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 23, 2019

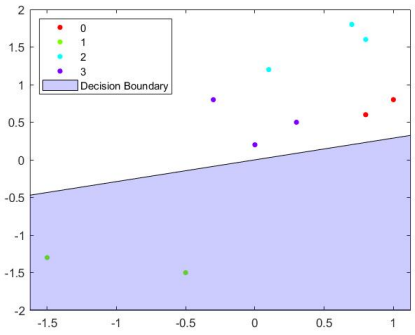
This page has been intentionally left blank

Contents

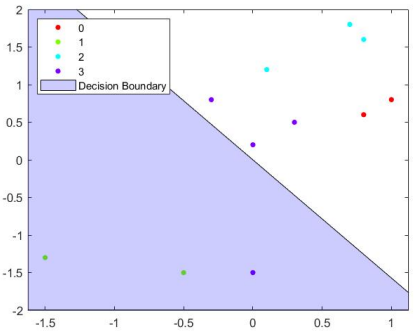
1 Ejercicio 1. Perceptron

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].

2 Ejercicio 2. Aproximación de funciones

En todos los casos, y con los diferentes métodos de entrenamiento, el incrementar el tamaño de la capa oculta implica una mayor precisión, número de iteraciones para completar el proceso y un pequeño incremento del tiempo de procesado.

En cuanto a las funciones de entrenamiento, se han empleado:

trainbr Bayesan Regularization

trainrp Resilient Backpropagation

trainoss One Step Secant

traingd Gradient Descent

Cambiar la función de entrenamiento puede dar resultados completamente diferentes. Como se observa en 2 emplear diferentes funciones puede afectar al tiempo de entrenamiento; al número de iteraciones (siendo en este caso *traingd* la más costosa); y a la precisión de los valores obtenidos (siendo *trainbr* la más precisa).

3 Ejercicio 3. Aproximación de funciones (II)

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

[1] Xor Problem.

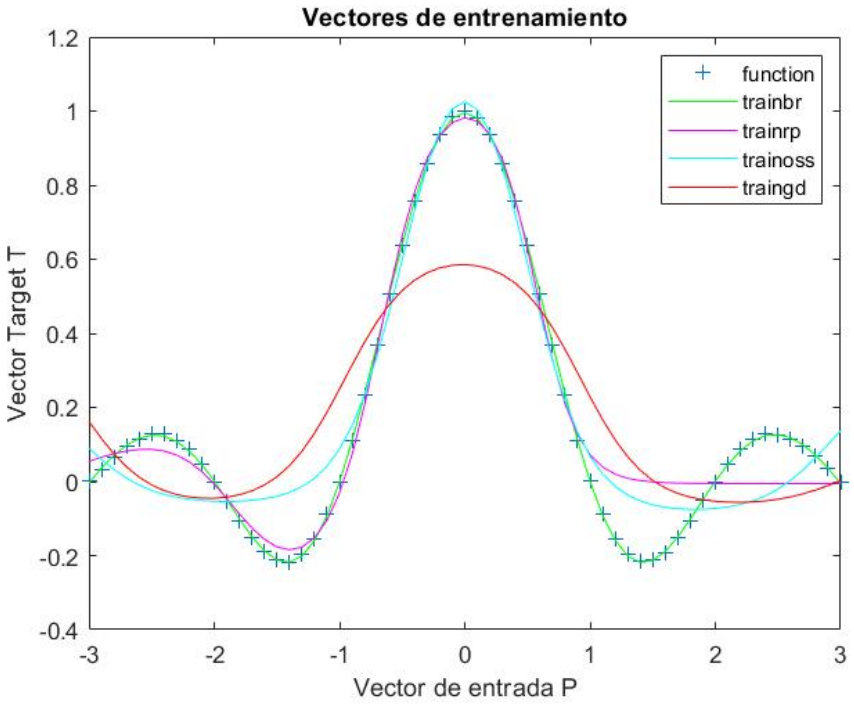
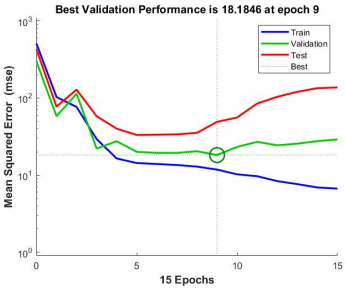
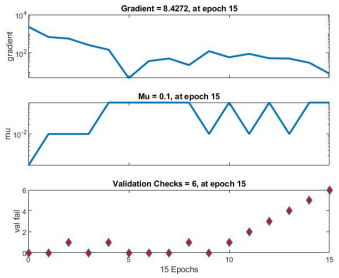


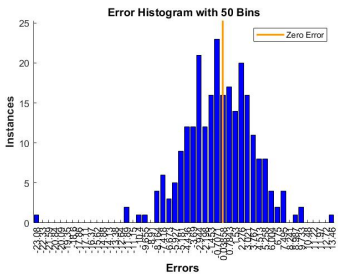
Figure 2: Funciones de entrenamiento empleadas.



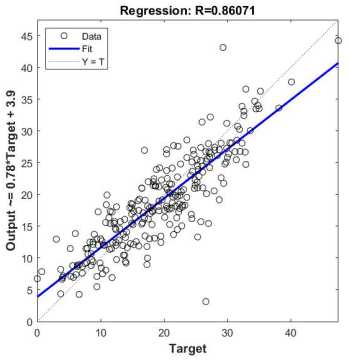
(a) Rendimiento



(b) Estado de Entrenamiento



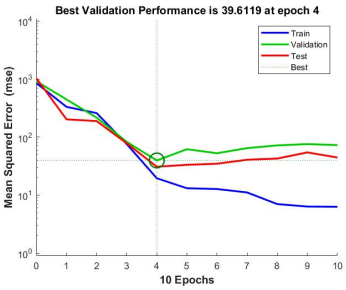
(c) Histograma de Errores



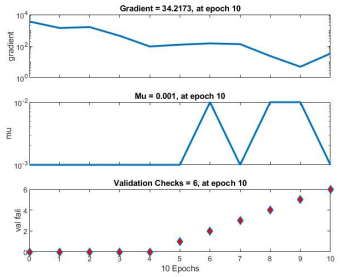
(d) Regresión

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).

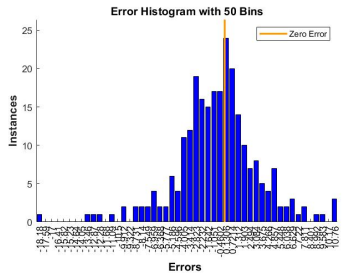
Para solucionar este problema, se puede probar una distribución de los datos diferente. En este caso se ha escogido 60/20/20 como valores para el entrenamiento (*train*), la validación (*validation*) y el testeo (*test*), respectivamente, de manera arbitraria.



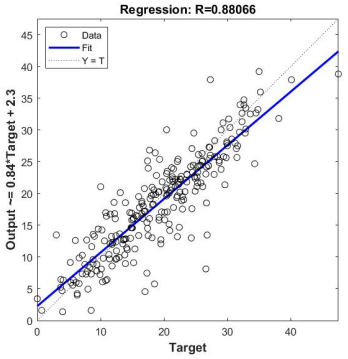
(a) Rendimiento



(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

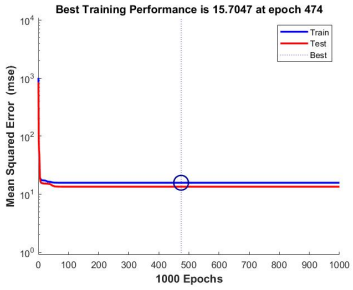
trainrp Resilient Backpropagation

trainoss One Step Secant

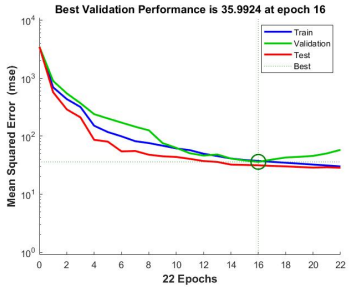
traingd Gradient Descent

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

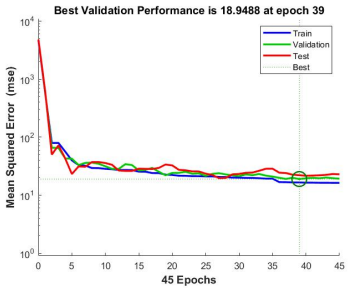
Referente al rendimiento, la función bayesiana ha conseguido los mejores resultados, consigue el mejor rendimiento de entre las cuatro gráficas. Por otro lado, la función *trainrp* consigue un rendimiento que empeora conforme se supera la época 16. *trainoss*, con un rendimiento entre las dos anteriores, 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.



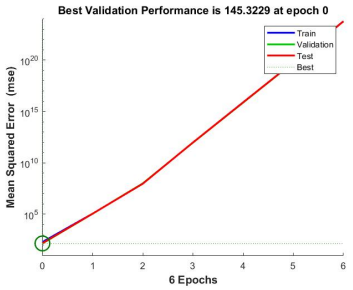
(a) Rendimiento *trainbr*



(b) Rendimiento *trainrp*

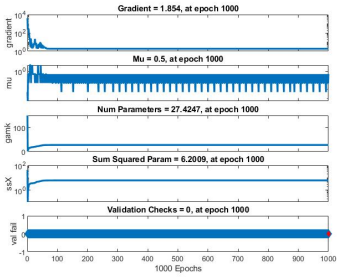


(c) Rendimiento *trainoss*

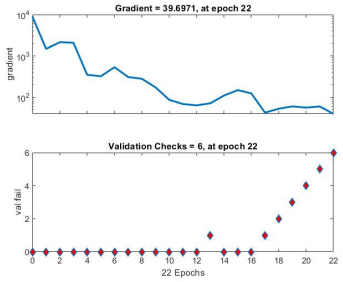


(d) Rendimiento *traingd*

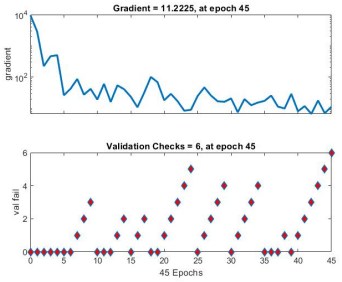
El valor del gradiente [2] se muestra una vez más incremental para *traingd*[3] En el caso de *trainbr*, el gradiente se mantiene constante a partir de cierto punto, lo que hace que el entrenamiento dure 1000 *epochs*; esto se debe a la manera en la que está codificado el algoritmo, la parada por valiación 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.



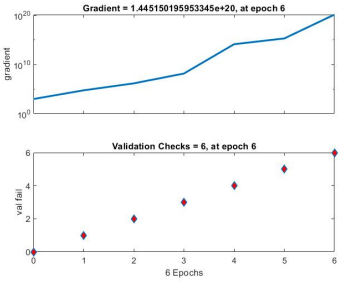
(a) Estado de Entrenamiento *trainbr*



(b) Estado de Entrenamiento *trainrp*



(c) Estado de Entrenamiento *trainoss*



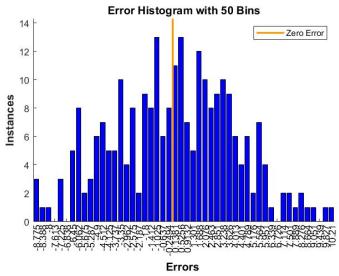
(d) Estado de Entrenamiento *traingd*

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

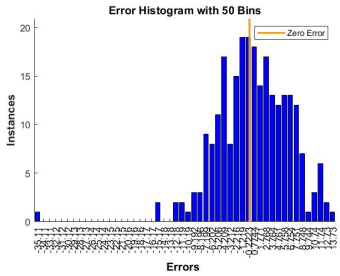
[2] El valor del gradiente por propagación inversa se encuentra en escala logarítmica. El valor en la parte superior de la gráfica es el valor para el cual se ha alcanzado el punto mínimo de un mínimo local.

[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*.

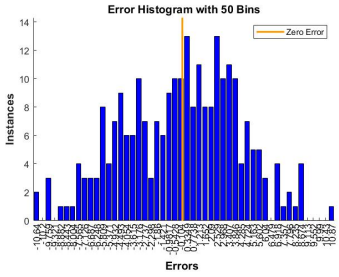
manera cercana al indicador *Zero Error*, mejor predicción dará el modelo. Atendiendo a este criterio, los extremos de *trainrp* están prácticamente libres, y los valores cercanos a $Error = 0$ están más poblados. Por esto mismo, se sabe que la función del gradiente es la que ha obtenido peores resultados, 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.



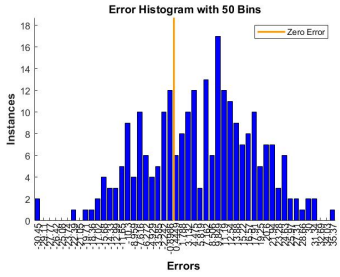
(a) Histograma de Errores *trainbr*



(b) Histograma de Errores *trainrp*

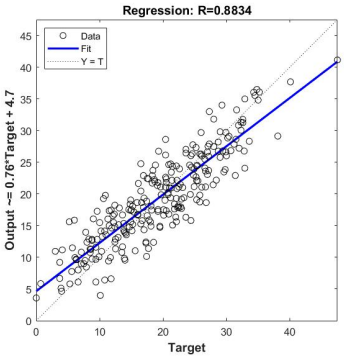


(c) Histograma de Errores *trainoss*

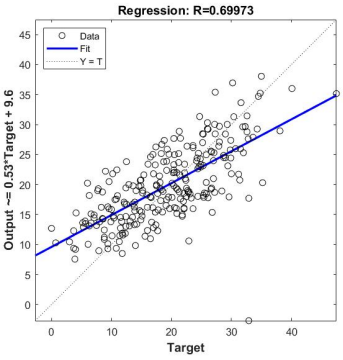


(d) Histograma de Errores *traingd*

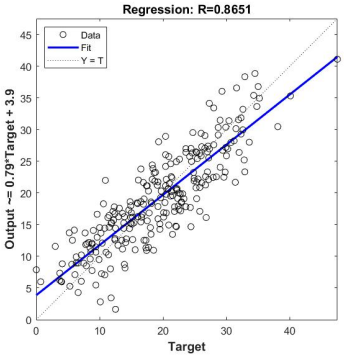
En la parte superior de cada gráfica aparece un valor, R, el cual indica lo óptima que es la regresión, siendo 1 el valor más óptimo. No hace falta observar mucho los resultados obtenidos para percatarse de que el método de entrenamiento más óptimo es en realidad la regularización bayesiana, tal y como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



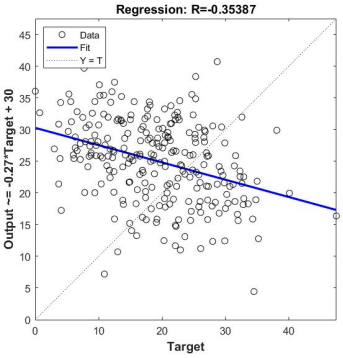
(a) Regresión *trainbr*



(b) Regresión *trainrp*



(c) Regresión *trainoss*



(d) Regresión *traingd*