Universidad de Burgos

Computación Neuronal y Evolutiva

P3: Multilayer Perceptron (MLP)

Diseñar y entrenar distintos Perceptron Multicapa (MLP), con el objetivo de hacer una comparativa respecto al rendimiento de estos para la misma tarea y aplicación estudiada en P1_Thyroid.

Estudiantes:

David Miguel Lozano
Javier Martínez Riberas

Profesor de la asignatura:

Álvaro Herrero Cosío

1º semestre 2016

Índice

A. Introduction

El objetivo de la práctica es diseñar y entrenar distintos Perceptron Multicapa (MLP), con el objetivo de hacer una comparativa respecto al rendimiento de estos para la misma tarea y aplicación estudiada en P1_Thyroid (clasificación de patrones: tiroides).

Se realizará un estudio sobre los distintos algoritmos de aprendizaje que implementan backpropagation y el ajuste de los correspondientes parámetros de estos. En concreto:

- `net.trainParam.max_fail`: máximo número de fallos de validación.
- `net.trainParam.alpha`: *learning rate* de los pesos.
- `net.trainParam.beta`: *learning rate* de los bias.

**En el estudio se explicará porque se han selecionado estos parámetros para realizar el estudio y no todos los propuestos en el enunciado de la práctica.*

La topología de la red se corresponderá con la configuración más óptima encontrada en la práctica P1_Thyroid.

B. Descripción del conjunto de datos

El conjunto de datos utilizados se ha obtenido del dataset de ejemplo *Thyroid* que provee Matlab. Los datos provienen del *UCI Machine Learning Repository* [1] y fueron donados por la Universidad de California.

El dataset contiene datos de 7200 pacientes agrupados en dos matrices:

- $thyroidInputs$: matriz de 21x7200 con los datos de los 7200 pacientes caracterizados por 15 atributos binarios y 6 atributos continuos.
- $thyroidTargets$: matriz de 3x7200 en donde se asocia un vector de tres clases a cada paciente. En este vector se define a cuál de las tres clases pertenece el paciente.

Las tres clases que contiene el dataset son:

1. Paciente sano.
2. Paciente con hipertiroidismo.
3. Paciente con hipotiroidismo.

C. Descripción del procedimiento

Para automatizar el estudio lo máximo posible, se ha realizado un script que realiza varios entrenamientos con los diferentes algoritmos de entrenamiento y variando los parámetros mencionados.

Por cada combinación se ejecutan 20 experimentos y se toma el valor medio para reducir el impacto de la aleatoriedad en la inicialización de los pesos y bias.

Se han utilizado los siguientes algoritmos de entrenamiento:

- `traincgb`: conjugate gradient backpropagation with Powell-Beale restarts. [3]
- `traincgf`: conjugate gradient backpropagation with Fletcher-Reeves updates. [4]

Como se puede observar ambos utilizan el algoritmo de retropropagación para la actualización de los pesos. Y por lo tanto, son adecuados para el entrenamiento de un perceptrón multicapa.

Se modifican los siguientes parámetros para cada algoritmo:

- `net.trainParam.max_fail`: maximum validation failures. Rango [10:10:100].
- `net.trainParam.alpha`: scale factor that determines sufficient reduction in performance. Rango [0.001:0.001:0.004].
- `net.trainParam.beta`: scale factor that determines sufficient large step size. Rango [0.1:0.2:0.8].

Como indicaderes generales se han empleado:

- Confusion value: fraction of samples misclassified.
- `plotperform`: plot network performance. [2]

**Todo el código generado en esta práctica se encuentra disponible en el siguiente repositorio: P2_SelfOrganizingMap.*

D. Estudio

En primer lugar, se han utilizado los parámetros descritos para realizar el estudio y no todos los propuestos en el enunciado de la práctica (*epochs*, `goal` y tiempo de aprendizaje) porque, debido a la experiencia obtenida durante la realización de la práctica 1, somos conscientes que los criterios de parada elegidos son los que realmente detienen el entrenamiento para este problema en concreto.

Todos los datos generados durante los 160 experimentos del estudio (archivos CSV con los datos recogidos, imágenes con las gráficas de rendimiento y tablas excel de resumen) se encuentran en el repositorio: Data. No se incluyen en el informe por no aumentar demasiado la longitud de este.

Tras analizar los datos obtenidos, hemos seleccionado el mejor caso para algoritmo y cada parámetro basándonos en el que tuviese el menor valor de confusión medio.

Mejor caso para el algoritmo `traincgb`:

- `max_fail`: 50.
- `alpha`: 0.001.
- `beta`: 0.1.

Su gráfico de rendimiento fue:

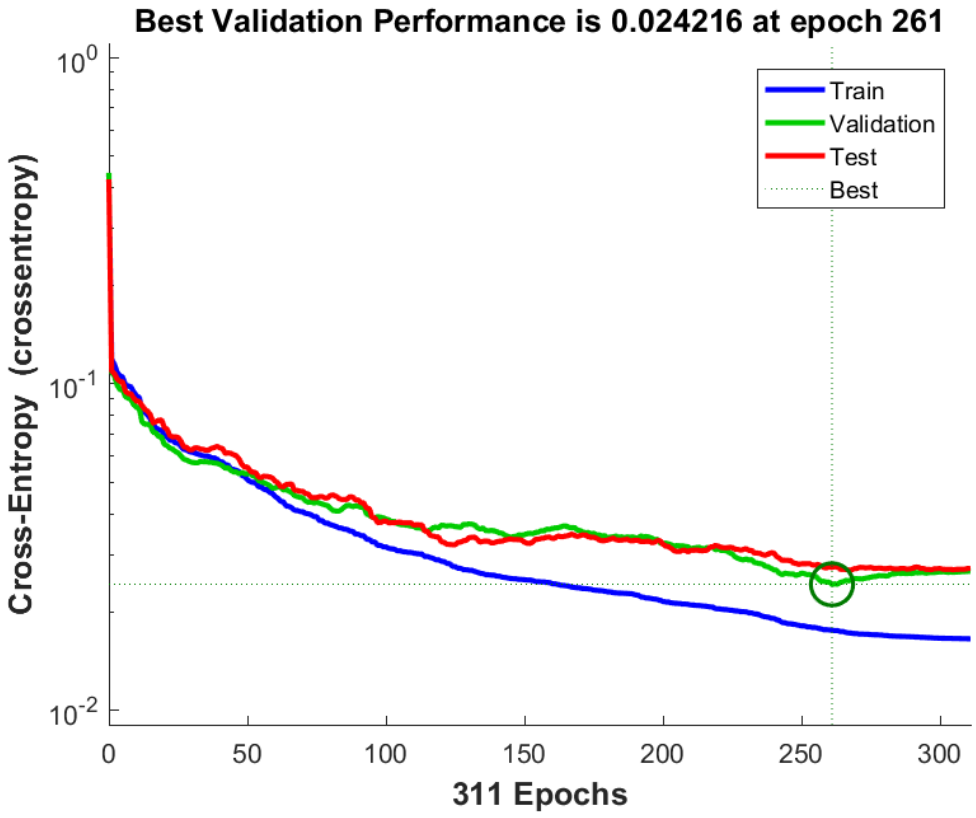


Figura 1: Rendimiento mejor caso `traincgb`.

Mejor caso para el algoritmo `traincgf`:

- `max_fail`: 100.
- `alpha`: 0.002.
- `beta`: 0.1.

Su gráfico de rendimiento fue:

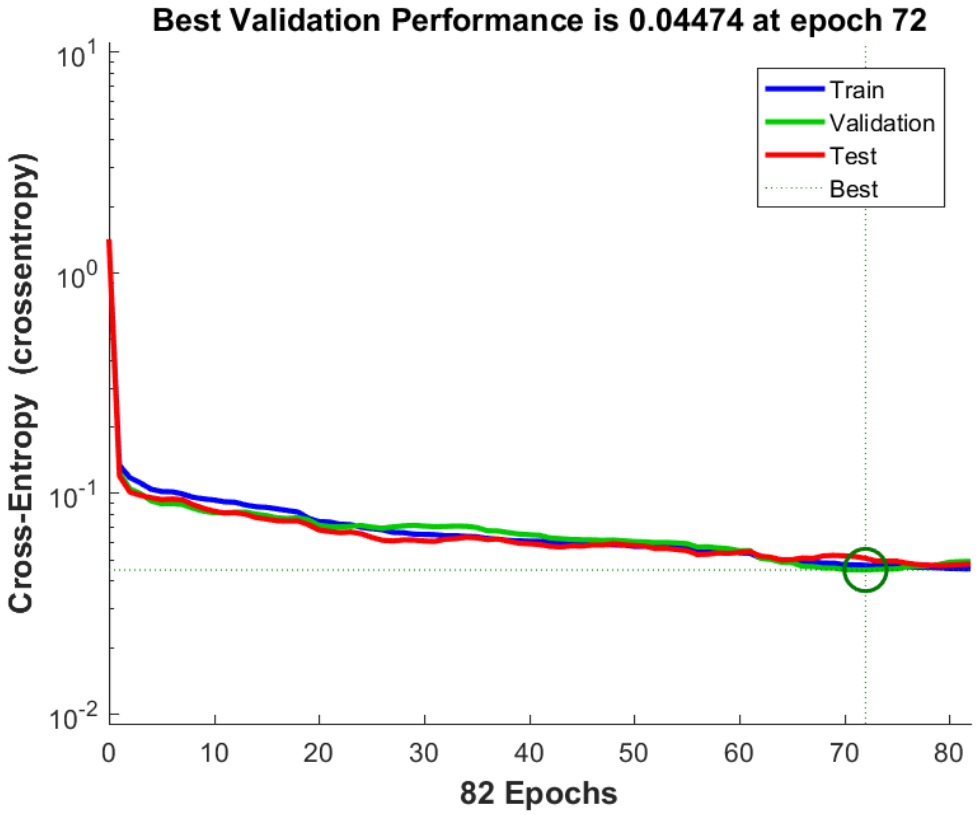
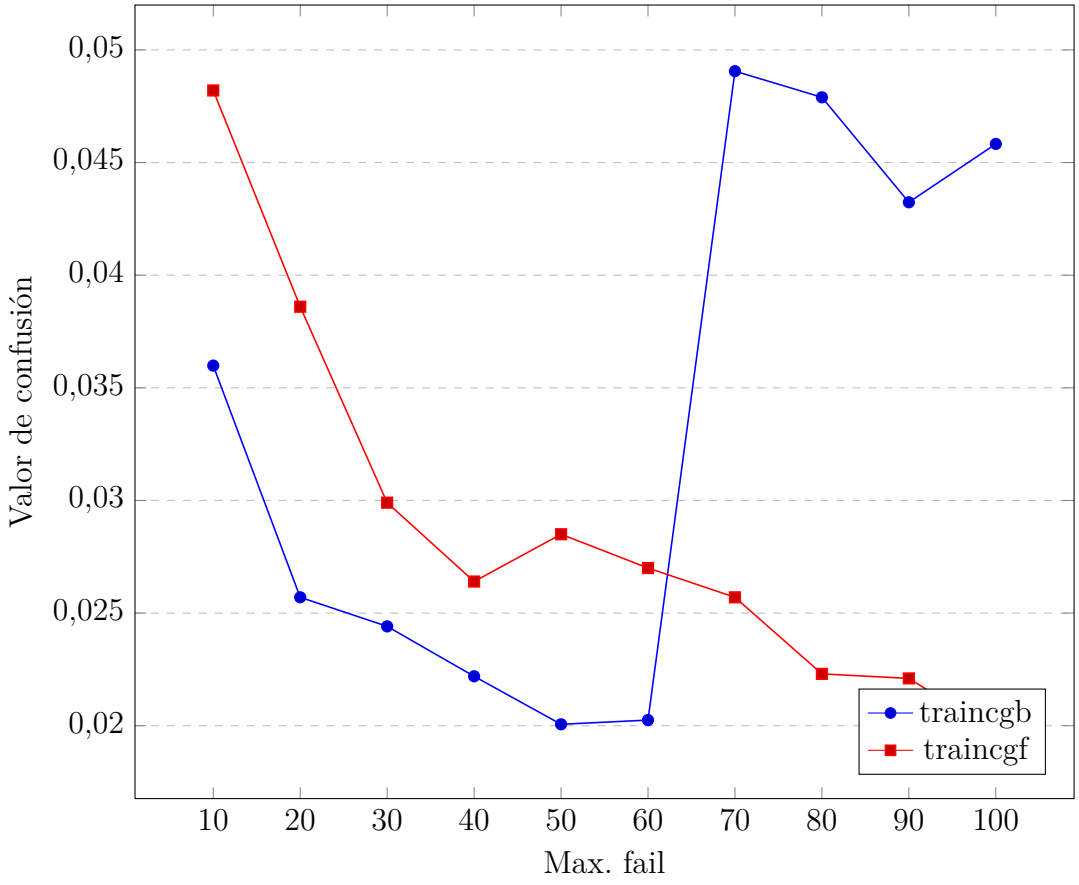


Figura 2: Rendimiento mejor caso `traincgb`.

Hemos obserdo que los valores `alpha` y `beta` no afectan en la medida que afecta `max_fail`. Por esto, nos hemos limitado a graficar este último parámetro.



E. Conclusiones

El rendimiento del algoritmo es muy sensible al ajuste del *learning rate*. Si se establece un valor muy alto el algoritmo oscila y se vuelve inestable. Por el contrario, si se establece un valor muy bajo, la red neuronal puede demorarse demasiado en converger.

TODO

Referencias

[1] D.J. Newman A. Asuncion. UCI machine learning repository, 2007. URL https://archive.ics.uci.edu/ml/datasets/Thyroid+Disease.

[2] Matlab documentation. traincgf command, 2016. URL https://www.

mathworks.com/help/nnet/ref/plotperform.html. [Online; accessed 08-November-2016].

[3] Matlab documentation. traincgb command, 2016. URL https://www.mathworks.com/help/nnet/ref/traincgb.html. [Online; accessed 08-November-2016].

[4] Matlab documentation. traincgf command, 2016. URL https://www.mathworks.com/help/nnet/ref/traincgf.html. [Online; accessed 08-November-2016].