Neurocomputación - Práctica 2

Introducción a las Redes Neuronales Artificiales

Jorge Arellano y Pablo Marcos

1. Detalles de implementación.

En esta práctica hemos programado en Python una red multicapa con una regla de aprendizaje de backpropagation. Esta red es capaz de aprender cualquier patrón de entrada-salida, por lo que es una herramienta muy potente para aprendizaje automático.

Para implementarla, hemos creado la clase *PerceptronMulticapa*, que hereda de la clase abstracta *RedNeuronal*, creada en la práctica anterior. Esta incluye los métodos típicos de cualquier red neuronal: fit (entrenar), evaluar y otras funciones para su representación.

El constructor del perceptrón recibe un array, en el que se especifica el número de neuronas que tiene cada capa oculta. Así, nuestra implementación permite una cantidad variable de capas ocultas, con tantas neuronas cada capa como queramos. También recibe la función de activación, y la expresión de su derivada en función de esta función de activación. También se puede especificar el tipo de normalización a aplicar, “bipolar” para hacer una transformación afín y estandarizar los datos a [-1, 1], “normal” para normalizar los datos para que tengan media 0 y desviación estándar igual a 1 o en caso de no especificar se utiliza la codificación de los datos leída.

Así, nuestra implementación trabaja con matrices (numpy.matrix), aplicando sobre ellas productos matriciales, escalares o coordenada a coordenada, según convenga.

La red neuronal puede ser ejecutada con el script *retro.py*, utilizando la opción -h se mostrará información del uso de los comandos, pero su uso básico es

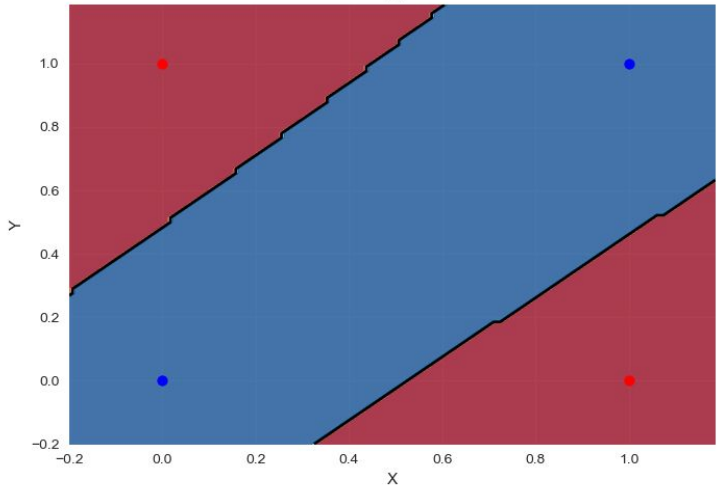
- retro.py data/fichero.txt: Utilizará un fichero como datos de train y test (equivalente a retro.py data/fichero.txt 100)
- retro.py data/fichero.txt 80: Utilizará un 80% (O el porcentaje especificado para el entrenamiento)
- retro.py data/train.txt data/test.txt: Ficheros de train y test diferenciados.

Los comando opcionales para configurar el entrenamiento serán:

- -l : Neuronas en cada capa oculta, por ejemplo, -l 10 10 2.
- --learning: Tasa de aprendizaje.
- --ecm: Criterio de parada utilizando como umbral el error cuadrático medio.
- -o: Fichero de volcado de datos.
- --normalize: Normalización de datos, “normal” o “bipolar”.
- -e: Número de épocas máximas de entrenamiento.

2. Comprobación de su funcionamiento

En primer lugar, para comprobar el correcto funcionamiento del perceptrón, hemos probado que fuera capaz de aprender el problema xor, el cual no era separable linealmente y no podía ser aprendido por el perceptrón sin capas ocultas. Mostramos la frontera de decisión para este problema utilizando una capa oculta, en la cual se observa que el perceptrón ahora es capaz de resolver el problema.

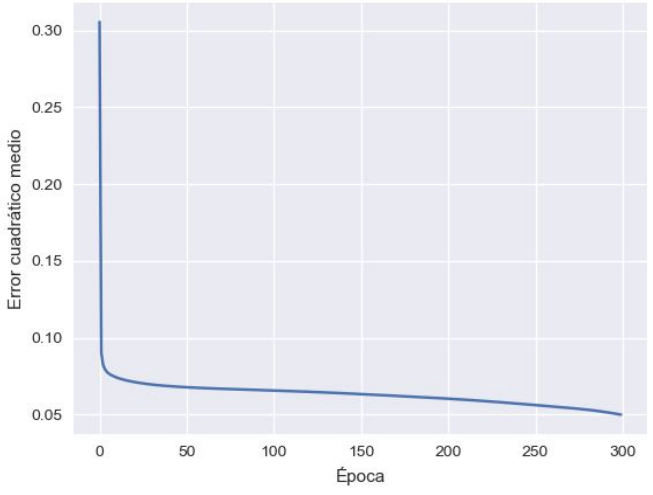


Tras estos estudiaremos los diferentes problemas reales proporcionados. En todos los casos hemos utilizado capas ocultas de 100 neuronas.

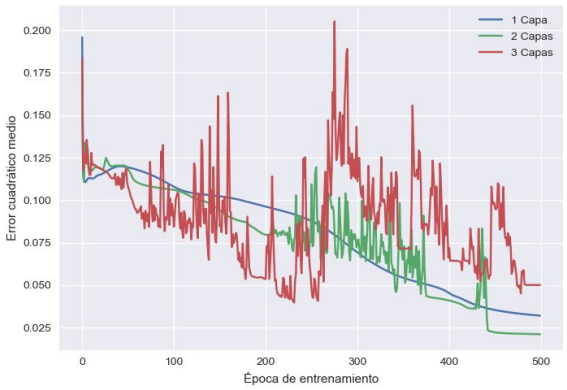
- **Problema real 1**

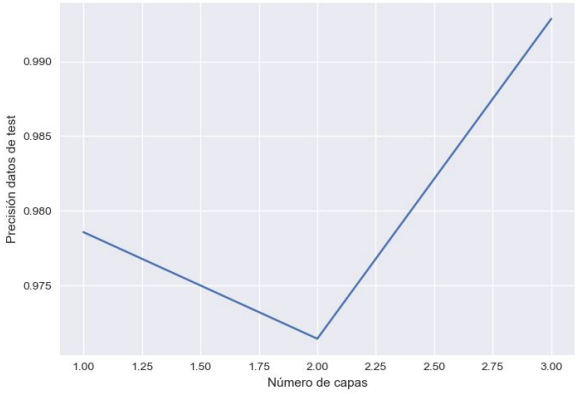
En primer lugar estudiamos la clasificación con una única capa oculta con 10 neuronas, utilizando un 80% de los datos para train y el 20% restante para validación, y sin realizar ningún tipo de normalización.

En la gráfica vemos como el error cuadrático medio desciende muy rápidamente, obteniendo un 98% de precisión en entrenamiento y un 92% con los datos de test.



A continuación estudiaremos si con más capas y más neuronas podemos conseguir mejores resultados, usaremos una, dos y tres capas con 100 neuronas cada una,

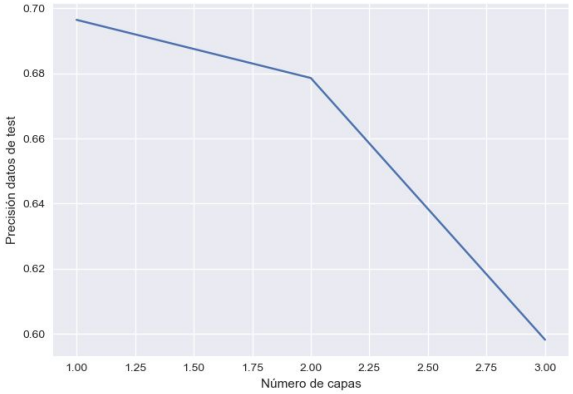


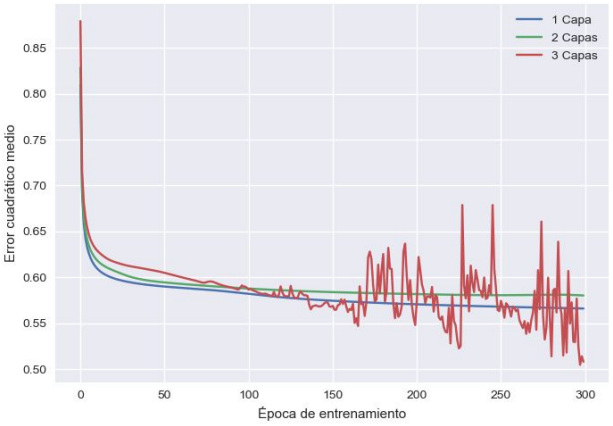


Observamos que, en cuanto a error cuadrático medio, este tiene más oscilaciones cuánto más capas ocultas hay. Esto se debe al crecimiento de la complejidad de la red, y por tanto del entrenamiento, para reducir estas oscilaciones sería necesario disminuir la constante de aprendizaje. La red con tan solo dos capas ocultas para las 500 épocas de entrenamiento es la que consigue un menor error cuadrático medio, sin embargo, se logra la mejor precisión con 3 capas, superando un 99% en test.

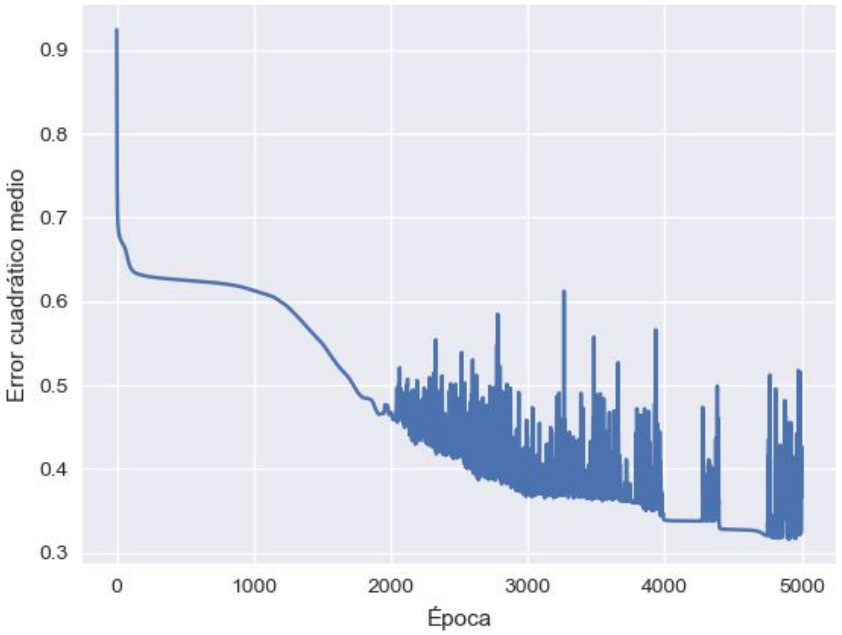
- **Problema real 2:**

Conseguimos los mejores resultados con 1 capa. Observamos que al aumentar el número de capas se reduce el ecm pero se produce overfitting y disminuye la precisión sobre los datos de test.



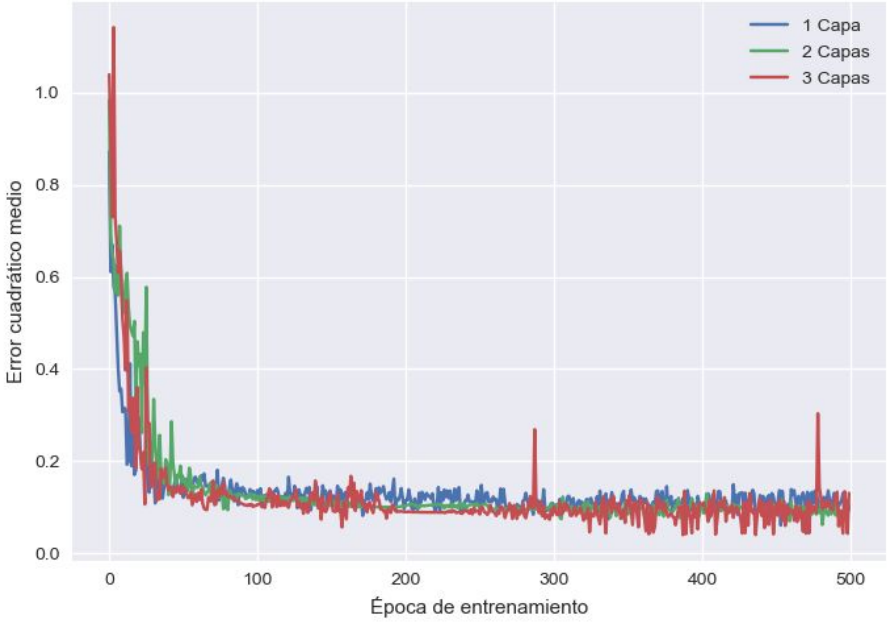


Probamos ahora a entrenar la red con una sola capa oculta durante 5000 épocas. Sin embargo obtenemos una disminución de la precisión de los datos de test, debido al *overffiting*.



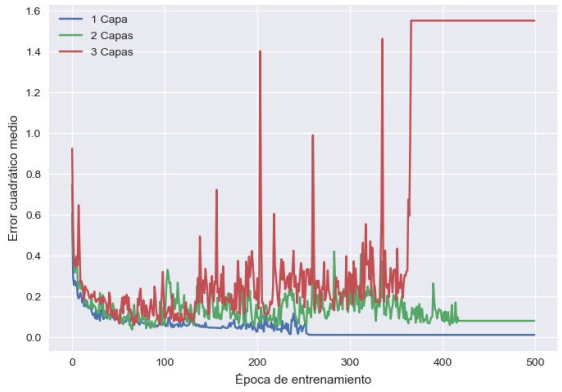
- **Problema real 3**

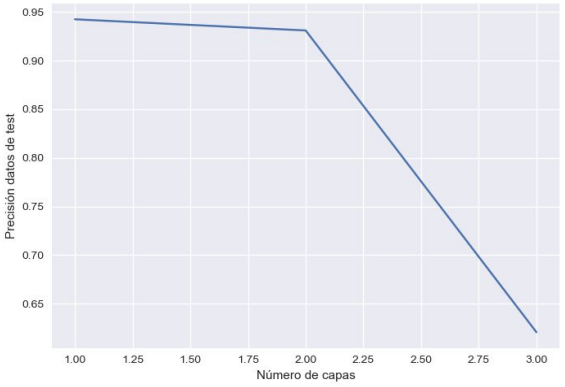
A continuación mostramos el error cuadrático de la red en entrenamiento con 1,2 y 3 capas ocultas. En todas se consigue un 100% de acierto sobre el 20% de datos de test. Deducimos que este problema es algo más fácil de resolver para una red neuronal.



- **Problema real 5**

Entrenados con 1,2 y 3 capas ocultas con 100 neuronas, el mejor resultado se logra para una capa oculta (99% de acierto para test y 90% para train). Este hecho probablemente se deba a un overfitting de la red, ya que al principio el error cuadrático medio es similar para los tres casos, pero hay un momento en el que empieza a dispararse, especialmente para 3 capas ocultas, y estabilizandose en un valor más alto del deseado. Otra posibilidad es que este problema sea muy difícil para la red y se necesitan más épocas con más capas para conseguir los mismos resultados, pero consideramos esto menos probable.





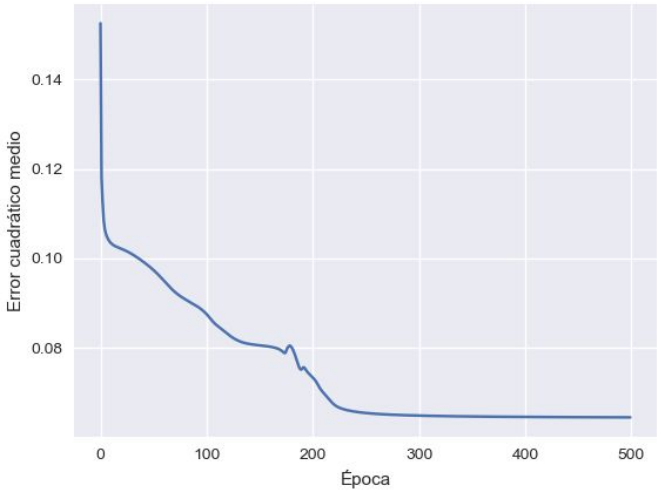
3. Normalización

Observamos que, al ejecutar los problemas reales 4 y 6 como hacíamos los otros, los resultados que obtenemos son pésimos… en torno a un 60% de acierto, que es casi el mismo acierto que tirar una moneda al aire para clasificar, y además nuestras funciones de activación eran saturadas en repetidas ocasiones. Observamos que los datos no están normalizados, y entonces algunos atributos ponderarán más que otros simplemente por estar en otro rango de valores. Así, decidimos normalizar, y a continuación exponemos nuestros resultados.

- **Problema real 4**

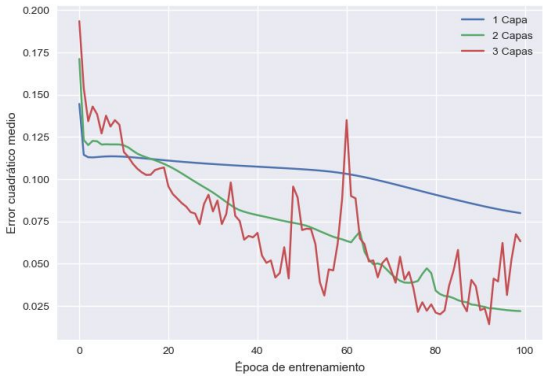
Primero, comprobamos con una capa oculta que la red, tras normalizar los datos (escalados en el intervalo [-1, 1]) pero usando los mismos parámetros, sí que aprende. Obtenemos un 97% de precisión en los datos de train, un 96% en los datos de test y un error cuadrático medio en entrenamiento de 0.06, que son resultados bastante más prometedores que con los que contábamos en un inicio. Obteniendo la siguiente matriz de confusión

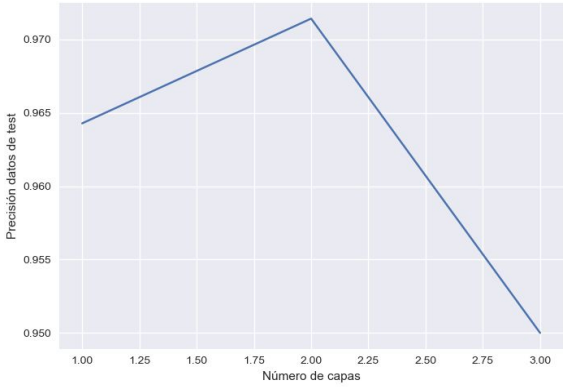
137	**3**
3	**137**



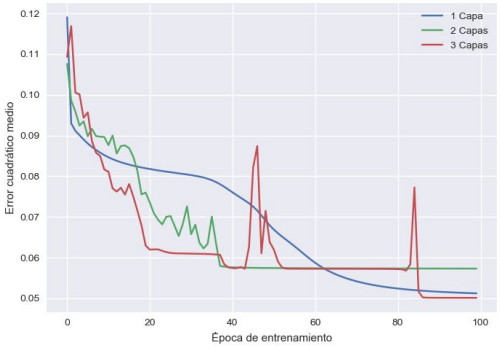
Realizamos la comparación de resultados ahora con normalización min-max, con estandarización a una distribución normal (0,1), y sin normalización. También variamos el número de capas ocultas entre 1, 2 y 3, como hacíamos antes. Las resultados obtenidos son los siguientes:

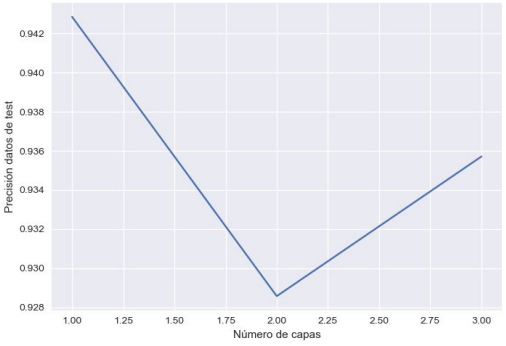
- **Con normalización min-max**



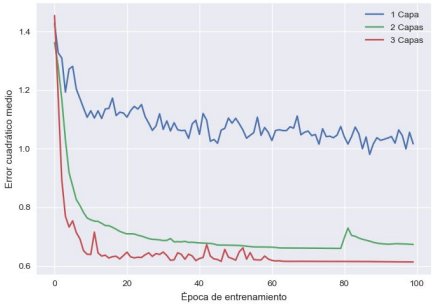


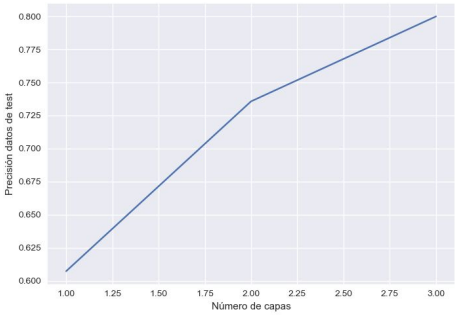
- **Con estandarización media 0 y std 1**





- **Sin normalización**





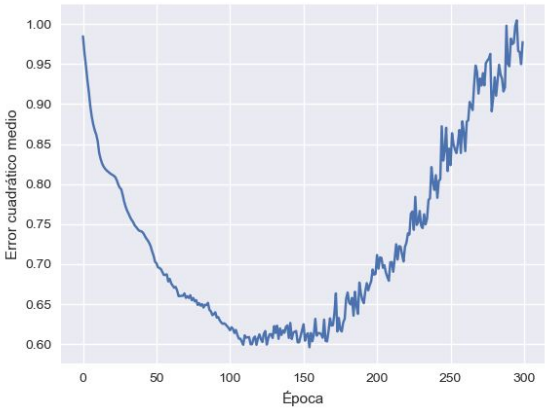
Hemos obtenido los mejores resultados para una normalización min-max con dos capas ocultas. Ha de considerarse que sólo hemos entrenado durante 100 épocas, con lo que con entrenamientos más largos, puede ser que la red con 3 capas ocultas acabe haciéndolo mejor. O quizás acabe sobreajustándose.

- **Problema real 6**

Este problema nos ha dado una serie de dificultades. En primer lugar, al tratarlo sin normalizar, nos encontrábamos un overflow. Así, tuvimos que realizar algún ajuste en la red neuronal para tomar precauciones ante este inconveniente. Así, al entrenar la red sin

normalizar, volvíamos a encontrarnos una precisión pésima (en torno al 60% de nuevo), lo que nos hizo tomar la decisión de normalizar.

Entrenamos la red con dos capas ocultas, de 100 neuronas cada una, durante 300 épocas solo, ya que al haber tantos datos cada época tardaba bastante tiempo.
La evolución del error cuadrático es la siguiente:



Como vemos, parece que volvemos a estar en un caso de sobreajuste. La precisión conseguida ha mejorado notablemente (un 77% en train y un 75% en test), aunque siguen sin ser prometedores. Con más capacidad de procesado, podríamos haber añadido más capas, que con el suficiente número de épocas, seguramente se hubiera conseguido mayor precisión.

4. Conclusiones

En esta práctica hemos experimentado la gran mejora que consiguen las redes multicapa sobre las monocapa, ya que son capaces de clasificar, en mejor o peor medida, cualquier dataset, a costa de más tiempo en el entrenamiento, y son capaces de resolver problemas que no son separables linealmente. En general, observamos que, a la larga, cuantas más capas y más neuronas utilizamos, a mejores resultados aspiramos. Sin embargo, el añadir más capas se potencia el riesgo de sobreajustar la red. Llegamos a la conclusión de que la mejor estrategia puede ser utilizar un gran número de capas ocultas, y probar con distintas inicializaciones de los pesos. También, sería buena idea guardar los pasados 100 pesos durante entrenamiento, para que si observamos que a partir de cierto punto la precisión empieza a descender, nos quedamos con la configuración de pesos en ese punto, lo que equivaldría a dejar de entrenar antes de sobreajustar. Sin embargo, en ocasiones ya se consiguen resultados prometedores tan solo con una capa (que, teóricamente, puede conseguir los resultados óptimos con suficiente entrenamiento). Así, dependiendo del problema, del tiempo de procesado disponible y de la precisión perseguida, habrá que decidir qué topología dotar a la red.