Universidad de Alcalá

Práctica 1. Identificación y control neuronal (I)

Sistemas de Control Inteligente

Grado en Ingeniería Computadores
Grado en Ingeniería Informática
Grado en Sistemas de Información
Universidad de Alcalá

Toolbox necesarias para esta práctica:

- **Redes neuronales:**
  - ***Deep Learning Toolbox*** (versiones R2018b y posteriores)
  - ***Neural Network Toolbox*** (versiones R2018a y anteriores)
- **Signal Processing Toolbox**

Ejercicio 1. Perceptron

Se desea clasificar un conjunto de datos pertenecientes a cuatro clases diferentes. Los datos y las clases a las que pertenecen con los que se muestra a continuación:

x_1	x_0	Clase
0.1	1.2	2
0.7	1.8	2
0.8	1.6	2
0.8	0.6	0
1.0	0.8	0
0.3	0.5	3
0.0	0.2	3
-0.3	0.8	3
-0.5	-1.5	1
-1.5	-1.3	1

Se desea diseñar un clasificador neuronal mediante un perceptron simple que clasifique estos datos. Diseñe el clasificador, visualice los parámetros de la red y dibuje los datos junto con las superficies que los separan.

¿Consigue la red separar los datos?, ¿cuántas neuronas tiene la capa de salida?, ¿por qué?

¿Qué ocurre si se incorpora al conjunto un nuevo dato: [0.0 -1.5] de la clase 3?

Ejercicio 2. Aproximación de funciones

Una de las aplicaciones inmediatas de las redes neuronales es la aproximación de funciones. Para ello, Matlab dispone de una red optimizada, *fitnet*, con la que se trabajará en este ejercicio. El objetivo en este caso es aproximar la función $f = sinc(t)$ tal y como se muestra a continuación:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% APROXIMACIÓN DE FUNCIONES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all; close all;

%    DEFINICIÓN DE LOS VECTORES DE ENTRADA-SALIDA
%    ============================================

t = -3:.1:3; % eje de tiempo
F=sinc(t)+.001*randn(size(t)); % función que se desea aproximar

plot(t,F,'+');
title('Vectores de entrenamiento');
xlabel('Vector de entrada P');
ylabel('Vector Target T');

%    DISEÑO DE LA RED
%    ================

hiddenLayerSize = 4;
net = fitnet(hiddenLayerSize,'trainrp');

net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

net = train(net,t,F);

Y=net(t);

plot(t,F,'+'); hold on;
plot(t,Y,'-r'); hold off;
title('Vectores de entrenamiento');
xlabel('Vector de entrada P');
ylabel('Vector Target T');
```

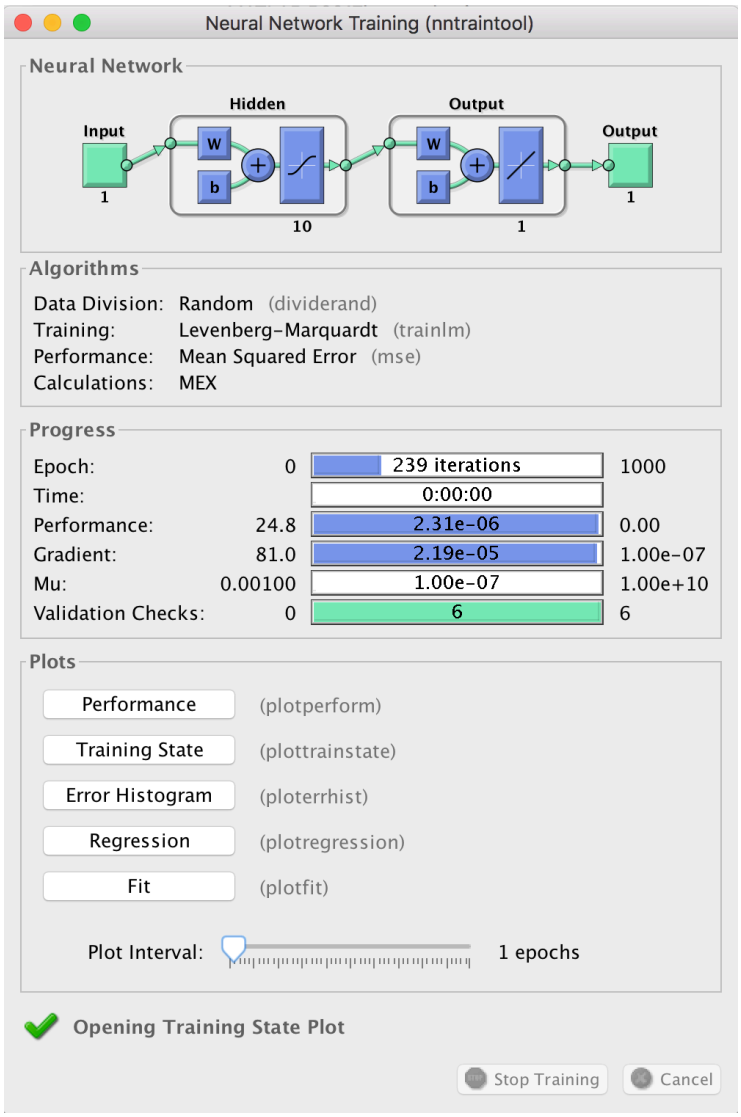
Estudie los efectos sobre la solución final de modificar el método de entrenamiento (consulte la ayuda de Matlab y pruebe 4 métodos diferentes) y el número de neuronas de la capa oculta.

Ejercicio 3. Aproximación de funciones (II)

En este ejercicio, se estudiarán en detalle las herramientas que facilita Matlab para el diseño y prueba de redes neuronales ejecutando el siguiente código de ejemplo:

```
% Carga de datos de ejemplo disponibles en la toolbox
[inputs,targets] = simplefit_dataset;

% Creación de la red
hiddenLayerSize = 10;
net = fitnet(hiddenLayerSize);

% División del conjunto de datos para entrenamiento, validación y test
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Entrenamiento de la red
[net,tr] = train(net,inputs,targets);

% Prueba
outputs = net(inputs);
errors = gsubtract(outputs,targets);
performance = perform(net,targets,outputs)

% Visualización de la red
view(net)
```

Al ejecutar el script, se muestra la siguiente ventana:



← Esquema de la red

← Detalles de los algoritmos

← Progreso del proceso de entrenamiento y test

← Gráficas con diversos resultados

Explore las gráficas disponibles:

- plotperform: gráfica que representa el error en función del número de épocas para los datos de entrenamiento, validación y test.
- plottrainstate: evolución del entrenamiento.
- plotrrhist: histograma del error.
- plotregression y plotfit: ajuste de los datos de entrenamiento, validación y test.

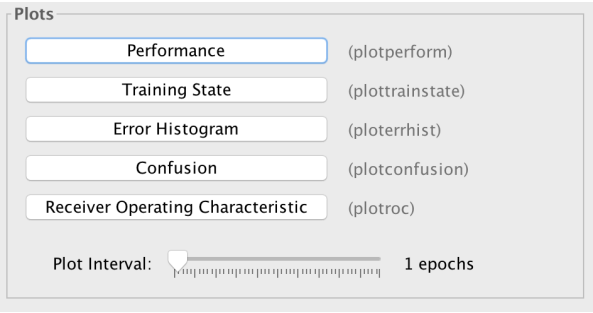
Pruebe este mismo script con el conjunto de datos bodyfat_dataset, y evalúe sus resultados. Estudie la mejora que supone utilizar distintos métodos de entrenamiento y una división diferente de los datos (entrenamiento, validación y test).

Ejercicio 4. Clasificación.

La clasificación de patrones es una de las aplicaciones que dieron origen a las redes neuronales artificiales. Como en el caso anterior, la toolbox de redes neuronales de Matlab dispone de una red optimizada para la clasificación, *patternnet*, que analizaremos en este ejemplo.

```
% Carga de datos de ejemplo disponibles en la toolbox
[inputs,targets] = simpleclass_dataset;

% Creción de una red neuronal para el reconocimiento de patrones
hiddenLayerSize = 10;
net = patternnet(hiddenLayerSize);

% División del conjunto de datos para entrenamiento, validación y test
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;

% Entrenamiento de la red
[net,tr] = train(net,inputs,targets);

% Prueba
outputs = net(inputs);
errors = gsubtract(targets,outputs);
performance = perform(net,targets,outputs)

% Visualización
view(net)
```

La ejecución del script muestra una ventana similar a la anterior en la que cambian algunas de las gráficas disponibles para mostrar los resultados como se ve en la siguiente figura:



En lugar de las gráficas específicamente relacionadas con la aproximación de una función, en el caso de una tarea de clasificación, se ofrecen:

- plotconfusion: matrices de confusión de los resultados.
- plotroc: curvas roc (característica operativa del receptor).

Pruebe este mismo script con el conjunto de datos cancer_dataset, y evalúe sus resultados. Estudie de nuevo la mejora que supone utilizar distintos métodos de entrenamiento y una división diferente de los datos (entrenamiento, validación y test).