164764	0.00273823	0.0164764	0.00273823
{n:64:k}	0.0587316	0.107693	0.0587316	0.107693
{n:100:k}	0.104873	0.1334	0.104873	0.1334
{n:2:2:k}	0.249881	0.000277139	0.249881	0.000277139
{n:4:4:k}	0.24301	0.0120244	0.24301	0.0120244
{n:8:8:k}	0.192439	0.0491734	0.192439	0.0491734
{n:32:32:k}	0.00702618	0.00107954	0.00702618	0.00107954
{n:64:64:k}	0.00304487	0.000374046	0.00304487	0.000374046
{n:100:100:k}	0.00171962	0.000203223	0.00171962	0.000203223

Tabla 1: Tabla de errores de la base de datos xor con distintas arquitecturas

En el caso de la base de datos del seno el mejor resultado sería el formado por dos capas de 64, teniendo en cuenta que aunque el error de la red neuronal formada por dos capas de 100 neuronas es menor, su desviación típica es muy alta, por lo que puede sacar resultados potencialmente peores que la de 64 neuronas, ademas de que el coste computacional se reduce.

Arquitectura	error-train	σ(error-train)	error-test	σ(error-test)
(n:2:k)	0.0297094	8.22369e-05	0.0364562	0.000309978
(n:4:k)	0.029758	6.12666e-05	0.036362	0.000239498
(n:8:k)	0.0297447	0.000401935	0.0363076	0.000406638
(n:32:k)	0.0291585	0.000409247	0.0361904	0.000930909
(n:64:k)	0.0279803	0.000137091	0.0354173	0.000314833
(n:100:k)	0.0283426	0.000778661	0.0365482	0.000252493
(n:2:2:k)	0.0297266	3.94678e-05	0.0363048	0.000210629
(n:4:4:k)	0.0297223	4.53382e-05	0.0363441	0.000285346
(n:8:8:k)	0.0299509	0.000126738	0.036167	0.000104973
(n:32:32:k)	0.0293756	0.000408474	0.0365286	0.000675989
(n:64:64:k)	0.0267517	0.000749347	0.033586	0.000751515
(n:100:100:k)	0.0225438	0.00694352	0.0305728	0.00680754

Tabla 2: Tabla de errores de la base de datos seno con distintas arquitecturas

En este caso elegimos como mejor solución la formada por una capa oculta y 100 neuronas.

Arquitectura	error-train	σ(error-train)	error-test	σ(error-test)
{n:2:k}	0.030144	6.11858e-05	0.0272387	9.799e-05
{n:4:k}	0.030003	7.52494e-05	0.0271007	7.39452e-05
{n:8:k}	0.0299017	7.70147e-05	0.0270063	4.50037e-05
{n:32:k}	0.0297689	3.46677e-05	0.0269924	2.27153e-05
{n:64:k}	0.0297746	4.95201e-05	0.027009	3.04414e-05
{n:100:k}	0.0297604	4.22055e-05	0.026995	4.42267e-05
{n:2:2:k}	0.0301755	2.17583e-05	0.0272869	2.9527e-05
{n:4:4:k}	0.0301034	7.97793e-05	0.0272091	7.38196e-05
{n:8:8:k}	0.0300488	5.13141e-05	0.0271334	6.0522e-05
{n:32:32:k}	0.029811	0.000114052	0.026973	9.09515e-05
{n:64:64:k}	0.0295418	6.72246e-05	0.0269774	5.04636e-05
{n:100:100:k}	0.0294435	5.04461e-05	0.0270882	6.06973e-05

Tabla 3: Tabla de errores de la base de datos quake con distintas arquitecturas

Por último, en este caso escogeremos la que está formada por dos capas ocultas y 100 neuronas en cada capa.

Arquitectura	error-train	σ(error-train)	error-test	σ(error-test)
{n:2:k}	0.0344364	7.98442e-05	0.0368947	0.000364991
{n:4:k}	0.0260354	0.000201014	0.0256289	8.57999e-05
{n:8:k}	0.0207091	0.000456061	0.0210962	0.000785644
{n:32:k}	0.0147497	0.00109129	0.0164725	0.000841465
{n:64:k}	0.0144767	0.000879269	0.0160682	0.000975874
{n:100:k}	0.015584	0.00305554	0.0173915	0.00185764
{n:2:2:k}	0.0312864	0.000845782	0.0327105	0.000880303
{n:4:4:k}	0.0233895	0.00188424	0.0235673	0.00161157
{n:8:8:k}	0.0164418	0.00372152	0.0173304	0.00314233
{n:32:32:k}	0.00844623	0.00174372	0.0105212	0.00226277
{n:64:64:k}	0.00561614	0.00144218	0.00792597	0.0017131
{n:100:100:k}	0.00373842	0.000896492	0.00569479	0.0012525

Tabla 4: Tabla de errores de la base de datos parkinsons con distintas arquitecturas

3.3. Pruebas variando el conjunto de validación y el factor de decremento

Tras haber obtenido lo que consideraremos son las mejores arquitecturas de la red por el momento, vamos a pasar a modificar el factor de decremento(F) y el conjunto de validación(v) y veremos los resultados que esto nos aporta.

En esta gráfica se puede ver como el F en este caso apenas varia los resultados del modelo en test. Sin embargo podemos ver como a mayor es el conjunto de validación mayor es el error de test. Eso se debe a que al seleccionar ciertos puntos para la validación se pierden datos que pueden ser relevantes.

v	F	error-train	σ(error-train)	error-test	σ(error-test)
0	1	0.0267517	0.000749347	0.033586	0.000751515
0.15	1	0.0703278	0.0102615	0.0549201	0.0133494
0.25	1	0.0576142	0.0184956	0.103991	0.059534
0	2	0.0282724	0.000268378	0.0353136	0.000456309
0.15	2	0.0697605	0.0107902	0.0534537	0.0121721
0.25	2	0.0578585	0.0183985	0.103746	0.057249

Tabla 5: Tabla de errores de la base de datos seno variando el conjunto de validación y decremento

Curiosamente, en esta tabla se puede ver como mientras menor es el error de entrenamiento, mayor es el de test. De nuevo se puede ver como el conjunto de validación empeora los datos de test.

v	F	error-train	σ(error-train)	error-test	σ(error-test)
0	1	0.0297604	4.22055e-05	0.026995	4.42267e-05
0.15	1	0.024493	0.00444711	0.0319426	0.00357464
0.25	1	0.0186005	0.00591481	0.0365678	0.00515206
0	2	0.0297521	2.50246e-05	0.026967	3.52326e-05
0.15	2	0.0244027	0.0042417	0.0313813	0.00340355
0.25	2	0.018559	0.00590074	0.0360389	0.00523062

Tabla 6: Tabla de errores de la base de datos quake variando el conjunto de validación y decremento

Aquí podemos ver como la F empeora nuestro modelo en la mayor parte

de los casos y de nuevo el como usar un conjunto de validación empeora el resultado pero también evita el sobre entrenamiento.

v	F	error-train	σ(error-train)	error-test	σ(error-test)
0	1	0.00373842	0.000896492	0.00569479	0.0012525
0.15	1	0.0760665	0.00457605	0.103	0.0314081
0.25	1	0.0665395	0.0150206	0.128992	0.0525545
0	2	0.0088328	0.00134245	0.0109749	0.00203569
0.15	2	0.0767007	0.00448053	0.0940991	0.0240841
0.25	2	0.0662583	0.015306	0.130848	0.0486523

Tabla 7: Tabla de errores de la base de datos parkinson variando el conjunto de validación y decremento

3.4. Gráficos de convergencia

Tras esto voy a mostrar la tendencia de los errores de test, train y validación a lo largo del entrenamiento y así podremos ver ejemplos de sobreentrenamiento y como las redes convergen.

Para comenzar podemos ver lo que sucede en la base de datos de seno con una validación de 0.25 y el resto de valores por defecto. Como se muestra cuando el error de test empieza a empeorar y deja de converger el error de validación el entrenamiento acabaría, siendo en este caso en la iteración 111. Esto es gracias a las condiciones de parada.

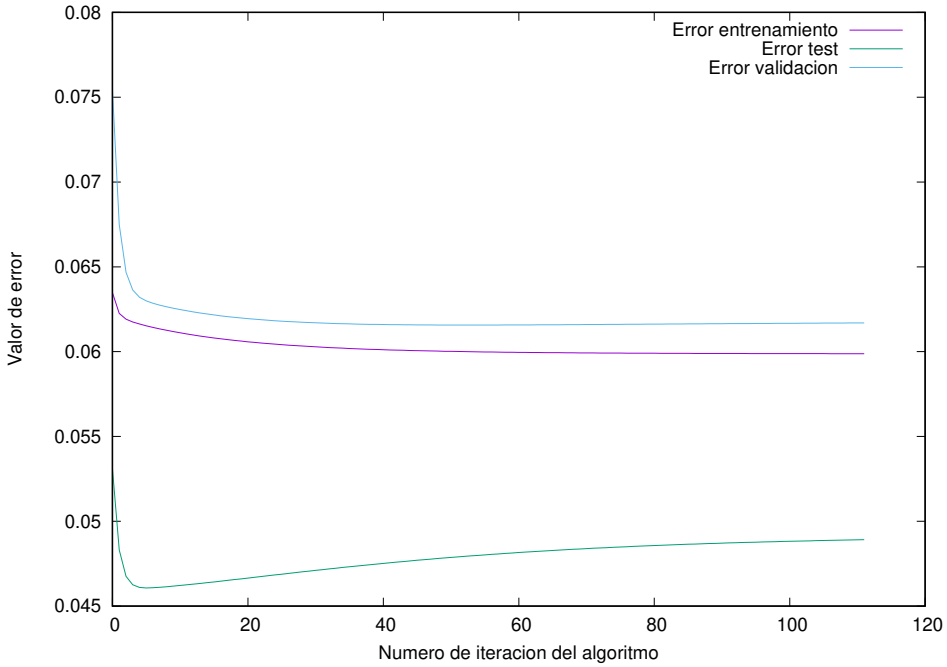

Figura 3: Errores en la base de datos del seno

Sin embargo, viendo el gráfico de arriba ¿Que pasaría si no hubiera condición alguna de parada? Para hacer esto he eliminado del código las condiciones de parada y he puesto 10000 iteraciones. Como se puede ver en la figura, el sobre entrenamiento es muy claro. Llega un momento en el que nuestra red neuronal se adapta demasiado bien a nuestro conjunto de entrenamiento y, sin embargo, muy mal a nuestro conjunto de validación y test.

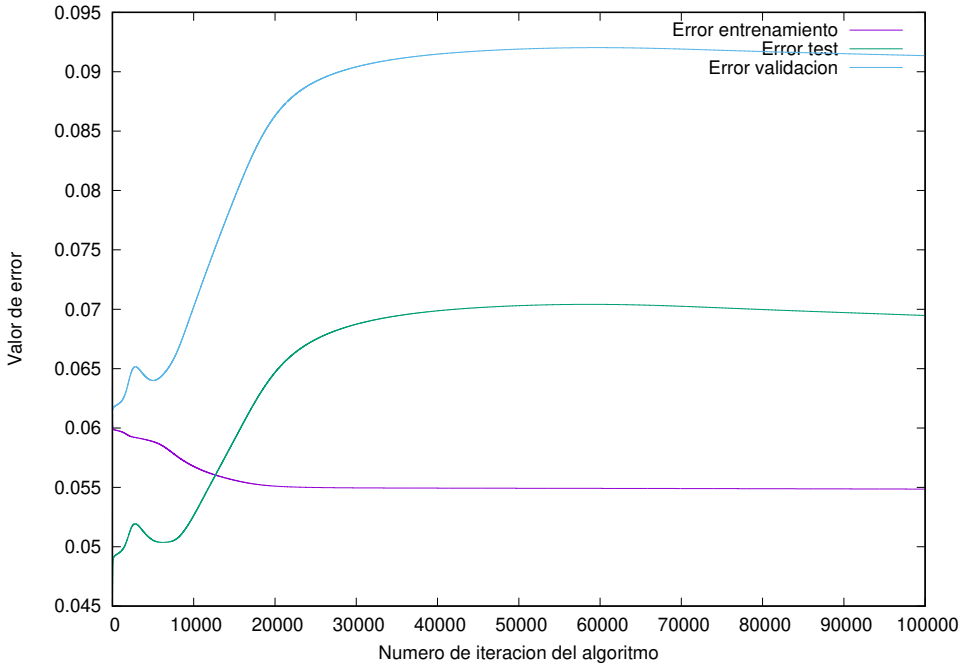

Figura 4: Sobreentrenamiento en la base de datos del seno

En este otro caso podemos ver el sobreentrenamiento en otra base de datos, la de parkinson.

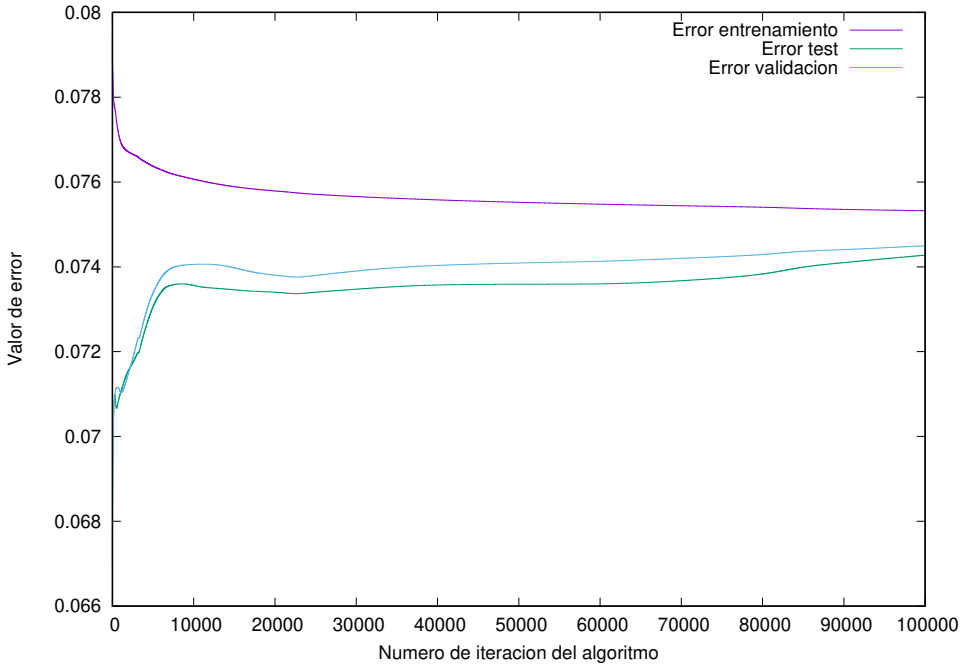

Figura 5: Sobreentrenamiento en la base de datos del parkinson

Algunas otras cosas que podemos variar son el valor de eta para comprobar como varía la convergencia con un eta alto. Para ello hemos usado la base de datos del seno con dos capas de 64 neuronas y usado un eta de 0.8.

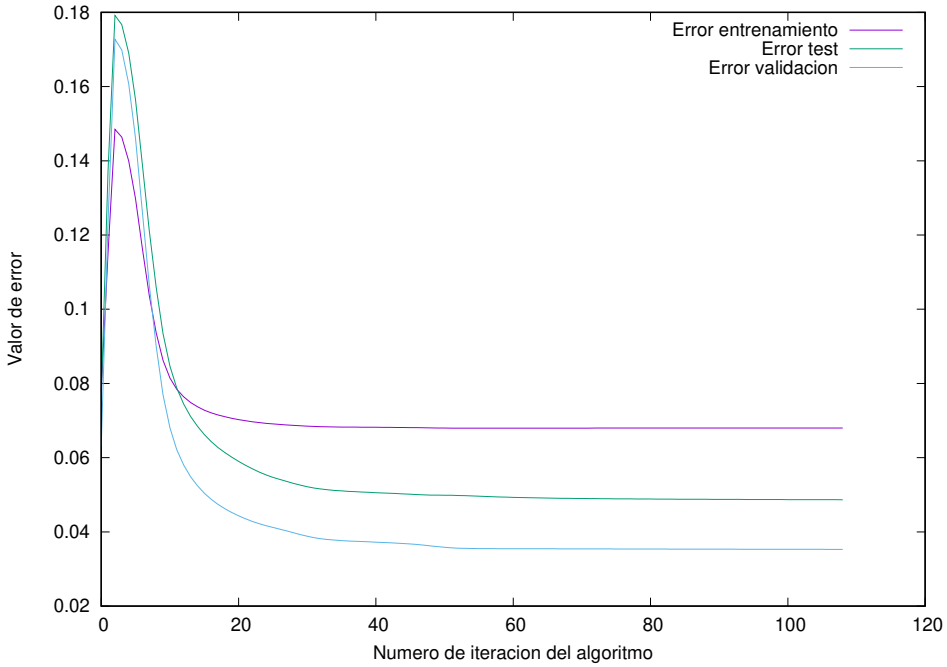

Figura 6: Entrenamiento en la base de datos del seno con eta alta

Cuando lo comparas con la gráfica con un eta bajo se nota el como la convergencia es mucho más alta cuanto más alta es la eta.

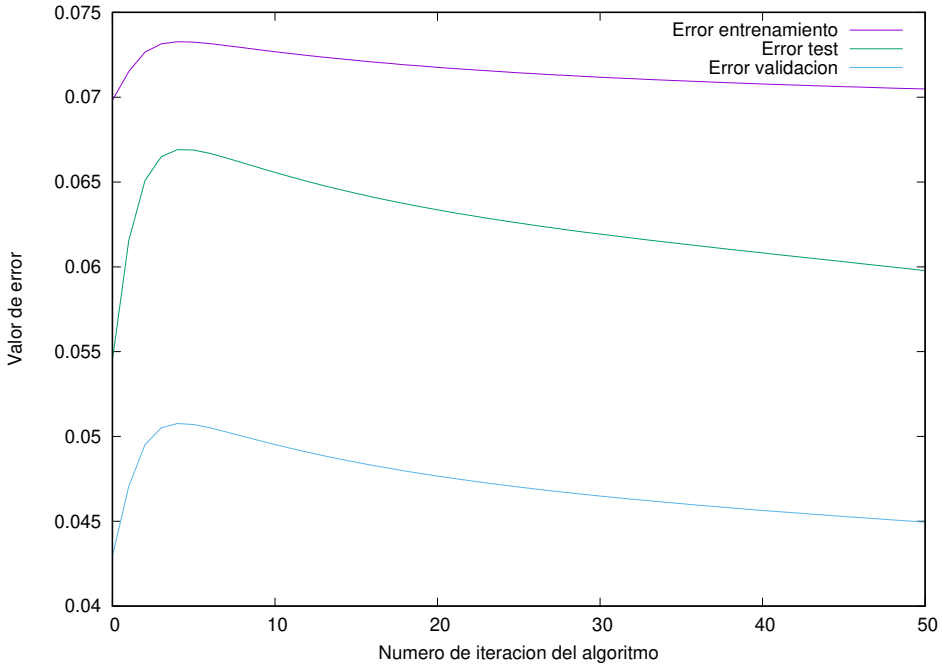

Figura 7: Entrenamiento en la base de datos del seno con eta baja

En la siguiente imagen podemos ver la convergencia en la base de datos de quake. Para ello hemos usado dos capas de 64 neuronas cada capa y un eta de 0.3 .

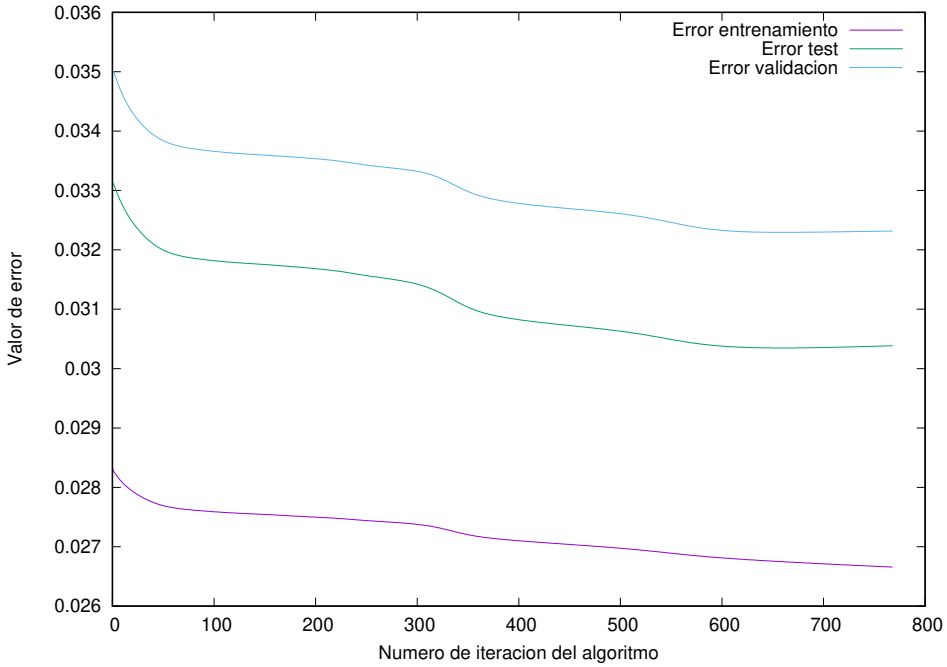

Figura 8: Entrenamiento en la base de datos quake

Tras esto he hecho la prueba con la base de datos quake y sin mu y he podido comprobar como en mi caso la diferencia es relativamente baja

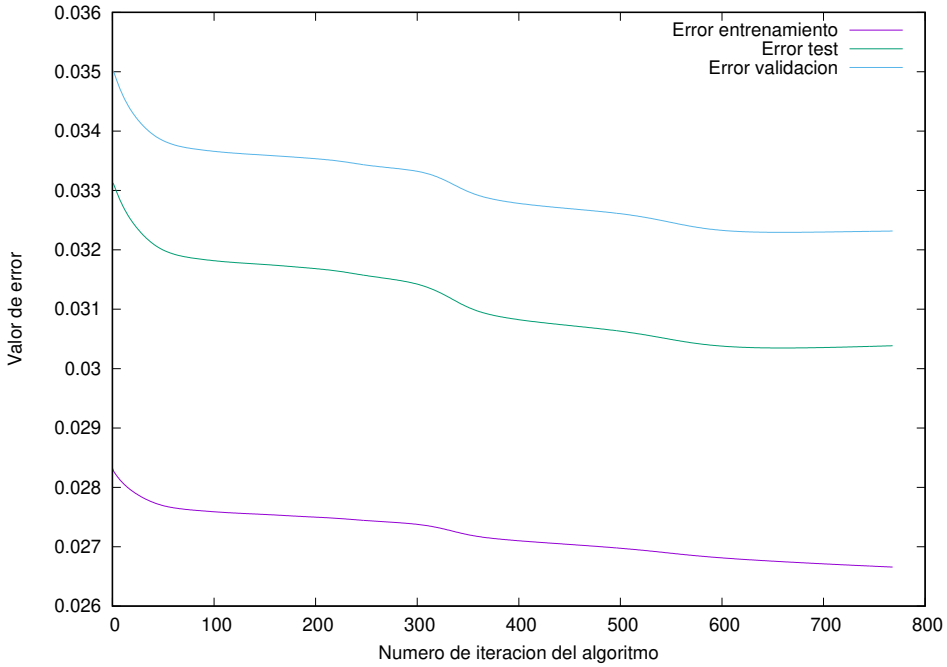

Figura 9: Entrenamiento en la base de datos quake sin mu

Por ultimo, el la siguiente gráfica se muestra la convergencia en la base de datos de parkinson.

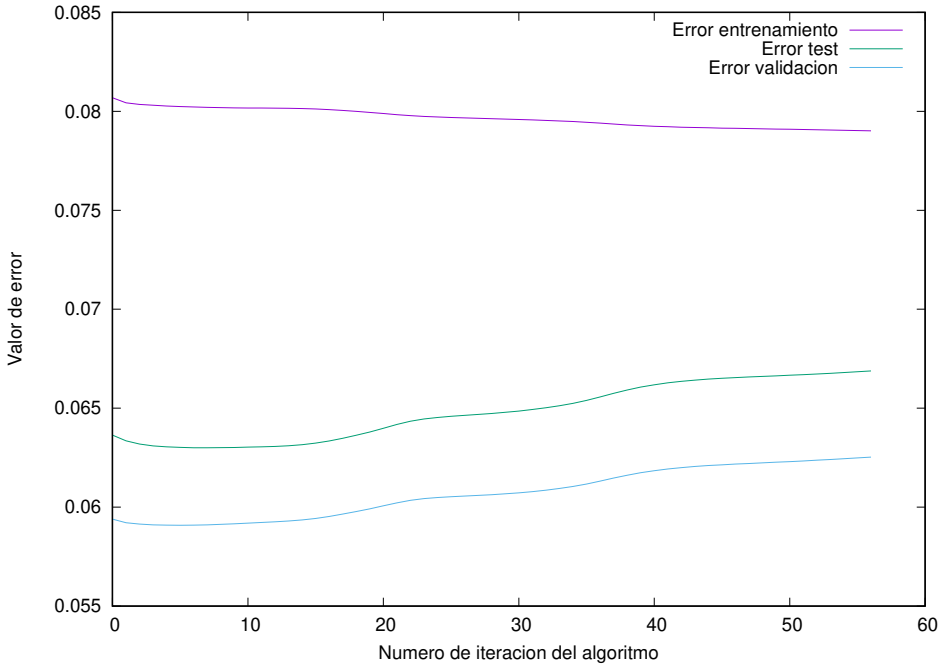

Figura 10: Entrenamiento en la base de datos parkinson.

Referencias

[1] Pedro Antonio Gutierrez Peña. *Práctica 1: Implementación del perceptrón multicapa.*