Universidad de Burgos

Computación Neuronal y Evolutiva

# P1: Thyroid

Diseñar y entrenar una red neuronal para clasificar pacientes en tres grupos clínicos dependiendo de la patología en la glándula tiroides (pacientes sanos, con hipertiroidismo o con hipertiroidismo).

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 una red neuronal para la clasificación de pacientes en tres grupos dependiendo de la patología en la glándula tiroides. Se trata de un problema de reconocimiento de patrones en el que dados unos datos de entrada, la red neuronal será capaz de asignarlos a una de las tres clases definidas (pacientes sanos, con hipertiroidismo o con hipotiroidismo). Haremos uso del dataset de ejemplo *Thyroid* que provee Matlab.

Realizaremos un estudio sobre el ajuste en el número de neuronas, ejecutando varios entrenamientos con la red y modificando el valor de este parámetro. También estudiaremos el rendimiento que proporcionan los diferentes algoritmos de entrenamiento incluidos en Matlab. Por último, discutiremos qué configuración nos proporciona los mejores resultados, explotaremos la red y analizaremos las salidas devueltas.

# B. Motivación

La motivación que nos llevó a elegir el tema del trabajo fue el que ambos poseíamos algún familiar que sufría alguna alteración en la glándula tiroides. Además, el gran desarrollo que están teniendo las redes neuronales aplicadas a la medicina en los últimos años (ej. IBM Watson [2]) nos desató interés para adentrarnos en este tema.

# C. Descripción del conjunto de datos

El conjunto de datos utilizados los hemos 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.

La descripción precisa del significado real de cada atributo y los valores que puede tomar la podemos encontrar en la página web del repositorio [1]. Para nuestra tarea no nos es relevante.

Nuestro objetivo es construir una red neuronal y entrenarla con los datos reales proporcionados para que posteriormente sea capaz de clasificar a nuevos pacientes.

Dividiremos aleatoriamente los datos que contiene el data set en tres grupos:

- $Training(70\,\%)$: lo utilizaremos para el entrenamiento supervisado.
- $Validation(15\,\%)$: lo utilizaremos para comprobar cuándo la red está generalizando y detener el entrenamiento para evitar el sobreajuste (*overfitting*).
- $Test(15\,\%)$: lo utilizaremos de forma totalmente independiente al entrenamiento para comprobar la generalización de la red.

# D. Descripción del procedimiento

Para resolver la tarea planteada creamos una red neuronal con una topología patternnet. Esta topología posee dos capas *feedforward*, con una función de activación sigmoidal en la capa oculta y una función *softmax* en la capa de salida. Se caracteriza por poder clasificar vectores con una precisión arbitrariamente buena si posee suficientes neuronas en su capa oculta.

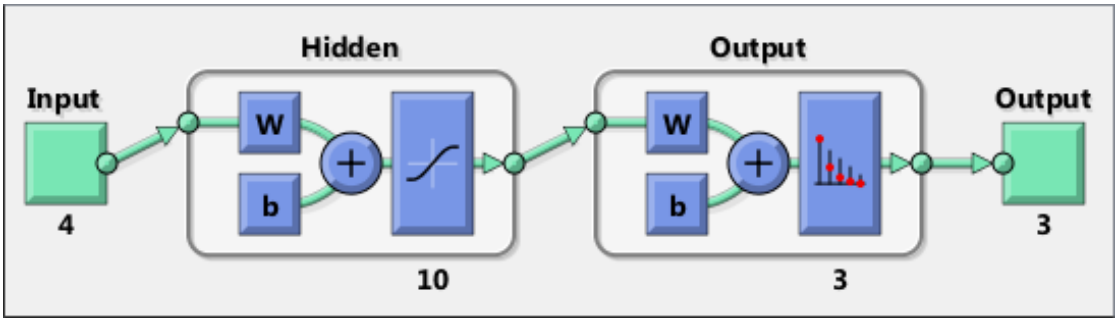



Figura 1: Pattern recognition network.

En nuestro estudio vamos a variar el número de neuronas de la capa oculta para intentar conseguir los mejores resultados (tanto en error, como en tiempo de

entrenamiento). La capa de salida siempre va a tener tres neuronas, cada una se encargará de evaluar la pertenencia a una de las tres clases.

Otro elemento que puede afectar al rendimiento de nuestra red es la función de entrenamiento utilizada. Probaremos ocho funciones distintas que implementa Matlab para comparar resultados.

Para realizar el estudio lo más automatizado posible, hemos creado una función ($autoTrain$) que se encarga de entrenar la red de acuerdo con los parámetros pasados. Por otro lado tenemos un script ($trainAndShow.m$) que llama a esta función variando el número de neuronas de la capa oculta (de 1 a 50 con paso 1) y los algoritmos de entrenamiento ($trainbfg$, $trainrp$, $trainscg$, $traincgb$, $traincgf$, $traincgp$, $trainoss$ y $traingdx$). Además, repite el entrenamiento de cada caso 10 veces para disminuir la varianza y la aleatoriedad.

El script proporciona como salida el valor de confusión medio en cada caso, la matriz de confusión completa y el tiempo medio de entrenamiento. Con tres formatos de salida, un primer formato legible para humanos, otro formateado para la librería de gráficos *pgfplots* y un tercer formato como CSV.

**Todo el código generado en esta práctica se encuentra disponible en el siguiente repositorio: P1_Thyroid.*

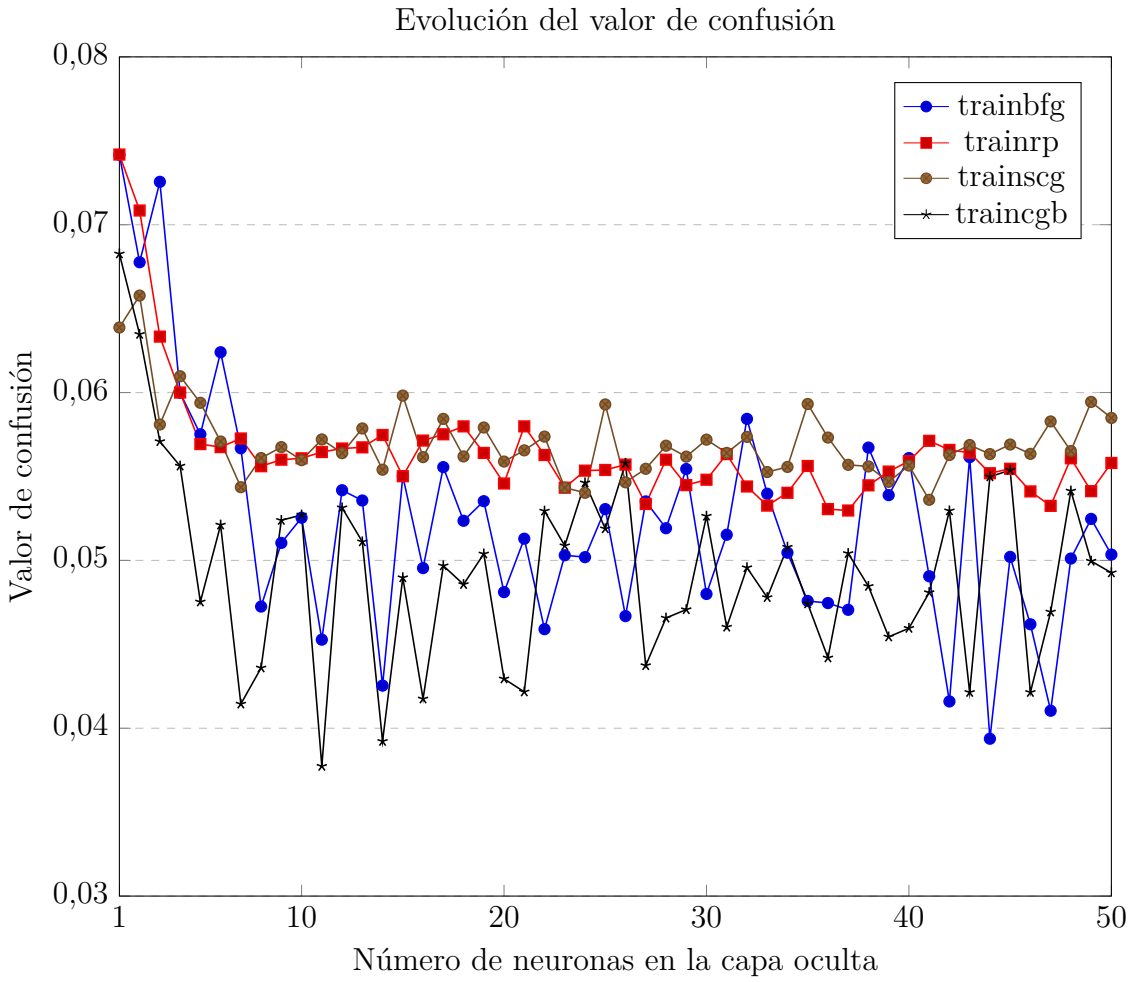
# E. Estudio

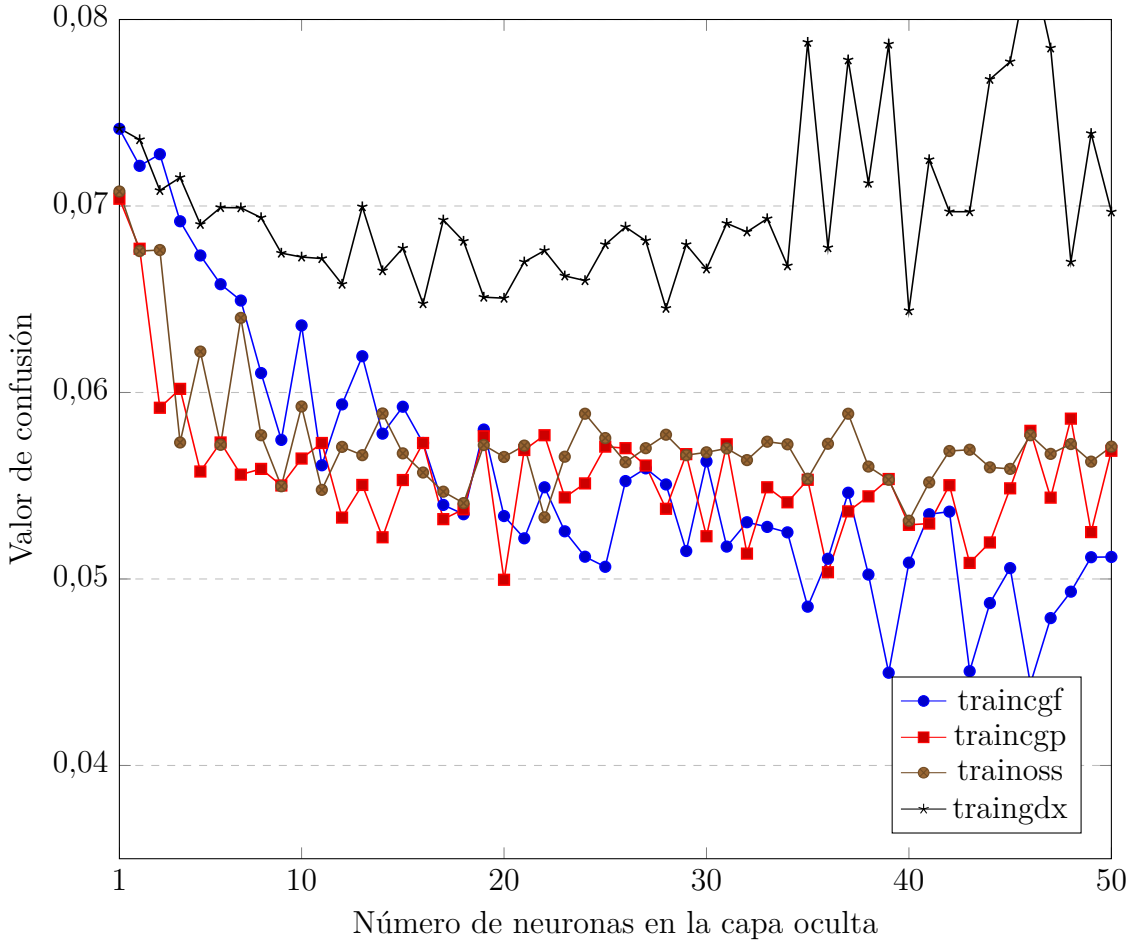
Tras realizar la ejecución del programa descrito en la sección anterior, hemos obtenido los datos que se muestran en las siguientes gráficas.

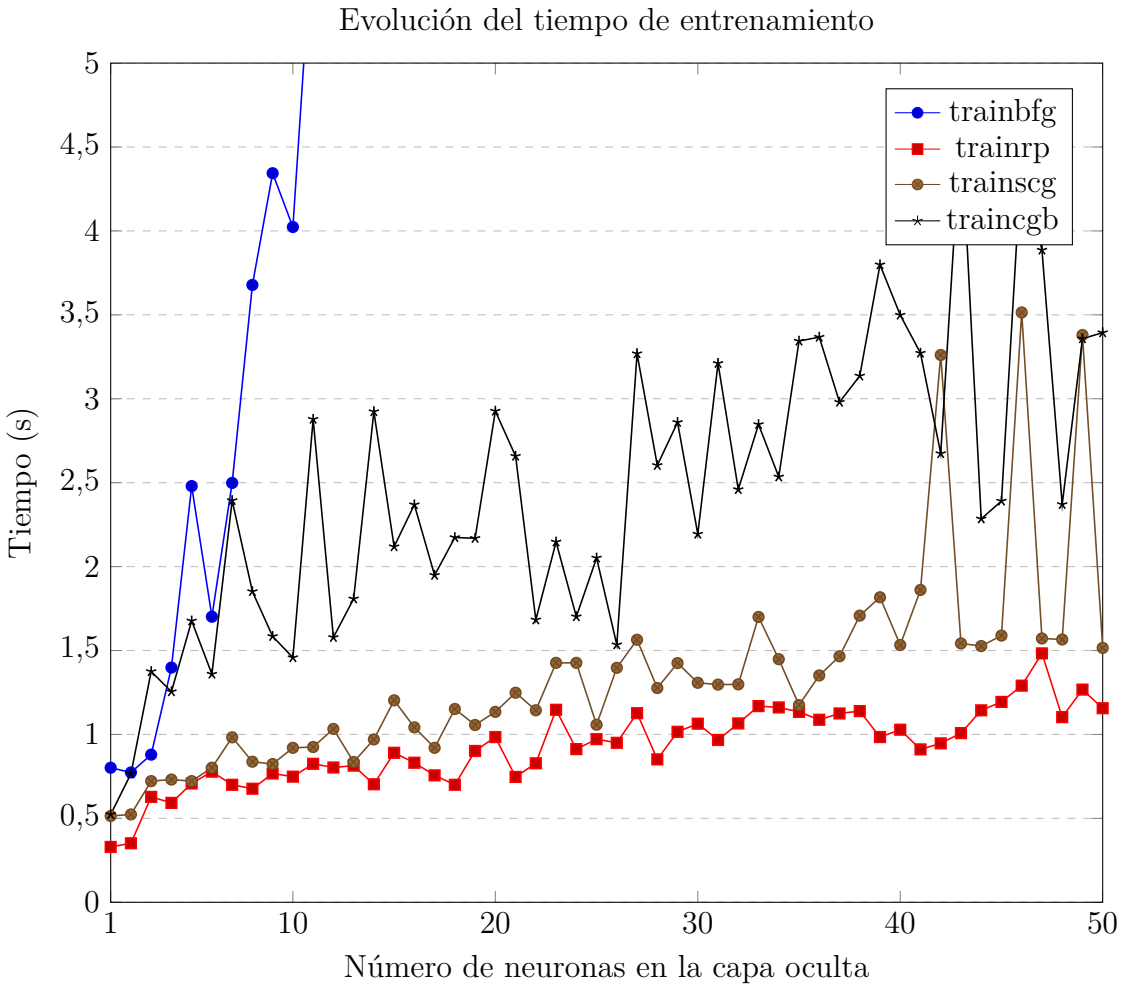
En las dos primeras se muestra, para cada algoritmo de entrenamiento, la evolución del valor de confusión total ponderado de todos los conjuntos de datos (entrenamiento, validación y pruebas) según se incrementa el número de neuronas en la capa oculta.

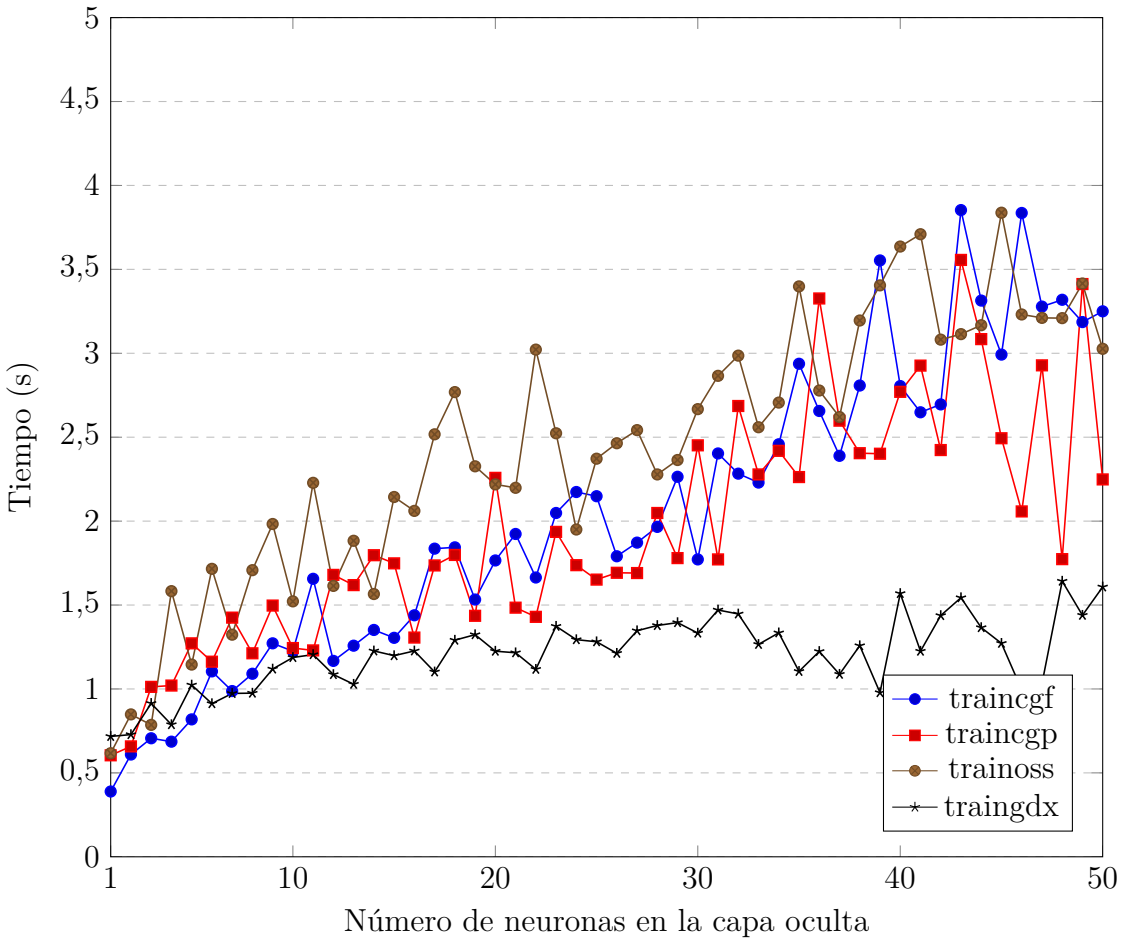
**Hemos dividido los resultados en dos gráficas para mejorar la legibilidad de las mismas.*

En las dos segundas se muestra la evolución del tiempo empleado en el entrenamiento de la red neuronal según se aumenta el número de neuronas en la capa oculta.


Evolución del valor de confusión
Valor de confusión
Número de neuronas en la capa oculta
0,08
0,07
0,06
0,05
0,04
0,03
1
10
20
30
40
50
trainbfg
trainrp
trainscg
traincgb


Valor de confusión
Número de neuronas en la capa oculta
0,08
0,07
0,06
0,05
0,04
1
10
20
30
40
50
traincgf
traincgp
trainoss
traingdx


Evolución del tiempo de entrenamiento
Tiempo (s)
Número de neuronas en la capa oculta
trainbfg
trainrp
trainscg
traincgb
0
0,5
1
1,5
2
2,5
3
3,5
4
4,5
5
1
10
20
30
40
50


Tiempo (s)
Número de neuronas en la capa oculta
traincgf
traincgp
trainoss
traingdx
0
0,5
1
1,5
2
2,5
3
3,5
4
4,5
5
1
10
20
30
40
50

En primer lugar, destacamos la desequilibrada distribución de pacientes de cada clase que contiene el conjunto de datos utilizado. Existen 166 pacientes sanos, 368 con hipertiroidismo y 6.666 pacientes con hipotiroidismo. Lo cual ha provocado que hasta las redes neuronales más simples (pocas neuronas) tengan un error relativamente bajo.

Por ejemplo, una red neuronal de una neurona entrenada con el algoritmo *trainbfg* obtiene un valor de confusión total ponderado del 7,4 %. Clasificando todos los datos como pertenecientes al grupo de hipotiroidismo (el más numeroso).

Respecto a los resultados observados en los dos primeros gráficos, cabe destacar que no hay mucha diferencia en el valor de confusión de un algoritmo de entrenamiento a otro, con la excepción del algoritmo *traingdx* que produce unos resultados bastante peores que la media.

Se observa una mejora en el aprendizaje según se aumenta el número de neuronas de la capa oculta. Empezando en torno al 7 %, mejorando hasta aproximadamente un 5,7 % de media con diez neuronas y estabilizándose en torno al 5,5 %.

Los resultados más descados nos los proporcionan los algoritmos *trainbfg, traincgb* y *traincgf*. Los dos primeros aprenden rápidamente, obteniendo valores entre 4 % y 4,5 % en torno a las 10 neuronas. El tercero tiene una pendiente menos pronunciada pero que se mantiene según se aumenta el número de neuronas, llegando a valores en torno al 4,5 %.

Desde el punto de vista del tiempo necesario para entrenar las redes neuronales en relación al algoritmo de entrenamiento usado, cabe destacar que el algoritmo *trainbfg*, a pesar de que en el apartado anterior poseía unos buenos resultados, su tiempo de entrenamiento es completamente irrazonable llegando a 140 segundos con 50 neuronas, cuando el resto de algoritmos no supera los 5 segundos.

Por otro lado, el algoritmo *traingdx* que obtenía uno de los peores resultados en valor de confusión, presenta uno de los mejores tiempos de entrenamiento junto con *trainnrp* y *trainscg*.

El resto de algoritmos tienen un tiempo medio de unos 3-3,5 segundos para 50 neuronas, lo cual es un tiempo más que aceptable.

# F. Mejor resultado obtenido

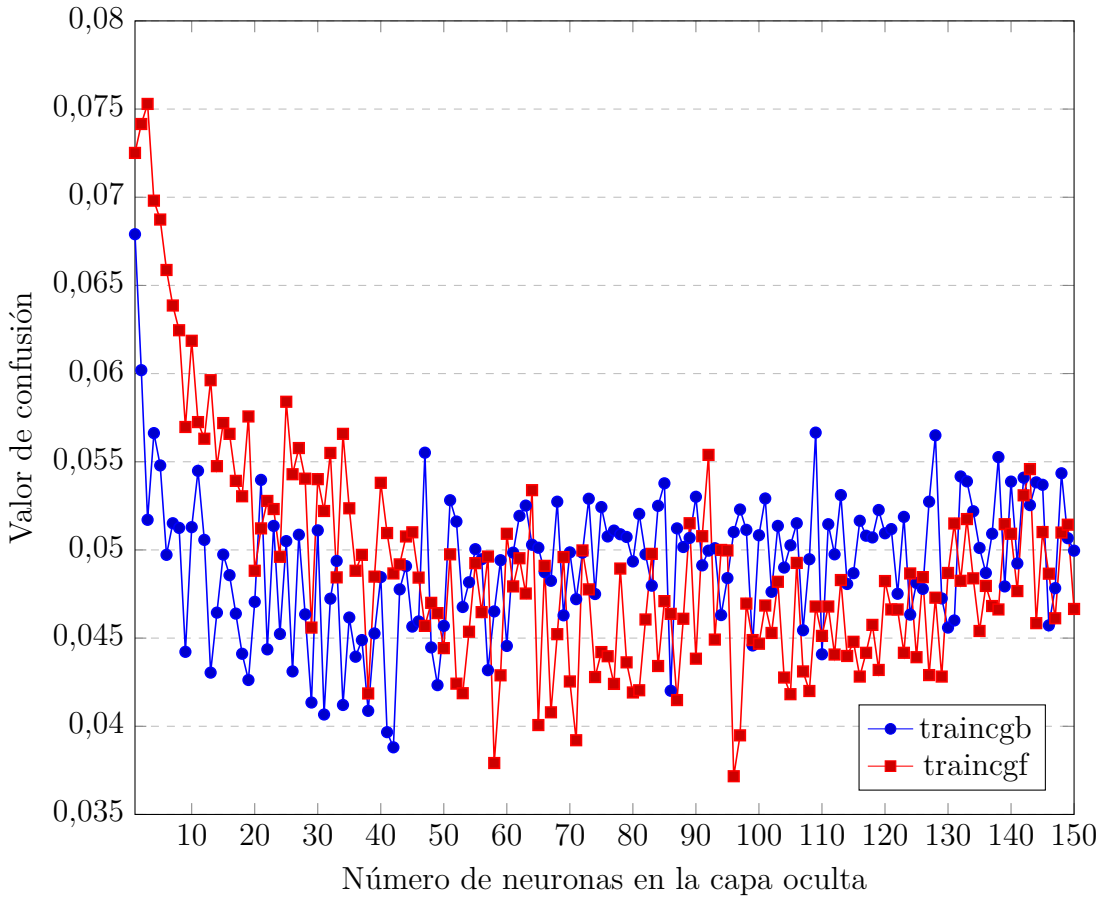
Los mejores resultados obtenidos, teniendo en consideración tanto el tiempo como el error total ponderado, son *traincgb* y *traincgf*.

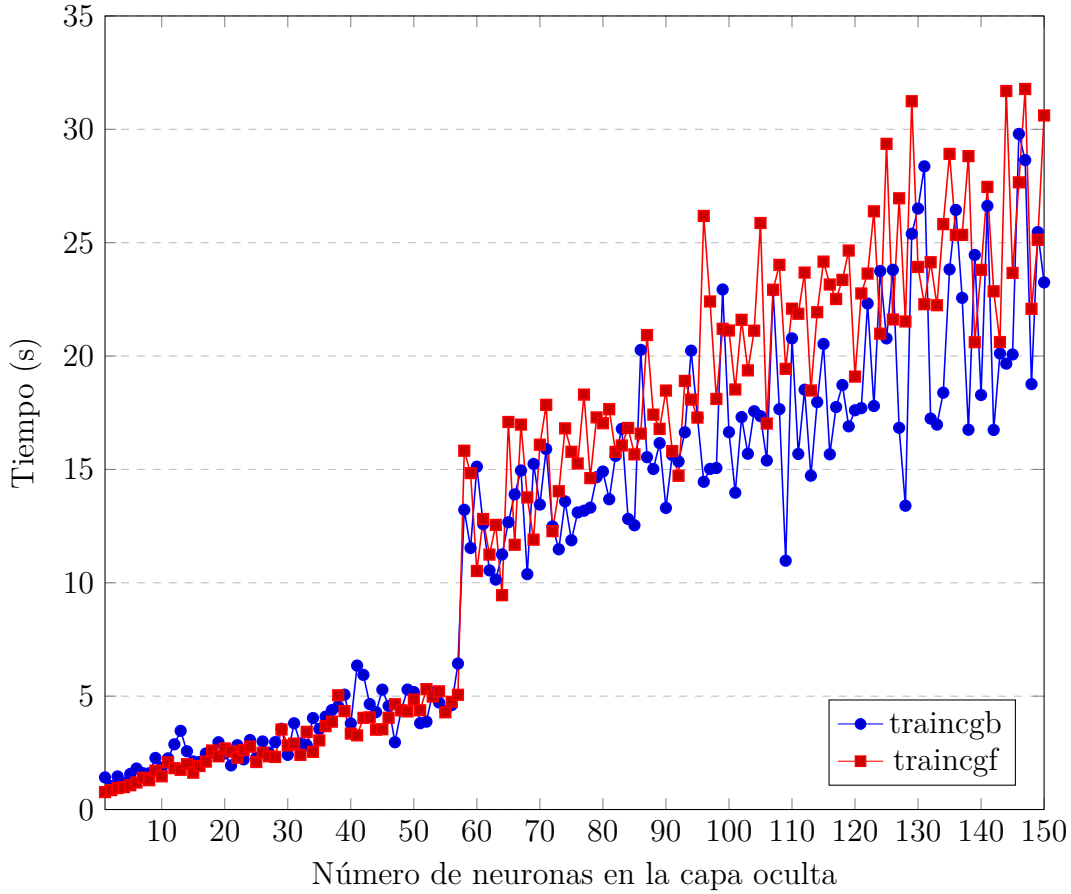
Decidimos ampliar el estudio con estos dos algoritmos para comprobar si se mantenía el descenso del error total ponderado observado. También nos interesaba comprobar si llegaría a haber diferencias significativas entre ambos.

Ampliamos el número de neuronas en la capa oculta progresivamente hasta 150. Tras observar los resultados, vimos que no existe una mejoría al seguir aumentando el número de neuronas y que lo observado anteriormente era un artefacto de la varianza en un conjunto de entrenamiento tan pequeño.

La diferencia entre ambos algoritmos de entrenamiento no ha llegado a ser suficiente para concluir inequívocamente que uno es mejor que el otro. Pero debido a que no disponemos de tiempo ilimitado para comprobar descartar uno de los dos, hemos decidido tomar los resultados obtenidos como significativos.

En ellos se puede observar una ligera ventaja de *traincgb* con respecto a *traincgf* con un número de neuronas menor de 20. Por lo que nos decantamos por *traincgb* como el algoritmo que mejor rendimiento tiene en el problema tratado.






# G. Explotación de la red

Para la explotación de la red intentamos buscar otros datasets del mismo tema pero con distintas fuentes, para poder evaluar la calidad real con nuevos datos. Sin embargo, todos los datasets sobre tiroides que encontramos no coincidían en los atributos usados.

Por lo tanto, procedimos a creat datos falsos con el procedimiento que seguimos en la clase de prácticas: multiplicar datos que ya teníamos por un porcentaje alto para ver si al reintroducirlos en la red neuronal se consiguen los mismos resultados.

Se ha creado un script para hacer esta última parte automáticamente. La ejecución del mismo nos ha dado los siguientes resultados con los datos reales:

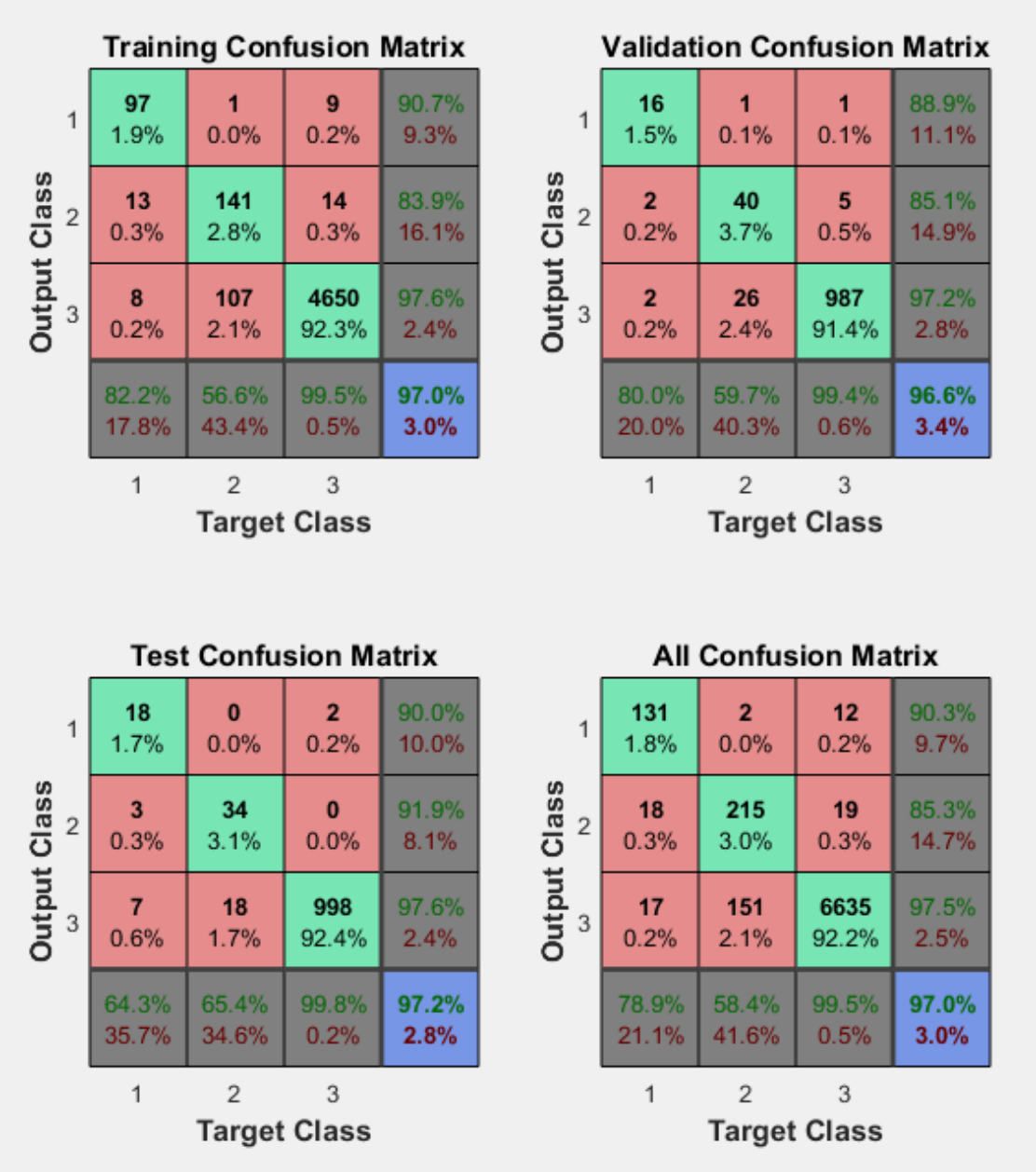



Figura 2: Matriz de confusión.

Al introducir datos multiplicados por 0.95 obtenemos 0.031806 de error total. Mientras que al multiplicarlos por 1.05 obtenemos 0.029306. Como vemos, se obtienen unos resultados bastante buenos.

# Referencias

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

[2] Wikipedia. Watson (computer) — wikipedia, the free encyclopedia, 2016. URL https://en.wikipedia.org/w/index.php?title=Watson_(computer)#Healthcare&oldid=741165271. [Online; accessed 29-September-2016].