Práctica 1: Implementación del perceptrón multicapa

Asignatura: Introducción a los modelos computacionales, 4º Grado Ingeniería Informática (Escuela Politécnica Superior de Córdoba - Universidad de Córdoba)

Trabajo realizado por:

-Antonio Gómez Giménez (32730338G)
i72gogia@uco.es

Índice:

1. Descripción de los modelos de redes neuronales utilizados:

Principalmente, los modelos que usamos en estas redes neuronales se basan en número de capas y el número de neuronas que nos pide el usuario, es decir, aparte de la capa de entrada y la capa de salida, ambas siempre están presentes, el usuario indica cuantas capas oculta desea, como mínimo una capa oculta (sin incluir la capa de salida).

Aparte del número de capas, el usuario proporciona el número de neuronas por capas, es decir, si dice por ejemplo cinco, entonces en todas las capas del modelo (exceptuando capa de entrada y salida, ya que se especifican las neuronas de esta capa en la propia base de datos proporcionada por el usuario) hay cinco neuronas en total, no puede haber capas con cinco neuronas y otras capas con un número distinto de cinco neuronas.

En resumen, la arquitectura de todo el modelo depende de los datos introducidos por el usuario (número de capas y número de neuronas por cada capa, que en este caso siempre es el mismo).

2. Descripción en pseudocódigo de los pasos del algoritmo de retropropagación:

Voy a obviar la parte de extracción de datos y la ejecución del main y voy a centrarme en la parte donde se realiza el run del algoritmo:

runOnlineBackPropagation(DataSet entrenamiento, DataSet test, número iteraciones, error iteraciones, error test){

```
    if(comprobamos_validacion()){
        dividimos_dataset_entrenamiento();
    }

    hacer{
        trainOnline(entrenamiento);
        test(test);
    }mientras_que(no se cumpla las condiciones de parada);

    if(comprobamos_validacion()){
        hacer{
            test(validacion);
        }mientras_que(no se cumpla las condiciones de parada);
    }

}
```

*Cabe destacar en esta parte del algoritmo que dividimos_dataset_entrenamiento() se encuentra ya implementado en esa parte del código (no es una función en sí, está puesta para generalizar lo que habría que realizar en ese punto). Las condiciones de parada son las tres especificadas en el pdf.

Se van a explicar la función **trainOnline** y **test** ya que son bastante interesantes:

trainOnline es una función que contiene un bucle que recorre todos los patrones del dataset y llama a la función **performEpochOnline** al cual se le pasan las entradas, las salidas y el número total de patrones.

test es igual que la función anterior pero en vez de llamar a **performEpochOnline** llama directamente a **feedInputs** y **forwardPropgate** para poder así sacar el error medio de todos los patrones, es decir, el error entre el valor que debería salir y el que nuestra red nos ha dado.

performEpochOnline es una función importante donde se aloja el algoritmo de retropropagación error, por tanto el pseudocódigo sería el siguiente:

```
performEpochOnline(valores entradas, valores salidas, numero_patrones){

        limpiarAcumulador();
        feedInputs(entradas);
        fordwardPropagate();
        backPropagateError();
        accumulateChange();
        weightAdjustment();

}
```

limpiarAcumulador es una función donde todos los *deltaW*(de todas las capas menos la primera y para todas las neuronas de esas capas) los ponemos a cero. Esto es útil para la versión online, ya que para cada iteración se resetea a cero aplicando esos pesos en **weightAdjustment**. Cuando esto sea offline se aplica fuera del bucle permitiendo así que se no se aplique cada cambio y que se aplique el cambio acumulandose. En resumen, el acumulador se resetea para cada patrón.

feedInputs es una función donde se alimentan las entradas, es decir, para la primera capa, para todas sus neuronas, se rellenan todos los *out* con los valores de entradas.

fordwardPropagate esta función realiza la propagación hacia delante. Para todas las capas menos la primera y para todas las neuronas de esas capas, acumulamos en su *out* el sesgo (almacenado en *w[0]*, vector de pesos), y añadimos para todas los pesos anteriores por sus respectivas salidas (bucle for desde 1 ya que no cogemos el sesgo). Una vez tenemos todo el net (el *out* acumulado, el sumatorio en sí), le aplicamos la sigmoide *(1/(1+exp(-net)))*. Este valor se guarda en *out*.

backPropagateError esta función realiza la propagación hacia atrás. Se divide en dos partes.
La primera parte consiste un un bucle que recorre todas las neuronas de la última capa donde almacenamos en *delta* de esa neurona la siguiente fórmula:

*-2(salida-out(salida de nuestra neurona de salida)*out*(1-out));*

De esta forma calculamos el *delta* de la neuronas de la capa de salida ya que la fórmula no es exactamente igual en las capas ocultas.

En la segunda parte, para cada capa desde la penúltima hasta la segunda capa, para todas las neuronas, más uno por el sesgo, y para todas las neuronas de la capa siguiente, acumulamos el peso de de todas las neurona de la capa siguiente, excepto el peso por los *deltas* de la capa siguiente.
Una vez acumulado todo eso, lo multiplicamos por la salida de la capa actual por 1-la salida actual. Por último, limpiamos el acumulador.
En si lo que hemos hecho en esa acumulación es calcular los pesos de las neuronas por las salidas posteriores a la actual.

accumulateChange esta función acumula los cambios realizados por un patrón, por ello es necesario antes de realizar los cambios guardar los *deltasW* actuales en *lastDeltaW* para almacenarlos.
Para cada capa excepto la primera, para cada neurona y para todas las neuronas de la capa anterior, guardamos en *deltaW* lo que ya había en *deltaW* más el *delta* de la neurona actual por la salida de la neurona de la capa anterior.
El *deltaW* que calcula el sesgo se calcula de una forma un poco diferente ya que no se multiplica por la salida de la neurona de la capa anterior si no que es por 1 ya que es el sesgo.

weightAdjustment esta función ajusta los pesos de toda la red para cada capa excepto la primera y para cada neurona de esas capas. Donde el nuevo peso *W* pasa a ser el peso *W* anterior más el decremento de la tasa de aprendizaje por el *deltaW* menos el factor de momento por el decremento de la tasa de aprendizaje por el *lastDeltaW*.
Es decir, calculamos el momento del peso de esa neurona. le aplicamos el decremento de la tasa de aprendizaje por capas y se lo restamos a el peso actual, cabe destacar que el peso también hay que ajustarlo para el sesgo (*W[0]*).

3. Experimentos y análisis de resultados:

En el **primer experimento** se ha utilizado la base de datos del problema XOR. Esta base de datos representa el problema de clasificación no lineal del XOR. Se utiliza el mismo fichero para entrenamiento y para test. Consta de dos entradas, una salida y cuatro patrones.
En esta primera parte, utilizaremos el factor de momento, no utilizaremos el conjunto de validación, no emplearemos decremento de la tasa de aprendizaje (F= 1) y el número de iteraciones es de 1000.

Momento
1 capa oculta
-momento 0,5

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.114262	0.0181597	0.00731603	0.00580858	0.00392239	0.0522612	0.101729
	0.0073279	2.17158e-05	1.0635e-06	1.00943e-06	1.24574e-07	0.0098346	0.0147476
Test(media y desviación típica)	0.114262	0.0181597	0.00731603	0.00580858	0.00392239	0.0522612	0.101729
	0.0073279	2.17158e-05	1.0635e-06	1.00943e-06	1.24574e-07	0.0098346	0.0147476

1 capa oculta
-momento 0,9

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.114262	0.0181597	0.00731603	0.00580858	0.00392239	0.0522612	0.101729
	0.0073279	2.17158e-05	1.0635e-06	1.00943e-06	1.24574e-07	0.0098346	0.0147476
Test(media y desviación típica)	0.114262	0.0181597	0.00731603	0.00580858	0.00392239	0.0522612	0.101729
	0.0073279	2.17158e-05	1.0635e-06	1.00943e-06	1.24574e-07	0.0098346	0.0147476

1 capa oculta

-momento 1,5

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.114262	0.0181597	0.00731603	0.00580858	0.00392239	0.0522612	0.101729
	0.0073279	2.17158e-05	1.0635e-06	1.00943e-06	1.24574e-07	0.0098346	0.0147476
Test(media y desviación típica)	0.114262	0.0181597	0.00731603	0.00580858	0.00392239	0.0522612	0.101729
	0.0073279	2.17158e-05	1.0635e-06	1.00943e-06	1.24574e-07	0.0098346	0.0147476

2 capas ocultas

-momento 0,5

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.236795	0.192081	0.00950605	0.00559973	0.00218362	0.00113643	0.000720157
	0.000304678	0.00625068	2.3834e-06	1.22086e-06	3.55609e-08	6.22418e-10	1.37982e-09
Test(media y desviación típica)	0.236795	0.192081	0.00950605	0.00559973	0.00218362	0.00113643	0.000720157
	0.000304678	0.00625068	2.3834e-06	1.22086e-06	3.55609e-08	6.22418e-10	1.37982e-09

2 capas ocultas

-momento 0,9

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.236795	0.192081	0.00950605	0.00559973	0.00218362	0.00113643	0.000720157
	0.000304678	0.00625068	2.3834e-06	1.22086e-06	3.55609e-08	6.22418e-10	1.37982e-09

Test(media y variandesviación típicaza)	0.236795	0.192081	0.00950605	0.00559973	0.00218362	0.00113643	0.000720157
	0.000304678	0.00625068	2.3834e-06	1.22086e-06	3.55609e-08	6.22418e-10	1.37982e-09

2 capas ocultas
-momento 0,9

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.236795	0.192081	0.00950605	0.00559973	0.00218362	0.00113643	0.000720157
	0.000304678	0.00625068	2.3834e-06	1.22086e-06	3.55609e-08	6.22418e-10	1.37982e-09
Test(media y desviación típica)	0.236795	0.192081	0.00950605	0.00559973	0.00218362	0.00113643	0.000720157
	0.000304678	0.00625068	2.3834e-06	1.22086e-06	3.55609e-08	6.22418e-10	1.37982e-09

Tras ver estos resultados, podemos obviar la variable momento ya que sea cual sea el momento introducido no afecta a la arquitectura (probablemente no haya mínimos locales). La mejor arquitectura es 100 nodos para 2 capas ocultas, ofreciendo un error de 0.000720157 ya que da igual el sobreentreno porque los datos de test y entrenamientos son los mismos. No se penaliza entonces el sobreentrenamiento. Comprobamos la mejor arquitectura con F=2 ya que no hay validación en este caso, no tendría sentido.

2 capas ocultas con 100 neuronas para F=1 y F=2

Nodos	F=1	F=2
Entrenamiento (media y desviación típica)	0.000720157	0.00379216
	1.37982e-09	6.83007e-08
Test(media y desviación típica)	0.000720157	0.00379216
	1.37982e-09	6.83007e-08

En conclusión, con todos los datos obtenidos anteriormente podemos ver que tanto el momento en esta base de datos no tienen importancia ya que al ser un problema tan sencillo es probable que no haya mínimos locales donde en esos casos es donde más útil es el momento. El decremento de la tasa de aprendizaje empeora este modelo en concreto ya que con este número de iteraciones probablemente no le da tiempo a ajustar los pesos iniciales.
El test y el entrenamiento son iguales ya que los datos son los mismos, por tanto, lo que nos interesa es sobreentrenar de tal forma que a más iteraciones mejor ya que se reduce el error ajustándose a la solución (por ejemplo con 5000 iteraciones para el entrenamiento como el test tienen un error medio y varianza de 0.00014554 +- 4.34502e-11).

En el **segundo experimento** se ha utilizado la base de datos de la función seno donde esta base de datos está compuesta por 120 patrones de entrenamiento y 41 patrones de test. Ha sido obtenida añadiendo cierto ruido aleatorio a la función seno. Esta base de datos consta de una entrada y una salida.

En esta base de datos ocurre igual que con el momento ya que para m=0,5, 0,9 y 1,5, los resultados son los mismos, aun así se van a calcular por si se obtuviera algún cambio.
Calculamos todas las medias y todas las varianzas para ver las mejor estructura posible, para, sobre ella, realizar las pruebas de validación y de decremento de la tasa de aprendizaje.

1 capa oculta
-momento 0,5

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.0296618	0.0293966	0.029394	0.0288663	0.0279422	0.0285211	0.0292513
	2.24637e-07	5.70048e-07	7.50661e-07	5.6495e-07	2.04824e-08	1.63148e-06	7.31577e-07
Test(media y desviación típica)	0.0361994	0.0358996	0.0355651	0.0349588	0.0342904	0.036518	0.0388019
	3.33251e-08	2.23204e-07	4.84314e-07	4.58189e-07	2.76407e-08	3.33739e-06	4.00061e-06

1 capa oculta
-momento 0,9

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.0296618	0.0293966	0.029394	0.0288663	0.0279422	0.0285211	0.0292513
	2.24637e-07	5.70048e-07	7.50661e-07	5.6495e-07	2.04824e-08	1.63148e-06	7.31577e-07
Test(media y desviación típica)	0.0361994	0.0358996	0.0355651	0.0349588	0.034290 4	0.036518	0.038801 9
	3.33251e-08	2.23204e-07	4.84314e-07	4.58189e-07	2.76407e-08	3.33739e-06	4.00061e-06

1 capa oculta
-momento 1,5

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.0296618	0.0293966	0.029394	0.0288663	0.0279422	0.0285211	0.0292513
	2.24637e-07	5.70048e-07	7.50661e-07	5.6495e-07	2.04824e-08	1.63148e-06	7.31577e-07
Test(media y desviación típica)	0.0361994	0.0358996	0.0355651	0.0349588	0.034290 4	0.036518	0.038801 9
	3.33251e-08	2.23204e-07	4.84314e-07	4.58189e-07	2.76407e-08	3.33739e-06	4.00061e-06

2 capas ocultas
-momento 0,5

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.0298902	0.0298567	0.0296298	0.0300067	0.0284219	0.0199247	0.0138609
	1.06329e-09	1.02983e-08	6.74807e-08	3.42303e-08	1.80613e-06	3.44799e-05	1.10485e-07

Test(media y desviación típica)	0.036006	0.0361708	0.0366069	0.0364279	0.0354835	0.0263005	0.019979
	9.33555e-09	1.469e-08	6.19859e-08	2.17063e-09	2.74781e-06	4.22101e-05	3.61838e-07

2 capas ocultas
momento 0,9

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.0298902	0.0298567	0.0302206	0.0300067	0.0284219	0.0199247	0.0138609
	1.06329e-09	1.02983e-08	5.36034e-08	3.42303e-08	1.80613e-06	3.44799e-05	1.10485e-07
Test(media y desviación típica)	0.036006	0.0361708	0.0361824	0.0364279	0.0354835	0.0263005	0.019979
	9.33555e-09	1.469e-08	4.94313e-09	2.17063e-09	2.74781e-06	4.22101e-05	3.61838e-07

2 capas ocultas
momento 1,5

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.0298902	0.0298567	0.0302206	0.0300067	0.0284219	0.0199247	0.0138609
	1.06329e-09	1.02983e-08	5.36034e-08	3.42303e-08	1.80613e-06	3.44799e-05	1.10485e-07
Test(media y desviación típica)	0.036006	0.0361708	0.0361824	0.0364279	0.0354835	0.0263005	0.019979
	9.33555e-09	1.469e-08	4.94313e-09	2.17063e-09	2.74781e-06	4.22101e-05	3.61838e-07

Tras analizar estas tablas podemos observar que la mejor configuración es 2 capas con 100 neuronas por capa oculta y que el factor de momento es irrelevante en este caso, ocurre lo mismo que con la base de datos de la XOR. Usaremos la mejor arquitectura de la red (dos capas ocultas con cien neuronas por capa) para comprobar si se realiza una mejora modificando los parámetros de validación y de F.

Cabe destacar que en este caso sí tiene sentido hacer validación ya que el disponemos de varios patrones y el fichero de entrenamiento y test es diferente ya que en el caso anterior al ser igual nos interesaba sobreentrenar ya que buscamos únicamente las soluciones de la XOR y son pocos patrones.
Realizamos las pruebas con distintas validaciones y distintos valores de F.

Validación 0 probando F1 y F2

Nodos	F=1	F=2
Entrenamiento(media y desviación típica)	0.0138609	0.0273926
	1.10485e-07	1.97879e-07
Test(media y desviación típica)	0.019979	0.0331969
	3.61838e-07	0,0006671

Validación 0.15 probando F1 y F2

Nodos	F=1	F=2
Entrenamiento(media y desviación típica)	0.0276061	0.0399456
	6.52888e-05	6.75407e-05
Test(media y desviación típica)	0.031705	0.0387452
	2.49477e-05	2.27884e-05

Validación 0.25 probando F1 y F2

Nodos	F=1	F=2
Entrenamiento(media y desviación típica)	0.0287871	0.0386645
	8.14369e-05	3.66387e-05
Test(media y desviación típica)	0.0494352	0.0583314
	0.000854701	0.00124021

Tras realizar las pruebas podemos observar que la mejor estructura es aquella en la que no se realiza validación y F es 1. Esto se debe a que no le da tiempo a aprender a las neuronas de la capa 1 lo suficiente, por ello con F=1 si le permite un mejor entrenamiento.
Respecto a la validación, al eliminar patrones para su uso a la validación, estamos eliminando patrones de entrenamiento. Esto hace que si no hay muchos patrones de entrenamiento, nuestra red aprenda peor. Esto se aprecia, ya que a medida que aumentamos el ratio de patrones para validación, aumenta el error resultante que nos da nuestra red.

En el **tercer experimento** se ha utilizado la base de datos quake donde esta base de datos está compuesta por 1633 patrones de entrenamiento 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, por tanto, consta de tres entradas y una salida.

En esta base de datos ocurre igual que con el momento ya que para m=0,5, 0,9 y 1,5, los resultados son los mismos, aun así se van a calcular por si se obtuviera algún cambio.
Calculamos todas las medias y todas las varianzas para ver las mejor estructura posible, para, sobre ella, realizar las pruebas de validación y de decremento de la tasa de aprendizaje.

1 capa oculta
-momento 0,5

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.0300477	0.0299628	0.02974	0.0296278	0.0295724	0.0295996	0.0295511
	5.0806e-09	1.00488e-08	8.89824e-10	2.80532e-09	4.17144e-09	3.07729e-09	2.17945e-09
Test(media y desviación típica)	0.0271501	0.0270997	0.0270088	0.0269324	0.0270357	0.0269945	0.0270875
	6.95785e-09	5.34929e-09	3.84852e-09	9.92953e-09	1.1647e-09	2.5834e-09	1.48474e-09

1 capa oculta
-momento 0,9

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.0300477	0.0299628	0.02974	0.0296278	0.0295724	0.0295996	0.0295511
	5.0806e-09	1.00488e-08	8.89824e-10	2.80532e-09	4.17144e-09	3.07729e-09	2.17945e-09
Test(media y desviación típica)	0.0271501	0.0270997	0.0270088	0.0269324	0.0270357	0.0269945	0.0270875
	6.95785e-09	5.34929e-09	3.84852e-09	9.92953e-09	1.1647e-09	2.5834e-09	1.48474e-09

1 capa oculta
-momento 1,5

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.0300477	0.0299628	0.02974	0.0296278	0.0295724	0.0295996	0.0295511
	5.0806e-09	1.00488e-08	8.89824e-10	2.80532e-09	4.17144e-09	3.07729e-09	2.17945e-09
Test(media y desviación típica)	0.0271501	0.0270997	0.0270088	0.0269324	0.0270357	0.0269945	0.0270875
	6.95785e-09	5.34929e-09	3.84852e-09	9.92953e-09	1.1647e-09	2.5834e-09	1.48474e-09

2 capas ocultas
-momento 0,5

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.0299749	0.0298693	0.029766	0.0296445	0.0295478	0.0293941	0.0291782
	5.43271e-09	9.37831e-09	6.63436e-09	1.04114e-08	8.50589e-09	2.60575e-08	1.82489e-08

Test(media y desviación típica)	0.0270621	0.0269948	0.0269043	0.0269292	0.0269546	0.0270462	0.0270438
	5.01654e-09	2.51314e-09	3.2552e-09	1.37316e-09	0,000006074	1.95913e-08	3.16951e-09

2 capas ocultas
momento 0,9

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.0299749	0.0298693	0.029766	0.0296445	0.0295478	0.0293941	0.0291782
	5.43271e-09	9.37831e-09	6.63436e-09	1.04114e-08	8.50589e-09	2.60575e-08	1.82489e-08
Test(media y desviación típica)	0.0270621	0.0269948	0.0269043	0.0269292	0.0269546	0.0270462	0.0270438
	5.01654e-09	2.51314e-09	3.2552e-09	1.37316e-09	0,000006074	1.95913e-08	3.16951e-09

2 capas ocultas
momento 1,5

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.0299749	0.0298693	0.029766	0.0296445	0.0295478	0.0293941	0.0291782
	5.43271e-09	9.37831e-09	6.63436e-09	1.04114e-08	8.50589e-09	2.60575e-08	1.82489e-08
Test(media y desviación típica)	0.0270621	0.0269948	0.0269043	0.0269292	0.0269546	0.0270462	0.0270438
	5.01654e-09	2.51314e-09	3.2552e-09	1.37316e-09	0,000006074	1.95913e-08	3.16951e-09

En este caso, sorprendentemente, tras analizar las tablas, la mejor estructura no es de dos capas con cien neuronas sino que la mejor estructura es de dos capas con nueve neuronas, esto ocurre ya que el entrenamiento siempre mejora su error pero si nos fijamos, el error del test empieza a incrementar, esto ocurre por que se está empezando a sobreentrenar, es decir, nuestra red se está empezando a ajustar

demasiado a nuestro modelo y por consecuente, nuestro modelo empezará a generalizar peor.
Por tanto, la elección correcta como mejor estructura es aquella donde no se empieza a sobreentrenar y esta es con dos capas ocultas con nueve neuronas por capa.
Usaremos esta arquitectura de la red para comprobar si se realiza una mejora modificando los parámetros de validación y de F.

Validación 0 probando F1 y F2

Nodos	F=1	F=2
Entrenamiento(media y desviación típica)	0.029766	0.0301729
	6.63436e-09	3.47915e-11
Test(media y desviación típica)	0.0269043	0.0272865
	3.2552e-09	3.71298e-11

Validación 0.15 probando F1 y F2

Nodos	F=1	F=2
Entrenamiento(media y desviación típica)	0.0226564	0.0229613
	3.11424e-05	3.0239e-05
Test(media y desviación típica)	0.0282778	0.0285661
	3.59646e-07	5.34941e-07

Validación 0.25 probando F1 y F2

Nodos	F=1	F=2
Entrenamiento(media y desviación típica)	0.0178831	0.0183702
	6.33076e-05	6.27883e-05
Test(media y desviación típica)	0.0287387	0.0289998
	5.22033e-07	7.48867e-07

Tras realizar las pruebas podemos observar que la mejor estructura es aquella en la que no se realiza validación y F es 1. De hecho, el error de entrenamiento es bastante peor que en validación 0.25 con F=1 pero el error de test en mucho menor en validación cero con F=1, esto ocurre por que con la validación se generaliza mucho peor, lo que me sorprende es que en estos casos con menos patrones esta realizando sobreentrenamiento, supongo que no puede generalizar mejor por la poca variación de los patrones recibidos al limitar el número de patrones por culpa de la validación. Respecto a el parámetro F ocurre lo mismo que en el experimento anterior, donde las neuronas de la primera capa no aprenden lo suficiente.

En el **cuarto experimento** se ha utilizado la base de datos parkinsons donde esta base de datos está compuesta por 4406 patrones de entrenamiento 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. Por tanto tenemos dos entradas y dos salidas.

En esta base de datos ocurre igual que con el momento ya que para m=0,5, 0,9 y 1,5, los resultados son los mismos, aun así se van a calcular por si se obtuviera algún cambio.
Calculamos todas las medias y todas las varianzas para ver las mejor estructura posible, para, sobre ella, realizar las pruebas de validación y de decremento de la tasa de aprendizaje.

1 capa oculta
-momento 0,5

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.0347872	0.0289687	0.0187598	0.0169087	0.0150069	0.013207	0.0127331
	2.19245e-07	4.01793e-06	4.50777e-07	2.9039e-06	6.92032e-07	1.17366e-06	1.70371e-06
Test(media y desviación típica)	0.0366313	0.0293154	0.0197846	0.0182164	0.0167045	0.015895	0.0147893
	2.97079e-07	4.23059e-06	8.91716e-07	2.85223e-06	5.87734e-07	1.75214e-06	2.18482e-06

1 capa oculta
-momento 0,9

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.0347872	0.0289687	0.0187598	0.0169087	0.0150069	0.013207	0.0127331
	2.19245e-07	4.01793e-06	4.50777e-07	2.9039e-06	6.92032e-07	1.17366e-06	1.70371e-06
Test(media y desviación típica)	0.0366313	0.0293154	0.0197846	0.0182164	0.0167045	0.015895	0.0147893
	2.97079e-07	4.23059e-06	8.91716e-07	2.85223e-06	5.87734e-07	1.75214e-06	2.18482e-06

1 capa oculta
-momento 1,5

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.0347872	0.0289687	0.0187598	0.0169087	0.0150069	0.013207	0.0127331
	2.19245e-07	4.01793e-06	4.50777e-07	2.9039e-06	6.92032e-07	1.17366e-06	1.70371e-06
Test(media y desviación típica)	0.0366313	0.0293154	0.0197846	0.0182164	0.0167045	0.015895	0.0147893
	2.97079e-07	4.23059e-06	8.91716e-07	2.85223e-06	5.87734e-07	1.75214e-06	2.18482e-06

2 capas ocultas
-momento 0,5

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.0322799	0.0228121	0.0134504	0.00913982	0.00824027	0.00380575	0.00360742
	6.07372e-07	2.07187e-06	3.66059e-06	1.77143e-06	8.46237e-06	1.27519e-06	7.2039e-07

Test(media y desviación típica)	0.0335933	0.0234906	0.0147677	0.0110509	0.0110235	0.0062 1885	0.00615301
	1.0736e-06	3.2121e-06	5.56696e-06	2.85418e-06	8.4375e-06	1.52914e-06	2.72816e-06

2 capas ocultas
momento 0,9

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.0322799	0.0228121	0.0134504	0.00913982	0.00824027	0.00380575	0.00360742
	6.07372e-07	2.07187e-06	3.66059e-06	1.77143e-06	8.46237e-06	1.27519e-06	7.2039e-07
Test(media y desviación típica)	0.0335933	0.0234906	0.0147677	0.0110509	0.0110235	0.0062 1885	0.00615301
	1.0736e-06	3.2121e-06	5.56696e-06	2.85418e-06	8.4375e-06	1.52914e-06	2.72816e-06

2 capas ocultas
momento 1,5

Nodos	2	4	9	16	32	64	100
Entrenamiento (media y desviación típica)	0.0322799	0.0228121	0.0134504	0.00913982	0.00824027	0.00380575	0.00360742
	6.07372e-07	2.07187e-06	3.66059e-06	1.77143e-06	8.46237e-06	1.27519e-06	7.2039e-07
Test(media y desviación típica)	0.0335933	0.0234906	0.0147677	0.0110509	0.0110235	0.0062 1885	0.00615301
	1.0736e-06	3.2121e-06	5.56696e-06	2.85418e-06	8.4375e-06	1.52914e-06	2.72816e-06

Tras ver estos resultados, este caso es similar al de la base de datos seno donde la mejor estructura es dos capas ocultas con cien neuronas por capa ocultas.
Como era de esperarse el momento es irrelevante ya que las tablas para cada momento son idénticas (aunque se me hace un poco raro que para esta base de datos no le afecte).

Cabe destacar que para esta base de datos al aumentar su número de iteraciones es muy probable que siga mejorando su error pero el tiempo de cómputo es muy elevado, solo en una ejecución de dos capas con cien neuronas por capa puede llegar a tardar cerca de cuarenta minutos. Posteriormente comprobaremos si la validación soluciona este problema.

Usaremos esta arquitectura de la red para comprobar si se realiza una mejora modificando los parámetros de validación y de F.

Validación 0 probando F1 y F2

Nodos	F=1	F=2
Entrenamiento(media y desviación típica)	0.00360742	0.0110653
	7.2039e-07	5.31489e-06
Test(media y desviación típica)	0.00615301	0.0131777
	2.72816e-06	6.84293e-06

Validación 0.15 probando F1 y F2

Nodos	F=1	F=2
Entrenamiento(media y desviación típica)	0.0053372	0.00902755
	3.24185e-06	5.11616e-06
Test(media y desviación típica)	0.0105457	0.015275
	1.15317e-05	4.94121e-06

Validación 0.25 probando F1 y F2

No he podido sacar estos resultados por problemas de ejecución probablemente por problemas de espacio al crear el conjunto de validación. El programa ejecuta tres seed pero a partir de la cuarta ya no funciona.

4. Resumen:

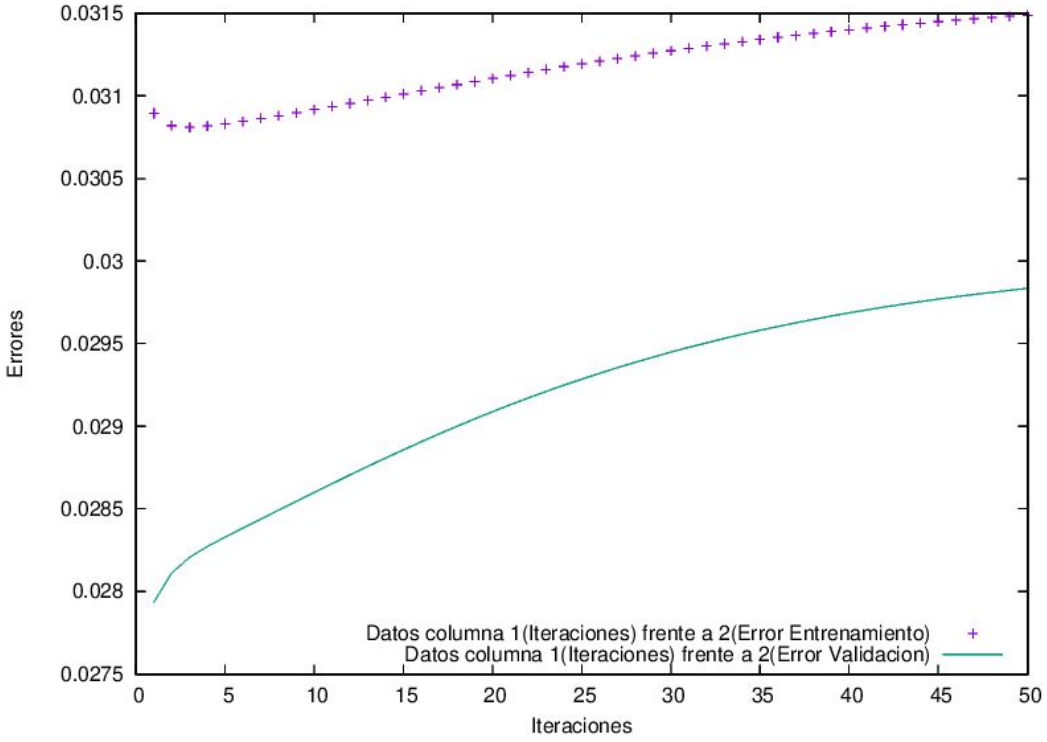
Tras analizar todo lo visto en las tablas podemos sacar en conclusión que :

-El **momento** en las redes que se entrenan de forma online es inutil. Esto se debe a que el momento multiplica el estado anterior de variación de los pesos y este es cero ya que de forma online los cambios se aplican en cada iteración. Para ser más exactos, el momento multiplica deltaW en nuestro programa y en la versión online cada vez que realizamos una iteración esta deltaW la reseteamos a cero. Por consiguiente, el momento solo tiene sentido cuando se tenga en cuenta el momento anterior y esto sucederá en el algoritmo offline.

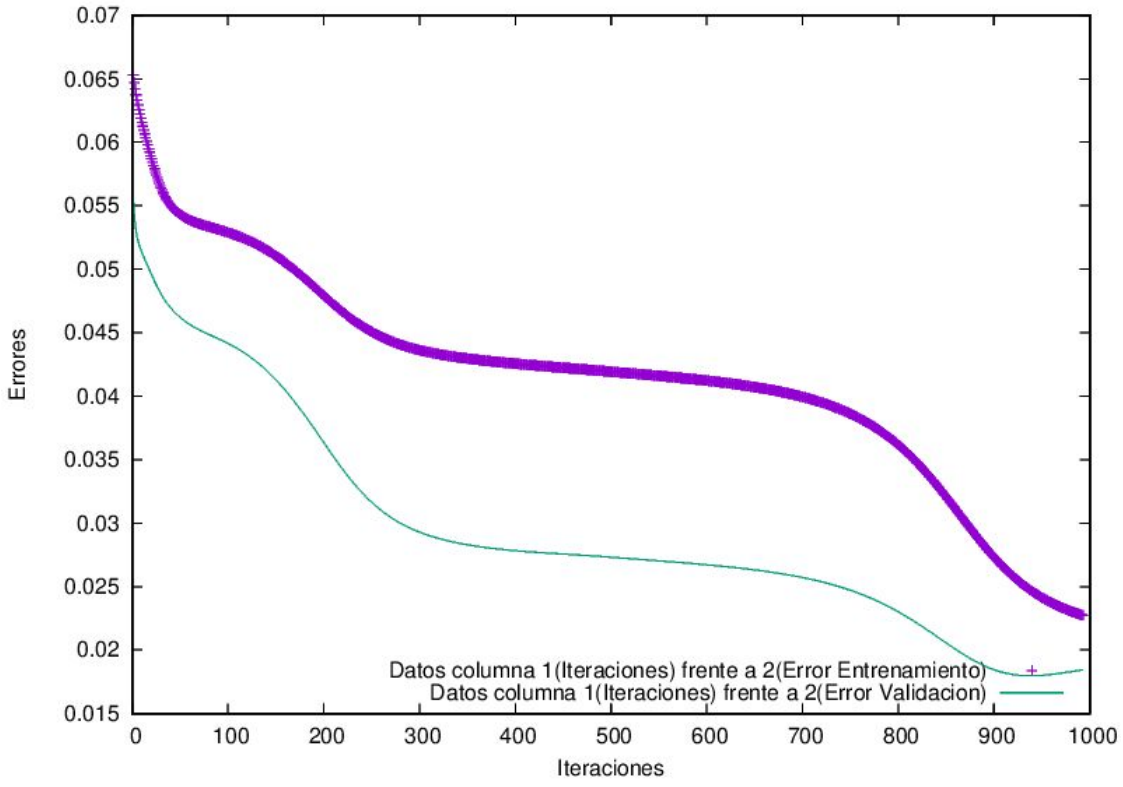
-Para **F** nos hemos dado cuenta de que sucede y lo hemos explicado.
A mi parecer F tiene más sentido en redes con más número de iteraciones de esta forma es mejor modificar los pesos de forma más importante aquellos que están en las entradas, ya que con 1000 iteraciones no le da tiempo prácticamente a entrenar correctamente los pesos más cercanos a las entradas en nuestro caso cuando hay dos capas nos referimos a la primera capa.

-Respecto a la **validación**, todos los resultado donde se realiza la validación son peores pero la utilidad de la validación se encuentra en evitar el sobreentrenamiento(en nuestros casos no se aplica por el pequeño número de iteraciones y por la poca cantidad de patrones, ya que si hay pocos patrones y nos excedemos con los patrones de validación, en el entrenamiento no será capaz de generalizar por falta de patrones) y reducir el número de iteraciones.
Esto último se nota mucho sobre todo en aquellas bases de datos donde hay muchos patrones ya que por cada iteración se tarda bastante y al reducir el número de iteraciones mejoramos enormemente la velocidad de cómputo respecto a aumentar el error obtenido.
Para apreciar cómo funciona validación vamos a observar las siguientes gráficas, estas gráficas representan la base de datos seno con dos capas y cien nodos por capas donde se ha escogido la mejor solución (menor error) y hemos observado que valores da de error de entrenamiento y validación con respecto a la iteración para cada una de las semillas.

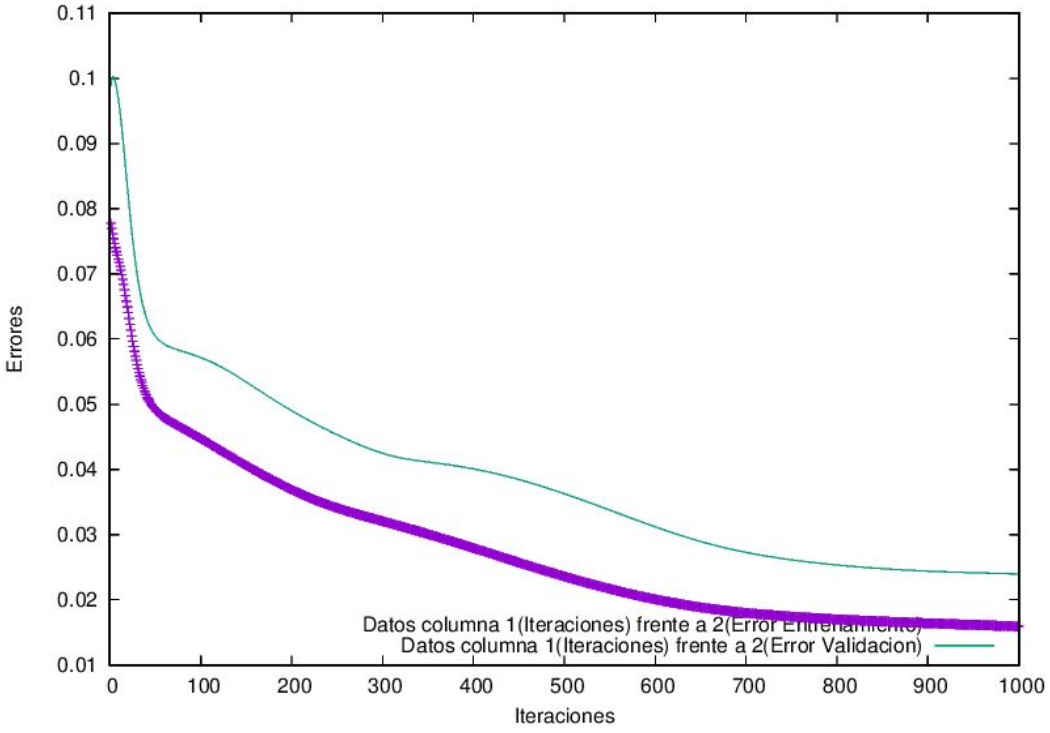
Seed1:


0.0315
0.031
0.0305
0.03
0.0295
0.029
0.0285
0.028
0.0275
Errores
0
5
10
15
20
25
30
35
40
45
50
Iteraciones
Datos columna 1(Iteraciones) frente a 2(Error Entrenamiento)
Datos columna 1(Iteraciones) frente a 2(Error Validacion)

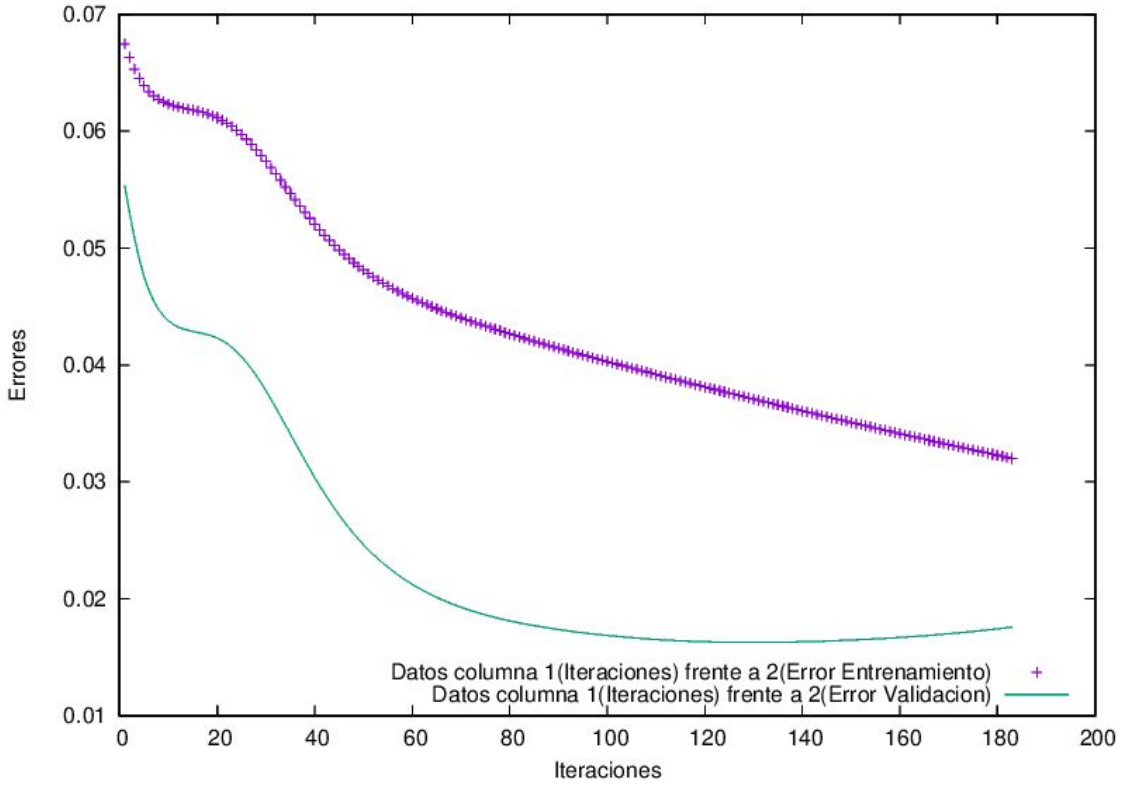
Seed2:


0.07
0.065
0.06
0.055
0.05
0.045
0.04
0.035
0.03
0.025
0.02
0.015
Errores
0
100
200
300
400
500
600
700
800
900
1000
Iteraciones
Datos columna 1(Iteraciones) frente a 2(Error Entrenamiento)
Datos columna 1(Iteraciones) frente a 2(Error Validacion)

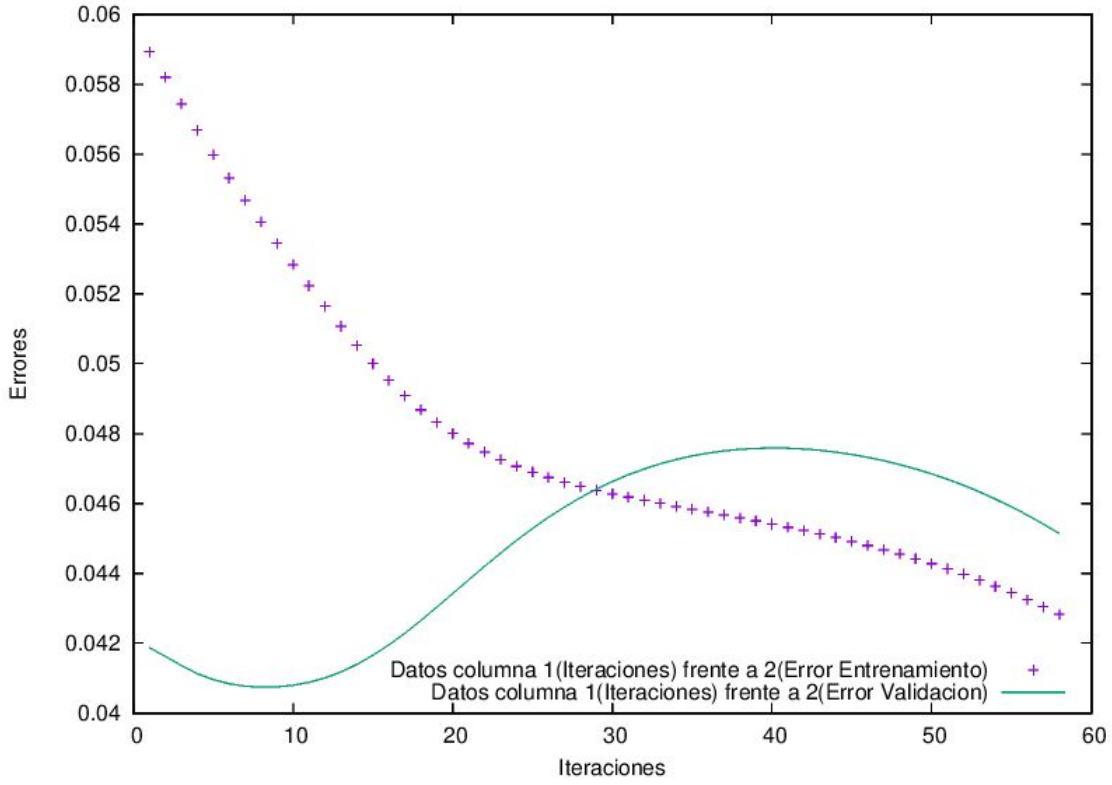
Seed3:


0.11
0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
Errores
0
100
200
300
400
500
600
700
800
900
1000
Iteraciones
Datos columna 1(Iteraciones) frente a 2(Error Entrenamiento)
Datos columna 1(Iteraciones) frente a 2(Error Validacion)

Seed4:


0.07
0.06
0.05
0.04
0.03
0.02
0.01
Errores
0
20
40
60
80
100
120
140
160
180
200
Iteraciones
Datos columna 1(Iteraciones) frente a 2(Error Entrenamiento)
Datos columna 1(Iteraciones) frente a 2(Error Validacion)

Seed5:

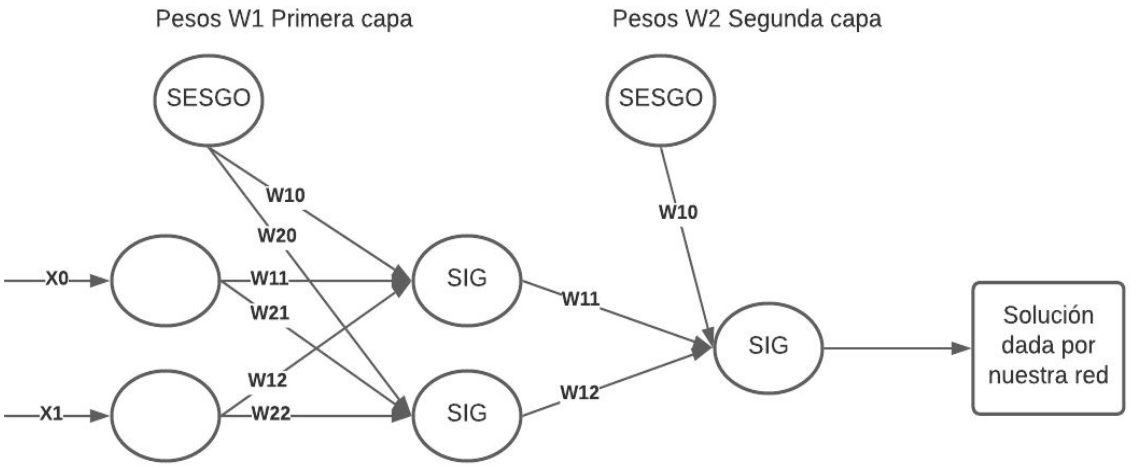


De estas gráficas se puede extraer mucha información.
En la primera gráfica tanto el error en la validación como el error en el entrenamiento sube por tanto se para el número de ejecuciones.
En la segunda gráfica siempre disminuye el error de ambas por tanto actúa como la seed3, es decir, como si no existiera validación realizando así todas las iteraciones posibles.
Respecto a la cuarta ocurre algo interesante, la validación corta el número de iteraciones ya que cuando llega a cierto punto el error de validación aumenta y la diferencia entre el error y el siguiente es notorio, sino lo fuera la validación no pararía las iteraciones.
Por último, en la quinta iteración, ocurre algo similar al caso anterior pero tras realizar una subida, posteriormente hace una bajada y esto me da que pensar.

Tras revisar código he comprobado que cuando la validación baje a la validación esta se debe resetear, es decir, en este caso lo que sucede con la validación es que siempre es peor al mínimo aunque al final decrece y nosotros debemos de contemplar que cuando se realice cualquier bajada resetee este contador por que la validación solo se debe parar cuando aumente durante 50 iteraciones seguidas, es decir, no influye el error mínimo de la validación. Por ello, es necesario realiza un

cambio en el código para arreglar este error, ya que en ciertas ocasiones puede darse que termine el número de iteraciones antes de tiempo.

Por último, para que quede todo claro, se va a mostrar gráficamente cómo quedaría la red neural con una estructura de una capa con dos neuronas en la base de datos de XOR.



Las entradas que se le darán a nuestra red son las siguientes:
1 -1
-1 -1
-1 1
1 1

Y los pesos de nuestra red los siguientes:

Capa 1

W10-> 2.79585
W20-> -2.95944
W11-> -2.95888
W21-> -2.72116
W12-> -2.74063
W22-> -2.74222

Capa 2

W10-> -2.05467
W11-> 4.70256
W12-> -5.01248

Las salidas esperadas respecto a las salidas de nuestra red neuronal son las siguientes:

1 -- 0.887675
0 -- 0.112299
1 -- 0.887796
0 -- 0.134979

Como podemos observar, en esta red más o menos se acerca a los valores reales pero dista bastante los valores que deseamos que son aquellos que se acercan más a 1 cuando debe salir 1 y 0 cuando debe ser 0.
Con más capas y más neuronas se acerca a esos valores, ya que entrena mejor la red para alcanzarlos.