Universidad de Burgos

Computación Neuronal y Evolutiva

P3: Multilayer Perceptron (MLP)

Diseñar y entrenar distintos Perceptrón 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. Introducción

El objetivo de la práctica es diseñar y entrenar distintos Perceptrón 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 seleccionado 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 los 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 indicadores generales se han empleado:

- `C`: *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: P3b_MLP.*

D. Estudio

En primer lugar, se han utilizado los parámetros descritos en la sección anterior 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 parámetros elegidos son los más relevantes para este problema en concreto y, por tanto, los más interesantes de estudiar.

Todos los datos generados durante los 320 experimentos del estudio (archivos CSV con los datos recogidos, imágenes con las gráficas de rendimiento y tablas Excel de resumen) se encuentran en el directorio del 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 cada 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:

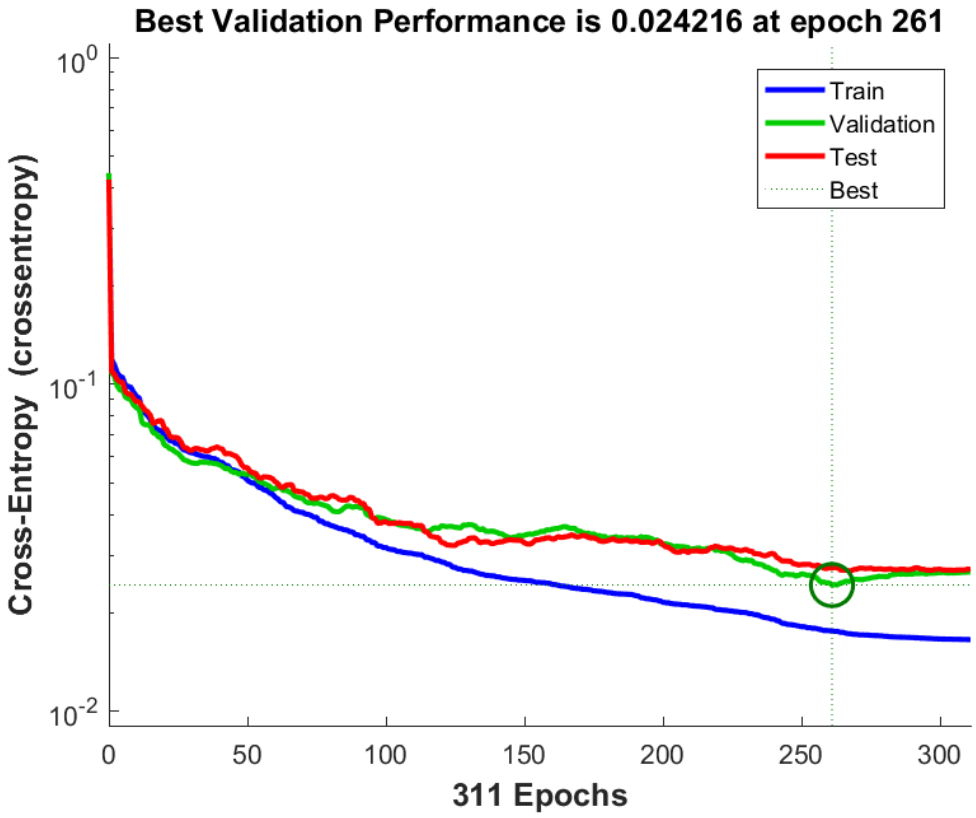


Figura 1: Rendimiento mejor caso `traincgb`.

Mejor caso para el algoritmo `traincgf`:

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

Su gráfico de rendimiento fue:

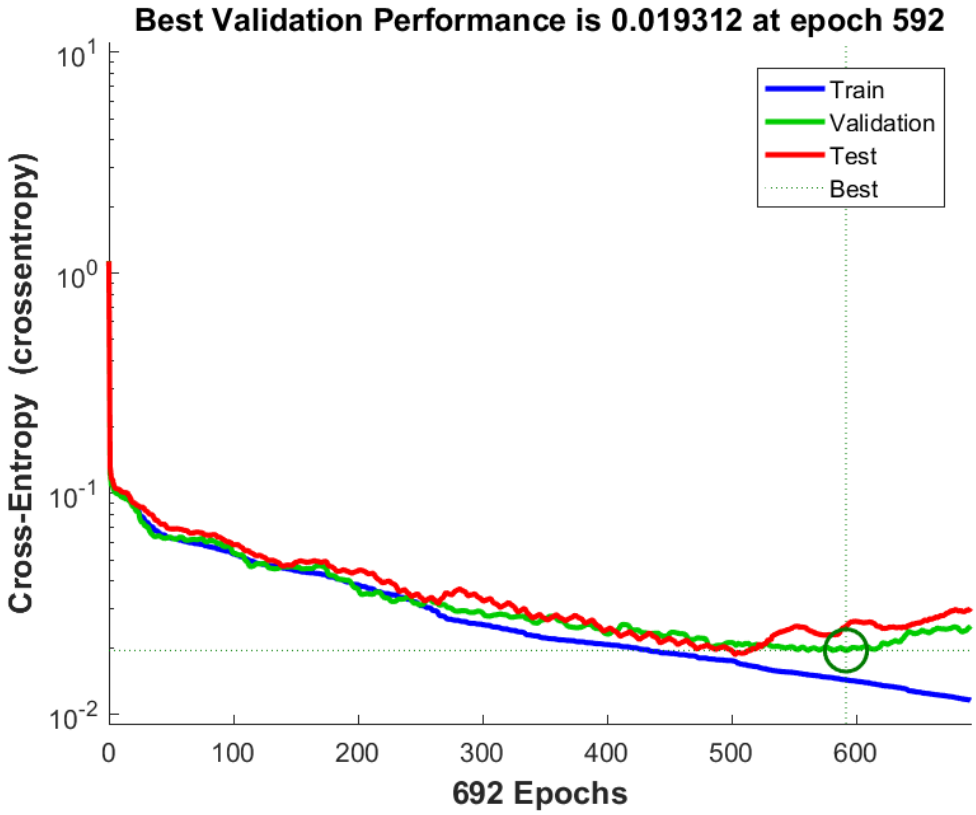
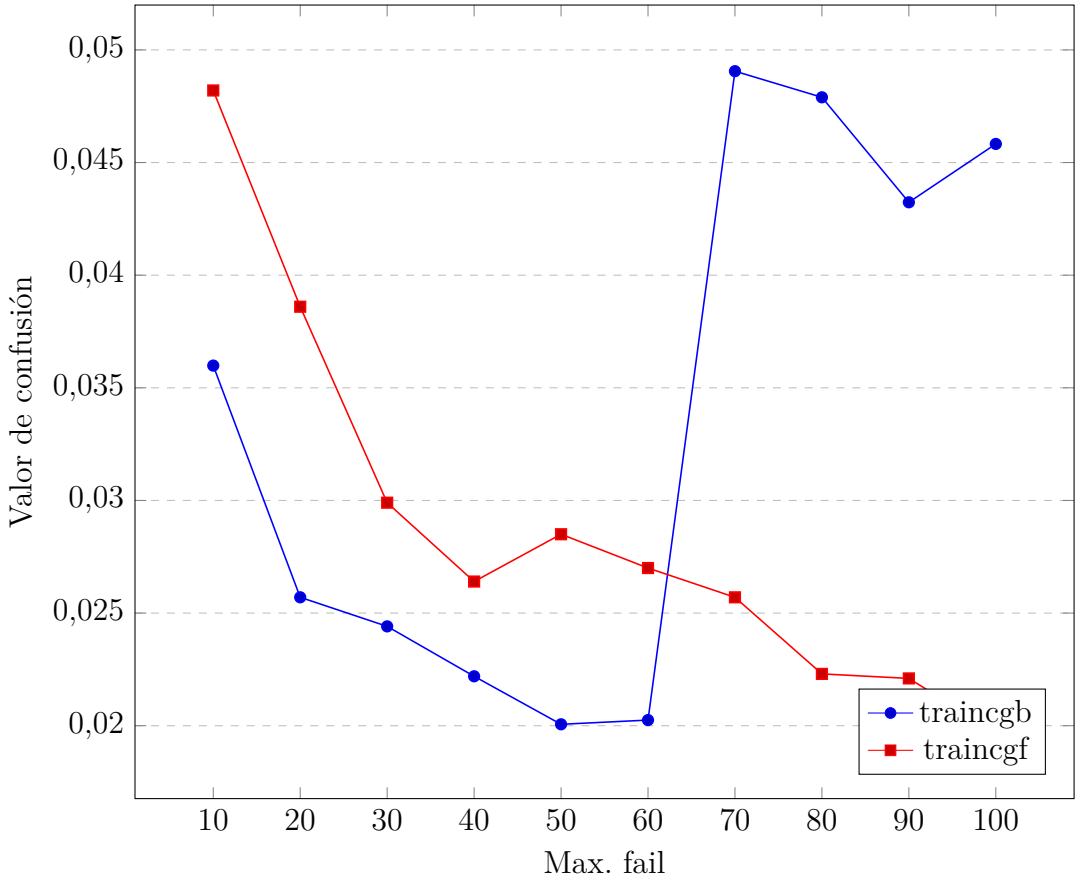


Figura 2: Rendimiento mejor caso `traincgf`.

Hemos observado 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.



Como podemos observar el algoritmo `traincgb` alcalza sus valores mínimos en torno a 50 y luego se dispara. En cambio, el valor de confusión con el algoritmo `traincgf` tiende a cero según se incrementa el parámetro `max_fail`.

Para comprobar la tendencia decreciente del segundo algoritmo, hemos realizado otra prueba incrementando el límite superior del parámetro `max_fail` a 10.000. De manera que el criterio de parada es el número de *epochs* (1000). El mejor caso obtenido es:

- `max_fail`: 10.000.
- `alpha`: 0.004.
- `beta`: 0.5.

Su gráfico de rendimiento fue:

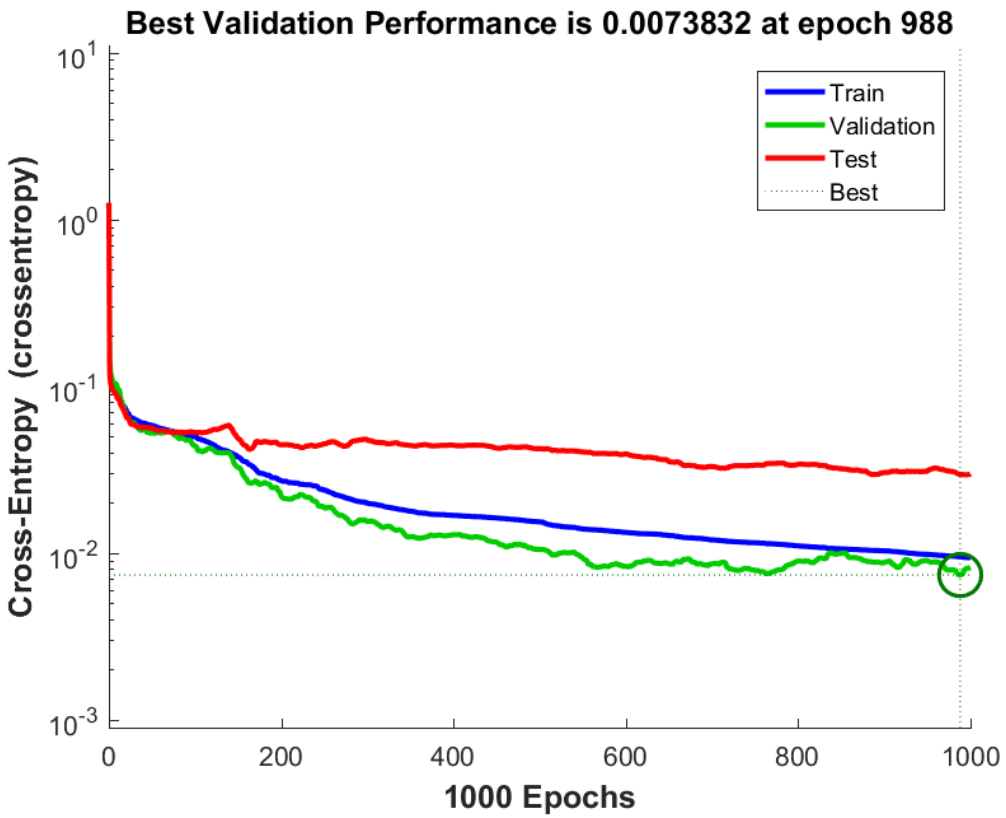


Figura 3: Rendimiento mejor caso `traincgf`.

Vemos como se ha reducido el error hasta un 0,7 %. Lo cual es una gran mejora respecto a los resultados obtenidos en la primera práctica que estaban en torno al 4,5 % o el 2 % obtenido en el primer experimento de este estudio.

*El experimento se ha repetido 20 veces para descartar que este buen rendimiento obtenido fuese cosa del azar.

E. Conclusiones

Tras el análisis de los datos obtenidos, podemos concluir que:

1. El límite de *epochs* por defecto (1.000) es un límite suficientemente alto como no influir en el entrenamiento.
2. El límite de la función de rendimiento por defecto es 0, lo cual significa que hasta que la red neuronal no sea perfecta, esta no va a parar de entrenar. Obviamente, según se aumenta este parámetro los resultados van a empeorar (aunque mejorará el tiempo de entrenamiento).
3. El tiempo de aprendizaje por defecto es infinito. En nuestro caso, hasta 8 segundos no empezaría a afectar al entrenamiento, y obviamente, cuanto más se reduzca peor entrenada estará la red.
4. En nuestro caso, con los *learning rates* (para el bias y los pesos) estudiados, no hemos encontrado variación significativa. En pruebas posteriores, se ha visto que estos producían variaciones significativas entre órdenes de magnitud diferentes. Cuanto más altos eran, más rápido era el entrenamiento, pero más varianza había en el entrenamiento de la red.
5. El único criterio de parada que realmente detenía el entrenamiento en nuestro problema era el parámetro `max_fail`. Aumentando este y utilizando `traincgf` se consiguieron mejorar los resultados notablemente. Sin embargo, el algoritmo `traincgb` conseguía su mejor rendimiento entorno a valores de 50, pero empeoraba según se aumentaban más.

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].