PERCEPTRÓN MULTICAPA

Práctica 1

Introducción a los Modelos Computacionales
Grado en Ingeniería Informática
Universidad de Córdoba – Escuela Politécnica Superior

Índice

Descripción del modelo neuronal

El modelo de red neuronal utilizados en esta práctica es conocido como perceptrones multicapa. Es un modelo de red neuronal formado por múltiples capas, lo que le permite resolver problemas linealmente no separables, siendo ésta la principal limitación del perceptrón. En este caso hemos utilizado un perceptrón multicapa totalmente conectado, es decir, cada salida de una neurona de una capa es entrada de todas las neuronas de la capa siguiente. Éstas capas son la de entrada, la oculta y la de salida. Dentro de la capa oculta puede haber tantas capas como se deseen.

Para que un perpceptrón sea capaz de resolver los problemas linealmente no separables se inicia con unos pesos aleatorios, con un número determinado de capas y un número determinado de neuronas por capa. Cada neurona está conectada a otra u otras de la capa siguiente. Dicha conexión indica que la salida de esa neurona va a influir de alguna manera en la salida de la neurona a la que está conectada. Esa influencia es conocida como **peso**, que es un valor que pondera la salida de la neurona para la neurona a la que está conectada.

Para obtener las salidas en las diferentes neuronas se usa una función de transferencia, en este caso hemos usado una sigmoide. Las funciones de transferencia han de ser siempre derivables. Éstas indican cómo se calcula la salida de las neuronas teniendo en cuenta el valor y los pesos de las neuronas de la capa anterior que están conectadas con ellas.

Una vez que se han propagado las entradas, es decir, todas las neuronas tienen sus valores de salida y éstos se han ido calculando hasta llegar a la capa de salida, se compara la salida obtenida con la salida deseada. El resultado de esta comparación es el error que se ha cometido en la estimación. Para ir disminuyendo este error a lo largo de las iteraciones del entrenamiento se calcula la derivada de la función de transferencia para cada una de las neuronas de la red con respecto a los pesos que la componen, exceptuando las entradas (no tienen pesos). A esta operación se le conoce como **retropropagación del error.**

Luego se calcula el cambio que se va a realizar en cada uno de los pesos de la red multiplicando la derivada de la salida de la neurona por la salida de cada una de las neuronas con la que está conectada de la capa anterior. Una vez calculado el cambio de cada peso se aplica restando dicho cambio ponderado al valor actual del peso. Dicha ponderación es dada por el usuario y es la tasa de aprendizaje de la red, si el valor es cercano a 1, se está haciendo búsqueda en anchura, si es cercano a 0, en profundidad. En este caso también usamos un factor de momento al aplicar el cambio en los pesos, que tiene en cuenta el último cambio realizado en ese peso. Si el valor del momento es cercano a 1, este cambio se tiene muy en cuenta, si el valor es cercano a 0 el último cambio apenas tiene influencia. A la hora de ajustar los pesos se puede hacer de dos formas: *online* u *offline*.

- **Online:** los pesos se modifican por cada patrón que pasa de la iteración.
- **Offline:** los pesos se modifican cuando han pasado todos los patrones de la iteración.

Una vez que ha finalizado el entrenamiento se le pasan los patrones del conjunto de test, con los que ya no se realiza ningún cambio en la red. Si el la función de error tiene mínimos locales, el perceptrón se puede quedar atrapado en alguno. Si esto pasa habría que reconsiderar la estructura de la red, el valor inicial de los pesos o el orden en el que se introducen los patrones.

Pseudocódigos

En este apartado se muestran los pseudocódigos de las funciones más relevantes del código implementado. Las funciones seleccionadas han sido las que se encargan de simular la red online, alimentar las entradas, propagar esas entradas, retropropagar el error cometido en la estimación, acumular el cambio que se le va a aplicar a los pesos y ajustar los pesos.

SIMULAR RED ONLINE

Para 1 (i) hasta número de capas
    Para 1 (j) hasta número de neuronas de la capa i
        Para 1 (k) hasta número de neuronas de la capa i-1
            Δw_{jk}^{i} (t-1) ←Δw_{jk}^{i}
            Δw_{jk}^{i} ←0
        finPara
    finPara
finPara
alimentarEntradas(entrada)
propagarEntradas()
retropropagarError(objetivo)
acumularCambio()
ajustarPesos()

ALIMENTAR ENTRADAS

Para 1 (i) hasta número de neuronas de la capa de entrada
    x_i^0←input[i]
finPara

PROPAGAR ENTRADAS

Para 1 (i) hasta número de capas
    Para 1 (j) hasta número de neuronas de la capa i
        Para 1 (k) hasta número de neuronas de la capa i-1
            $x_j^i \leftarrow x_j^i + x_k^i * w_{jk}^i$
        finPara
        $x_j^i \leftarrow x_j^i + x_k^i * w_{jn}^i$ (n es el número de neuronas de la capa anterior)
        $x_j^i \leftarrow x_j^i * -1$
        $x_j^i \leftarrow \frac{1}{1+e^{x_j^i}}$
    finPara
finPara

RETROPROPAGAR ERROR

Para 1 (i) hasta número de neuronas de la capa de salida
$\delta_i^n \leftarrow$-(objetivo[i]- x_i^n)* x_i^n*(1-x_i^n) (n es el número de neuronas de la última capa)
finPara
Para número de capas -2 (i) hasta 0 (decremento)
Para 1 (j) hasta número de neuronas de la capa i
//Pesos
Para 1 (k) hasta número de neuronas de la capa i+1
$\delta_j^i \leftarrow \delta_j^i + \Delta w_{kj}^{i+1} * \delta_j^{i+1}$
finPara
//Sesgo
for(int k=0; k<pCapas[i+1].nNumNeuronas;k++){
$\delta_j^i \leftarrow \delta_j^i + \Delta w_{kn}^{i+1} * \delta_j^{i+1}$ (n es el número de neuronas de esta capa)
finPara
$\delta_j^i \leftarrow \delta_j^i * x_j^i * (1 - x_j^i)$
finPara
finPara

ACUMULAR CAMBIO

Para 1 (i) hasta número de capas
Para 1 (j) hasta número de neuronas de la capa i
Para 1 (k) hasta número de neuronas de la capa i-1
$\Delta w_{jk}^i \leftarrow \Delta w_{jk}^i + \delta_j^i * x_j^i$
finPara
$\Delta w_{jn}^i \leftarrow \Delta w_{jn}^i + \delta_j^i * bSesgo$ (n es el número de neuronas de la capa i-1)
finPara
finPara

AJUSTAR PESOS

Para 1 (i) hasta número de capas
Para 1 (j) hasta número de neuronas de la capa i
Para 1 (k) hasta número de neuronas de la capa i-1
$w_{jk}^i \leftarrow w_{jk}^i - \eta * \Delta w_{jk}^i - \mu * (\eta * \Delta w_{jk}^i(t-1))$
finPara
$w_{jn}^i \leftarrow w_{jn}^i - \eta * \Delta w_{jn}^i - \mu * (\eta * \Delta w_{jn}^i(t-1))$ (n es el número de neuronas de la capa i-1)
finPara
finPara

Experimentos y análisis de los resultados

Descripción de las bases de datos

Todos los ficheros que contienen las bases de datos tienen el mismo formato:

- En la primera línea se encuentran 3 enteros que indican el número total de entradas del problema(*n*), el número total de salidas del problema(*k*) y el número total de patrones en dicho fichero(*p*).

- A partir de la segunda línea, cada línea corresponde a un patrón:
  - Los primeros *n* valores corresponden a las entradas.
  - Los siguientes *k* valores son las salidas deseadas.

Descripción de los parámetros considerados

Los parámetros que se han considerado en esta práctica para el estudio del comportamiento del perceptrón son los siguientes:

- Nº de iteraciones: Es un entero que indica el número máximo de iteraciones que se van a realizar para uno de los patrones de la base de datos. El número total de iteraciones de una ejecución del programa será 1000**p.*
- Nº de capas ocultas: Es un entero que indica el número de capas ocultas de la red neuronal. Mientras mayor sea el número, más compleja será la red.
- Nº de neuronas en capa oculta: Es un entero que indica el número de neuronas que tendrá cada una de las capas ocultas. Mientras mayor sea el número, más compleja será la red.
- Valor de *eta*(η): Es un valor real entre 0 y 1 que indica el factor de aprendizaje del modelo.
- Valor de *mu*(μ): Es un valor real entre 0 y 1 que indica el factor de momento del modelo.
- Sesgo: Es un booleano que indica si el modelo tendrá o no sesgo.

Resultados obtenidos

En este apartado se explicarán los diferentes experimentos que se han realizado con el programa desarrollado y sus resultados, con el fin de estimar la configuración de los parámetros óptima para cada uno de los problemas, es decir, encontrar la serie de parámetros con la que obtendremos los mejores resultados posibles. Se harán experimentos con la arquitectura de la red neuronal utilizada, con la activación y desactivación del sesgo y del momento y con el cambio en el factor de aprendizaje. Finalmente se mostrará la configuración con mejores resultados que será la que se analizará en el apartado siguiente.

Comparación de arquitecturas

En este apartado se muestran los resultados de las pruebas realizadas con diferentes arquitecturas del perceptrón multicapa, con el fin de averiguar cuál es la más apropiada para cada uno de los problemas planteados. Se han probado las mismas arquitecturas para cada uno de los problemas, recogiendo los valores medios y la desviación típica del MSE del entrenamiento y del test. Estos valores se han calculado con el MSE obtenido con cada una de las semillas utilizadas en el programa (10, 20, 30, 40 y 50). Todas se han realizado teniendo activado el sesgo, con un número de iteraciones de 1000 y η = 0.1 μ = 0.9.

XOR

	Entrenamiento		Test	
Estructura	Media	Desv	Media	Desv
{n : 2 : k}	0,140759	0,0541244	0,140893	0,0541884
{n : 5 : k}	0,0182761	0,0114866	0,0182786	0,0114888
{n : 10 : k}	0,00941709	0,00113589	0,00941824	0,00113619
{n : 25 : k}	0,0050031	0,00034659	0,00500001	0,00034718
{n : 50 : k}	0,00391411	0,00029648	0,0039094	0,00029746
{n : 100 : k}	0,101915	0,135607	0,101912	0,135604
{n : 2 : 2 : k}	0,24649	0,00445858	0,246535	0,00440771
{n : 5 : 5 : k}	0,223066	0,0205838	0,223126	0,020505
{n : 10 : 10 : k}	0,0727347	0,0841001	0,072695	0,0840522
{n : 25 : 25 : k}	0,00337156	0,00046524	0,00337083	0,0004671
{n : 50 : 50 : k}	0,0018466	0,00034192	0,00184349	0,00034229
{n : 100 : 100 : k}	0,0009562	3,3703E-05	0,00095043	3,24E-05

En la tabla se puede apreciar que el modelo nunca sobrentrena para este problema en concreto. Esto se debe a que la base de datos utilizada para entrenamiento y test es la misma, así como que ésta tiene un tamaño muy reducido (sólo hay cuatro posibles combinaciones de entradas y cuatro únicas salidas). Si la red es más compleja, el modelo va a ser más preciso, lo que tiene sentido en este caso debido a las mismas razones por las que no sobrentrena. La mejor estructura en este caso sería la última configuración, ya que el error de salida no llega a la milésima. Al ser un problema simple, aunque la estructura sea la más compleja, se puede asumir el coste computacional que conlleva, por lo que se escogerá ésta.

CPU

	Entrenamiento		Test	
Estructura	Media	Desv	Media	Desv
{n : 2 : k}	0,00144329	0,00019592	0,00527275	0,00134697
{n : 5 : k}	0,00106592	6,07E-05	0,00572958	0,000912973
{n : 10 : k}	0,00108108	3,27E-05	0,00469538	0,000636367
{n : 25 : k}	0,00115335	8,45E-05	0,00568541	0,000482257
{n : 50 : k}	0,0011998	6,17E-05	0,00654248	0,000552143
{n : 100 : k}	0,00126497	0,00015554	0,00617595	0,000410995
{n : 2 : 2 : k}	0,00578616	0,00753586	0,00927035	0,00682025
{n : 5 : 5 : k}	0,00159354	0,00020689	0,00533122	0,000267262
{n : 10 : 10 : k}	0,0015083	0,00010352	0,00564723	0,00100907
{n : 25 : 25 : k}	0,00123789	0,00014499	0,00570728	0,000640275
{n : 50 : 50 : k}	0,00122513	0,00014559	0,00628834	0,00027812
{n : 100 : 100 : k}	0,00114371	0,00023251	0,00760334	1,17E-03

En este caso, a diferencia del anterior, la red sí llega a sobrentrenar, ya que el error en el entrenamiento es menor que en el test. Esto significa que a partir de una iteración el modelo empieza a fallar en la clasificación del conjunto de test. Para evitar este sobrentrenamiento hay que realizar una gráfica para ver en qué iteración sucede esto. Una vez se sepa este valor, realizar el entrenamiento un número de iteraciones igual a éste, así el modelo nunca sobrentrenará. Todas las estructuras obtienen aproximadamente el mismo error de entrenamiento, pero a medida que la red es más compleja, el error de test es mayor, es decir, sobrentrena mucho más. Esto se debe a la complejidad de la red, ya que tiene muchos más parámetros que controlar y la base de datos del problema es bastante más compleja que la del XOR. Por esta razón, y por coste computacional, se escogerá la opción de una sola capa oculta con 10 neuronas, ya que posee el error de test más pequeño y el error de entrenamiento es el segundo mejor.

SIN

	Entrenamiento		Test	
Estructura	Media	Desv	Media	Desv
{n : 2 : k}	0,0301277	0,00019692	0,0360185	0,00014587
{n : 5 : k}	0,0293023	0,00132386	0,0341265	0,0027073
{n : 10 : k}	0,0302702	0,00015985	0,0368056	0,00060438
{n : 25 : k}	0,028364	0,00079199	0,0326455	0,00247829
{n : 50 : k}	0,0273758	0,00012071	0,0308952	0,00044932
{n : 100 : k}	0,0290203	0,00050762	0,03461	0,00099997
{n : 2 : 2 : k}	0,0301238	0,00022335	0,0360113	5,64E-05
{n : 5 : 5 : k}	0,0302764	0,00029845	0,03609	0,00016755
{n : 10 : 10 : k}	0,0303356	0,00030496	0,0360203	0,00037593
{n : 25 : 25 : k}	0,0298775	0,00071136	0,0362823	0,00048594
{n : 50 : 50 : k}	0,0275993	0,0024728	0,0313275	0,00465713
{n : 100 : 100 : k}	0,0270017	0,00145281	0,0362746	3,46E-03

En esta ocasión el modelo llega a sobrentrenar ligeramente, a diferencia del anterior. La base de datos es algo más simple que el problema anterior, aunque más compleja que el primero, por lo que se obtienen dichos resultados. La tendencia es la misma que en el anterior problema: a mayor complejidad, mejores resultados de entrenamiento, pero mayor sobrentrenamiento. En este caso se encuentra una estructura que obtiene el mejor resultado tanto en entrenamiento como en test, por lo que se escogerá esta configuración para continuar con los experimentos, teniendo una sola capa oculta y cincuenta neuronas en dicha capa.

Sesgo: activado y desactivado

En este apartado se mostrarán los resultados de las pruebas realizadas con las estructuras seleccionadas en el apartado anterior, activando y desactivando el sesgo. Finalmente se decidirá si se deja activado o desactivado el sesgo para las pruebas finales y análisis.

XOR

	Entrenamiento		Test	
Sesgo	Media	Desv	Media	Desv
Activado	0,0009562	3,3703E-05	0,00095043	3,24E-05
Desactivado	0,00106283	0,00014157	0,00105566	0,00014095

CPU

	Entrenamiento		Test	
Sesgo	Media	Desv	Media	Desv
Activado	0,00108108	3,27E-05	0,00469538	0,00063637
Desactivado	0,00120913	6,74E-05	0,00523122	0,00045374

SIN

	Entrenamiento		Test	
Sesgo	Media	Desv	Media	Desv
Activado	0,0273758	0,00012071	0,0308952	0,00044932
Desactivado	0,0326952	0,00153912	0,0414376	0,00235324

Como se puede observar, en todos los problemas, la red que tiene el sesgo activado produce un error mucho menor que el que lo tiene desactivado. Es lógico que ocurra esto, ya que el sesgo es un ruido que se le introduce a la red para que vaya evolucionando a la vez que ella y vaya anulando el ruido que se va produciendo en la misma red, o simplemente porque el tener este parámetro de más hace que la red sea más precisa.

Momento: activado y desactivado

En este apartado se van a exponer los resultados de los experimentos activando y desactivando el factor de momento. El momento, como se explicó anteriormente, indica en qué medida el peso de los sesgos va a seguir la trayectoria de cambio que ha seguido hasta entonces, de forma que, si se queda en un óptimo local, pero empieza a salir, pueda llegar a salir de él y así poder llegar al óptimo global. Se probarán además diferentes valores para el momento activado, siendo el 0 cuando está desactivado.

XOR

	Entrenamiento		Test	
Momento	Media	Desv	Media	Desv
0	0,00221115	0,00010954	0,00220922	0,00010781
0,1	0,00194263	8,72E-05	0,00193914	8,62E-05
0,5	0,00128621	4,92E-05	0,00128256	4,75E-05
0,9	0,0009562	3,3703E-05	0,00095043	3,24E-05

CPU

	Entrenamiento		Test	
Momento	Media	Desv	Media	Desv
0	0,00128306	6,42E-05	0,00423553	0,00029264
0,1	0,00124879	5,40E-05	0,00431142	0,00032937
0,5	0,00115536	3,69E-05	0,00453332	0,00050061
0,9	0,00108108	3,27E-05	0,00469538	0,00063637

SIN

	Entrenamiento		Test	
Momento	Media	Desv	Media	Desv
0	0,0275703	0,00014541	0,0318687	0,00038092
0,1	0,0274997	0,00011897	0,0316899	0,00040209
0,5	0,0284044	0,00229599	0,0333514	0,00502449
0,9	0,0273758	0,00012071	0,0308952	0,00044932

En las tablas se puede observar que mientras mayor es el factor de momento, es decir, mientras más se tiene en cuenta la trayectoria del cambio de los pesos, mejores son los resultados. En el caso del XOR, en un espacio de soluciones mucho menor que en el del problema del seno, las mejoras son sustanciales, mientras que, en este último, la mejoría se nota mucho más en el test, que indica que mientras mayor sea el factor de momento, menos va a sobrentrenar el sistema. Algo que no ocurre con el test del CPU. Es decir, en todos tiende a mejorar en el entrenamiento, pues en la fase en la que se usa el factor de momento, pero no tiene porqué mejorar el error del test.

Mejor configuración

XOR

- Nº de iteraciones: 1000
- Nº de capas ocultas: 2
- Nº de neuronas en capa oculta: 100
- Valor de *eta*(η): 0.1
- Valor de *mu*(μ): 0.9
- Sesgo: Activado

CPU

- Nº de iteraciones: 1000
- Nº de capas ocultas: 1
- Nº de neuronas en capa oculta: 50
- Valor de *eta*(η): 0.1
- Valor de *mu*(μ): 0.9
- Sesgo: Activado

SIN

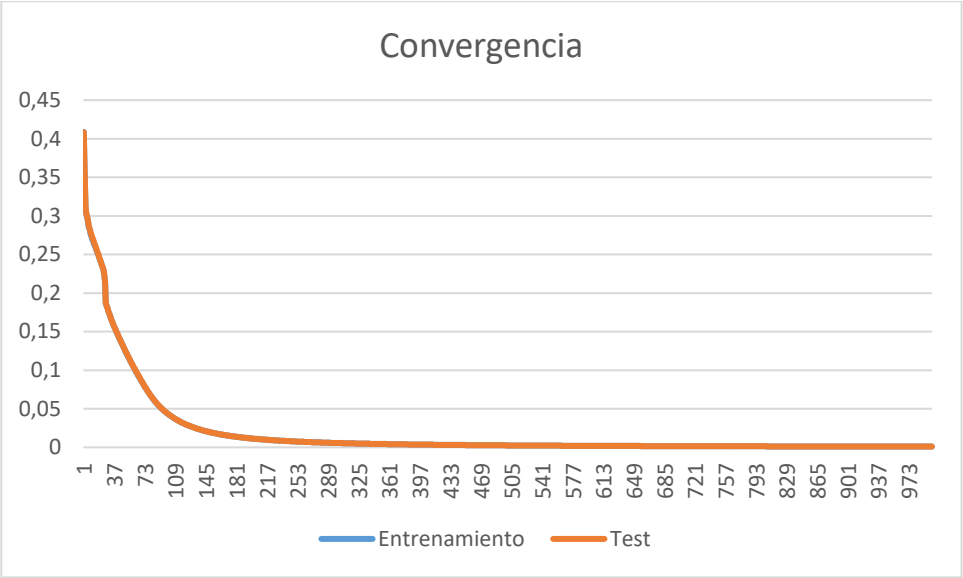
- Nº de iteraciones: 1000
- Nº de capas ocultas: 1
- Nº de neuronas en capa oculta: 10
- Valor de *eta*(η): 0.1
- Valor de *mu*(μ): 0.9
- Sesgo: Activado

Análisis de los resultados

En este último apartado se mostrarán las gráficas de convergencia del perceptrón multicapa para cada uno de los problemas con los valores de los parámetros acordados anteriormente. En las gráficas se representarán en el eje de abcisas el número de iteraciones y en el de ordenadas el error cometido en la estimación. El error cometido en entrenamiento se representa con la línea azul y el cometido por el de test se representa por una línea naranja.

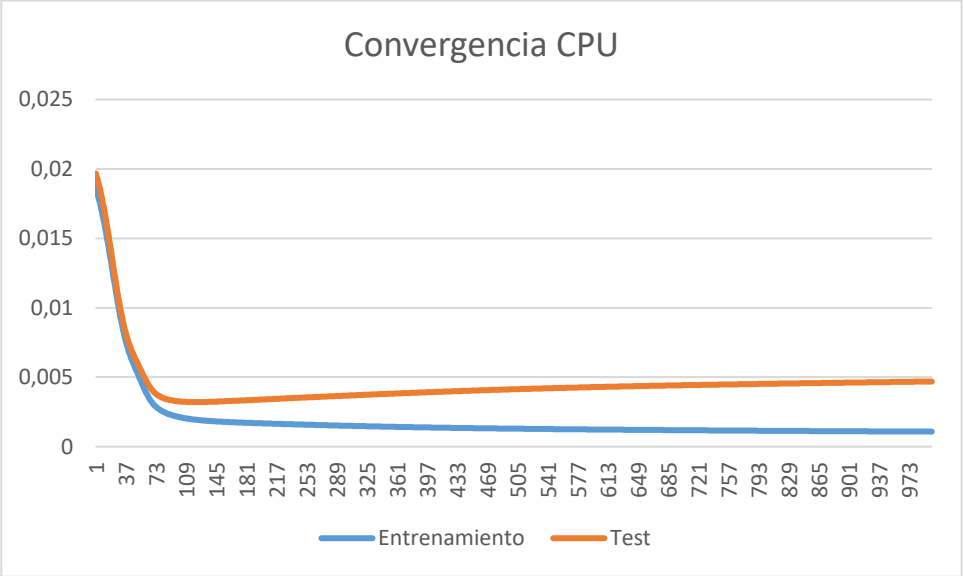
Gráficas de convergencia

XOR



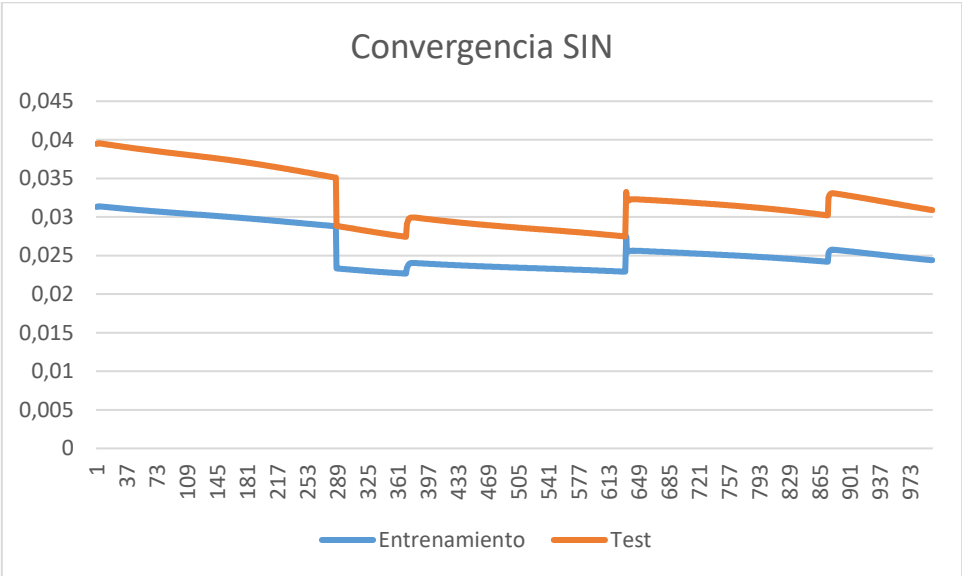
En este caso, como ya se determinó anteriormente, no se produce sobrentrenamiento por ser un problema con tan sólo 4 posibles entradas y 4 posibles salidas, algo que se puede apreciar claramente en la gráfica, donde la línea que marca el error de test está justo encima de la línea que marca el error de entrenamiento. Se podría concluir que, debido a la simpleza del problema, éste nunca llegaría a sobrentrenar, por muy compleja que fuera la red neuronal utilizada. Si bien tampoco hace falta una red demasiado compleja, ya que con redes más simples se consiguen resultados bastante buenos, en los que, si se fija un umbral para cada una de las posibles salidas, podría simular perfectamente el comportamiento de una puerta XOR (ej: toda salida por debajo de 0,05 es un 0 y toda salida por encima de 0,95 es un 1).

CPU



Con el problema del CPU sí encontramos un sobrentrenamiento, concretamente a partir de la iteración 90 aproximadamente, en la que el error de test comienza a aumentar en lugar de disminuir, al contrario que el error de entrenamiento, que siempre está descendiendo. Esto significa que si en la fase de entrenamiento se realizan unas 90 iteraciones en lugar de las 1000 actuales, se obtendría un error de test menor que el que se obtiene ahora. Por lo que el número de iteraciones ideal para este problema es 90.

SIN



Al contrario de como se vaticinó anteriormente, en esta ocasión no se produce sobrentrenamiento. Esto se puede saber porque se aprecia en la gráfica que cuando el error de entrenamiento aumenta o disminuye, el error de test hace exactamente lo mismo, manteniendo la diferencia entre ambos aproximadamente. Se podría cortar el entrenamiento en la iteración 390 aproximadamente, ya que es en la que se consigue el error más bajo de ambos errores.

Bibliografía

- Apuntes, transparencias y material referente a esta práctica de la asignatura en la plataforma Moodle.