Identificación y Control Neuronal

Pablo Acereda García
Department of Computer Science
University of Alcala
28805 - Alcalá de Henares, Madrid, Spain
pablo.acereda@edu.uah.es
Laura Pérez Medeiro
Department of Computer Science
University of Alcala
28805 - Alcalá de Henares, Madrid, Spain
l.perezm@edu.uah.es

October 24, 2019

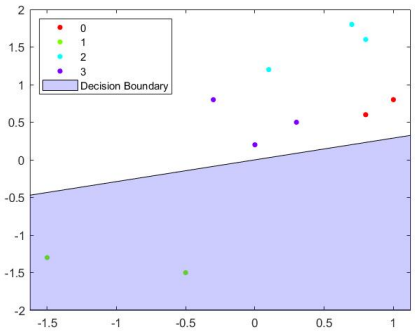
This page has been intentionally left blank

Contents

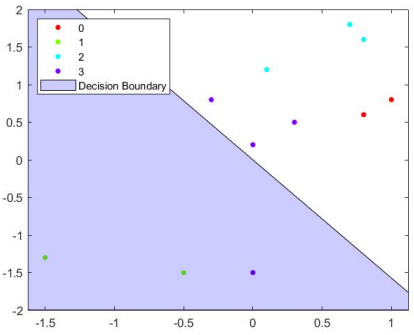
Ejercicio 1. Perceptron

El Perceptron es una función de tipo lineal, que puede resultar indicada cuando solo hay 2 clases dentro del conjunto de datos. Por este mismo motivo, este tipo de red neuronal no es perfectamente funcional cuando hay diferentes clases (en este caso 4).

Debido a que no se ha definido con anterioridad a la creación de la red neuronal, el número de neuronas de la capa de salida es una (al igual que el número de capas y el número de entradas).



(a) Datos Originales



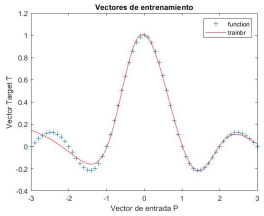
(b) Datos Extendidos

En la gráfica 1a la pendiente obtenida por el perceptrón separa los elementos de la clase *1* del resto de elementos. Pero al añadir el dato $[0.0, -1.5]$ de la clase *3*, el perceptrón obtenido cambia completamente su pendiente (llegando incluso a invertirla). Con la nueva pendiente es incapaz de separar completamente la clase *1* de la clase *3*. Resultado a esperarse debido a que hay más clases y más *clusters* de los que puede separar un perceptrón[1].

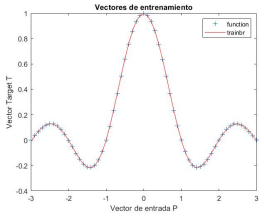
Ejercicio 2. Aproximación de funciones

La función sobre la que se quiere experimentar es $\sin(t)$. Se desea comprobar si el modificar el número de neuronas de la capa ocula o cambiar el método de entrenamiento posee algún efecto sobre un conjunto de datos.

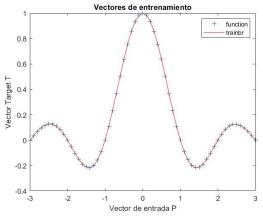
En primera instancia, se ejecuta el *script* con el método de entrenamiento *trainrp*, modificando el número de neuronas de la capa oculta. Para las pruebas, se han empleado valores desde 3 hasta 5.



(a) *trainbr*. Número neuronas de la capa oculta = 3.



(b) *trainbr*. Número neuronas de la capa oculta = 4.



(c) *trainbr*. Número neuronas de la capa oculta = 5.

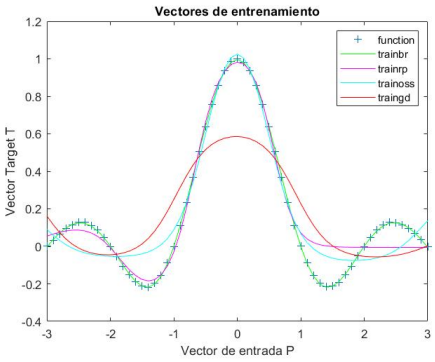
Se comprobará en la figura 3b como puede afectar el número de neuronas de la capa oculta conforme aumenta su tamaño.

Además de modificar el número de neuronas de la capa oculta también se ha modificado el método de entrenamiento, empleándose los siguientes:

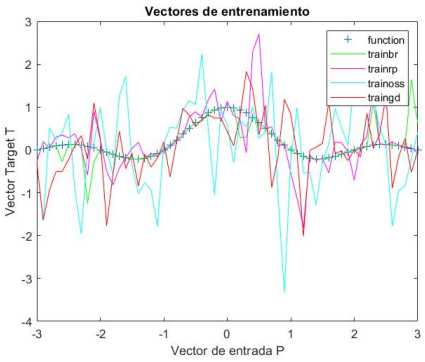
trainbr Bayesan Regularization

trainrp Resilient Backpropagation

[1] Xor Problem.



(a) Todas las funciones. Número de neuronas de la capa oculta = 4.



(b) Todas las funciones. Número de neuronas de la capa oculta = 200.

trainoss One Step Secant

traingd Gradient Descent

El resultado se puede observar en las siguientes gráficas (3a y 3b).

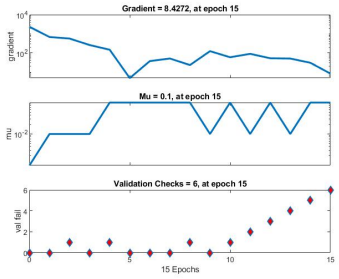
El modificar el número de neuronas de la capa oculta puede acabar en una red neuronal que esté demasiado adaptada al problema, y que por lo tanto no tenga un buen rendimiento al comprobar nuevos datos.

Además, el cambiar el método de entrenamiento supone el tener aproximaciones diferentes. El elegir un método u otro siempre dependerá de la función original o del conjunto de datos con los que se desea entrenar la red neuronal.

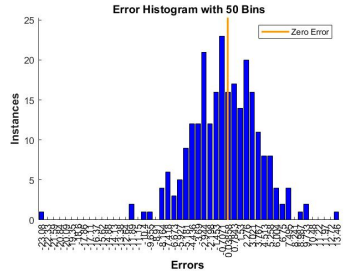
Ejercicio 3. Aproximación de funciones (II)

Tras la ejecución del algoritmo proporcionado y aplicarlo a *bodfat_ dataset* se obtienen las siguientes gráficas:

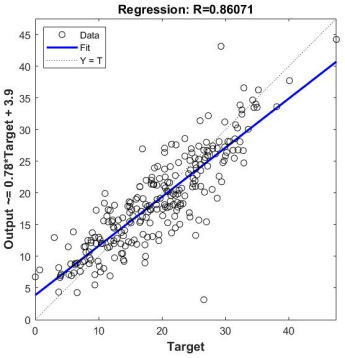
(a) Rendimiento



(b) Estado de Entrenamiento



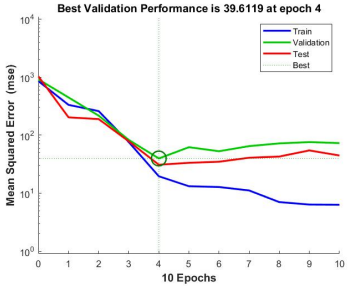
(c) Histograma de Errores



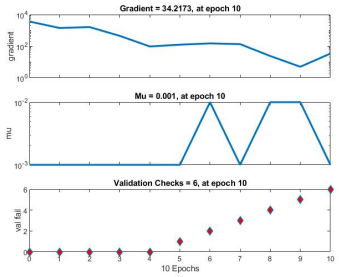
(d) Regresión

Los resultados demuestran que el entrenamiento de la red neuronal aumenta en precisión conforme avanzan las épocas (*epoch*); la validación encuentra su error mínimo durante el periodo 9, momento a partir del cual solo aumenta; en el testing, el error aumenta tras durante el testeo, llegado a alcanzar más de 10^2 (un valor demasiado alto).

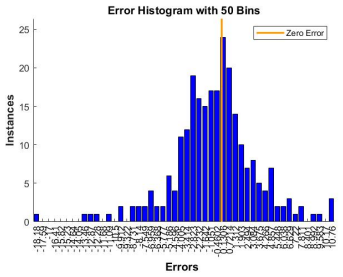
Para solucionar este problema, se puede probar una distribución de los datos diferente. En este caso se ha escogido 60/20/20 como valores para el entrenamiento (*train*), la validación (*validation*) y el testeo (*test*), respectivamente, de manera arbitraria.



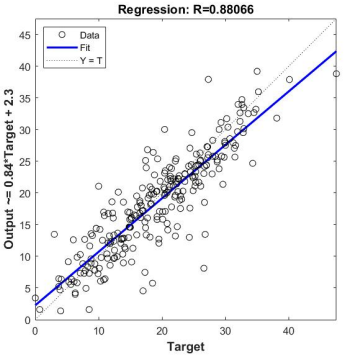
(a) Rendimiento



(b) Estado de Entrenamiento



(c) Histograma de Errores



(d) Regresión

Tras la nueva división de los datos, empleando el mismo algoritmo de entrenamiento, se consigue que los errores (teste, validación y entrenamiento) disminuyen, aunque la valicación y el test siguen teniendo una cierta predisposición a incrementar levemente superada la cuarta época.

La función lineal mostrada en la regresión pasa a ser de $Output = 0.78 * Target + 3.9$ a valer $Output = 0.84 * Target + 2.3$. Siendo, aparentemente, la segunda una ecuación más fiel a los datos representados.

Para realizar un estudio más exhaustivo, se han empleado diferentes métodos de entrenamiento.

trainbr Bayesan Regularization

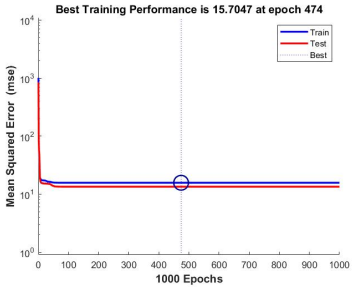
trainrp Resilient Backpropagation

trainoss One Step Secant

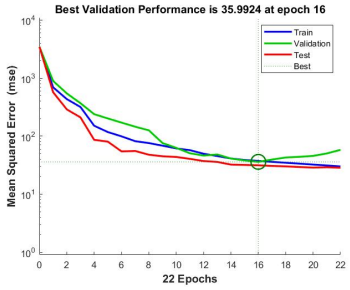
traingd Gradient Descent

A continuación se especificará y comparará cada uno de los grupos de gráficas obtenidos a lo largo de la ejecución del *script*

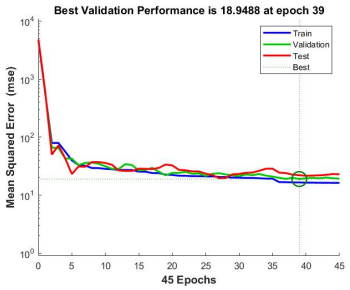
Referente al rendimiento, la función bayesiana ha conseguido los mejores resultados, consigue el mejor rendimiento de entre las cuatro gráficas. Por otro lado, la función *trainrp* consigue un rendimiento que empeora conforme se supera la época 16. *trainoss*, con un rendimiento entre las dos anteriores, posee un MSE muy similar entre los conjuntos de datos *train*, *validation* y *test*. Por último, el método de entrenamiento del gradicente, consigue el peor resultado, empeorando conforme recibe más entrenamiento.



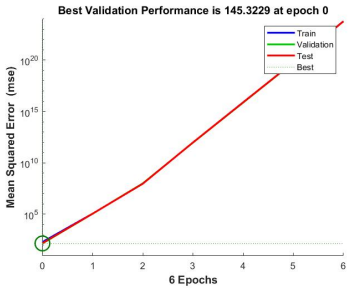
(a) Rendimiento *trainbr*



(b) Rendimiento *trainrp*

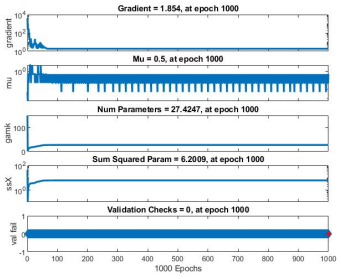


(c) Rendimiento *trainoss*

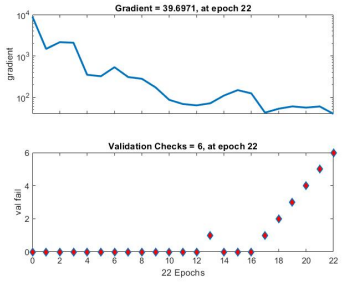


(d) Rendimiento *traingd*

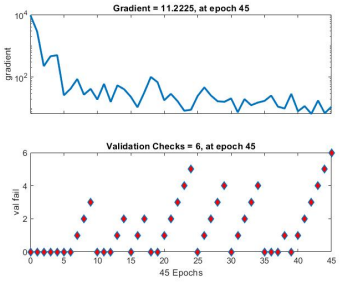
El valor del gradiente [2] se muestra una vez más incremental para *traingd*[3] En el caso de *trainbr*, el gradiente se mantiene constante a partir de cierto punto, lo que hace que el entrenamiento dure 1000 *epochs*; esto se debe a la manera en la que está codificado el algoritmo, la parada por valiación se encuentra desactivada por defecto. El entrenamiento de *trainrp* y *trainoss* discurre de manera normal, alcanzando los mínimos locales en los valores 11.2225 y 39.6971, respectivamente.



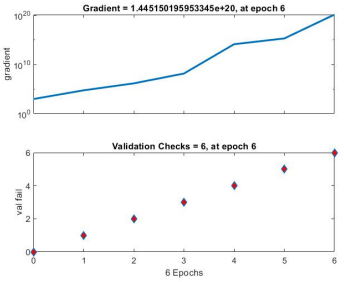
(a) Estado de Entrenamiento *trainbr*



(b) Estado de Entrenamiento *trainrp*



(c) Estado de Entrenamiento *trainoss*



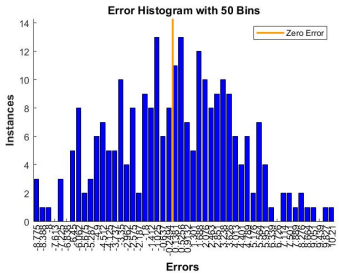
(d) Estado de Entrenamiento *traingd*

El histograma es una gráfica que representa el error cometido al estimar los valores empleados para el entrenamiento de la red neuronal, por tanto, cuanto más esté centrada la distribución de lo errores de

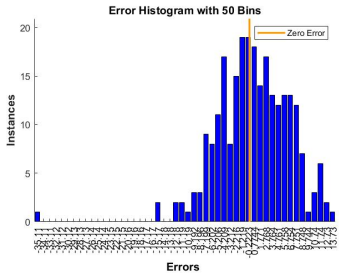
[2] El valor del gradiente por propagación inversa se encuentra en escala logarítmica. El valor en la parte superior de la gráfica es el valor para el cual se ha alcanzado el punto mínimo de un mínimo local.

[3] MATLAB detiene el entrenamiento del modelo cuando el valor de MSE - Mean Square Error - resulta incremental 6 veces de manera consecutiva, también conocido como *validation fails*.

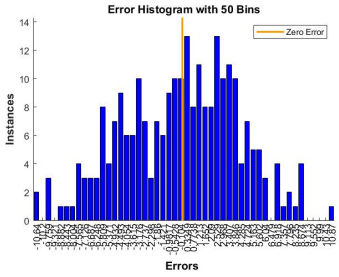
manera cercana al indicador *Zero Error*, mejor predicción dará el modelo. Atendiendo a este criterio, los extremos de *trainrp* están prácticamente libres, y los valores cercanos a $Error = 0$ están más poblados. Por esto mismo, se sabe que la función del gradiente es la que ha obtenido peores resultados, por tener su mayor concentración de errores más cercana a 1 que a 0. *trainoss* tiene, con alguna excepción, forma de pirámide escalonada, lo que también sugiere un entrenamiento exitoso. *trainbr* posee una forma bastante irregular, pero los errores que más se repiten se encuentran principalmente en la zona central, sugiriendo un resultado mejor que el resto de algoritmos de entrenamiento.



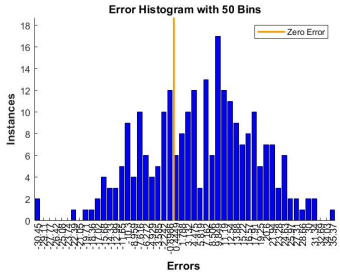
(a) Histograma de Errores *trainbr*



(b) Histograma de Errores *trainrp*

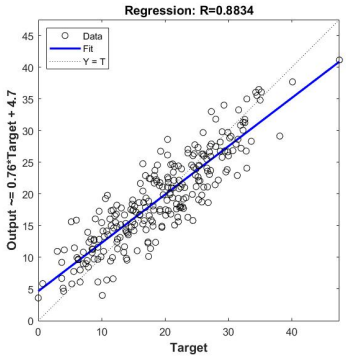


(c) Histograma de Errores *trainoss*

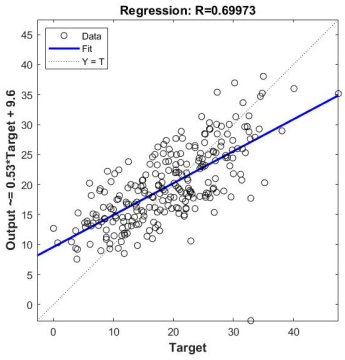


(d) Histograma de Errores *traingd*

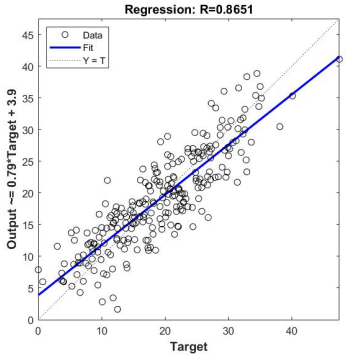
En la parte superior de cada gráfica aparece un valor, R, el cual indica lo óptima que es la regresión, siendo 1 el valor más óptimo. No hace falta observar mucho los resultados obtenidos para percatarse de que el método de entrenamiento más óptimo es en realidad la regularización bayesiana, tal y como se venía adelantando a lo largo del ejercicio.



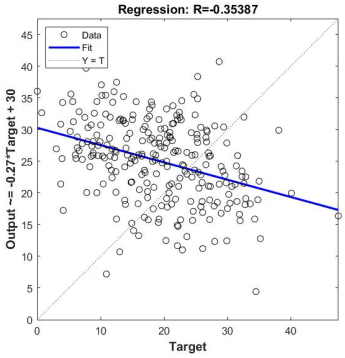
(a) Regresión *trainbr*



(b) Regresión *trainrp*



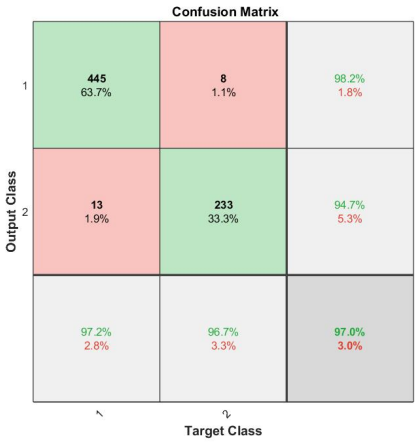
(c) Regresión *trainoss*



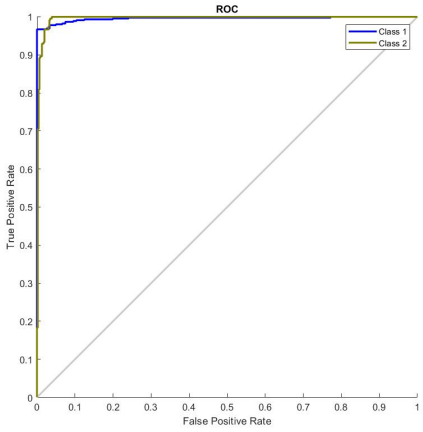
(d) Regresión *traingd*

Ejercicio 4. Clasificación

Tras la ejecución del *script* adaptado para los datos de *cancer_ dataset* se obtienen las siguientes gráficas.



(a) Matriz de confusión



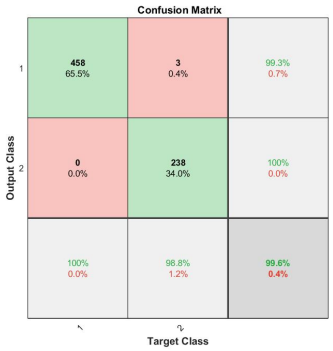
(b) *Receiver Operating Characteristic*

En 10a, las filas indican el valor que se ha predicho; lsa columnas indican el valor real. Por tanto, la diagonal corresponde a los valores que han sido predecidos de manera correcta. Interpretando el resultado obtenido, se sabe que el 97% de las estimaciones han sido correctas.

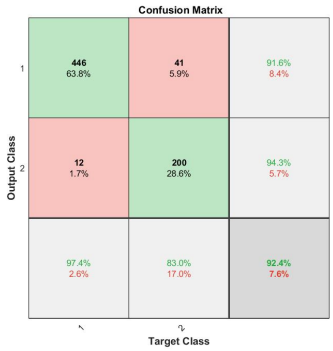
En la segunda gráfica (10b), es una métrica empleada para comprobar la calidad de los clasificadores. Cuanto más se acerquen las funciones a 1 en el eje de ordenadas. Por este mismo motivo, se conoce que

los clasificadores empleados para este set de datos son una buena opción.

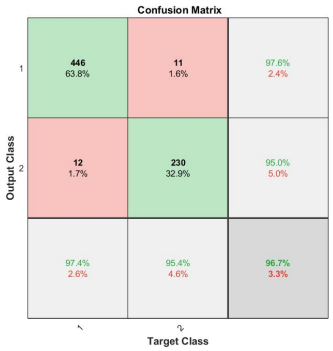
Para realizar un estudio más acertado de estas herramientas, se han empleado diferentes métods de entrenamiento (los mismos empleados para los ejercicios 2 y 3 de este mismo apartado).



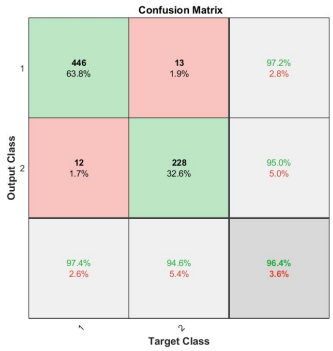
(a) Matriz de confusión *trainbr*



(b) Matriz de confusión *trainrp*

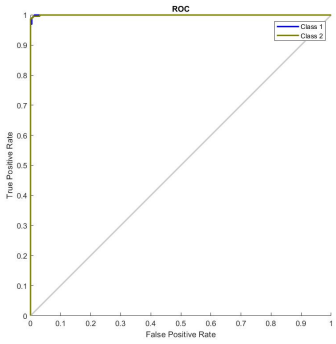


(c) Matriz de confusión *trainoss*

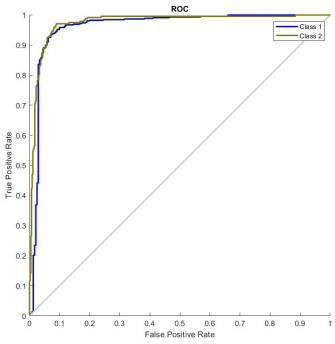


(d) Matriz de confusión *traingd*

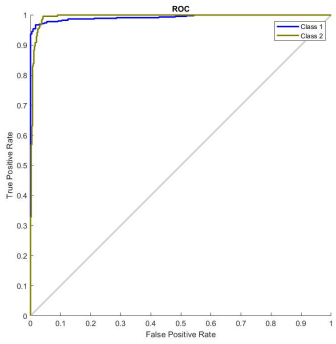
De entre todos los resultados obtenidos, el más acertado ha sido el de el algoritmo de entrenamiento *trainbr*, con un 99.6% de acierto sobre el set. El peor algoritmo que se podría emplear sobre este set es *trainrp*, que a pesar de poseer el peor resultado tiene un 92.4% de acierto.



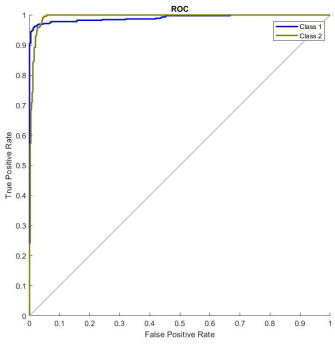
(a) ROC *trainbr*



(b) ROC *trainrp*



(c) ROC *trainoss*



(d) ROC *traingd*

Evaluando el resultado obtenido de entre las representaciones de ROC se corrobora que el mejor método de entrenamiento pertenece a *trainbr*, y el peor a *trainrp*; debido a la proximidad/distancia de los valores de *True Positive Rate* para *False Positive Rate* entre 0 y 0.2.