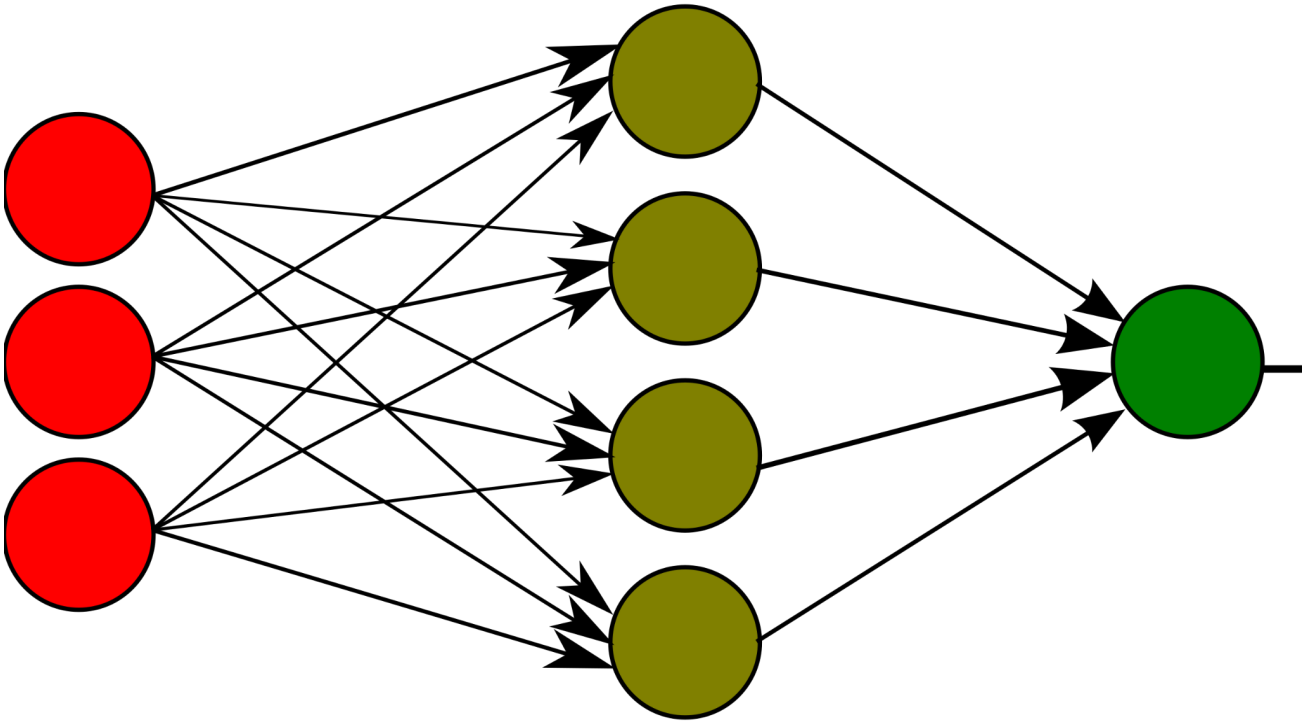
Perceptrón Multicapa



Práctica 2

Alejandro Benimeli Miranda y Rodrigo Juez Hernández

UAM
UNIVERSIDAD AUTONOMA
DE MADRID

Escuela Politécnica Superior

ALGORITMO DE RETROPROPAGACIÓN

Descripción algoritmo de retropropagación

En primer lugar hemos creado una función de feedforward que usamos tanto en la ejecución del entrenamiento con la retropropagación como para calcular las predicciones como para calcular el ECM y accuracy.

Después de realizar el feedforward para una entrada, calculamos los deltas de cada salida y los metemos en el vector “deltas”, y actualizamos los pesos de la capa de salida inmediatamente ya que hemos guardado los pesos originales en la variable “peso_anterior” de la conexión y se podrán usar en el resto del proceso.

Tras realizar el cálculo para la capa de salida hemos abstraído el paso de la retropropagación para un número de capas ocultas ilimitado. Esto lo hemos hecho con un bucle que va por todas las capas ocultas calculando los deltas_in, (y deltas a partir de estos deltas_in) y finalmente el cambio de peso de cada conexión de las capas ocultas.

Tras realizar toda la retropropagación y el cambio de pesos tenemos una sección que es evaluar el rendimiento de la red, tanto para graficar más tarde los rendimientos como para, en caso de que haya partición de validación evaluar la condición de parada.

La condición de parada consiste en un sistema que calcula el accuracy en la partición de validación y usa una “paciencia”. Esa “paciencia” es la cantidad de iteraciones a realizar desde que se obtuvo la máxima puntuación en la partición de validación. De esta manera si tenemos una “paciencia” de 10 y en la iteración 9 obtenemos el máximo accuracy y luego baja o se mantiene eso significa que en la iteración 19 se corta el entrenamiento. Cuando se corta el entrenamiento se restauran los pesos de la iteración que obtuvieron el máximo accuracy.

Lo hemos diseñado para que si el entrenamiento no recibe partición de validación simplemente se ejecuten todas las iteraciones marcadas (el máximo).

Este algoritmo está diseñado para un número de capas ocultas variables e ilimitadas, permitiendo incluso tener un número diferente de neuronas en cada capa oculta.

Se han implementado funciones específicas de la función de activación sigmoide bipolar y su derivada.
Por cada ejecución en cada iteración se imprime el ECM, el accuracy sobre el train, el accuracy sobre la validación (si se ha usado validación) y la matriz de confusión para esa época, estos datos se usarán para graficar los experimentos de los siguientes ejercicios.

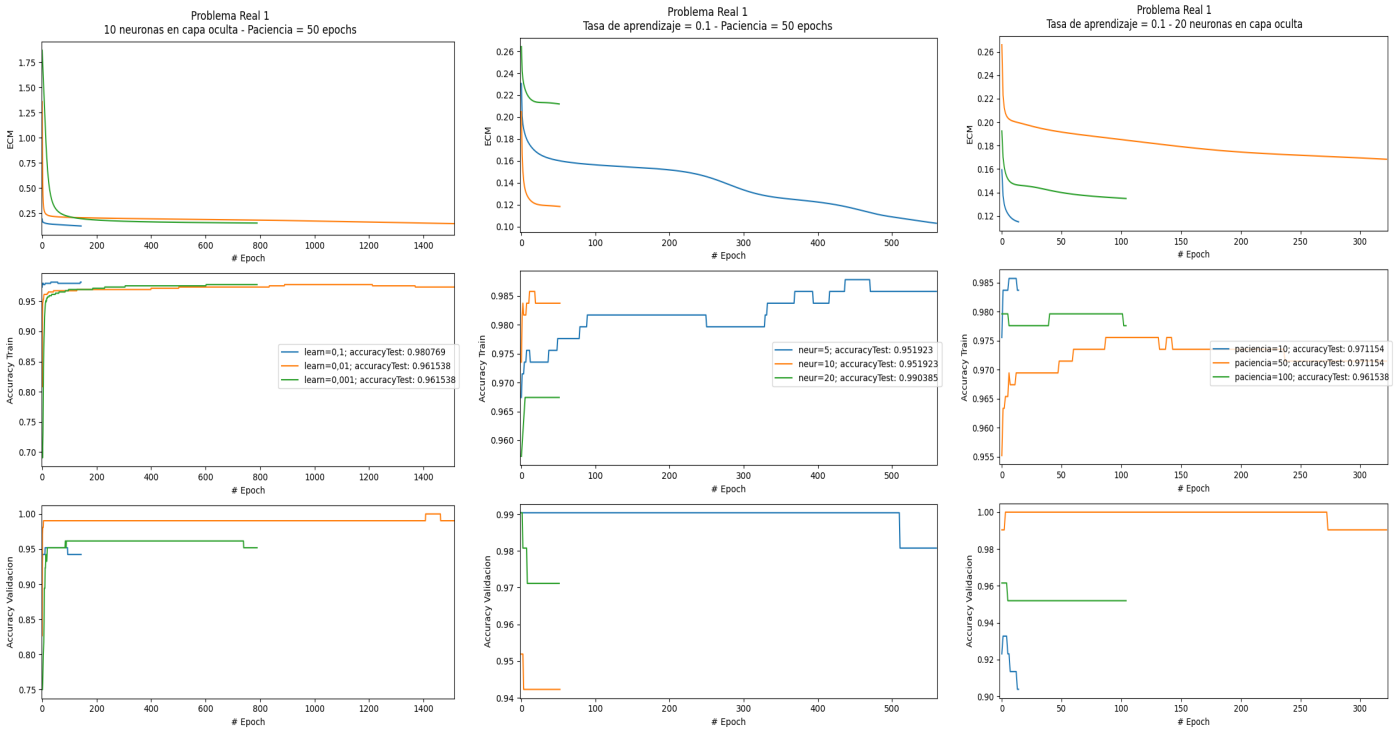
Problemas Reales

En primer lugar se ha definido un procedimiento experimental común para todas los problemas reales.
Se ha definido un split del 70% para entrenamiento, 15% validación y 15% test. La puntuación final se obtiene en la partición del test la cual es completamente independiente mientras que la partición de validación se usa para obtener una evolución del accuracy a lo largo de las épocas y para evaluar la condición de parada.

Las pruebas incluyen primero probar el mejor learn rate, luego con ese learn rate comparar la cantidad de neuronas en la capa oculta (aunque hemos implementado que se puedan usar varias capas hemos decidido fijar una única capa oculta y variar la cantidad de neuronas) y finalmente la paciencia de la condición de parada.

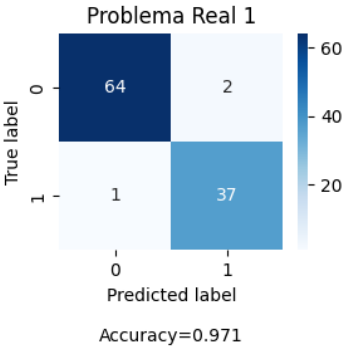
Para cada problema real también mostramos una matriz de confusión, con su accuracy, este es el resultado de elegir los 3 mejores hiper parámetros obtenidos de las pruebas.

Problema Real 1



Los mejores hiper parámetros para el problema real 1 fueron:

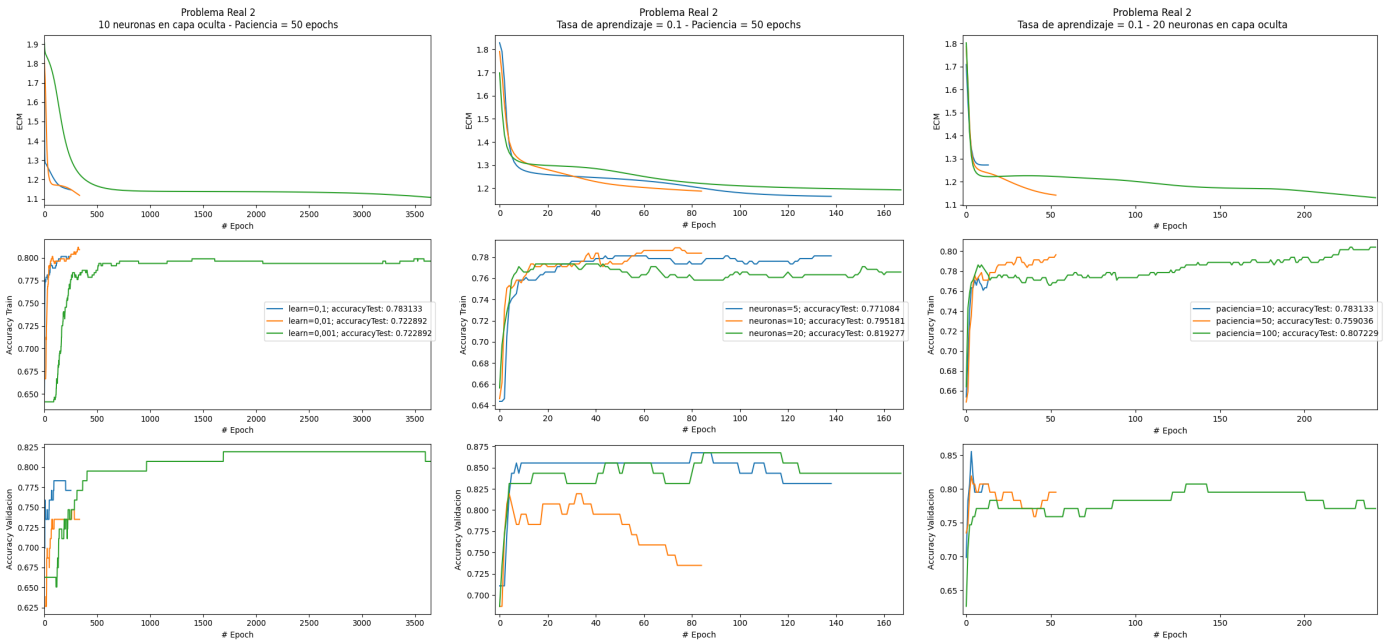
- Learn Rate: **0.1**
- Neuronas en Capa Oculta: **20**
- Paciencia de condición de parada: **50**



Se puede ver como un learn rate mayor hace que el entrenamiento converge mucho más rápido, esto es debido a que los pasos son mayores y por eso la condición de parada se activa mucho antes ya que empieza a bajar la validación más rápido que los demás.

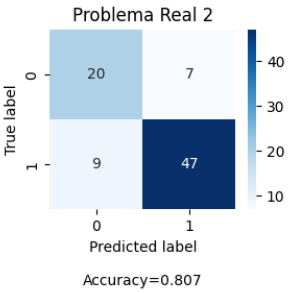
El mejor resultado se obtuvo también con 20 neuronas y además no provoca overfitting. Esto se puede ver porque la validación no empeora al mejorar el accuracy sobre train.
En cuanto a la condición de parada se ha elegido 50 ya que aunque tanto 10 como 50 tienen el mismo accuracy en test 50 mejora sustancialmente en validación, esto puede ser porque la validación de 10 provoca que se corte demasiado pronto y no deja entrenar tanto como la paciencia de 50. Sin embargo, la de 100 simplemente empeora inicialmente y luego se mantiene, aunque hubiese sido 50 sucedería lo mismo, este es el problema de la aleatoriedad en la inicialización. Aún así consideramos que la paciencia de 50 épocas es un buen término medio.
Respecto a la matriz de confusión sobre el test vemos que apenas comete errores (solo 3).

Problema Real 2



Los mejores hiper parámetros para el problema real 1 fueron:

- Learn Rate: **0.1**
- Neuronas en Capa Oculta: **20**
- Paciencia de condición de parada: **100**

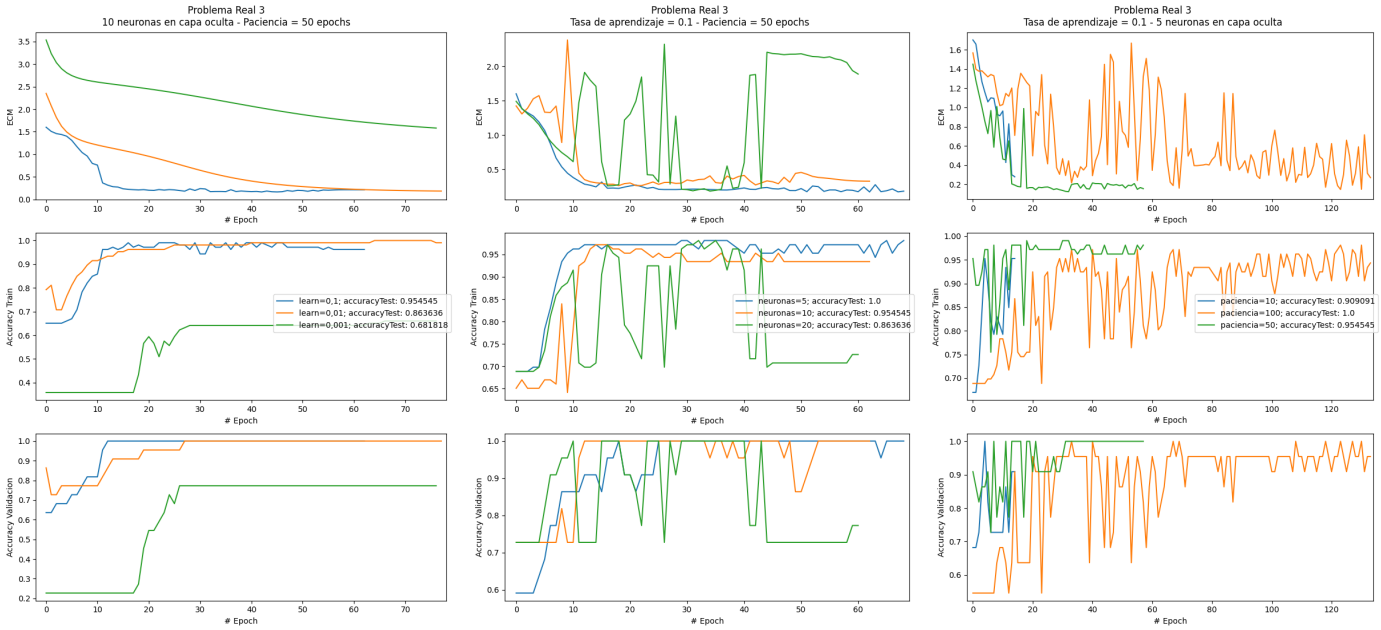


En este caso la learning rate de 0.1 vuelve a ganar, sin embargo esta vez no es de las que más rápido converge, pero es la que mejor accuracy en validación consigue, esto puede ser porque otras learning rates más pequeñas se queden en mínimos locales. Respecto al número de neuronas, se eligió 20 por la razón de que obtiene el mejor resultado en el test, además vemos que generaliza muy bien porque tiene el peor accuracy en train, pero el mejor en validación y test.

La paciencia es la más interesante de observar ya que se puede ver justamente que un mayor número de iteraciones provoca overfitting bajando el accuracy en la validación y aumentando demasiado en el train, pero gracias al sistema de condición de parada que hemos implementado cuando vemos que ha estado bajando prolongadamente restauramos los pesos que obtenían el mayor accuracy sobre la validación antes de empezar a crear overfitting y por eso generaliza tan bien en la partición de test.

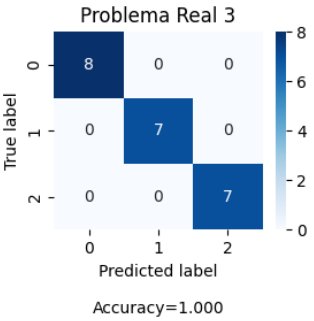
El accuracy final sobre el test es de 0.807, lo que es un buen resultado. Los errores están balanceados ya que hay más o menos los mismos falsos positivos que falsos negativos, aunque las clases positivas y negativas en la partición de test no estaban bien balanceadas.

Problema Real 3



Los mejores hiper parámetros para el problema real 1 fueron:

- Learn Rate: **0.1**
- Neuronas en Capa Oculta: **5**
- Paciencia de condición de parada: **100**

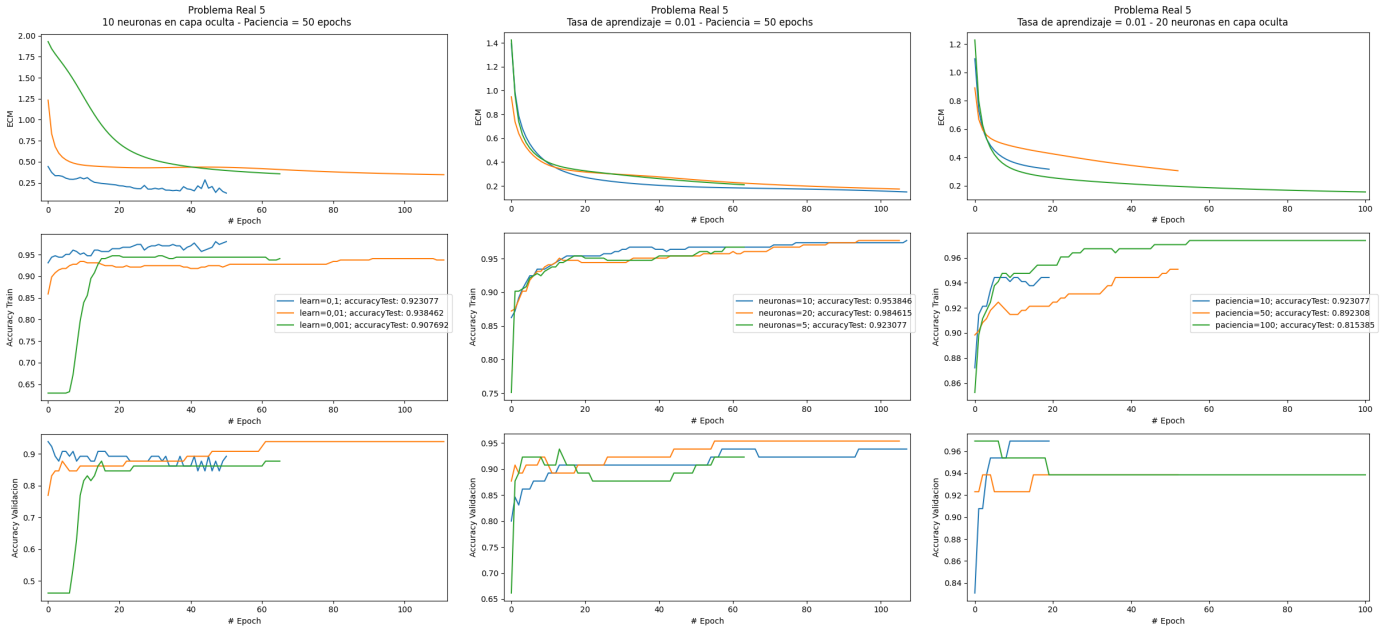


El mejor learn rate de nuevo es el más alto, de hecho vemos que 0,001 obtiene un accuracy muy bajo y ECM alto, y se estanca. Seguramente se haya alcanzado un mínimo local y no sea capaz de salir de él. Respecto a la cantidad de neuronas esta vez vemos que 20 neuronas tiene un comportamiento muy errático y da accuracies muy bajos tanto en la validación como en el entrenamiento. Con 10 neuronas se estabiliza un poco más pero no consigue tan buenos resultados como 5 neuronas, está claro que este problema necesita mucha generalización.

Este comportamiento tan aleatorio también hace que la mejor opción de la paciencia sea la más alta ya que con tantas subidas y bajadas con una paciencia muy baja podemos no llegar al máximo rendimiento debido a que se corte demasiado pronto cuando podría haber un pico incluso mayor. Por eso el accuracy en el test es directamente proporcional al tamaño de la paciencia.

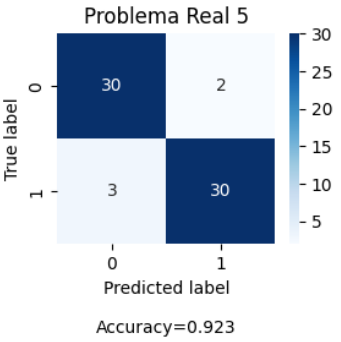
Además como muestra la matriz de confusión, la accuracy sobre el test da perfecta y no clasifica erróneamente ningún dato.

Problema Real 5



Los mejores hiper parámetros para el problema real 1 fueron:

- Learn Rate: **0.01**
- Neuronas en Capa Oculta: **20**
- Paciencia de condición de parada: **10**



Parece que la red neuronal entiende muy bien este problema ya que la evolución es muy suave, el mejor learn rate es el intermedio con 0.01, esto probablemente sea porque 0.001 no es capaz de escapar de algunos mínimos locales y 0.1 mueve demasiado los pesos. Este problema además requiere de una frontera de decisión muy compleja ya que se requieren 20 neuronas, en el entrenamiento son todos excelentes sin embargo en validación y test es donde realmente se ve que para obtener buen resultado necesitas más neuronas.

Como tenemos muchas neuronas tiene sentido que para compensar y que no haya tanto overfitting se requiera una paciencia menor.

En este caso la matriz de confusión muestra que es excelente clasificando ambas clases y el error está muy balanceado.

Problemas Reales y Normalización

Normalización de los datos

La normalización que hemos aplicado a los datos se conoce como *Z-Score Normalization*. Esta normalización consiste en transformar los atributos de forma que la media de cada atributo sea 0 y la desviación típica 1. Esta transformación se realiza mediante la siguiente fórmula:

$$x_{norm} = (x - \mu) / \sigma$$

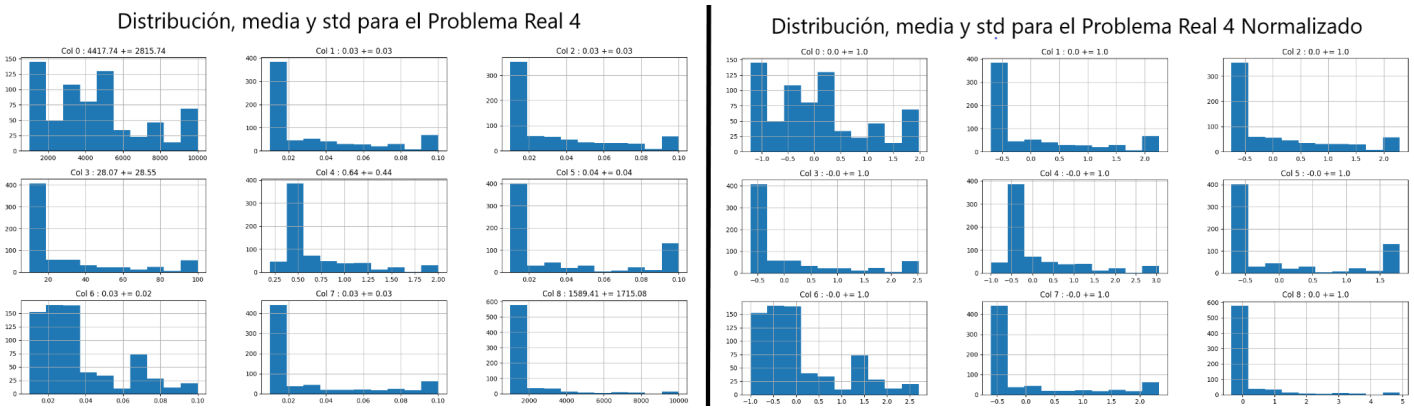
Donde x es el atributo sin normalizar, μ es la media de todos los valores de ese atributo y σ es la desviación típica.

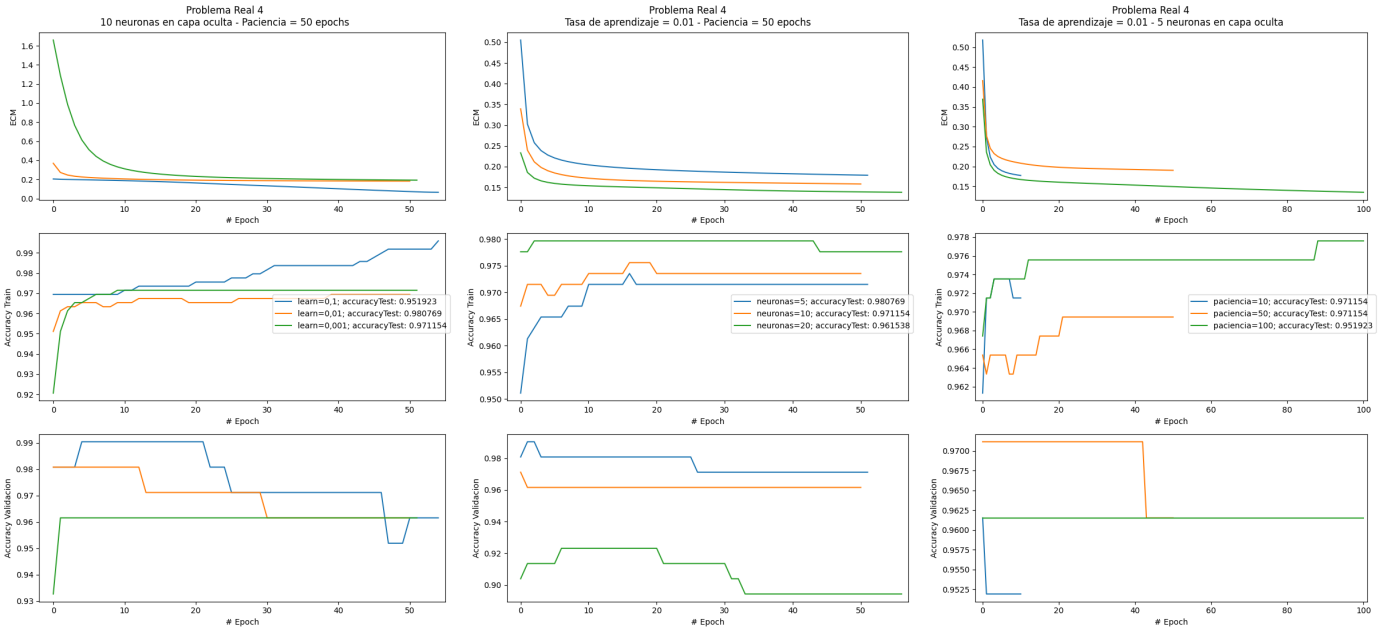
La razón por la que se normalizan los datos es porque ayudan al algoritmo de la retropropagación a converger más rápidamente. Al perceptrón multicapa le cuesta más aprender si hay diferentes atributos con escalas muy diferentes o muy grandes o si los valores de un atributo son todos positivos o todos negativos.

En los apartados del problema real 4 y 6 mostramos un histograma de cada atributo con y sin normalización, además de la media y desviación típica de cada uno. Vemos que varios de estos atributos están centrados respecto a valores muy grandes y/o tienen una desviación muy grande. Al transformar los datos, la escala de los mismos se reduce significativamente y quedan centrados respecto al 0, con una desviación de 1.

En nuestro código, cuando normalizamos los datos de entrenamiento, guardamos las medias y desviaciones de cada atributo, las cuales luego usamos para normalizar los datos de validación y test.

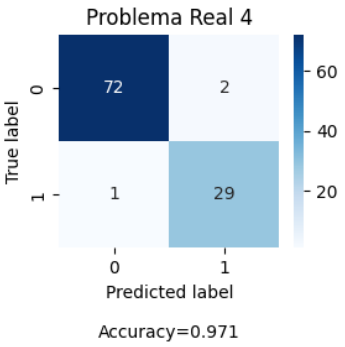
Problema Real 4





Los mejores hiper parámetros para el problema real 1 fueron:

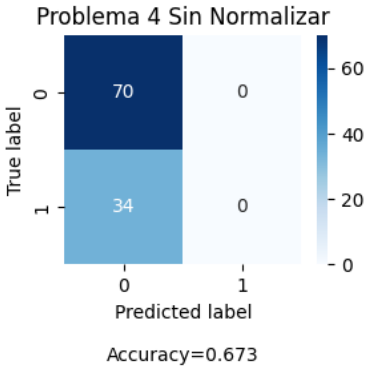
- Learn Rate: **0.1**
- Neuronas en Capa Oculta: **5**
- Paciencia de condición de parada: **50**



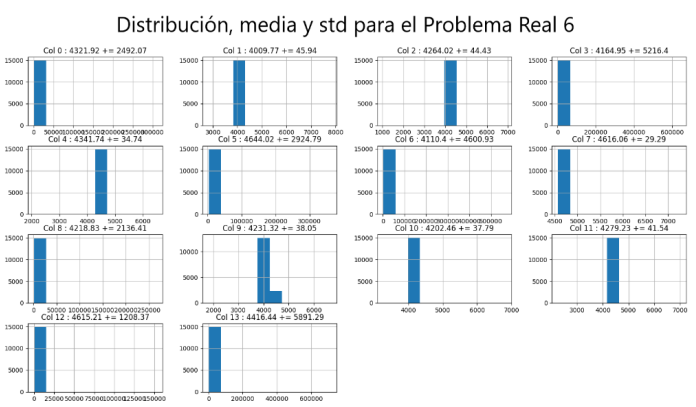
La red aprende rápidamente el problema, incluso con una tasa de aprendizaje pequeña. No parece que requiera de una frontera muy compleja, ya que con tan solo 5 neuronas en la capa oculta obtiene un muy buen accuracy. Vemos que con 20 neuronas en la capa oculta obtiene también buen accuracy sobre train, pero el accuracy sobre la validación decrece. Esto se debe a que se está produciendo overfitting.

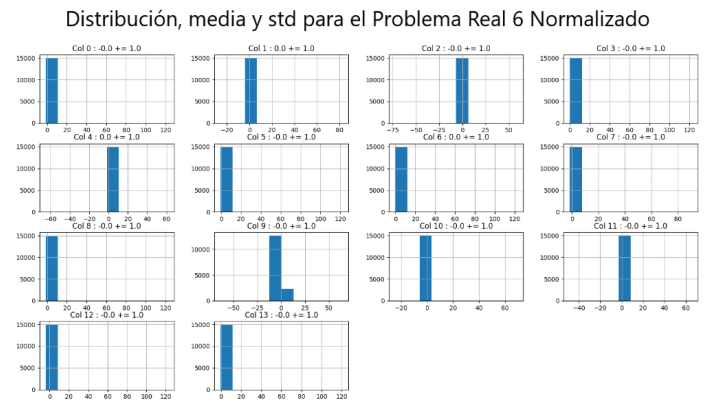
En la matriz de confusión sobre el test se ve que tan solo se cometen 3 errores y están balanceados (hay más o menos el mismo número de false positives y false negatives). El accuracy final sobre el test es de 0.97, lo que es un resultado excelente.

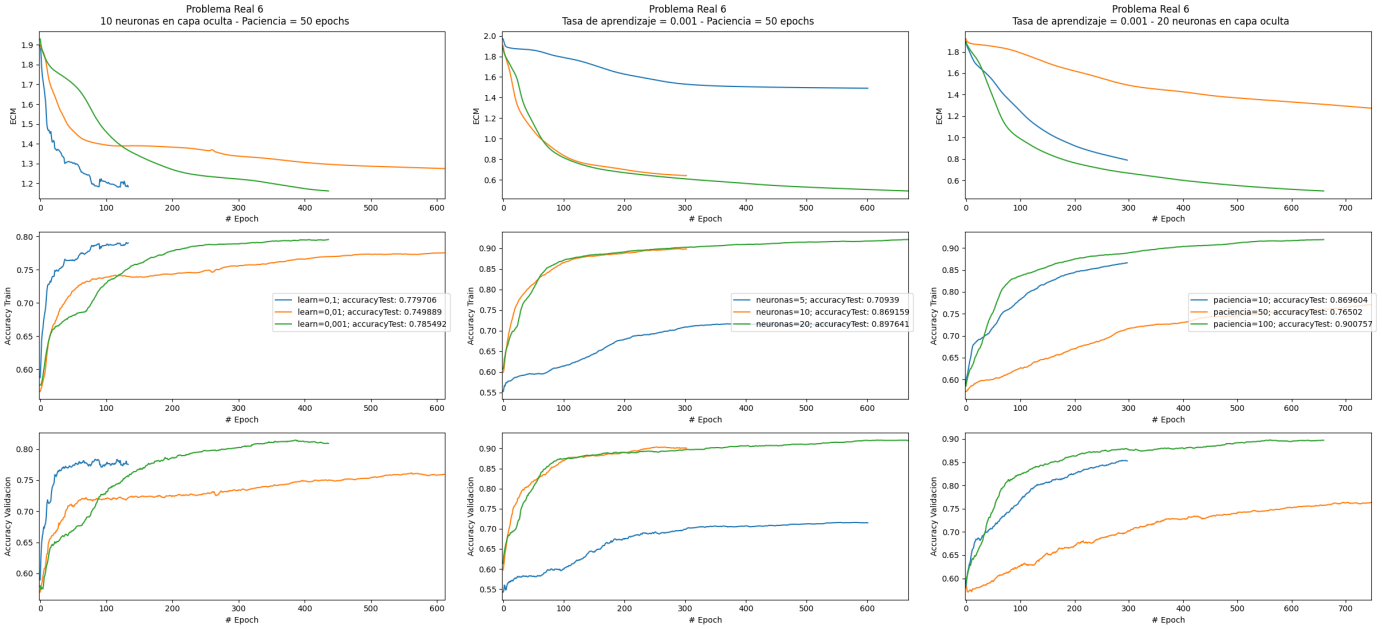
Después de obtener los mejores hiperparámetros, decidimos probar a ejecutar la red con esos parámetros y los datos sin normalizar, para ver si había una gran diferencia en la accuracy o no. Como vemos en la matriz de confusión sobre el test sin normalizar, la accuracy es muy inferior. La razón de esto es que la red ha clasificado todos los datos como *clase 1* y ninguno como de *clase 2*. La normalización de los datos parece haber arreglado este problema, ya que como hemos visto antes, se obtuvo un accuracy de 0.97 normalizando.



Problema Real 6

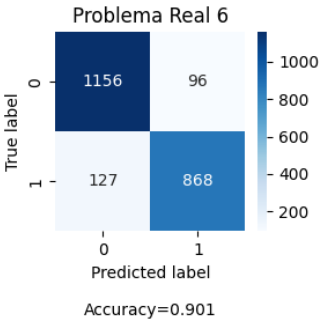






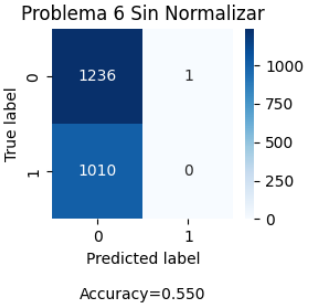
Los mejores hiper parámetros para el problema real 1 fueron:

- Learn Rate: **0.001**
- Neuronas en Capa Oculta: **20**
- Paciencia de condición de parada: **100**



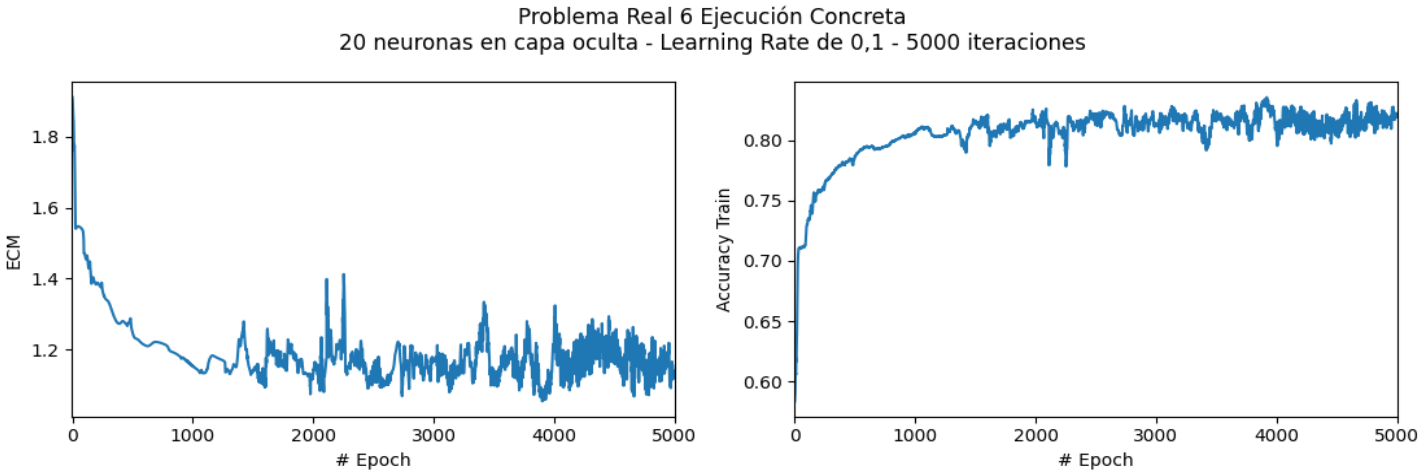
La red aprende mucho más rápido con una learn rate de 0.1, pero en general aprende mejor con 0.001. Con una learn rate de 0.1 es con la que peores resultados se obtienen. En la ejecución variando las neuronas en la capa oculta se obtuvieron mejores resultados con 20, seguido muy de cerca de 10, mientras que con 5 se obtiene un resultado pésimo. Esto parece indicar que el dataset es algo complejo. En ninguno de los casos parece que haya overfitting, porque en ningún momento decrecen las accuracies de validación.

En la matriz de confusión sobre el test se ve que los errores están balanceados (hay más o menos el mismo número de false positives y false negatives). El accuracy final sobre el test es de 0.90, lo que es un resultado bastante bueno.



Al probar los mejores hiperparámetros con los datos sin normalizar, ocurre lo mismo que en el problema real 4: La red aprende a clasificar todo como clase 1. Solo clasifica un dato como clase 2 y lo clasifica erróneamente. El accuracy sobre el test que se obtiene es pésimo.

Problema Real 6 Ejecución Concreta



Para la ejecución concreta del problema real 6 en el ejercicio 3, graficamos la evolución del error cuadrático medio (ECM) y del accuracy sobre el conjunto de train durante las 5000 épocas de entrenamiento de un perceptrón con los hiper parámetros especificados en dicho ejercicio.

Se aprecia cómo el ECM disminuye de forma muy rápida en las primeras 1000 iteraciones. A partir de ese punto la gráfica fluctúa mucho pero no mejora, manteniéndose cerca de 1,2. Algo similar pasa con el accuracy: Crece hasta llegar a 0,81 en las primeras 1000 iteraciones y luego se estanca, fluctuando alrededor de 0.8. Debido a lo poco que mejora, lo ideal hubiese sido cortar la ejecución cuando se llegase a unas 1000 épocas.

Con tantas iteraciones es posible que se esté produciendo overfitting, ya que no se permite a la red acabar antes. Sin embargo, en la matriz de confusión obtenida con la partición de test una vez se ejecutaron las 5000 iteraciones, vemos que este no es el caso. La precisión sobre el conjunto de test es prácticamente la misma que la del entrenamiento.