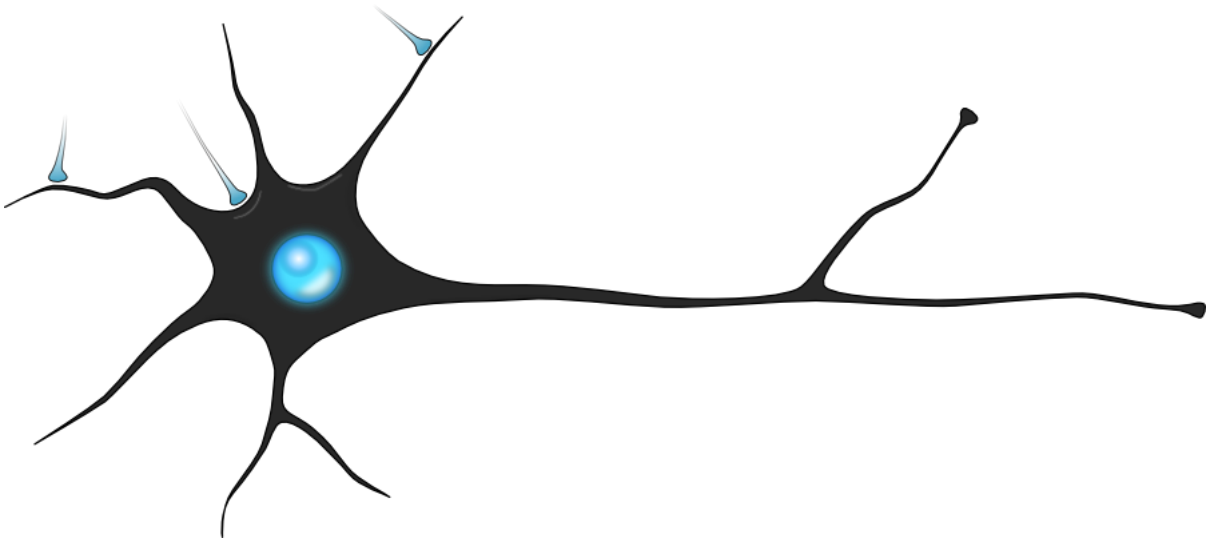
UNIVERSIDAD DE CÓRDOBA
Escuela Politécnica Superior
Grado en Ingeniería Informática
Introducción a los Modelos Computacionales
Curso académico 2016-2017
Víctor Monserrat Villatoro

PRÁCTICA 1:
IMPLEMENTACIÓN DEL **PERCEPTRÓN MULTICAPA**.



Víctor Monserrat Villatoro
i32moviv@uco.es
45887876R

1. Descripción de los modelos de redes neuronales utilizados.

Una red neuronal prealimentada es una red neuronal artificial (RNA) donde las conexiones no son cíclicas.

El perceptrón multicapa es una clase de red neuronal prealimentada que consta de múltiples capas de unidades computacionales, normalmente interconectados de forma prealimentada. Esto le permite resolver problemas que no son linealmente separables, lo cual es la principal limitación del perceptrón simple.

El perceptrón multicapa puede ser totalmente o localmente conectado, en nuestro caso estará totalmente conectado. Cada capa está formada por una serie de neuronas que tienen conexiones dirigidas con todas las neuronas de la capa siguiente. Todo perceptrón multicapa está compuesto y en este orden por las siguientes capas:

- Capa de entrada: sólo se encarga de recibir las señales de entrada y propagarla a la siguiente capa. Está constituida por aquellas neuronas que introducen los patrones de entrada en la red. En estas neuronas no se produce procesamiento.

- Capas ocultas: realizan un procesamiento no lineal de los datos de entrada. Están formadas por aquellas neuronas cuyas entradas provienen de capas anteriores y cuyas salidas pasan a neuronas de capas posteriores.

- Capa de salida: proporciona al exterior la respuesta de la red para cada patrón de entrada. Los valores de las neuronas de salida se corresponden con las salidas de toda la red.

En el perceptrón multicapa, la propagación de los patrones de entrada define una relación entre las variables de entrada y variables de salida de la red. Esta relación se obtiene propagando hacia delante los valores de entrada. Cada neurona de la red procesa la información recibida por sus entradas y produce una respuesta o activación que se propaga, a través de las conexiones correspondientes, a las neuronas de la siguiente capa.

2. Descripción en pseudocódigo del algoritmo de retropropagación.

El algoritmo de retropropagación aplicado en una red neuronal prealimentada, concretamente de la clase perceptrón multicapa, se divide en tres fases.

2.1. Fase 1: propagación hacia delante.

```
c ← 2
Mientras c <= numeroCapas
     Para Cada neurona n ∈ capa c Hacer
          net ← 0
          Para Cada neurona m ∈ capa c-1 Hacer
               net ← net + peso_mn · salida_m
          Si sesgo = verdadero
               net ← net + peso_m+1n
          salida_n ← 1 ÷ 1+e^-net
     c ← c + 1
```

2.2. Fase 2: propagación hacia atrás.

```
c ← numeroCapas
Para Cada neurona n ∈ capa c Hacer
     δ_n ← -1·(objetivos_n-salida_n)·salida_n·(1-salida_n)
c ← numeroCapas-1
Mientras c > 0
     Para Cada neurona n ∈ capa c Hacer
          s ← 0
          Para Cada neurona m ∈ capa c+1 Hacer
               s ← s + peso_nm · δ_m
          δ_n ← s · salida_n · (1 - salida_n)
     c ← c - 1
```

2.3. Fase 3: actualización de pesos.

```
c ← 2
Mientras c <= numeroCapas
     Para Cada neurona n ∈ capa c Hacer
          Para Cada neurona m ∈ capa c-1 Hacer
               peso_mn ←
peso_mn - η · Δpeso_mn - μ · ( η · uΔpeso_mn )
          Si sesgo = verdadero
               m ← m + 1
               peso_mn ←
peso_mn - η · Δpeso_mn - μ · ( η · uΔpeso_mn )
     c ← c + 1
```

Existen dos tipos de algoritmos de retropropagación (on-line y off-line), analizaremos el implementado, el algoritmo de retropropagación on-line.

2.4. Retropropagación on-line.

```
inicializar()
p ← 1
Mientras p <= numeroPatrones
     c ← 2
     Mientras c <= numeroCapas
          Para Cada neurona n ∈ capa c Hacer
               Para Cada neurona m ∈ capa c-1 Hacer
               uΔpeso_mn ← Δpeso_mn
               Δpeso_mn ← 0
          c ← c + 1
     alimentarEntradas(entradas_p)
     propagarHaciaDelante()
     propagarHaciaAtrás(objetivos_p)
     calcularAjustePesos()
     actualizarPesos()
     p ← p + 1
```

3. Descripción en pseudocódigo de operaciones relevantes.

Estas son algunas de las operaciones más relevantes para la realización de la retropropagación.

3.1. Inicialización

```
c ← 2
Mientras c <= numeroCapas
     Para Cada neurona n ∈ capa c Hacer
          Para Cada neurona m ∈ capa c-1 Hacer
               peso_mn ← realAleatorio(-1, 1)
          Si sesgo = verdadero
               peso_m+1n ← realAleatorio(-1, 1)
     c ← c + 1
```

3.2. Alimento de entradas.

```
c ← 1
Para Cada neurona n ∈ capa c Hacer
     salida_n ← entradas_n
```

3.3. Cálculo de ajuste de pesos.

```
c ← 2
Mientras c <= numeroCapas
     Para Cada neurona n ∈ capa c Hacer
          Para Cada neurona m ∈ capa c-1 Hacer
               Δpeso_mn ← Δpeso_mn + δ_n · salida_m
          Si sesgo = verdadero
               Δpeso_m+1n ← Δpeso_m+1n + δ_n
     c ← c + 1
```

La operación realAleatorio(low, high) generará un número real aleatorio del intervalo [low, high].

4. Experimentos y análisis de resultados.

4.1. Descripción de las bases de datos utilizadas.

4.1.1. Problema XOR.

Esta base de datos representa el problema de clasificación no lineal del XOR. Está compuesta por 4 patrones, que se utilizaran tanto para entrenamiento como para test. Tiene 2 variables independientes, las entradas, y una dependiente, la salida.

4.1.2. Función seno.

Esta base de datos está compuesta por 120 patrones de entrenamiento y 41 patrones de test. Ha sido obtenido añadiendo cierto ruido aleatorio a la función seno. Tiene una variable independiente y otra dependiente.

4.1.3. Base de datos CPU.

Esta base de datos está compuesta por 109 patrones de entrenamiento y 100 patrones de test. Contiene, como entradas o variables independientes, MYCT, MMIN, MMAX, CACH, CHMIN, CHMAX, y, como salida o variable dependiente, el rendimiento relativo del procesador (PRP).

4.1.4. Base de datos forest.

Esta base de datos está compuesta por 387 patrones de entrenamiento y 130 patrones de test. Contiene, como entradas o variables independientes, una serie de datos meteorológicos y otras variables descriptoras (27) sobre incendios en bosques al norte de Portugal, y, como salida o variable dependiente, el área quemada.

4.2. Descripción de los valores de los parámetros considerados.

Para la la topología de la red neuronal se han considerado los siguientes valores.

- Número de capas ocultas. Se generarán redes de una y dos capas ocultas.

- Número de neuronas por capa oculta. Se van a generar cada una de las configuraciones anteriores con 2, 5, 10, 50 y 100 neuronas.

La capa de entrada tendrá tantas neuronas como entradas hay en la base de datos y la de salida como salidas haya.

La actualización de pesos podremos controlarla a través de los valores de η y μ.

- Tasa de aprendizaje (η). Controla el tamaño de los pasos que se van dando, que no sean muy grandes o muy pequeños. El valor deberá ser entre 0 y 1. Si este valor es demasiado grande, se pueden provocar oscilaciones y si es demasiado pequeño, se necesitaran muchas iteraciones. Se utilizará un valor de 0,1.

- Factor de momento (μ). Controla el efecto del momento, que hace que se pueda escapar de óptimos locales a través de la "inercia". Deberá tener valores entre 0 y 1, se utilizará uno de 0,9.

Además se utilizará sesgo en las neuronas de nuestra red neuronal.

4.3. Resultados obtenidos.

4.3.1. Problema XOR.

		Error de entrenamiento		Error de test	
		Media	Desviación	Media	Desviación
L = 1	{n : 2 : k}	0.106384	0.0977149	0.106384	0.0977149
	{n : 5 : k}	0.0370568	0.0494994	0.0370549	0.0495003
	{n : 10 : k}	0.00827663	0.00312442	0.00827699	0.00312377
	{n : 25 : k}	0.00410268	0.00040795	0.00410268	0.00040795
	{n : 50 : k}	0.00456539	0.00065797	0.00456177	0.00065623
	{n : 100 : k}	0.00346904	0.00067752	0.00347019	0.00067812
L = 2	{n : 2 : k}	0.300941	0.045005	0.300941	0.045005
	{n : 5 : k}	0.053355	0.0787193	0.053355	0.0787193
	{n : 10 : k}	0.00445083	0.00019023	0.0044518	0.00018972
	{n : 25 : k}	0.00266827	0.00076487	0.00266748	0.00076584
	{n : 50 : k}	0.00190115	0.00033317	0.00189989	0.00033348
	{n : 100 : k}	0.00130926	0.00021842	0.00130848	0.00021893

4.3.2. Función seno.

		Error de entrenamiento		Error de test	
		Media	Desviación	Media	Desviación
L = 1	{n : 2 : k}	0.074038	0.0844666	0.077066	0.0771118
	{n : 5 : k}	0.115101	0.104366	0.114421	0.0952887
	{n : 10 : k}	0.0302789	0.00057806	0.0369754	0.00143721
	{n : 25 : k}	0.0293767	0.00126014	0.0360615	0.00049914
	{n : 50 : k}	0.0308608	0.00015133	0.0363693	0.00033569
	{n : 100 : k}	0.0307196	0.00135935	0.0362969	0.00042799
L = 2	{n : 2 : k}	0.0309337	5.97275e-6	0.0361873	2.81426e-6
	{n : 5 : k}	0.0303163	0.00092728	0.0364829	0.00056789
	{n : 10 : k}	0.0727444	0.0850892	0.0750532	0.078035
	{n : 25 : k}	0.0303304	0.00108478	0.0362613	0.00039784
	{n : 50 : k}	0.032343	0.0012409	0.0370173	0.00069078
	{n : 100 : k}	0.0310214	0.0012009	0.0366225	0.00049348

4.3.3. Base de datos CPU.

		Error de entrenamiento		Error de test	
		Media	Desviación	Media	Desviación
L = 1	{n : 2 : k}	0.0258483	2.10381e-7	0.0287843	2.31225e-7
	{n : 5 : k}	0.0258485	8.73905e-9	0.0287844	9.80917e-9
	{n : 10 : k}	0.0207993	0.0100955	0.0237061	0.0101534
	{n : 25 : k}	0.0258484	5.77748e-8	0.0287844	6.73041e-8
	{n : 50 : k}	0.0258483	3.1865e-07	0.0287842	3.7287e-07
	{n : 100 : k}	0.0207927	0.0101076	0.0238261	0.00991452
L = 2	{n : 2 : k}	0.0258485	2.3774e-11	0.0287844	2.7289e-11
	{n : 5 : k}	0.0258485	1.02478e-8	0.0287844	1.16702e-8
	{n : 10 : k}	0.0258484	9.72731e-8	0.0287844	1.11522e-7
	{n : 25 : k}	0.0157384	0.0123781	0.0183558	0.0127712
	{n : 50 : k}	0.00056555	1.63331e-5	0.00315188	0.00038479
	{n : 100 : k}	0.0207877	0.0101172	0.0236811	0.0102016

4.3.4. Base de datos forest.

		Error de entrenamiento		Error de test	
		Media	Desviación	Media	Desviación
L = 1	{n : 2 : k}	0.00164968	0	0.00915703	0
	{n : 5 : k}	0.00164968	0	0.00915703	0
	{n : 10 : k}	0.00164968	0	0.00915703	0
	{n : 25 : k}	0.00164968	0	0.00915703	0
	{n : 50 : k}	0.00164968	0	0.00915703	0
	{n : 100 : k}	0.00164968	2.1142e-15	0.00915703	1.2987e-14
L = 2	{n : 2 : k}	0.00164968	0	0.00915703	0
	{n : 5 : k}	0.00164968	0	0.00915703	0
	{n : 10 : k}	0.00164968	5.47105e-9	0.00915702	1.8182e-08
	{n : 25 : k}	0.00164968	0	0.00915703	0
	{n : 50 : k}	0.00164968	0	0.00915703	0
	{n : 100 : k}	0.00164968	2.7004e-14	0.00915703	9.6043e-14

4.4. Análisis de resultados.

4.4.1. Problema XOR.

Con los datos obtenidos para este problema podemos decir que, para entre 5 y 10 neuronas, empieza a ser mejor el modelo el más complejo.

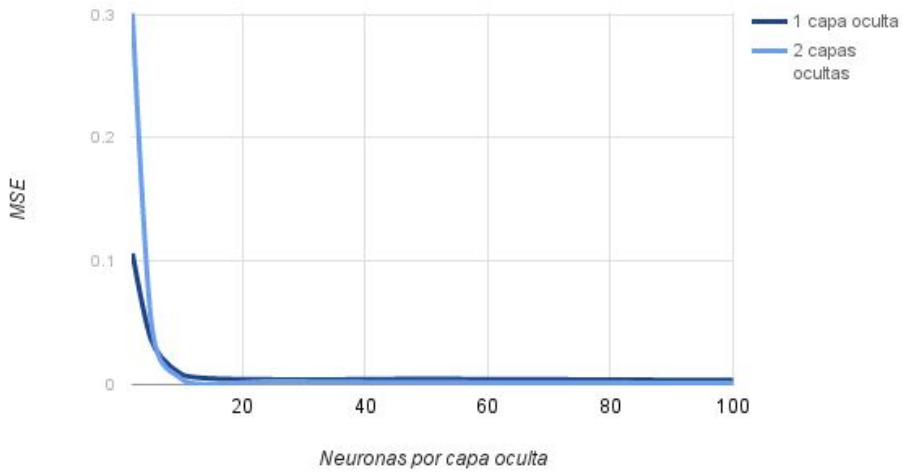


Figura 1. Error de test en la base de datos “XOR”.

Al ser un problema sencillo, la mejora de modelos más complejos no es significativamente grande por lo que utilizaremos el modelo más sencillo posible (1 capa oculta con 2 neuronas). Podríamos minimizar el error aumentando el número de capas ocultas y/o el número de neuronas por capa oculta.

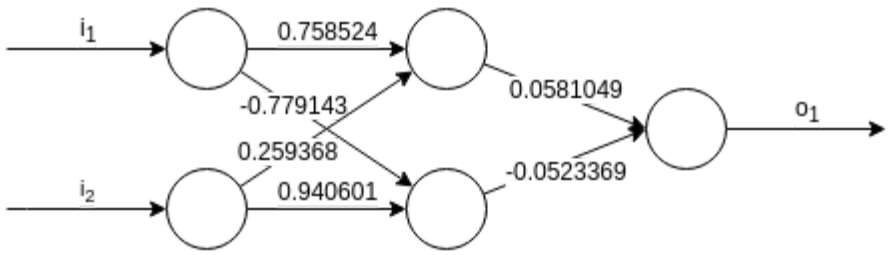


Figura 2. Modelo más sencillo para la base de datos “XOR”.

Comparando la salida obtenida y la esperada podemos ver el error de test cometido al usar este modelo con la topología especificada.

	Salida esperada	Salida obtenida
Patrón 1	1	0.507051
Patrón 2	0	0.497841
Patrón 3	1	0,494391
Patrón 4	0	0,503601
	Error de test final	0.250025

A continuación se muestra una gráfica en la que podemos observar como el error de entrenamiento disminuye cuando aumenta el número de iteración.

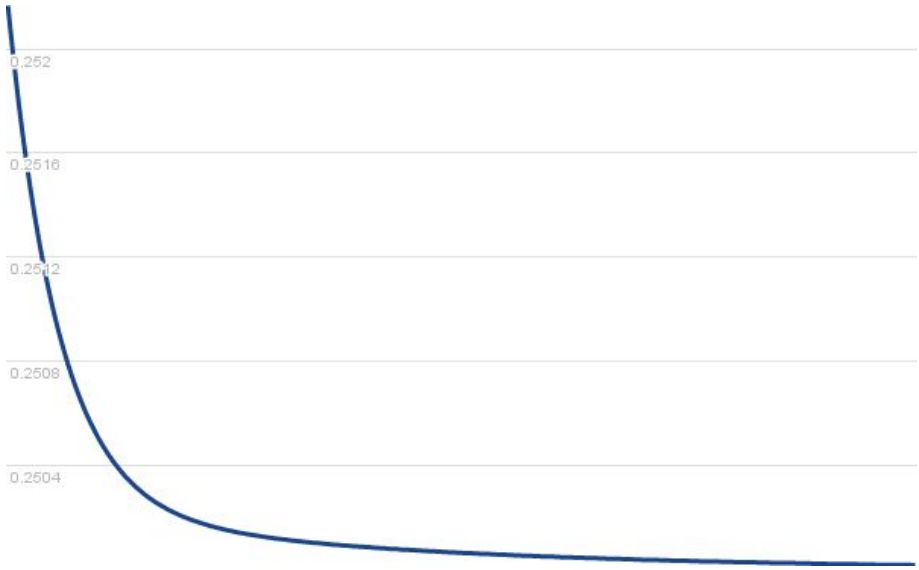


Figura 3. Error de entrenamiento para la base de datos “XOR”.

Como se puede apreciar el mínimo error se obtendría para iteraciones infinitas.

4.4.2. Función seno.

Para la función seno, el modelo de red con el que se han obtenido errores de test menores ha sido el modelo con una capa oculta con 100 neuronas y sesgo.

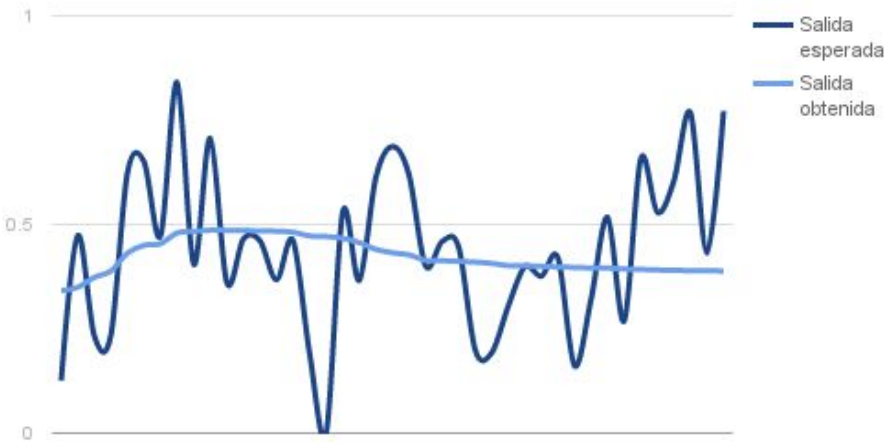


Figura 4. Error de test para la base de datos “SIN”.

La salida esperada por los patrones utilizados es la función seno con un ruido aleatorio añadido.

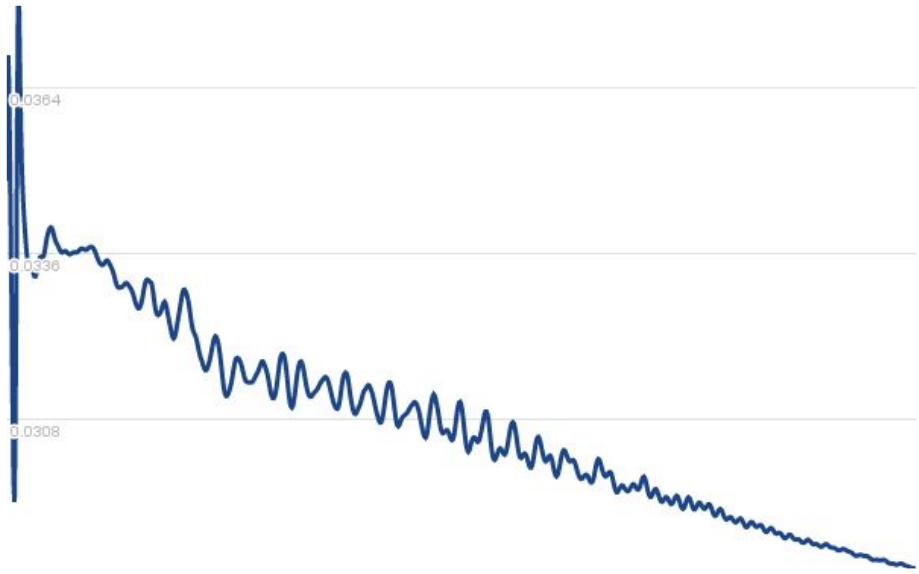


Figura 5. Error de entrenamiento para la base de datos “SIN”.

4.4.3. Base de datos CPU.

Para esta base de datos, el modelo de red con el que se han obtenido errores de test menores ha sido el modelo con dos capas ocultas con 50 neuronas y sesgo.

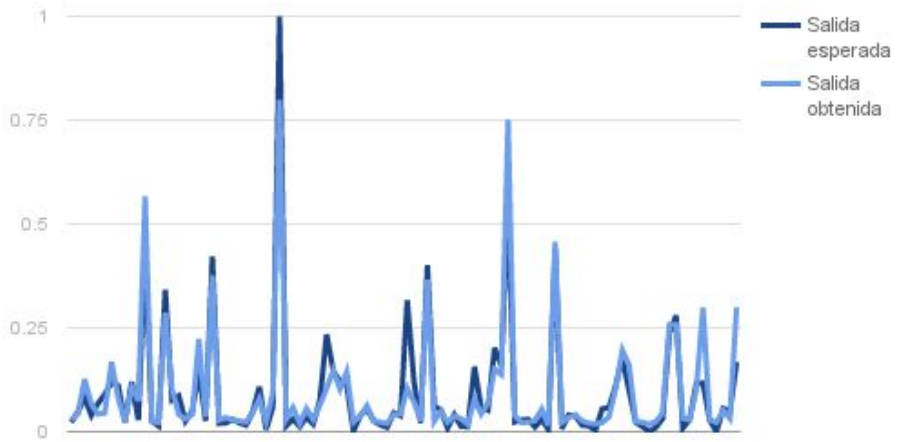


Figura 6. Error de test para la base de datos “CPU”.

Ahora veremos el error cometido en el entrenamiento para cada iteración.

Figura 7. Error de entrenamiento para la base de datos “CPU”.

4.4.4. Base de datos forest.

Por último, para esta base de datos, el modelo de red con el que se han obtenido errores de test menores ha sido el modelo con dos capas ocultas con 10 neuronas, sin sesgo.

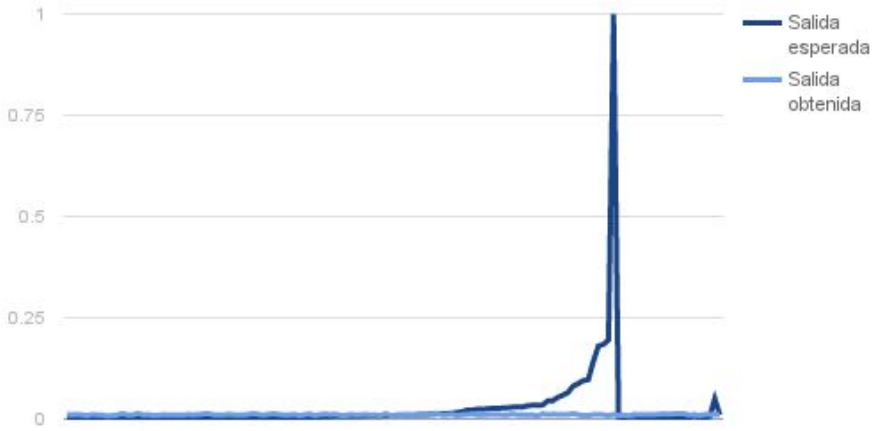


Figura 8. Error de test para la base de datos “CPU”.

Ahora veremos el error cometido en el entrenamiento para cada iteración.

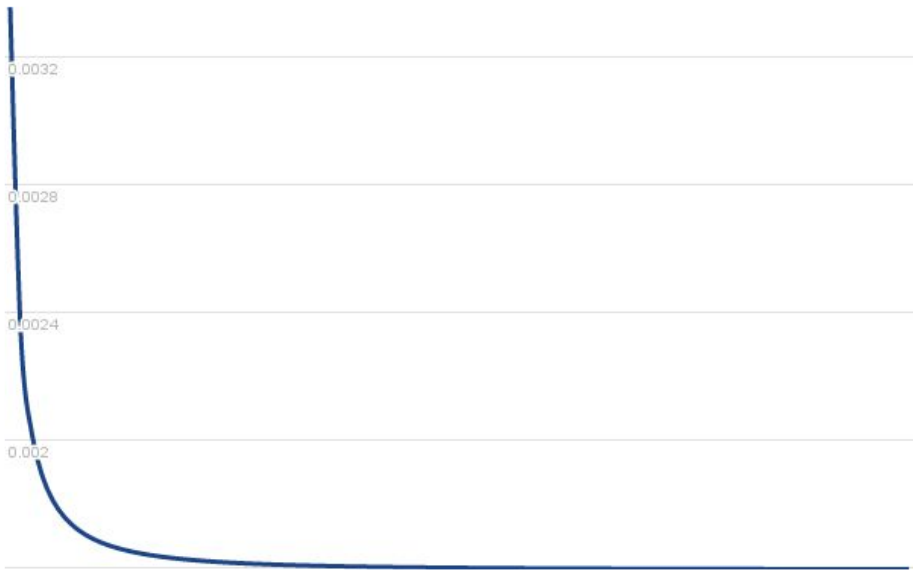


Figura 9. Error de entrenamiento para la base de datos “CPU”.

5. Referencias.

- https://es.wikipedia.org/wiki/Perceptr%C3%B3n_multicapa
- http://www.lab.inf.uc3m.es/~a0080630/redes-de-neuronas/perceptron-multicapa.html
- https://es.wikipedia.org/wiki/Red_neuronal_prealimentada
- Auer, Peter; Harald Burgsteiner; Wolfgang Maass (2008). «A learning rule for very simple universal approximators consisting of a single layer of perceptrons». Neural Networks 21 (5): 786-795.
- Roman M. Balabin, Ravilya Z. Safieva, and Ekaterina I. Lomakina (2007). «Comparison of linear and nonlinear calibration models based on near infrared (NIR) spectroscopy data for gasoline properties prediction». Chemometr Intell Lab 88 (2): 183-188.
- http://archive.ics.uci.edu/ml/datasets/Computer+Hardware
- https://archive.ics.uci.edu/ml/datasets/Forest+Fires