Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Introducción a la Inteligencia Artificial

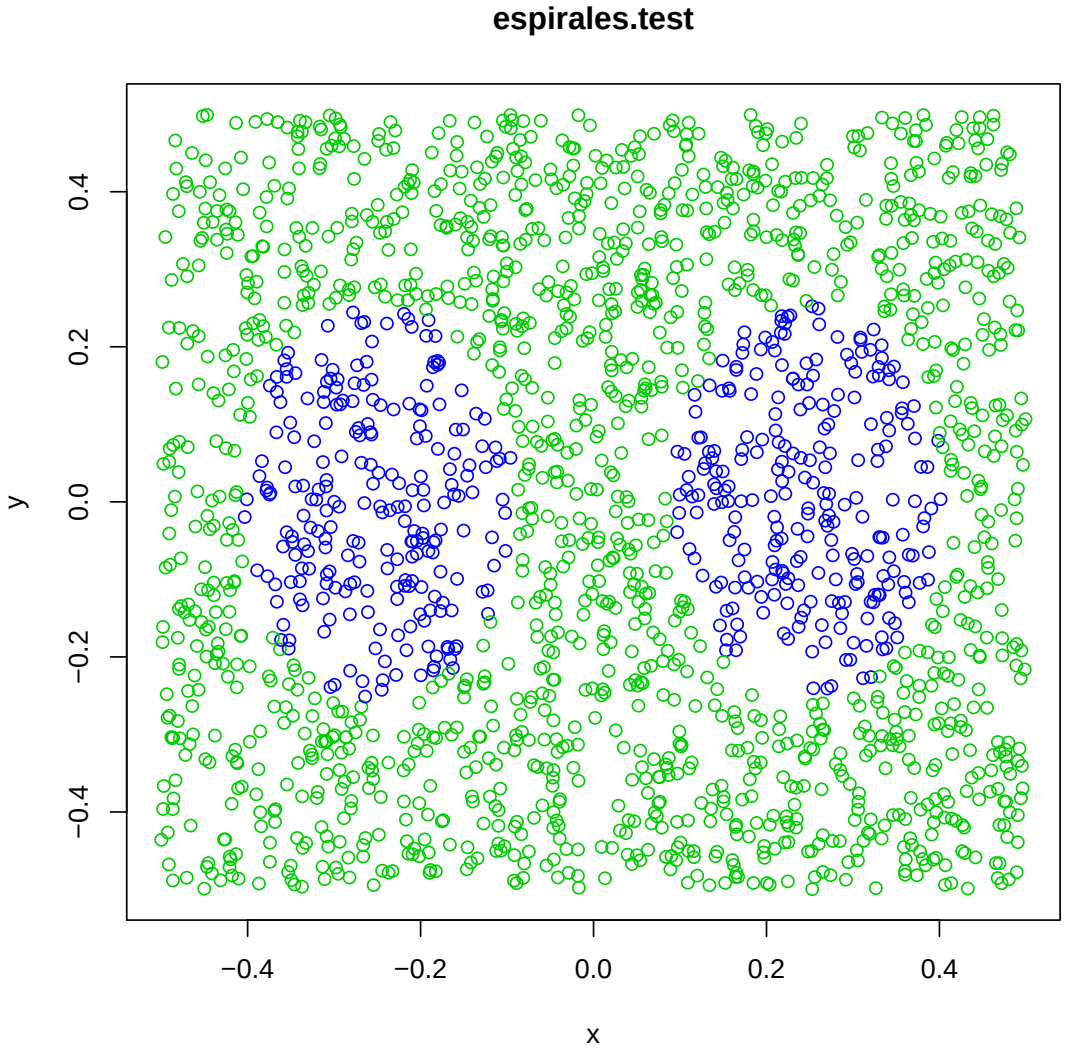
TRABAJO PRÁCTICO 2

Alumno: Rodríguez Jeremías

28 de mayo de 2017

1. Ejercicio A: Mínimos locales

Usamos redes neuronales para aprender a clasificar si un punto en el plano está dentro de dos elipses determinadas. El conjunto de test es el siguiente:

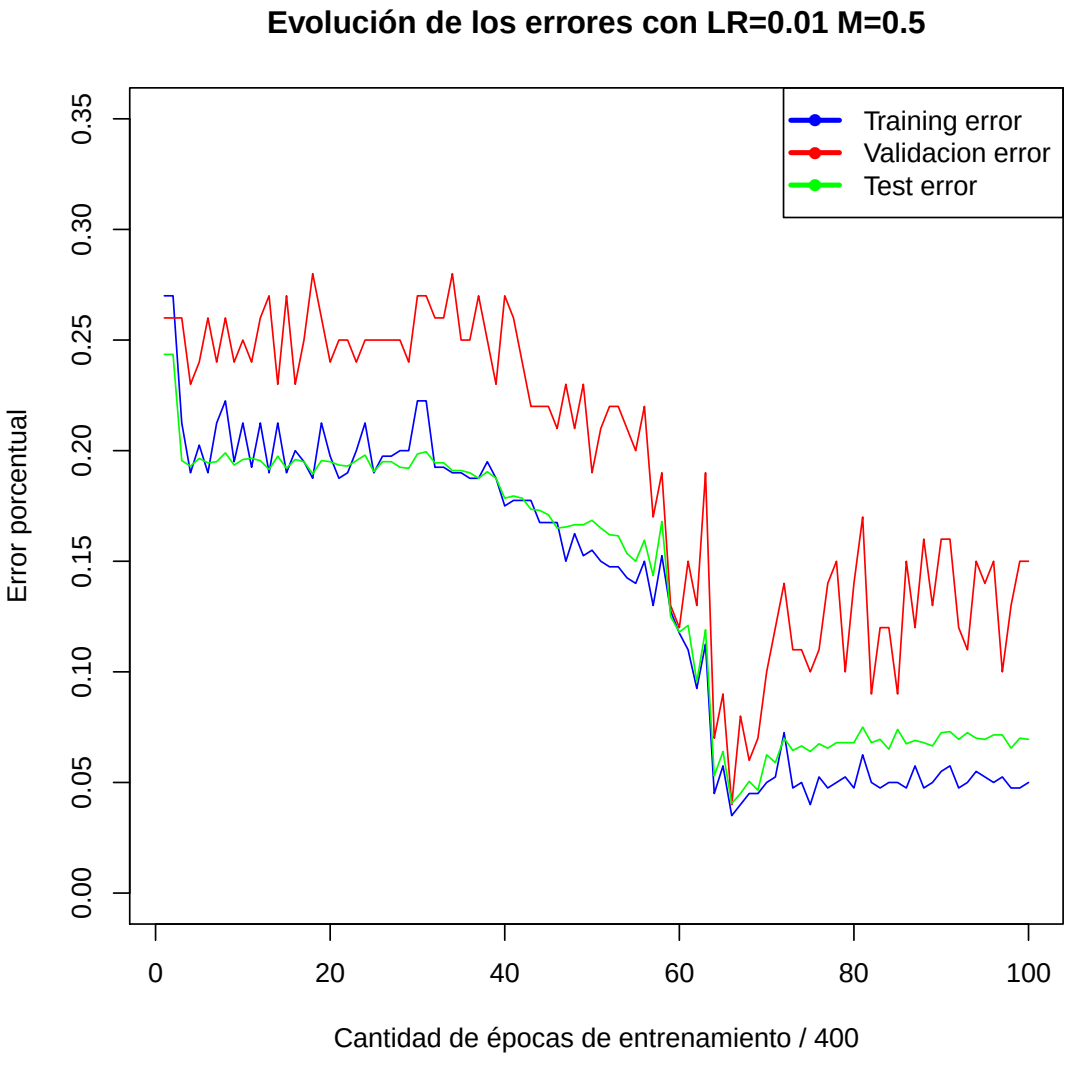


En este ejercicio analicé cómo varía el error porcentual obtenido por la red neuronal final que retorna el algoritmo *bp* en función de los parámetros **momentum** y **learning rate** en el conjunto de training.

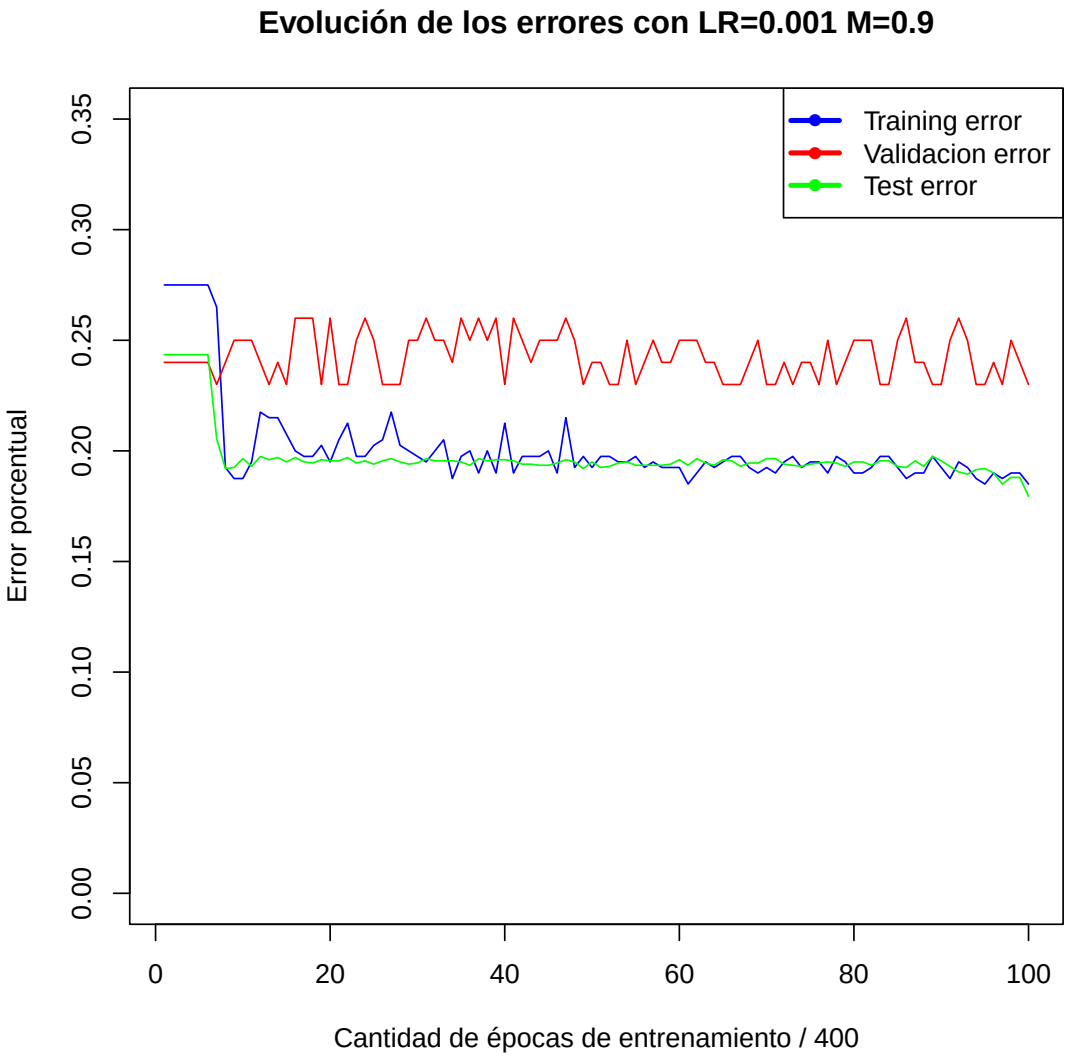
Usé 3 valores para cada uno de estos parámetros, y para cada una de las 9 combinaciones ejecuté el algorimo *bp* 20 veces. De las 20 ejecuciones, se puede ver en la siguiente tabla la mediana de ellos (en %):

	m=0	m=0.5	m=0.9
lr=0.001	19.700000	19.400001	17.950000
lr=0.01	16.550000	6.450000	14.700000
lr=0.1	6.700000	15.200000	19.550000

Como vemos, el menor error se dió para $lr = 0{,}01$ y $m = 0{,}5$. La siguiente gráfica muestra los errores de test, validación y training y su evolución a través de las iteraciones del algoritmo en la ejecución que obtuvo como resultado el valor óptimo (6,45 %).



Para comparar, veamos la gráfica de la ejecución que obtuvo como valor 17.95 % en la tabla:



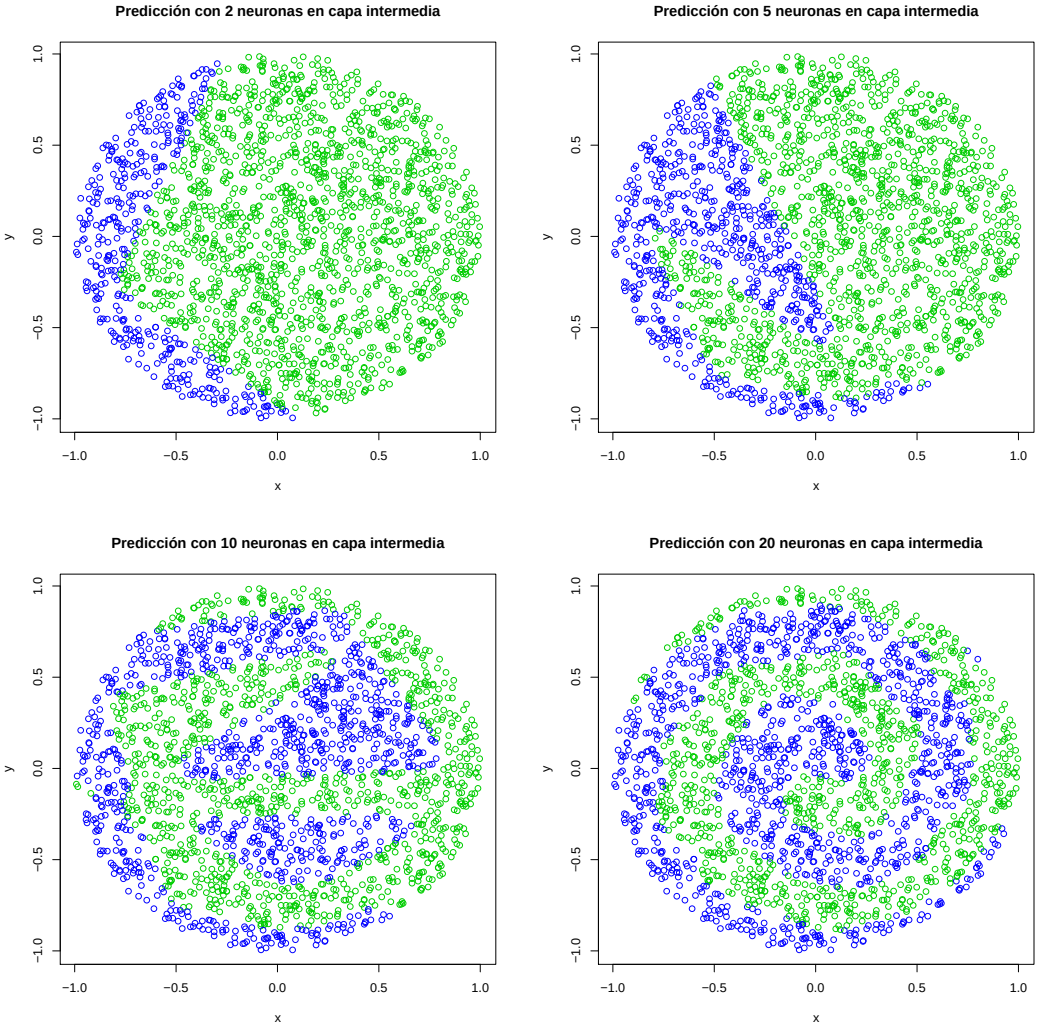
En primer lugar puedo observar que el error varía mucho para distintas combinaciones de learning rate y momentos. Por lo tanto, es crucial elegir valores adecuados. Este problema en particular (dos elipses) obtuvo los mejores resultados con LR=0.01 y M=0.5. Esto parece indicar que valores muy similares a estos producen resultados óptimos para este problema en concreto.

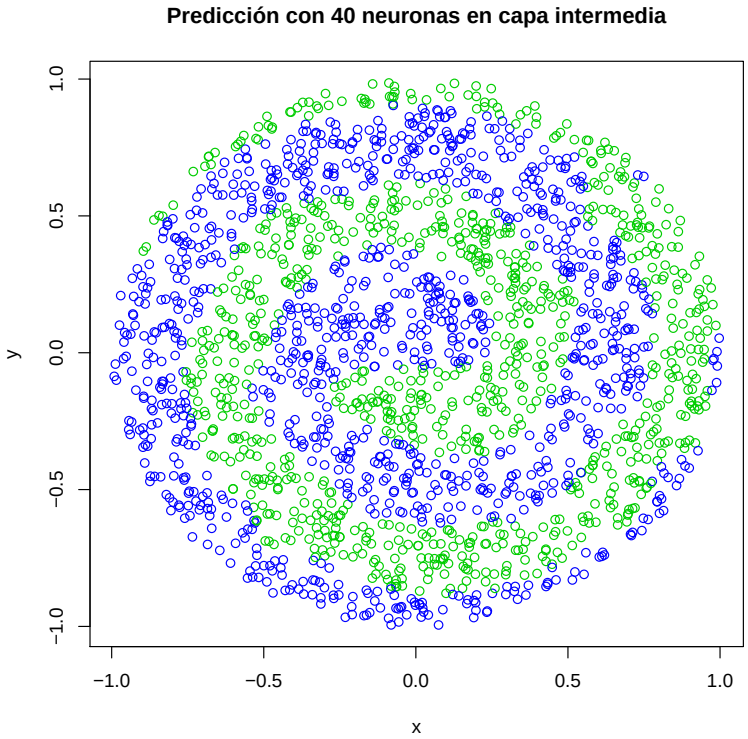
2. Ejercicio B

En este ejercicio analizaremos el error porcentual en el problema de clasificación de las espirales anidades, de acuerdo a la cantidad de neuronas en la capa intermedia. El procedimiento fue análogo al ejercicio anterior, realizando 7 entrenamientos para cada configuración de neuronas en la capa intermedia y seleccionando la mediana. La tabla obtenida fue la siguiente:

n	Error %
2	39.649999
5	36.100000
10	23.000000
20	9.800000
40	8.450000

Y las gráficas de las predicciones de las redes de donde provienen esos porcentajes son:



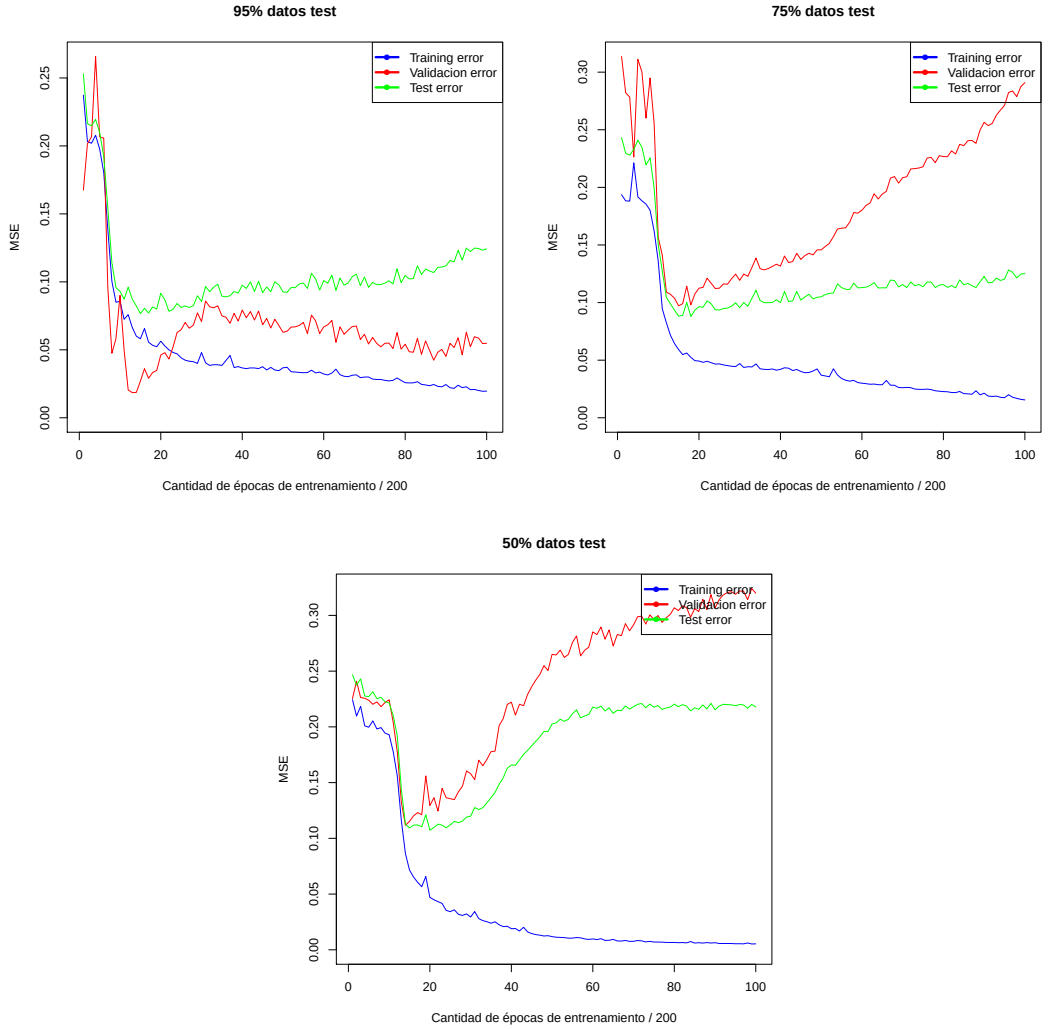


Como podemos observar, cuando aumenta la cantidad de neuronas en la capa intermedia la red clasifica cada vez mejor los puntos en el conjunto de test. La diferencia se hace mucho más notoria al principio. Si siguieramos agregando neuronas el error seguiría bajado, aunque correríamos el riesgo de aprender ruido.

3. Ejercicio C

Este es el primer problema de regresión al cuál nos enfrentamos. En este ejercicio, vamos a ir modificando la proporción de los datos que se utilizan para training-validación, y viendo como responde el error a estos cambios. Se toman 100 elementos del conjunto de training a repartir entre validación y entrenamiento de la red.

Se utilizan tres proporciones: 95-5, 75-25 y 50-50 (training - validación). Para cada una de estas proporciones, se realizaron 20 entrenamientos y se eligieron los entrenamientos cuyo error era la mediana del grupo para realizar las siguientes gráficas:



Como ya hemos estudiado, la función del conjunto de validación es estimar la curva de test para evitar sobreajuste. En estas tres gráficas podemos ver claramente como la curva verde (validación) estima mejor la curva roja (test) a medida que el conjunto de validación es mayor.

La siguiente tabla muestra la mediana de los errores:

% valid	Error %
5	0.087282
25	0.088184
50	0.112632

Lo cual parece indicar que, para este ejemplo en particular, tener datos de validación para evitar el sobreajuste no ayuda demasiado porque quedan muy pocos puntos en training. Si bien se estima la curva de test mejor, el error es muy alto.

4. Ejercicio D

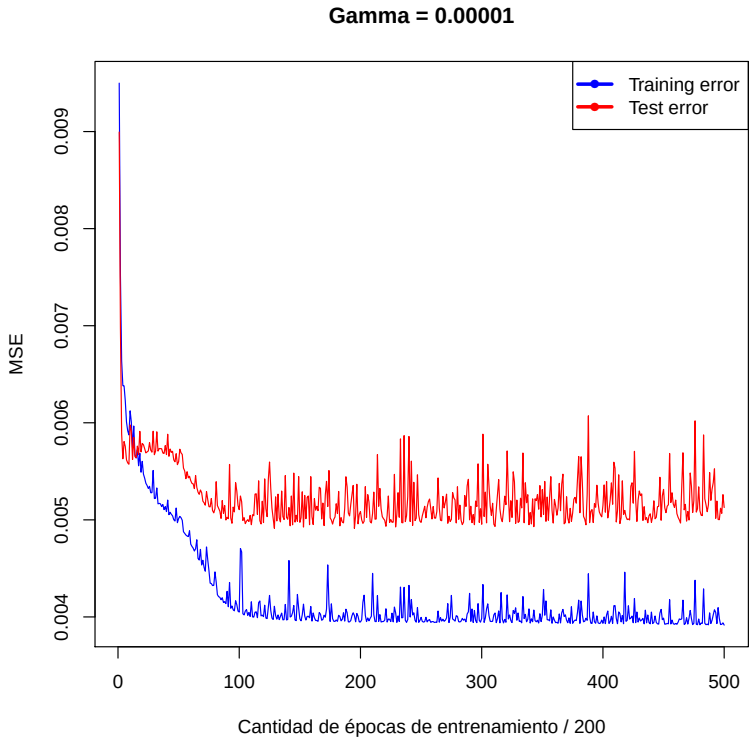
Realicé las modificaciones en el algoritmo *bp* indicadas en el libro (ver código, hay comentarios aclarando dónde modifiqué) para implementar weight decay.

Luego, al igual que los ejercicios anteriores, entrené redes neuronales variando un parámetro γ para resolver el programa de regresión Sunspots. En este caso, el parámetro del weight decay varió entre 10^0 y 10^{-8}. Recordemos que no se usaron datos de validación.

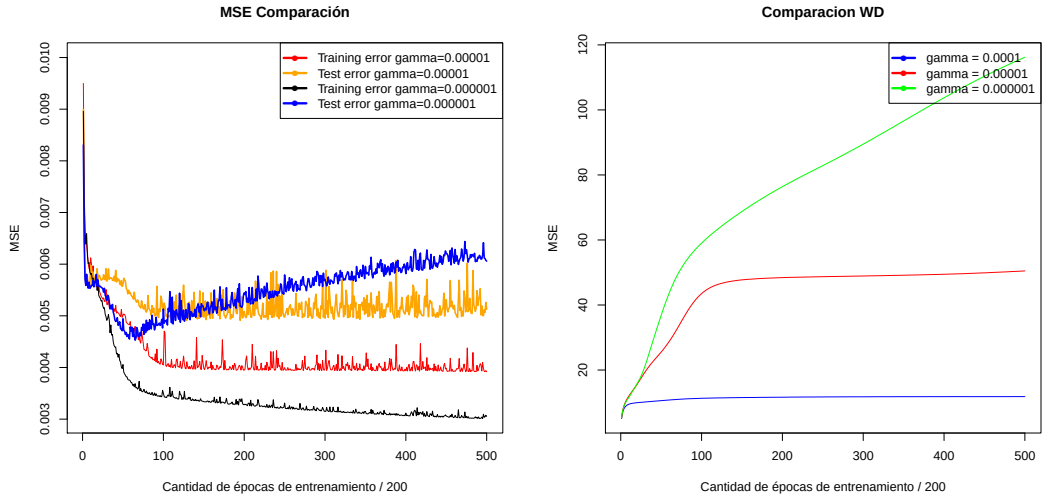
Se entrenaron 20 redes para cada valor de γ y elegí la mediana como red representativa. La siguiente tabla muestra los resultados:

γ	Error
0.0000001	0.006335
0.000001	0.006058
0.00001	0.005127
0.0001	0.006636
0.001	0.013038
0.01	0.035926
0.1	0.040284
1	0.055126

Como vemos, este problema se beneficia más de un valor $\gamma = 0{,}00001$ pues minimiza el error y evita el sobreajuste: vemos que la curva de error de test y la curva de error de training se comportan de forma similar, además la curva de test se mantiene no creciente.



Veamos esta misma gráfica comparada con otro valor de γ (izquierda), y el comportamiento de los valores de weight decay comparando ambos valores de gamma:



Vemos que con el otro valor de gamma la red sobreajusta. En general, viendo todas las gráficas, puedo conlcuir que usando valores de gamma muy pequeños la red sobreajusta y el errore de test crece rápidamente para este problema (además no tenemos conjunto de validación). Sin embargo, usar gamma muy grandes conducen a redes muy rígidas donde el error de training es mayor al error en test.

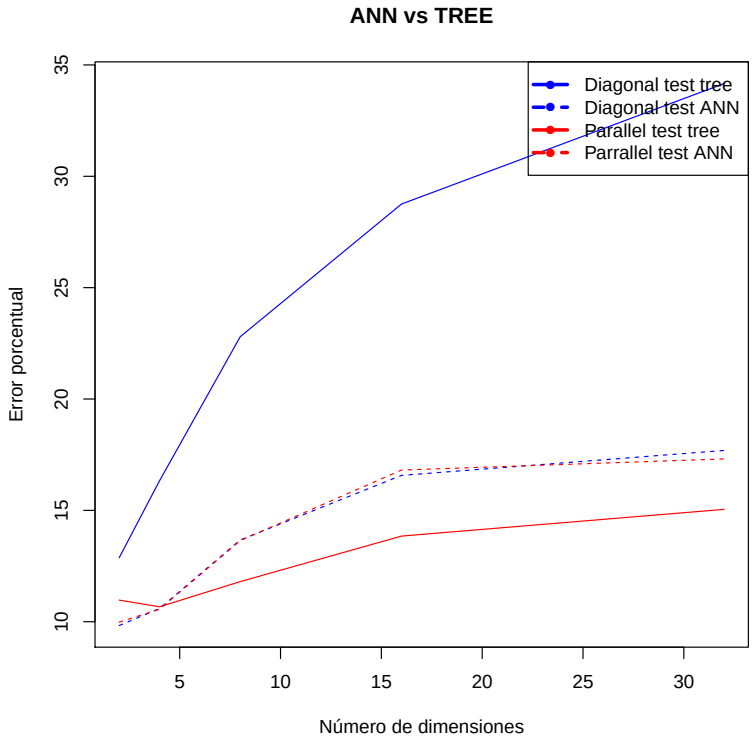
OBS! El rótulo correcto del eje y del segundo gráfico es decay, no MSE. Esta gráfica muestra el término agregado al error ANTES de multiplicarlo por γ.

El código está en la carpeta D y las modificaciónes estan precedidas por un comentario que dice AGREGADO.

5. Ejercicio E

Repetiré el ejercicio de la práctica anterior donde se comparaba el problema paralelo con el diagonal variando la cantidad de dimensiones. Para cada número de dimensiones de los problemas diagonal y paralelo, entrenaré 20 redes neuronales y elegiré la correspondiente a la mediana del error discreto como representante.

Además compararé los resultados obtenidos en este trabajo con los obtenidos en el anterior usando árboles.



Vemos que para abordar el problema diagonal es mucho mejor utilizar una red neuronal, ya que como vimos antes los árboles de decisión requieren mucho trabajo y árboles muy grandes para representar este problema. Por otro lado, para abordar el problema paralelo es un poco mejor usar árboles, ya que el problema puede ser simplemente expersado por un árbol de decisión. De todos modos (en el caso paralelo) la red neuronal no muestra una gran diferencia respecto al árbol, a diferencia del caso diagonal.

6. Ejercicio F

Tuve dos ideas distintas para abordar este ejercicio. Ninguna de las dos dio buenos resultados para el dataset faces, pero voy a exponerlas igual.

6.1. Usando una sola red neuronal

Una primera idea que tuve fue, dado el input (.data .net y .test); modificar los archivos .data y .net para que en vez de tener un solo output, tuvieran n donde n es la cantidad de clases distintas del problema.

Escribí un programa en c que modifica los archivos .data y .net para, entonces, agregar una secuencia de ceros y unos reemplazando una linea como la siguiente:

```
0.45 1.34 -0.12 5
```

Donde el problema tiene clases 0,1,2,3,4,5 y 6 ; por la siguiente:

```
0.45 1.34 -0.12 0 0 0 0 0 1 0
```

Luego, en el archivo .net se modifica la cantidad de neuronas en la capa de salida; se entrena la red neuronal modificada y se lee el .predic devuelto.

Para ese .predic, símplemente elegí como clase final de cada punto la clase correspondienete a la columna de mayor valor.

Calcular el error discreto consistió en sumar las predicciones erradas y dividir por el total de elementos de test.

Este enfoque, sin embargo, parece no ser bueno porque si bien para el dataset iris cometió 0 % de error; par el dataset faces los resultados eran del orden del 70 % de error.

El código se encuentra en la carpeta F1. Hay un archivo .txt explicando brevemente como está organizado.

6.2. Usando muchas redes neuronales

Otro intento consitió en, dados los inputs (por ejemplo iris.data, iris.net e iris.test) donde hay n (3) clases, crear un .data y un .net por cada clase.

El archivo iris.2.data por ejemplo, contiene los mismos puntos pero cambian las clasificaciones: si el punto es de clase 2, entonces se le modifica la clase a 1. Si el punto es de clase distinta de 2, se le modifica a la clase a 0.

De este modo, entrené una red neuronal para cada clase de tal forma que cada red decidiera si un determinado punto está en una clase fija o no.

En base a esto, obtuve un archivo .predic por cada clase y clasifiqué a cada punto con la clase cuya predicción para ese punto fue mayor.

No sé si este enfoque es equivalente al anterior que mencioné, pero los resultados fueron casi iguales: para iris 0 % de error y para faces 70 % de error.

El código y predicciones se encuentran en la carpeta F2 y hay un .txt explicativo.

6.3. ¿Por qué no funcionan?

Revisando ejemplos a mano pude ver que estas redes, en el dataset faces, por algún motivo devuelven a casi todos los puntos un valor constante. De ahí proviene el 70 %, por pararse en una clase particular y errarle con probabilidad 3/4. No pude seguir intentando resolverlo porque ya es momento de entregar el trabajo.