EJERCICIO 1:

```
clearvars
clear all
close all
clc
```

Importamos la data:

```
load glass.data
Data = glass;
clearvars -except Data;
```

Le corresponde una altura:

```
hData = height(Data);
```

Dividimos la data en conjuntos de entrenamiento (80%), validación (10%) y prueba (10%). Para ello usamos la función `dividerand`. Tomamos los ratios:

```
trainRatio = .8;
valRatio = .1;
testRatio = .1;
```

Usamos la función correspondiente:

```
[trainInd,valInd,testInd] = dividerand(hData,trainRatio,valRatio,testRatio);
```

La Data de entrenamiento sería:

```
DataTrain = Data(trainInd',:);
```

Identificamos los valores de entrada correspondientes (input values):

```
InputDataTrain = DataTrain(:,1:end-1);
```

Y los valores objetivo de salida asociados (target values):

```
TargetDataTrain = DataTrain(:,end);
```

Tomamos la cantidad de neuronas como las que de tipo de vidrio existen:

```
nClases = 7;
hiddenSizes = nClases;
```

Entrenamos la red con la función de aprendizaje por defecto `'trainlm'`:

```
net = feedforwardnet(hiddenSizes,'trainlm');
net= train(net,InputDataTrain',TargetDataTrain');
perf = perform(net,net(InputDataTrain'),TargetDataTrain')
```

```
perf = 0.0295
```

Identificamos también la data de prueba:

```
DataTest = Data(testInd',:);
```

Identificamos los valores de entrada correspondientes (input values):

```
InputDataTest = DataTest(:,1:end-1);
```

Y los valores objetivo de salida asociados (target values):

```
TargetDataTest = DataTest(:,end);
```

Al ser una red neuronal feedforward, sabemos que tiene al menos una capa oculta. Y desde que el problema nos sugiere usar la función `feedforwardnet`, sabemos que `feedforwardnet(hiddenSizes,trainFcn)` es tal que `hiddenSizes` es el número de neuronas, y `trainFcn` es la regla de entrenamiento. El problema sugiere usar la data de prueba:

```
ibeg = 5; iend = 40;
for i = ibeg:2:iend
    hiddenSizes = i;
```

Entrenamos la red con la función de aprendizaje por defecto `'trainlm'`:

```
    net = feedforwardnet(hiddenSizes,'trainlm');
    net = train(net,InputDataTest',TargetDataTest');
```

La función performance es el MSE. Al usar la lógica feedforward, tengo más libertad para ver las funciones de transferencia, que en principio son las funciones de transferencia por defecto.

```
    perf = perform(net,net(InputDataTest'),TargetDataTest');
    if i == ibeg
        OptPerf = [i;perf];
    else
        OptPerf = [OptPerf [i;perf]];
    end
end
```

Dado que piden determinar el número de neuronas de la capa oculta como la correspondiente a la red con el mejor performance alcanzado en la data de prueba, buscamos el número de neuronas correspondiente a la menor performance (MSE) del conjunto de entrenamientos antes computados iterativamente:

```
[~,PosOptPerf] = min(OptPerf(2,:));
```

El número de neuronas de la capa oculta sería:

```
nNeurHiddenLayer = OptPerf(1,PosOptPerf);
```

Antes de usar distintas funciones de aprendizaje, debemos fijarnos en el número de neuronas de la cara oculta (exigencia del problema)

```
hiddenSizes = nNeurHiddenLayer;
```

'trainbr':

```
net = feedforwardnet(hiddenSizes,'trainbr');
net = train(net,InputDataTest',TargetDataTest');
perfbr = perform(net,net(InputDataTest'),TargetDataTest');
```

'trainbfg':

```
net = feedforwardnet(hiddenSizes,'trainbfg');
net = train(net,InputDataTest',TargetDataTest');
perfbfg = perform(net,net(InputDataTest'),TargetDataTest');
```

'trainrp':

```
net = feedforwardnet(hiddenSizes,'trainrp');
net = train(net,InputDataTest',TargetDataTest');
perfrp = perform(net,net(InputDataTest'),TargetDataTest');
```

'trainscg':

```
net = feedforwardnet(hiddenSizes,'trainscg');
net = train(net,InputDataTest',TargetDataTest');
perfscg = perform(net,net(InputDataTest'),TargetDataTest');
```

Podemos evaluar como cambia la eficiencia viendo el comportamiento:

```
plot([1:4],[perfbr perfbfg perfrp perfscg],'r*')
```

EJERCICIO 2:

```
clear all
clearvars
close all
clc
```

Importamos:

```
load data2.mat
Data = force;
TargetData = target;
clearvars -except InputData TargetData
```

Le corresponde una altura:

```
hData = height(Data);
```

Dividimos la data en conjuntos de entrenamiento (80%), validación (10%) y prueba (10%). Para ello usamos la función `dividerand`. Tomamos los ratios:

```
trainRatio = .7;
valRatio = .15;
testRatio = .15;
```

Usamos la función correspondiente:

```
[trainInd,valInd,testInd] = dividerand(hData,trainRatio,valRatio,testRatio);
```

Identificamos los valores de entrada correspondientes (input values):

```
InputDataTrain = Data(trainInd',:);
```

Y los valores objetivo de salida asociados (target values):

```
TargetDataTrain = TargetData(trainInd',:);
```

Tomamos la cantidad de neuronas como las que de clase existen:

```
nClases = 3;
hiddenSizes = nClases;
```

Entrenamos la red con la función de aprendizaje por defecto `'trainlm'`:

```
net = feedforwardnet(hiddenSizes,'trainlm');
net= train(net,InputDataTrain',TargetDataTrain');
perf = perform(net,net(InputDataTrain'),TargetDataTrain')
```

Identificamos también la data de prueba. Los valores de entrada correspondientes (input values):

```
InputDataTest = Data(testInd',:);
```

Y los valores objetivo de salida asociados (target values):

```
TargetDataTest = TargetData(testInd',:);
```

Variamos la cantidad de neuronas como pide el problema:

```
ibeg = 5; iend = 20;
for i = ibeg:2:iend
    hiddenSizes = i;
    net = feedforwardnet(hiddenSizes,'trainlm');
    net = train(net,InputDataTest',TargetDataTest');
    perf = perform(net,net(InputDataTest'),TargetDataTest');
    if i == ibeg
        OptPerf = [i;perf];
    else
        OptPerf = [OptPerf [i;perf]];
    end
end
[~,PosOptPerf] = min(OptPerf(2,:));
nNeurHiddenLayer = OptPerf(1,PosOptPerf);
```

La otra opción sugiere hacer uso del análisis PCA. Trataremos en principio con toda la data, para hallar las componentes principales:

```
mData = mean(Data);
rData = range(Data);
sData = (force-mData)./rData;
[P,S,V] = pca(sData);
```

Para encontrar qué tantos componentes principales son tales que su varianza acumulada es del 95 por ciento, hacemos:

```
Vnorm = V./sum(V);
for i = 1:length(Vnorm)
    VarAcum = sum(Vnorm(1:i));
    if VarAcum>0.95
```

`nComp` guardará el orden del último componente principal que cumple con las condiciones del problema:

```
        nComp = i;
        break;
    end
end
```

El número de neuronas para las diferentes capas son (dato):

```
nNeur1 = 12; nNeur2 = 8;
```

Hacemos el entrenamiento correspondiente:

```
net = feedforwardnet([nNuer1 nNuer2]);
```

```
net = train(net,P(:,1:nComp),TargetData(:,1:nComp));
```

EJERCICIO 3:

```
clear all
clearvars
close all
clc
```

Cargamos la data:

```
load CreditRating.mat
creditrating.Rating = categorical(creditrating.Rating);
creditrating.Rating(creditrating.Rating  == 'AAA') = categorical(7);
creditrating.Rating(creditrating.Rating  == 'AA') = categorical(6);
creditrating.Rating(creditrating.Rating  == 'A') = categorical(5);
creditrating.Rating(creditrating.Rating  == 'BBB') = categorical(4);
creditrating.Rating(creditrating.Rating  == 'BB') = categorical(3);
creditrating.Rating(creditrating.Rating  == 'B') = categorical(2);
creditrating.Rating(creditrating.Rating  == 'C') = categorical(1);
```

Identificamos la data de entrada y salida:

```
InputData = creditrating(:,1:end-1);
TargetData = creditrating(:,end);
```

Nuestro número de neuronas a proponer:

```
nClases = 7;
hiddenSizes = nClases;
```

Usamos diferentes funciones de entranamiento:

1)

```
net=feedforwardnet(hiddenSizes,'trainlm');
net=train(net,InputData',TargetData');
perf=perform(net,net(InputData'),TargetData');
```

2)

```
net=feedforwardnet(hiddenSizes,'traingd');
net=train(net,InputData',TargetData');
perf=perform(net,net(InputData'),TargetData');
```

3)

```
net=feedforwardnet(hiddenSizes,'traingdm');
net=train(net,InputData',TargetData');
perf=perform(net,net(InputData'),TargetData');
```

4)

```
net=feedforwardnet(hiddenSizes,'traingdx');
net=train(net,InputData',TargetData');
```

5)

```
perf=perform(net,net(InputData'),TargetData');
net=feedforwardnet(hiddenSizes,'trainoss');
```

6)

```
net=train(net,InputData',TargetData');
perf=perform(net,net(InputData'),TargetData');
```

Otras arquitecturas a usar, podrían ser **purelin, adaline, y perceptrón**.