* 1. Describe el algoritmo de retropropagación implementado:

El algoritmo de retropropagación implementado se trata de una serie de pasos que emula los pasos del implementado en teoría. Este consiste de la iteración múltiple de tres fases diferentes: propagación hacia delante, retropropagación del error y ajuste de pesos. Además, el modelo creado soporta múltiples capas ocultas puesto que la retropropagación tiene sintaxis parecido.

Otros cambios importantes que afecta al rendimiento de la ejecución:

1. Para ahorrar el tiempo de calcular los sigmoides que se tiene un componente exponencial, se ha creado una variable de la clase para almacenar los sigmoides que ya habían sido creados, para ello, se redondea a 5 decimales el valor de la entrada antes de aplicar el sigmoide. Se ha definido esta variable como de la clase porque hacemos múltiples simulaciones usando un script y se aprovecha todos los sigmoides calculados.
2. El modelo soporta múltiples capas ocultas, para ello se debe parar una lista de neuronas para capas ocultas, por ejemplo, si se pasa [2, 3], quiere decir que la primera capa oculta hay 2 neurona y la segunda 3. En el caso de pasar una lista vacía, ósea [], pues no se creará ninguna capa oculta.
3. Para lectura de ficheros, como se usa el sigmoide bipolar, se ha cambiado el código para que sustituya todos los 0 por -1, además, con el flag “norm”, se normalizará los valores de X con una desviación estándar del paquete Sklearn. Esto consiste en restar el valor con la media de todos los valores de esa columna y dividir entre la desviación estándar, es importante destacar que la media y la desviación estándar de cada columna sólo se obtiene a través de los datos de Train, y para los datos de Test, se usará los mismos valores de media y std para normalizar. Con el paquete Sklearn, es simplemente aplicar StandarScaler().fit(X\_train) y normalizar tanto X\_train como X\_test con scaler.tranform().

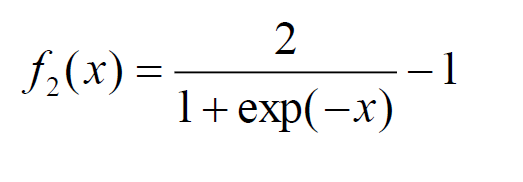
Propagación hacia delante

El algoritmo comienza inicializando la red, poniendo el valor de entrada de las diferentes neuronas y sesgos a valor 0, aunque en el sesgo la salida siempre valdrá 1 independientemente de la entrada. A continuación, mientras la condición de parada no se cumpla se ejecutan iterativamente los siguientes pasos que pasare a explicar, correspondiéndose este conjunto de pasos a una época.

Se iterará sobre el conjunto de pares X e Y de entrenamiento a partir de los cuales primeramente se inicializa con los valores de X la entrada de las neuronas de la capa de entrada, cada neurona con uno de los valores, al bias por otra parte no se le modifica la entrada.

Posteriormente se calcula la respuesta de la última capa oculta realizando el cálculo de la respuesta de la capa de entrada y de las diferentes capas ocultas, a partir de iterar desde la primera capa hasta la anterior a la de salida, disparando, inicializando y propagando para cada una de ellas.

En cada capa, cuando se dispara, se obtiene el valor de salida en función del valor de entrada de cada neurona haciendo uso de la función de activación de la sigmoide bipolar para este caso, la cual tiene la forma que se muestra más adelante. Este valor de salida se pasa como valor a cada una de las conexiones.



Una vez se ha disparado la capa se inicializa, estableciendo para las neuronas el valor de entrada a 0.

Posteriormente se propaga en la capa, a partir de lo cual el valor de entrada de la neurona se recalcula a partir de multiplicar el valor de la conexión por el peso de la conexión, haciendo el sumatorio para su conjunto de conexiones.

Una vez realizados los pasos anteriores iterando sobre las diferentes capas se obtiene la salida de la capa anterior a la capa de salida, a partir de lo cual se actualiza el valor de la capa de las conexiones de la capa de salida. En este punto, para la capa de salida solo se dispara, en vez de inicializar y propagar ya que para el ajuste de sus pesos hace falta y\_in, es decir su salida habiendo aplicado la función de activación.

Retropropagación del error:

La retropropagación se puede definir en 2 pasos, el primero sería retropropagar el error desde capa de salida hacia última capa oculta y el segundo paso que se hará de forma común para X números de capas ocultas que se retropropagará el error hasta que encuentre la capa de entrada.

Estos pasos se consisten en minimizar el error cuadrático medio y, por lo tanto, se debe calcular y posteriormente obtener correspondiente. Para retropropagación de la capa de salida hacia última capa oculta consiste en y para iteración común de resto de capas ocultas hasta capa de entrada, consiste en , obviamente los subíndices de cada iteración se cambiarán, pero el cálculo es el mismo.

El cálculo de las derivadas no lo vamos a hacer así, puesto que calcular una exponencial es una tarea costosa para ordenador y aplicar derivada también. Para ello, como estamos usando un sigmoide bipolar, la derivada se puede expresar de siguiente forma, .

Además, sabemos que es donde es el valor tras realizar el sumatorio de los pesos, por lo tanto se puede expresar como

o equivalente a . Esto pasa también para donde y es el valor tras disparar la neurona aplicando el sigmoide bipolar. La única diferencia que existe respecto primer paso es que el valor de es aplicar un sumatorio de los errores de la capa posterior que acabamos de calcular para iteración anterior, la fórmula consiste en .

La variación del peso se calcula bastante fácil siguiendo la fórmula , esta fórmula es común para cualquier paso de retropropagación pero con diferentes subíndices. Otra cosa que se debe destacar que puesto que la salida de bias es siempre 1, en tema de código no es necesario sacar un bloque para calcular .

Ajuste de pesos:

Con la matriz que contiene los cambios de pesos de cada de cada neurona de cada capa, para el conjunto de conexiones con las neuronas de la capa anterior se itera y actualizan los pesos de estas conexiones.

Finalmente se comprueba la condición de parada y se inicializan las entradas de la última capa a 0.

Criterio de parada:

Para nuestra implementación existe 4 criterios de parada, el primero que es el más simple con el uso de épocas; el segundo se va a definir una tolerancia, y se para cuando el error cuadrático medio es inferior que esa tolerancia especificada; el tercero es validación con los ejemplos de test, esto realmente se debería trocear el dataset en 3 partes (train, test y validación) pero sólo vamos a usar el test como validación, entonces tras cada época, se evaluará el score en Test y si alguna época supera 5% de error respecto la mejor score, se parará; el cuarto y el último, que también se usa validación, se parará cuando se repite el score en Test durante 30 épocas de entrenamiento. Obviamente los criterios no son los mejores puesto que existe casos donde cumpla estas condiciones, pero el score en test sale muy bajo, un ejemplo sería cuando tenemos muchas capas ocultas y con una tasa de aprendizaje muy bajo, esto hará que la red no consiga variar su peso durante X épocas y el score en Test se repetirá y se parará. Para usar la validación, se debe pasar el conjunto de Test de validación en la función train().

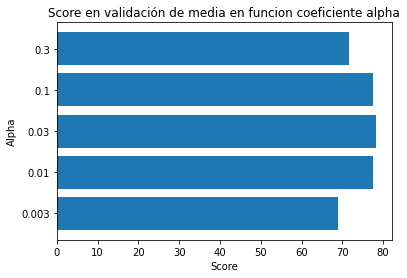
2.1. Describid todos los experimentos llevados a cabo y los resultados obtenidos:

Para este segundo ejercicio con el fin de obtener las mejores configuraciones de parámetros para los problemas 1, 2, 3 y 5 se realizó un hyperparameter tuning estilo grid a partir del cual se simularon diferentes configuraciones modificando los hiperparámetros como posteriormente se explicará y se escogió aquella que mostraba menor error cuadrático medio en la validación. Estas simulaciones se encuentran en el Excel “predicciones/hyperparameter\_tuning\_sin\_norm” donde se muestra el problema para el que se simula, la configuración de la red neuronal y de sus hiperparámetros, la época donde paró su entrenamiento y el porcentaje de Score para el Test.

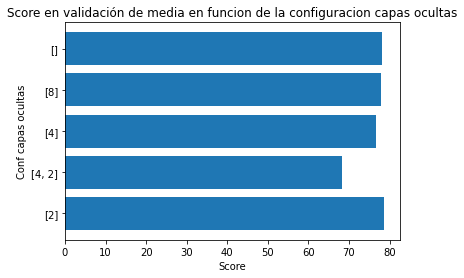
Hay que tener en cuenta que el entrenamiento depende en buena medida de la inicialización de los pesos, la cual en este caso se utiliza una distribución uniforme entre -0.5 y 0.5, en el sentido de que, dependiendo del valor del peso, se va a requerir más o menos épocas para converger a la solución, este suceso se puede encontrar en algunas de las simulaciones donde teniendo el mismo hiperparámetro, uno se para con pocas épocas y otro se entrena hasta época completa.

Por otra parte, durante el entrenamiento hay otra serie de problemas a solucionar, los cuales se tratan del problema de no sobre entrenar la red, que el entrenamiento no se quede estancado en mínimos locales y aprovechar potencia computacional no simulando de más. Para solucionar estos problemas por un lado se escogió un test de validación, con un tamaño del 25% del total de datos, con el fin escoger modelos más óptimos en producción que no se muestren sobre entrenados. También con el fin de ahorrar potencia computacional en cada simulación, se ha usado los 4 criterios de parada que habíamos definido en apartado 1.1 y con una tolerancia de 0.005.

Con las condiciones anteriores explicadas acerca de la configuración de la red neuronal, condiciones de parada y forma de obtener el score se simuló modificando el parámetro del coeficiente de aprendizaje con los siguientes valores [0.003 0.01 0.03 0.01 0.03 0.1 0.3]. Se observó que de media respecto todos los problemas, se obtuvieron mejores scores con valores de aprendizaje intermedios, sobre todo 0.03. Esto se observa en la siguiente gráfica.

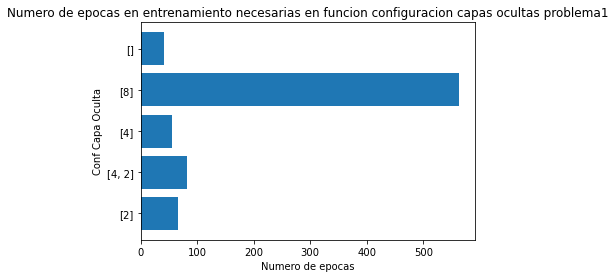


Por otra parte, se modificó la configuración de las capas ocultas de la red neuronal con los siguientes valores ([], [2], [4], [8], [4, 2]). Se observó que se obtuvieron mejores scores cuando no hay capas ocultas o únicamente hay una capa oculta, frente al caso de que haya más de una capa oculta. Esto se observa en la siguiente gráfica. También hay que destacar que, por cuestión de tiempo, no se ha podido simular con más configuraciones, ya que nos dimos cuenta tarde de que todos los problemas tienen al menos 9 atributos de entrada, y solo pusimos un máximo de 8 neuronas en capa oculta, por lo tanto, quizá si ponemos más neuronas, es posible que se mejore los resultados.



Problema 1:

Una vez realizadas las simulaciones anteriormente explicadas se observó que para el problema 1, se obtiene configurándolo con el coeficiente alpha a 0.03 necesitando de 35 épocas para obtener un score de 99.42%. Los entrenamientos en muchos casos requieren de muchas más épocas, ya que esto depende significativamente de la inicialización de los pesos como se muestra para ciertos entrenamientos de 1000 épocas con un score de 92.57%. Se observa que sin tener capas ocultas en menos de 100 épocas, se puede conseguir un score del 98% en validación por lo que a priori parece separable linealmente el problema. Ciertos valores explicados se muestran en las siguientes gráficas.







Problema 2:

Para el problema 2 se obtiene un score de valor 82.85% con un coeficiente alpha de 0.1, 2 neuronas en capa oculta y 250 épocas. Otra opción que obtiene buenos resultados se trata de una configuración de 8 neuronas con alpha 0.03, puesto que necesita menos de 300 épocas para llegar un score alrededor de 78%.

Problema 3:

Para el problema 3 se tiene que por un lado se cumple para determinados casos la condición de parada a partir de la cual se trata de si empeora en un 5% el score respecto al mejor score obtenido del modelo. Esto se muestra en el caso de coeficiente Alpha 0.01 con 8 neuronas ocultas, ya que existe un caso donde con dos épocas se finaliza el entrenamiento mientras que el resto necesita por debajo de 100 épocas con un score de 100%. Por lo tanto, se considera esta configuración como la óptima puesto que, si quitásemos los criterios de parada, seguramente volverá a conseguir un score bueno.

Es curioso tener un score de 100% para Test y posiblemente es por siguientes motivos:

* El score en Test está 100% pero en el Train no, pero como no hemos guardado ese resultado, no sabemos ese valor.
* Para predecir la clase, usamos argmax(), ósea se pone a 1 para el valor más grande de la salida y -1 para el resto. Esto quiere decir que si la salida es (-0.3, -0.4), se convertirá en (1, -1) pero realmente, los 2 valores de la salida son negativos y deberían corresponder a la clase -1. Por lo tanto, aunque algunos parecen predecir bien el resultado cuando realmente no lo es.

Problema 5:

Para el problema 5 se obtiene la mejor configuración con un coeficiente Alpha de 0.03, 2 neuronas en una capa oculta con un score en validación del 95% tras 150 épocas de entrenamiento.

Recuerda que el Excel “hyperparameter\_tuning\_sin\_norm” se puede encontrar en la carpeta **predicciones**.

**ejecuta\_problema\_real2**:

El fichero se encuentra en la carpeta **predicciones.**

Se ha ejecutado 295 épocas.

Porcentaje de aciertos: 80.71428571428572%

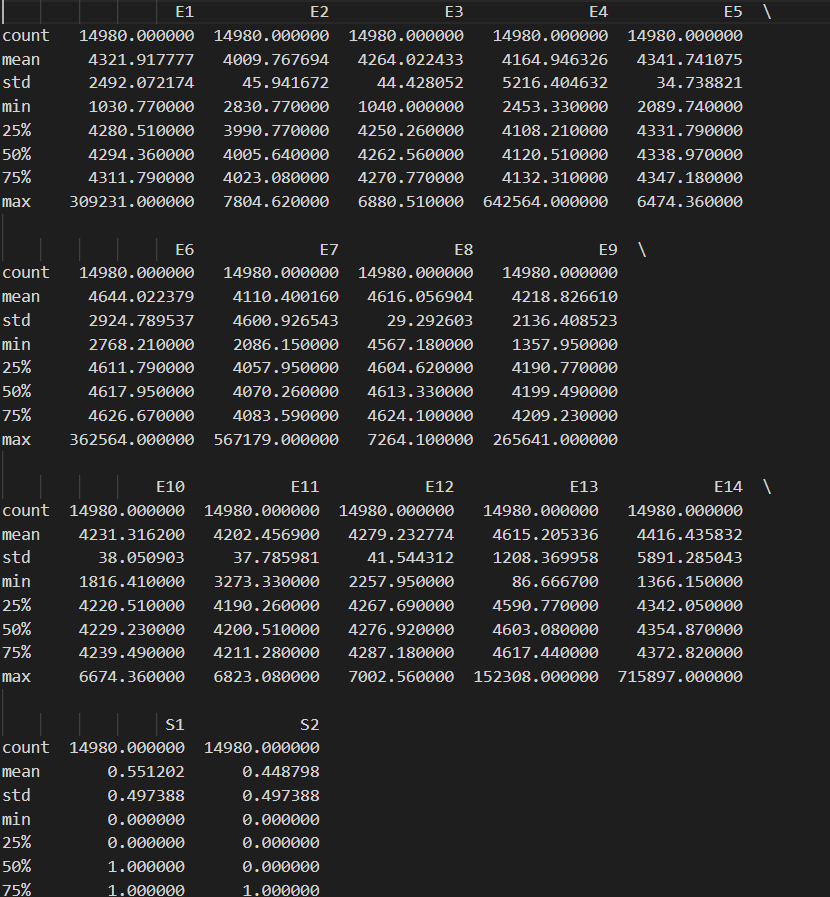
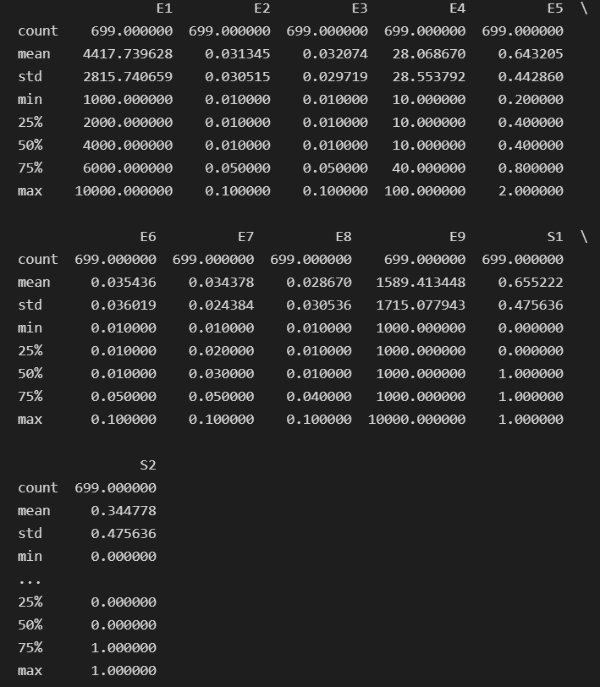
La matriz de confusión:

[[30 17]

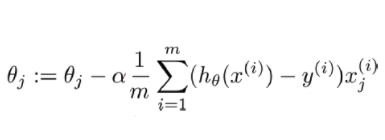
[10 83]]

3.1. Explicad la normalización aplicada a los dados, cómo se ha hecho y porqué se hace:

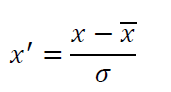
La descripción para el conjunto de datos relativos a las diferentes columnas para el problema 4 y el problema 6, en cuanto a la media y desviación estándar se muestran a continuación.



Como se aprecia para cada problema sus diferentes columnas muestran medias y desviaciones estándar dispares. Para las diferentes variables dependientes es necesario una normalización a través del escalamiento ya que, en caso de haber mucha disparidad entre los rangos de valores de una variable respecto de otra, los modelos lineales y las redes neuronales van a tener dificultades para obtener predicciones a partir de estos datos. Esto se debe a que estos modelos hacen uso del descenso por gradiente para el cual la diferencia de rangos entre características hace que en cada paso del recalculo de peso se va a quedar muy corto o muy largo en esta corrección. Es necesario que las características se encuentren en un mismo rango de valores para que el modelo llegue suavemente a la solución. Esta se trata de la función de descenso por gradiente.



Para este escalamiento hemos hecho uso de un escalamiento estándar, el cual a partir de su transformación hace que la variable obtenga una distribución normal, típicamente con media 0 y desviación estándar 1. Este escalador s de especial utilidad si la característica sigue una distribución normal y si hay puntos críticos.



Sin normalizar para el problema 4 se obtiene el mejor score con 76%, alpha 0.01 y 8 neuronas, mientras que para el problema 6 el mejor score es de solo 55%, alpha 0.01, 4 neuronas capa oculta 1 y 2 neuronas en la capa oculta 2. Estos datos se pueden encontrar en el Excel “hyperparameter\_tuning\_sin\_norm” de la carpeta **predicciones.**

Se puede notar que casi un 90% de los entrenamientos se han parado tras 31 épocas. Esto se debe a que uno de los criterios de parada se trata de que, si repite el score 30 veces, se finaliza el entrenamiento. Esto implica que la red no está consiguiendo aprender de forma correcta puesto que no está cambiando su score durante 30 épocas debido a los problemas de normalización explicados anteriormente.

Debido que el tiempo de ejecución de estos 2 ficheros normalizados tardan demasiado, sobre todo el problema real 6, hemos decidido reducir el número de las simulaciones y los hiperparámetros. Para problema real 6, en vez de tener ([], [2], [4], [8], [4, 2]) neuronas para capas ocultas, solo se ha usado ([], [4], [8], [16]), luego en vez de usar un alpha entre [0.003 0.01 0.03 0.01 0.03 0.1 0.3], se ha utilizado [0.01 0.03 0.01 0.3], finalmente solo se va a ejecutar una vez cada configuración. Se ha generado el fichero “hyperparameter\_tuning\_con\_norm” en la carpeta **predicciones.**

Con la normalización por el contrario debido a las mejoras que se producen debido a lo anteriormente explicado, se obtiene para el problema 4 un score en validación de 98.28% para una red neuronal con coeficiente de aprendizaje 0.003, 4 neuronas en la capa oculta y tras tan solo 40 épocas. Por otra parte, para el problema 6 se obtiene un score de 87.61% para una red neuronal con coeficiente de aprendizaje 0.01, 16 neuronas en la capa oculta y tras mil épocas.

Recuerda que el Excel “hyperparameter\_tuning\_sin\_norm” y “hyperparameter\_tuning\_con\_norm” se pueden encontrar en la carpeta **predicciones**.

3.2. Ejecucción concreta para el problema 6:

Por motivo de tiempo, hemos olvidado revertir los valores de X a la hora de escribir los resultados, por lo tanto, el fichero problema\_real6\_mul.txt que se encuentra en la carpeta predicción, todos sus valores de X están normalizados.

El porcentaje de aciertos para este problema se trata de 84.75%, por otra parte, la matriz de confusión es la siguiente.

[[2090 381]

[ 304 1719]]