Introducción a los Modelos Computacionales

Práctica 1

Juan José Méndez Torrero
31018712-S
i42metoj@uco.es

13 de octubre de 2018

Índice

Índice de cuadros

1. Introducción

En el presente documento podremos encontrar la solución al problema relacionado con la primera práctica de la asignatura *Introducción a los Modelos Computacionales*. En dicha práctica se requiere la construcción de una red neuronal la cual ha de clasificar los datos de una serie de conjuntos de datos. Este documento estará dividido en tres partes, *Descripción*, *Pseudocódigo* y *Experimentos*.

En el apartado *Descripción* hablaremos de qué consiste el problema y cómo hemos hecho para solucionarlo, además de qué tipo de red neuronal hemos utilizado. En el apartado *Pseudocódigo* veremos unos pseudocódigos que explicarán cómo hemos solucionado este problema pero desde el punto de vista de código. Por último, en el apartado *Experimentos*, veremos las soluciones que da nuestro programa, además de unas tablas que explicarán qué configuraciones son mejor.

2. Descripción

Para la solución de este problema hemos utilizado un modelo de red neuronal llamado *Multilayer Perceptron*(MLP), o perceptrón multicapa en español. Dicho modelo se caracteriza por poder dar una solución a problemas que no son linealmente separables. Esto se consigue a través de un conjunto de capas por las que va pasando los estímulos que le damos a nuestra red y nos proporciona una salida. Otra característica del modelo *MLP* es que presenta una arquitectura de conectividad total, es decir, que todas las neuronas de cada capa están conectadas con todas de la anterior y posterior capa.

Las capas que podemos ver en este modelo son las siguientes:

- **Capa de entrada**: Es la capa por la cuál estimularemos nuestra red neuronal con una serie de patrones encontrados en una base de datos.
- **Capa/s oculta/s**: Son un conjunto de capas en las cuales se irá realizando el procesamiento no lineal de los patrones introducidos en la capa de entrada.
- **Capa de salida**: Es la capa por la cuál veremos qué solución nos da a nuestro problema.

Además de las de las capas, nuestro modelo estará constituido por un número finito de neuronas, que se encontrarán en cada una de las capas de nuestra red neuronal.

Para el correcto funcionamiento de nuestra red, hemos utilizado un algoritmo de retropopagación. Este algoritmo consiste en propagar un estímulo desde la capa de entrada hasta la capa de salida, pasando por todas las capas ocultas. Esta salida se compara con la salida deseada y se calcula un error, cambiando la relación que existe entre las neuronas de cada capa, siendo esta propagación desde la capa de salida hasta la capa de entrada. Finalmente se calculará el error que produce y el algoritmo intentará que dicho error sea mínimo.

3. Pseudocódigo

En este apartado hablaremos sobre las funciones más relevantes de nuestro programa. Las funciones más relevantes consideradas son las siguientes:

- **simularRedOnline**: Simula el funcionamiento de nuestra red neuronal artificial.
- **alimentarEntradas**: Introduce unos estímulos para el funcionamiento de la red. Estos estímulos serán unos patrones encontrados en una base de datos.
- **propagarEntradas**: Propaga desde la capa de entrada hasta la capa de salida los patrones introducidos devolviendo una salida.
- **retropopagarError**: Propaga desde la capa de salida hasta la capa de entrada el error producido por nuestra red.
- **acumularCambio**: Acumula los cambios producidos por un patrón en la capa de salida.
- **ajustarPesos**: Ajusta los relación existente entre las neuronas en relación al error producido.

El pseudocódigo perteneciente a cada una de las anteriores funciones se puede encontrar a continuación:

1. **simularRedOnline**:

```
function simularRedOnline(vector entrada, vector objetivo)
BEGIN
  // Introduce patrones
    alimentarEntradas();

  // Propaga los patrones introducidos hacia delante
    propagarEntradas();

  // Propaga el error producido hacia atras
    retropropagarError();

  // Acumula el error producido por la salida
     acumularCambio();

  // Ajusta los pesos en relacion al error producido
     ajustarPesos();
END
```

2. **alimentarEntradas**:

```
function alimentarEntradas(vector input)
BEGIN
    // Para todas las neuronas de la capa de entrada
   for i=0 until i<numero_total_Neuronas do
       Neurona[i].salida = input[i];
   endfor
END
```

3. **propagarEntradas**:

```
function propagarEntradas()
BEGIN
    // Leemos todas las capas menos la de entrada
    for i=1 until i< numero_total_capas do

        // Leemos todas las neuronas de la capa
        for j=0 until j<numero_total_neuronas do

            // Inicializamos la activacion a 0
            activation=0;

            // Calculamos el numero de neuronas de la capa anterior
            number_neuron = Capa[i-1].number_neuron;

            // Calculamos la activacion de las neuronas
            for k=0 until k<number_neuron do
                activation += Capa[i].peso * Capa[i-1].salida;
            endfor

            // Aplicamos el sesgo
            activation += Capa[i].sesgo;

            // Aplicamos funcion sigmoide
            Capa[i].Neurona[j].salida = sigmoid(activation);
        endfor
    endfor
END
```

4. **retropropagarError**:

```
function retropropagarError(vector objetivo)
BEGIN
    // Leemos las neuronas de la capa de salida
    for i=0 until i<numero_neuronas do
        // Calculamos la derivada de las neuronas
        Capa[numero_capas-1].Neurona[i].dX = -(objetivo[i]-Capa[
            numero_capas-1].Neurona[i].salida)* Capa[numero_capas-1].
            Nuerona[i].salida * (1-Capas[numero_capas-1].Neurona[i].salida
            );
    endfor

    // Leemos desde la pen ltima capa
    for i=numero_capas-2 until i>0 do
        // Leemos todas las neuronas
        for i=0 until i<numero_neuronas do
            aux = 0;
            // Leemos todas las neuronas de la capa i+1
            for k=0 until k<numero_neuronas do
                //Calculamos w*delta
                aux+= Capa[i+1].Neurona[k].peso[j] * Capa[i+1].Neurona[j].
                    dX;
            endfor
            // Calculamos la derivada de esta capa
            Capa[i].Neurona[j].dX = aux*Capa[i].Neurona[j].salida * (1-
                Capa[i].Neurona[j].salida);
        endfor
    endfor
END
```

5. **acumularCambio**:

```
function acumularCambio()
BEGIN
    // Leemos todas las capas menos la de entrada
    for i=1 until i<numero_capas do
        // Leemos todas las neuronas de la capa
        for j=0 j<numero_neuronas do
            // Leemos neuronas de la capa anterior
            for k=0 until k<numero_neuronas do
                Capa[i].Neurona[j].deltaW[k] += Capa[i].Neurona[j].dX*Capa
                    [i-1].Neurona[k].salida;
            endfor
        endfor
    endfor
END
```

6. **ajustarPesos**:

```
function ajustarPesos()
BEGIN
    new_eta = 0;
    // Leemos todas las capas menos la de entrada
    for i=1 until i<numero_capas do
        // Decrementamos el learning rate
        new_eta = pow(factor_decremento, -(numero_capas-i)) *antigua_eta;
        // Leemos todas las neuronas de la capa
        for j=0 until j<numero_neuronas do
            // Leemos todas las neuronas de la capa anterior
            for k=0 until k<numero_neuronas do
                // Ajustamos los pesos de la capa actual
                Capa[i].Neurona[j].peso[k] -= (new_eta * Capa[i].Neurona[j
                    ].deltaW[k]) - (Momento *(new_eta * Capa[i].Neurona[j
                    ].ultimoDeltaW[k]));
                // Guardamos los cambios
                Capa[i].Neurona[j].ultimoDeltaW[k] = Capa[i].Neurona[j].
                    deltaW[k];
                // Reinicializamos los cambios
                Capa[i].Neurona[j].deltaW[k]=0;
            endfor
            // Incluimos el sesgo
            Capa[numero_capas-1].Neurona[j].peso[k] -= (new_eta * Capa[
                numero_capas-1].Neurona[j].deltaW[k]) - (Momento *(new_eta
                 * Capa[numero_capas-1].Neurona[j].ultimoDeltaW[k]));

            Capa[numero_capas-1].Neurona[j].ultimoDeltaW[k] = Capa[
                numero_capas-1].Neurona[j].deltaW[k];

            Capa[numero_capas-1].Neurona[j].deltaW[k]=0;
        endfor
    endfor
END
```

4. Experimentos

En este apartado se explicarán los conjuntos de datos utilizados para comprobar el buen funcionamiento de nuestro programa. Además, se expondrán un conjunto de tablas en las cuales podremos observar los resultados obtenidos por el programa junto con un análisis de cada uno de ellos.

4.1. Conjunto de datos

Para la comprobación del correcto funcionamiento de nuestro programa vamos a utilizar cuatro conjuntos de datos de diferentes dimensiones. A continuación se detallarán con más detalle:

1. **Conjunto de datos XOR**: Esta base de datos consta de 4 patrones, los cuales pueden tomar los valores -1, 0 y 1. Además, esta base de datos consta de tres atributos, 2 de entrada y 1 de salida. El conjunto test es igual que el conjunto train.
2. **Conjunto de datos Seno**: Esta base de datos cuenta con 120 patrones para el conjunto de train y 41 para el conjunto de test. Los dos conjuntos constan de dos atributos, uno de entrada y otro de salida, que pueden tomar valores comprendidos entre -1 y 1.
3. **Conjunto de datos Quake**: Esta base de datos consta de 1633 patrones para el conjunto train y 546 para el conjunto de test ambos constan de tres atributos de entrada y uno de salida. Los de entrada son, profundidad focal, latitud y longitud. El de salida es el equivalente en la escala de Richter.
4. **conjunto de datos Parkinson**: La base de datos consta de 4406 patrones dentro del conjunto train y 1469 en el conjunto test. Ambos cuentan con un total de 21 atributos, 19 de entrada y 2 de salida. Los de entrada son los siguientes:
   - Edad.
   - Sexo.
   - Tiempo de test.
   - 'Jitter per'.
   - 'Jitter abs'.
   - 'Jitter RAP'.
   - 'Jitter PPQS'.
   - 'Jitter DDP'.
   - Shimmer.
   - 'Shimmer dB'.
   - 'Shimmer APQ3'.
   - 'Shimmer APQ5'.
   - 'Shimmer APQ11'.

- 'Shimmer DDA'.
- 'NHR'.
- 'HNR'.
- 'RPDE'.
- 'DFA'.
- 'PPE'.

Los dos atributos de salida son 'motor_UPDRS' y 'total_UPDRS'.

Para evaluar el rendimiento de nuestro programa usando cada una de estas bases de datos, vamos a utilizar una configuración como la siguiente:

1. Un factor de momento con valor de 0,9.
2. Un learning rate con valor de 0,1.
3. Un valor para el conjunto de validación de 0,1.
4. 1000 iteraciones para cada entrenamiento.
5. 5 semillas diferentes.

El rendimiento de cada configuración será evaluado por su media y desviación típica de *Mean Squared Error*(MSE), tanto del entrenamiento como del test.

4.2. Resultados

1. **Conjunto de datos XOR**:
   Para este conjunto de datos no vamos a utilizar validación. En la tabla 1, se muestran los resultados obtenidos con las distintas configuraciones:

CONFIGURACIÓN	CAPAS OCULTAS	NEURONAS	MEDIA TRAIN	DESVIACIÓN TÍPICA TRAIN	MEDIA TEST	DESVIACIÓN TÍPICA TEST
1	1	2	0,104468	0,0950408	0,104468	0,0950401
2	1	5	0,0173961	0,0108963	0,0173961	0,0108963
3	1	10	0,00782709	0,000425221	0,00782709	0,000425221
4	1	25	0,00531671	0,000486476	0,00531318	0,000488704
5	1	50	0,00369215	0,000302392	0,00368861	0,000303177
6	1	100	0,101963	0,135632	0,10956	0,135632
7	2	2	0,242484	0,0119544	0,242482	0,0119529
8	2	5	0,122933	0,0732063	0,122933	0,0732063
9	2	10	0,0090491	0,00270599	0,0090491	0,00270599
10	2	25	0,00418375	0,000564516	0,00417916	0,000563202
11	2	50	0,0015057	3,31E-05	0,00150271	3,11E-05
12	2	100	0,000771677	4,64E-05	0,000767682	4,54E-05

Cuadro 1: Resultados obtenidos con el conjunto de datos XOR

Como vemos en la tabla 1, el mejor resultado lo conseguimos con una configuración de 2 capas ocultas y 100 neuronas en cada una. A continuación se intentará mejorar ese resultado cambiando el factor de decremento. En la tabla 2, se puede observar el mejor resultado:

CONFIGURACIÓN	CAPAS OCULTAS	NEURONAS	MEDIA TRAIN	DESVIACIÓN TÍPICA TRAIN	MEDIA TEST	DESVIACIÓN TÍPICA TEST
N : 100 : 100: k // F=1	2	100	0,000771677	4,64E-05	0,000767682	4,54E-05
N : 100 : 100: k // F=2	2	100	0,00407836	0,00049831	0,00407429	0,000501606

Cuadro 2: Resultados obtenidos con el conjunto de datos XOR cambiando el factor de decremento

Como vemos, la mejor configuración obtenida ha sido con 2 capas ocultas, 100 neuronas en cada una y un factor de decremento de 1.

2. **Conjunto de datos Seno**:
   Para este conjunto de datos sí vamos a utilizar el conjunto de validación de 0.1. En la tabla 3 pueden ser observados los resultados con las distintas configuraciones.

CONFIGURACIÓN	CAPAS OCULTAS	NEURONAS	MEDIA TRAIN	DESVIACIÓN TÍPICA TRAIN	MEDIA TEST	DESVIACIÓN TÍPICA TEST
1	1	2	0,0299495	7,39E-05	0,0360393	5,38E-05
2	1	5	0,0295685	0,000743823	0,0358969	0,000376811
3	1	10	0,292259	0,000734093	0,0358494	0,00111392
4	1	25	0,0284483	2,35E-04	0,0346922	4,42E-04
5	1	50	0,0274373	1,22E-04	0,0344361	3,83E-04
6	1	100	0,0298564	6,55E-04	0,0388174	9,98E-04
7	2	2	0,0299505	7,31E-05	0,0360235	1,07E-04
8	2	5	0,030055	8,51E-05	0,0360876	4,56E-05
9	2	10	0,0303572	0,000211651	0,0361074	6,73E-05
10	2	25	0,0298206	0,000291519	0,0365801	0,000213423
11	2	50	0,298124	3,19E-03	0,0305339	3,06E-03
12	2	100	0,0153846	3,65E-03	0,0219745	3,68E-03

Cuadro 3: Resultados obtenidos con el conjunto de datos Seno

Como vemos, la configuración que nos da mejores resultados es con 2 capas ocultas y 100 neuronas cada una. Como antes, vamos a intentar mejorarla cambiando el factor de decremento. En la tabla 4 podemos observar el resultado.

CONFIGURACIÓN	CAPAS OCULTAS	NEURONAS	MEDIA TRAIN	DESVIACIÓN TÍPICA TRAIN	MEDIA TEST	DESVIACIÓN TÍPICA TEST
N : 100 : 100: k // F=1 // y=0,0	2	100	0,0153846	3,65E-03	0,0219745	3,68E-03
N : 100 : 100: k // F=1 // y=0,1	2	100	0,0153846	3,65E-03	0,0219745	3,68E-03
N : 100 : 100: k // F=1 // y=0,2	2	100	0,0153846	3,65E-03	0,0219745	3,68E-03
N : 100 : 100: k // F=2 // y=0,0	2	100	0,0284488	5,43E-04	0,0371339	1,06E-03
N : 100 : 100: k // F=2 // y=0,1	2	100	0,0284488	5,43E-04	0,0371339	1,06E-03
N : 100 : 100: k // F=2 // y=0,2	2	100	0,0284488	5,43E-04	0,0371339	1,06E-03

Cuadro 4: Resultados obtenidos con el conjunto de datos Seno cambiando el factor de decremento

Como podemos observar en la anterior tabla, aun cambiando el factor de decremento y el porcentaje de conjunto de validación, no conseguimos que el error MSE disminuya, con lo que la mejor configuración será 2 capas ocultas, 100 neuronas en cada una, un factor de decremento de 1 y conjunto de validación del 0.1.

3. **Conjunto de datos Quake**:
   Para este conjunto de datos vamos a tener una configuración igual que para el conjunto de datos Seno. Los resultados obtenidos pueden ser observados en la

tabla 5.

CONFIGURACIÓN	CAPAS OCULTAS	NEURONAS	MEDIA TRAIN	DESVIACIÓN TÍPICA TRAIN	MEDIA TEST	DESVIACIÓN TÍPICA TEST
1	1	2	0,0301907	8,27E-06	0,0273008	1,96E-05
2	1	5	0,0300555	0,00019434	0,0271648	0,000187718
3	1	10	0,0298041	0,000156546	0,0269867	0,000118497
4	1	25	0,029611	7,17E-05	0,0270089	4,42E-05
5	1	50	0,0295814	3,16E-05	0,0270213	4,04E-05
6	1	100	0,0296058	8,60E-05	0,0270785	2,36E-05
7	2	2	0,0302305	7,29E-05	0,0273493	7,94E-05
8	2	5	0,0301866	8,03E-06	0,0272956	1,60E-05
9	2	10	0,0299764	0,000276519	0,0271084	2,44E-04
10	2	25	0,0298851	0,000196431	0,027011	0,000177078
11	2	50	0,0300378	2,57E-04	0,0272067	8,89E-05
12	2	100	0,0292315	4,24E-05	0,027145	5,96E-05

Cuadro 5: Resultados obtenidos con el conjunto de datos Quake

Como podemos observar, los mejores resultados son obtenidos con una configuración de 2 capas ocultas y con 100 neuronas cada una. Como antes, vamos a cambiar el factor de decremento y conjunto de validación y ver si se encuentra una mejora. En la tabla 6 podemos ver los resultados de este cambio:

CONFIGURACIÓN	CAPAS OCULTAS	NEURONAS	MEDIA TRAIN	DESVIACIÓN TÍPICA TRAIN	MEDIA TEST	DESVIACIÓN TÍPICA TEST
N : 100 : 100: k // F=1 // v=0,0	2	100	0,0292315	4,24E-05	0,027145	5,96E-05
N : 100 : 100: k // F=1 // v=0,1	2	100	0,0292315	4,24E-05	0,027145	5,96E-05
N : 100 : 100: k // F=1 // v=0,2	2	100	0,0292387	7,77E-05	0,0270901	5,49E-05
N : 100 : 100: k // F=2 // v=0,0	2	100	0,0296877	3,91E-05	0,0270131	7,56E-05
N : 100 : 100: k // F=2 // v=0,1	2	100	0,0296958	4,17E-05	0,0270138	7,47E-05
N : 100 : 100: k // F=2 // v=0,2	2	100	0,0296958	4,17E-05	0,0270138	7,47E-05

Cuadro 6: Resultados obtenidos con el conjunto de datos Quake cambiando el factor de decremento

Como vemos, no se produce ninguna mejora, con lo que podemos decir que la mejor configuración es la anteriormente comentada.

4. **Conjunto de dato Parkinson**:
   Como antes, vamos a utilizar la misma configuración, es decir, vamos a ir cambiando las neuronas por capas y el número de capas. En la tabla 7 podemos ver el resultado de estas configuraciones.

CONFIGURACIÓN	CAPAS OCULTAS	NEURONAS	MEDIA TRAIN	DESVIACIÓN TÍPICA TRAIN	MEDIA TEST	DESVIACIÓN TÍPICA TEST
1	1	2	0,0345873	1,35E-04	0,371918	1,37E-04
2	1	5	0,0238148	0,00113647	0,024135	0,0011046
3	1	10	0,0187655	0,00073448	0,0196097	0,00106549
4	1	25	0,0150333	0,0015264	0,0166003	0,00198785
5	1	50	0,01338308	0,000828743	0,0156963	0,000953465
6	1	100	0,0144086	0,000733413	0,0170298	0,000838865
7	2	2	0,0322341	7,42E-04	0,0337657	0,000816181
8	2	5	0,0174284	1,83E-03	0,0176884	1,79E-03
9	2	10	0,0128003	0,00279435	0,0135144	2,97E-03
10	2	25	0,008916	0,00307237	0,0115462	0,00359386
11	2	50	0,00440725	9,90E-04	0,00685887	1,20E-03
12	2	100	0,00349208	9,95E-04	0,00602255	1,61E-03

Cuadro 7: Resultados obtenidos con el conjunto de datos Parkinson

Como podemos observar en la anterior tabla, los mejores resultados se consiguen con una configuración de 2 capas ocultas y 100 neuronas en cada una. Como hasta ahora, vamos a intentar mejorar esos resultados con un incremento del factor de decremento. Dichos resultados pueden ser observados en la tabla 6.

CONFIGURACIÓN	CAPAS OCULTAS	NEURONAS	MEDIA TRAIN	DESVIACIÓN TÍPICA TRAIN	MEDIA TEST	DESVIACIÓN TÍPICA TEST
N : 100 : 100: k // F=1 // v=0,0	2	100	0,00367958	1,24E-03	0,0061998	1,50E-03
N : 100 : 100: k // F=1 // v=0,1	2	100	0,00367958	1,24E-03	0,0061998	1,50E-03
N : 100 : 100: k // F=1 // v=0,2	2	100	0,00367958	1,24E-03	0,0061998	1,50E-03
N : 100 : 100: k // F=2 // v=0,0	2	100	0,00745374	1,20E-03	0,00898193	1,10E-03
N : 100 : 100: k // F=2 // v=0,1	2	100	0,0296958	4,17E-05	0,0270138	7,47E-05
N : 100 : 100: k // F=2 // v=0,2	2	100	0,0296958	4,17E-05	0,0270138	7,47E-05

Cuadro 8: Resultados obtenidos con el conjunto de datos Parkinson cambiando porcentaje de validación y factor de decremento

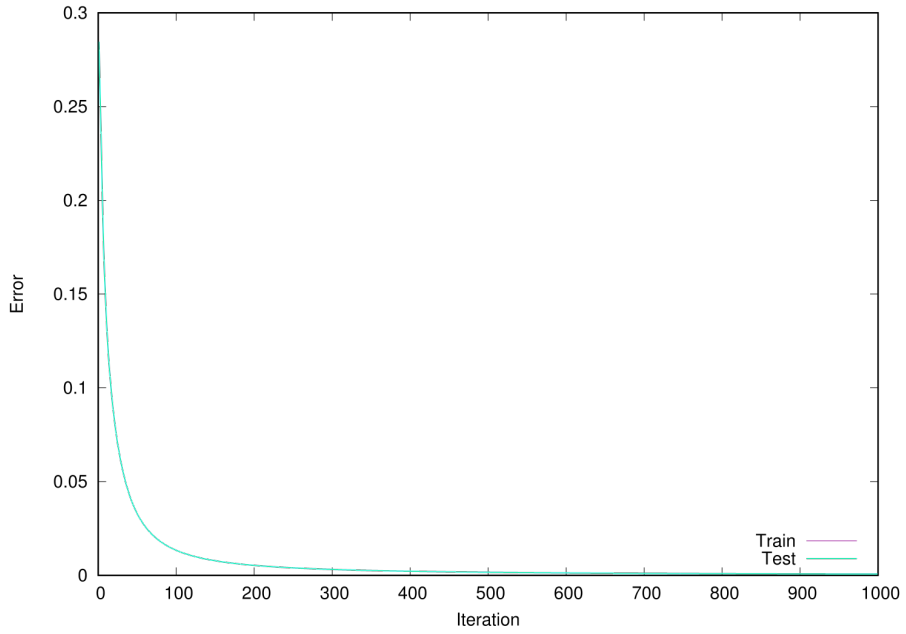
Viendo estos resultados, no encontramos ninguna mejora notable, con lo que podemos decir que la mejor configuración son 2 capas ocultas, 100 neuronas en cada una, un factor de decremento de 1 y un porcentaje de validación del 0.1.

4.3. Análisis de los resultados

En esta sección explicaremos los resultados obtenidos con las configuraciones comentadas y daremos información visual de cómo converge nuestro modelo de red neuronal.

1. **Conjunto de datos XOR**

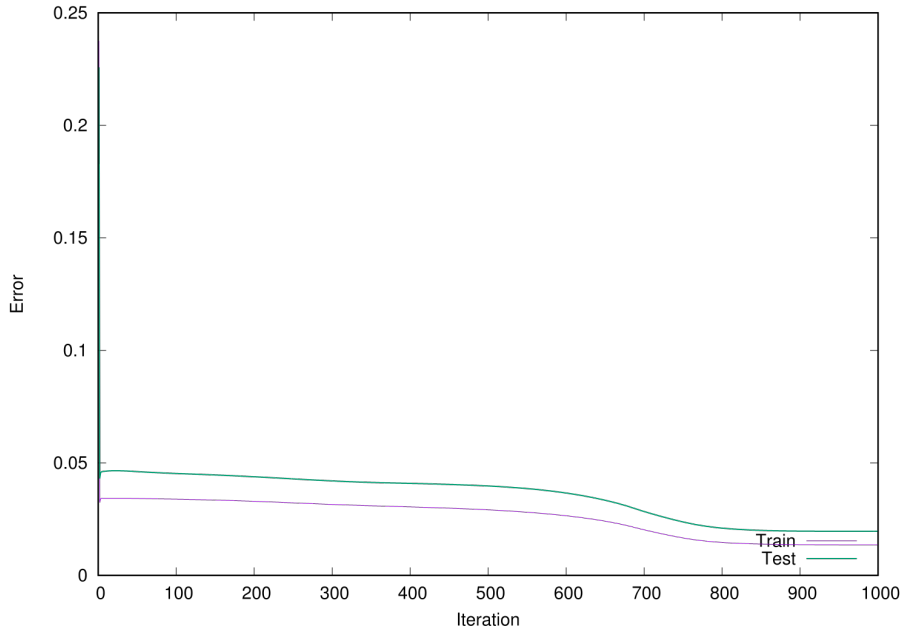
   Como podemos observar en las tablas 1 y 2, la mejor configuración de nuestra red para este conjunto de datos es de 2 capas ocultas con 100 neuronas cada una. Como se ve por los resultados, a mayor número de neuronas, menor es el error tanto del conjunto de train como del conjunto de test. Además, como vemos en la tabla 2, el aumentar el factor de decremento no hace que los errores obtenidos disminuyan, esto es debido a que aun aplicando mayor decremento a la tasa de aprendizaje, los errores, tanto de train como de test, no disminuyen, es más, aumentan.
   En la siguiente tabla podremos observar la convergencia de los errores cometidos por la red utilizando el conjunto de datos XOR. Para ello, hemos ejecutado nuestra red pero sólo con una semilla, 500, siendo ésta la que mejor resultados nos ha ofrecido. La tabla de convergencia resultante es la que podemos observar en la gráfica 9.



Cuadro 9: Tabla de convergencia del conjunto de datos XOR

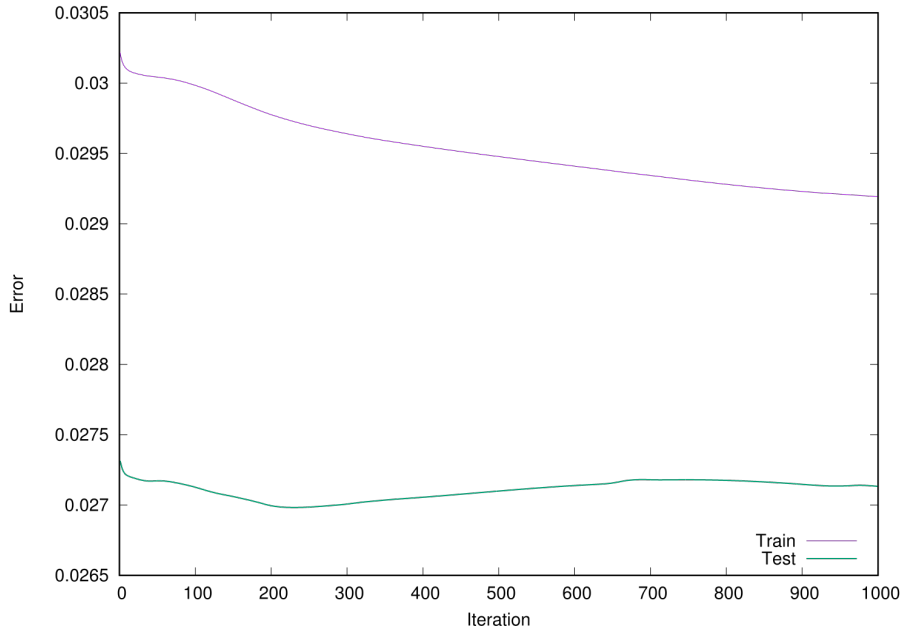
Como podemos observar en la gráfica anterior, el error de train y de test están superpuestos, esto es debido a que el conjunto de train es el mismo que el de test. Además, se puede observar que, a partir de la iteración 50 el error cometido por nuestra red disminuye drásticamente hasta llegar a ser casi 0 en la última iteración.

2. **Conjunto de datos Seno**:
   Para este conjunto de datos, como podemos ver en las tablas 3 y 4, la mejor configuración para nuestro modelo es de 2 capas, 100 neuronas en cada una, un factor de decremento de 1 y un conjunto de validación de 0.1. Esto es debido a que si aumentamos el factor de decremento el error obtenido también aumenta, siendo no éste el resultado buscado.
   A continuación hemos creado una gráfica en la que se puede observar cómo convergen el error tanto del conjunto de train como del conjunto de test.
   Como se puede observar en la gráfica 10, los errores de los dos conjuntos no convergen a la vez como pasaba con el conjunto de datos XOR. Esta vez, y como se puede observar, el error cometido por los dos conjuntos aumentan de manera no muy drástica al principio de nuestro entrenamiento, seguidamente se estabilizan en torno al 0.03 pero, a partir de la iteración 600/700 este error disminuye drásticamente y se vuelve a estabilizar. Esto es debido a que nuestro modelo ha conseguido superar un óptimo local llegando a convertirse en un óptimo global.



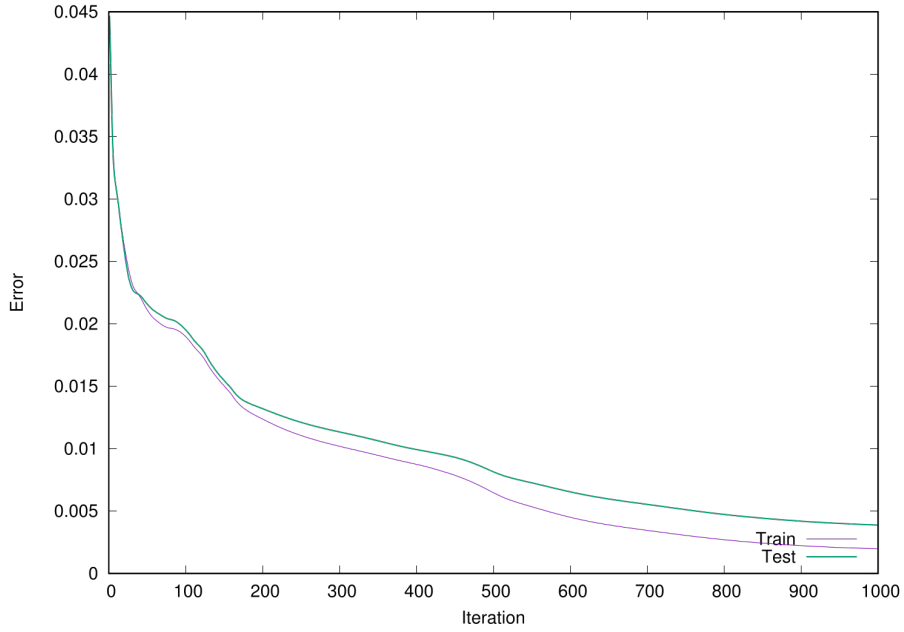
Cuadro 10: Tabla de convergencia del conjunto de datos Seno

3. **Conjunto de datos Quake**:
   En este conjunto de datos, los mejores errores obtenidos han sido con una configuración prácticamente igual que con las anteriores, es decir, 2 capas ocultas, 100 neuronas en cada una, un factor de decremento de 1 y un conjunto de validación del 0.1. Como antes, hemos decidido crear una gráfica de convergencia en la que podremos observar cómo convergen los errores tanto de test como de train. Dicha gráfica la hemos obtenido utilizando la semilla de 400, que es con la que mejor resultados hemos obtenido.
   Como podemos observar en la gráfica 11, los errores de train y de test son muy distintos, pero como observamos tienden a converger de la misma forma. En el conjunto de train vemos que al principio se estabiliza en torno al 0.03, pero después baja drásticamente para volver a estabilizarse. Esto puede ser debido, como hemos explicado antes a que nuestra red a entrado en un óptimo local que finalmente ha conseguido superar.



Cuadro 11: Tabla de convergencia del conjunto de datos Quake

4. **Conjunto de datos Parkinson**:
   Para este conjunto de datos, como vemos en las tablas 7 y 8, los mejores resultados obtenidos son con una configuración de 2 capas ocultas, 100 neuronas en cada una, un factor de decremento de 1 y un conjunto de validación del 0.1. Como hasta ahora, hemos recogido los errores del conjunto de train y test y hemos creado una gráfica para ver su convergencia.



Cuadro 12: Tabla de convergencia del conjunto de datos Parkinson

Como podemos observar en la gráfica 12, hasta la iteración 30 los dos conjuntos, train y test, convergen de la misma manera, éstos siguen disminuyendo de manera drástica hasta llegados a una iteración, 150, en la cuál el error cometido por el conjunto de train disminuye aún más, aunque como podemos observar los dos errores tienen la misma forma de convergencia. Además, como ha pasado con las anteriores tabals, vemos que los dos conjuntos han caído en un óptimo local, pero que después consiguen superarlo cayendo en un óptimo global. Como nota, para la realización de esta tabla hemos utilizado una semilla de 500, ya que con ella ha sido con la que mejores resultados hemos obtenido.

5. Bibliografía

1. Moodle
2. Gnuplot
3. Retropropagación