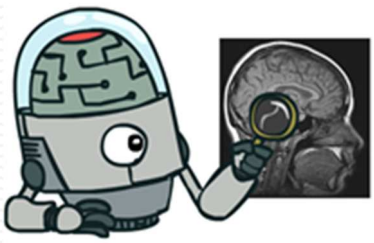
MACHINE LEARNING FOR DATA MINING
LECTURE 3-2: NEURAL NETWORKS

CLASSIFICATION USING MATLAB AND PYTHON

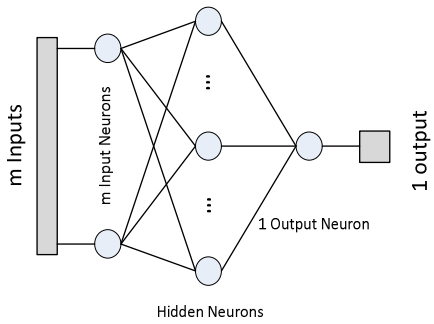
Siamak Sarmady (Urmia University of Technology)

Classification using Neural Networks in Matlab



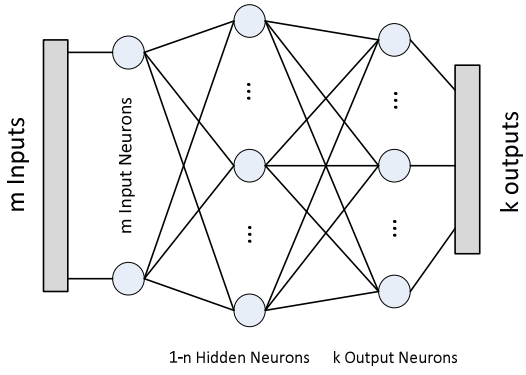
Binary Classifier

- For the binary classification using a neural network, a multilayer feed-forward network with a single neuron in the output layer is enough. By putting input values on the input nodes of the trained network, the output level will determine the class. Output levels below a threshold value are interpreted as class -1 (or 0) and higher values are treated as class 1.



Multiclass Classifier

- For multiclass classification with k possible classes, a feed-forward network with k output neurons may be used. In this method the neuron with highest output level determines the class.



Multiclass Classifier

- It is also possible to use binary coding for class identification. In this alternative method the classes are binary coded. For example if eight classes are used, the classes could be binary coded into three bits. A network with three output neurons can be used to specify any of the eight classes.

Class	Eight Output	Binary Coded
0	00000001	000
1	00000010	001
2	00000100	010
3	00001000	011
4	00010000	100
5	00100000	101
6	01000000	110
7	10000000	111

Sample Data

- We use the sample machine learning data sets at https://archive.ics.uci.edu/ml/datasets.html

- We use the "Car Evaluation" dataset which has 6 features at below URL. https://archive.ics.uci.edu/ml/datasets/Car+Evaluation

 Pittsburgh Bridges	Multivariate	Classification	Categorical, Integer	108	13	1990
 Car Evaluation	Multivariate	Classification	Categorical	1728	6	1997
 Census Income	Multivariate	Classification	Categorical, Integer	48842	14	1996

Data Definitions

- We first download the data file and save it under the name "car-eval-data.csv". Notice that the ".csv" extension will allow excel to open it.
- One of the files in the data set is called "car.c45-names" and contains the following information about the data. So apparently we have 6 input features and 4 possible classes.

class values:

unacc, acc, good, vgood

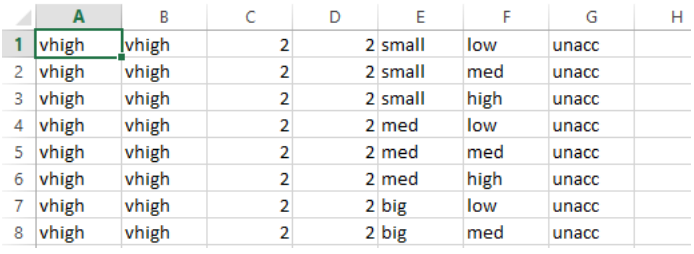
attributes:

buying: vhigh, high, med, low
maint: vhigh, high, med, low
doors: 2, 3, 4, 5more
persons: 2, 4, more
lug_boot: small, med, big
safety: low, med, high.

Data itself

- The following shows a few lines of the data file (in text format and inside Excel). The first 6 columns are the features and the last column is the output class.

vhigh,vhigh,2,2,small,low,unacc
vhigh,vhigh,2,2,small,med,unacc
vhigh,vhigh,2,2,small,high,unacc
vhigh,vhigh,2,2,med,low,unacc
vhigh,vhigh,2,2,med,med,unacc
vhigh,vhigh,2,2,med,high,unacc
vhigh,vhigh,2,2,big,low,unacc
vhigh,vhigh,2,2,big,med,unacc



	A	B	C	D	E	F	G	H
1	vhigh	vhigh	2	2	small	low	unacc	
2	vhigh	vhigh	2	2	small	med	unacc	
3	vhigh	vhigh	2	2	small	high	unacc	
4	vhigh	vhigh	2	2	med	low	unacc	
5	vhigh	vhigh	2	2	med	med	unacc	
6	vhigh	vhigh	2	2	med	high	unacc	
7	vhigh	vhigh	2	2	big	low	unacc	
8	vhigh	vhigh	2	2	big	med	unacc	

Data adjustments

- Most classification models (including Neural Networks which expects numerical values in its inputs) cannot work with strings. We should therefore convert the information to numerical. For this purpose we should assign numbers to classes (classes should start from 0) and features. We use the following conversions.

Attribute Values:

buying	v-high, high, med, low	-> 3,2,1,0
maint	v-high, high, med, low	-> 3,2,1,0
doors	2, 3, 4, 5-more	-> 2,3,4,5
persons	2, 4, more	-> 2,4,5
lug_boot	small, med, big	-> 0,1,2
safety	low, med, high	-> 0,1,2

Classes:

unacc	-> 0
acc	-> 1
good	-> 2
v-good	-> 3

Modified Data

- We produce a copy of the main data file by the name "car-eval-data-1.csv" and use sorting and value modifications in Excel to prepare and convert the data. You may write a small program to do that for you.

	A	B	C	D	E	F	G	H
1	2	2	2	2	2	2	0	
2	2	2	2	2	2	0	0	
3	2	2	2	2	2	1	0	
4	2	2	2	2	1	2	0	
5	2	2	2	2	1	0	0	
6	2	2	2	2	1	1	0	
7	2	2	2	2	0	2	0	
8	2	2	2	2	0	0	0	
9	2	2	2	2	0	1	0	
10	2	2	2	4	2	2	1	
11	2	2	2	4	2	0	0	
12	2	2	2	4	2	1	1	
13	2	2	2	4	1	2	1	
14	2	2	2	4	1	0	0	
15	2	2	2	4	1	1	0	
16	2	2	2	4	0	2	1	
17	2	2	2	4	0	0	0	
18	2	2	2	4	0	1	0	

Separating Inputs and Outputs

- The file we have created is usable in most software (specially if you develop yours using Python, C or Java). But for use in Matlab Neural Network wizards it is easier to use separate input and out put files. So we duplicate the file we produced in previous step into two files namely: "car-eval-inputs.csv" and "car-eval-out.csv", containing inputs and outputs respectively.

car-eval-inputs.csv

	A	B	C	D	E	F	G
1	2	2	2	2	2	2	
2	2	2	2	2	2	0	
3	2	2	2	2	2	1	
4	2	2	2	2	1	2	
5	2	2	2	2	1	0	
6	2	2	2	2	1	1	
7	2	2	2	2	0	2	
8	2	2	2	2	0	0	
9	2	2	2	2	0	1	
10	2	2	2	4	2	2	
11	2	2	2	4	2	0	
12	2	2	2	4	2	1	

car-eval-out.csv

	A	B	C
1	0		
2	0		
3	0		
4	0		
5	0		
6	0		
7	0		
8	0		
9	0		
10	1		
11	0		
12	1		

Converting output classes to Neuron Values

- One more conversion is needed for the output file. We need to represent each of the classes with neuron outputs. For this purpose we show class 0 with [0 0 0 1] , class 1 with [0 0 1 0], class 2 with [0 1 0 0] and class 3 with [1 0 0 0]. We first create a copy of output file and name it "car-eval-outputs.csv" and replace the classes with the neuron output vectors:

car-eval-out.csv

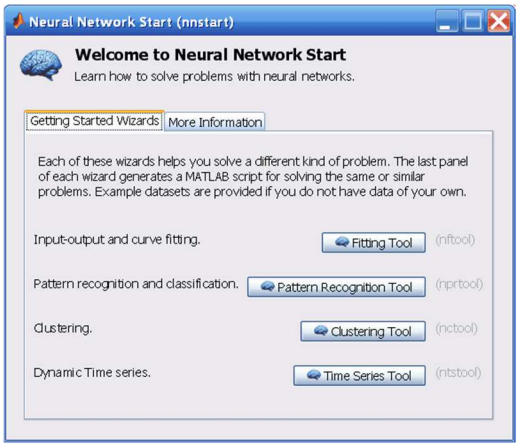
	A	B	C
1	0		
2	0		
3	0		
4	0		
5	0		
6	0		
7	0		
8	0		
9	0		
10	1		
11	0		
12	1		

car-eval-outputs.csv

	A	B	C	D
1	0	0	0	1
2	0	0	0	1
3	0	0	0	1
4	0	0	0	1
5	0	0	0	1
6	0	0	0	1
7	0	0	0	1
8	0	0	0	1
9	0	0	0	1
10	0	0	1	0
11	0	0	0	1
12	0	0	1	0

Starting the Neural Network Wizard

- Run Matlab and type nnstart in the command line. Neural network start menu appears. Select pattern recognition tool.



Fitting tool : Finds a network (a function) that fits your data. The Neural Network can then be used to predict the outcome for an input data.

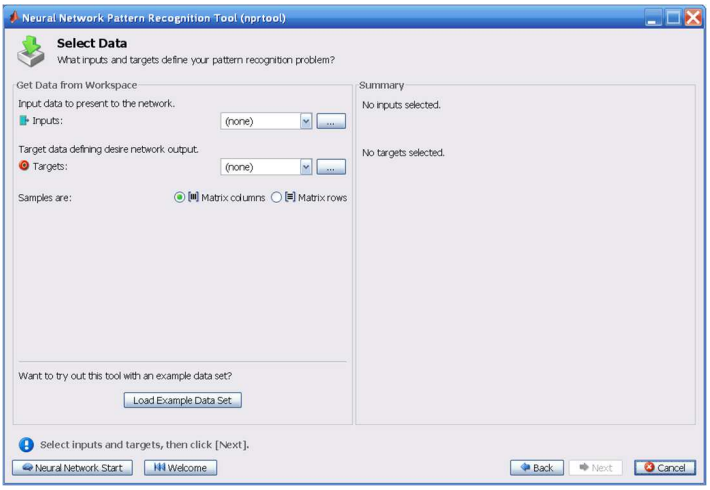
Pattern recognition Tool: Used for building classifiers.

Clustering Tool: Used to cluster data.

Dynamic Time Series Tool: Use past values to predict future values.

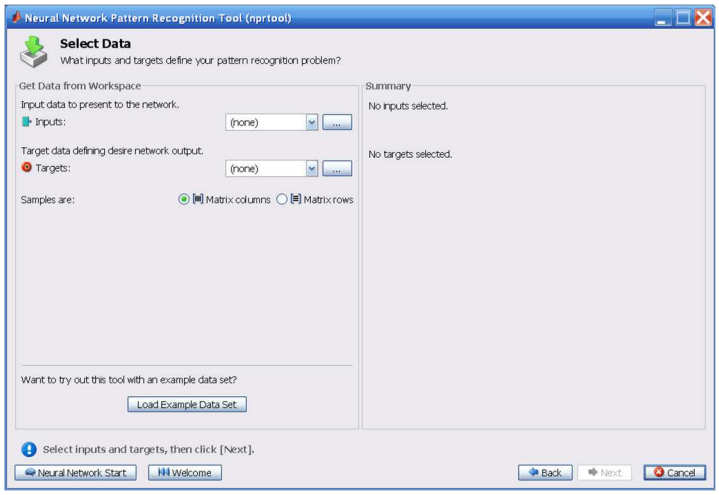
Staring the Pattern Recognition Wizard

- As described in the wizard's main page, the tool will build a 3 layer network (Input layer, hidden layer, output layer) for the purpose of classifying your data.



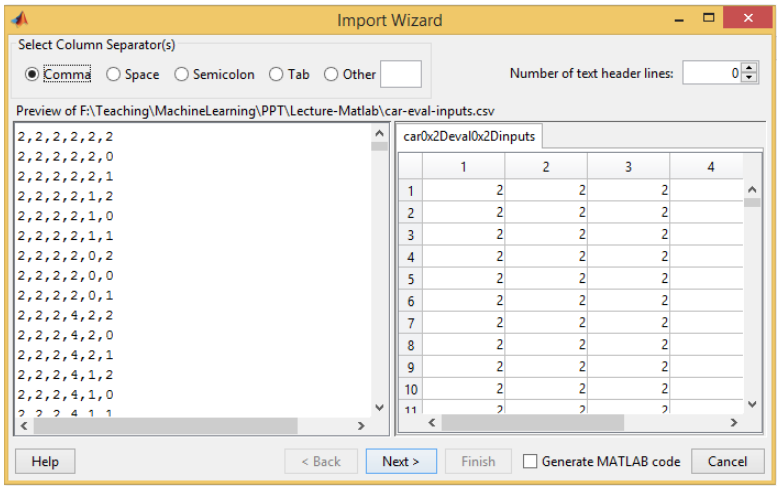
Loading Data

- In data selection tab, we can either select data arrays already loaded (or created) in Matlab. Or we can press "…" buttons to load data from files.



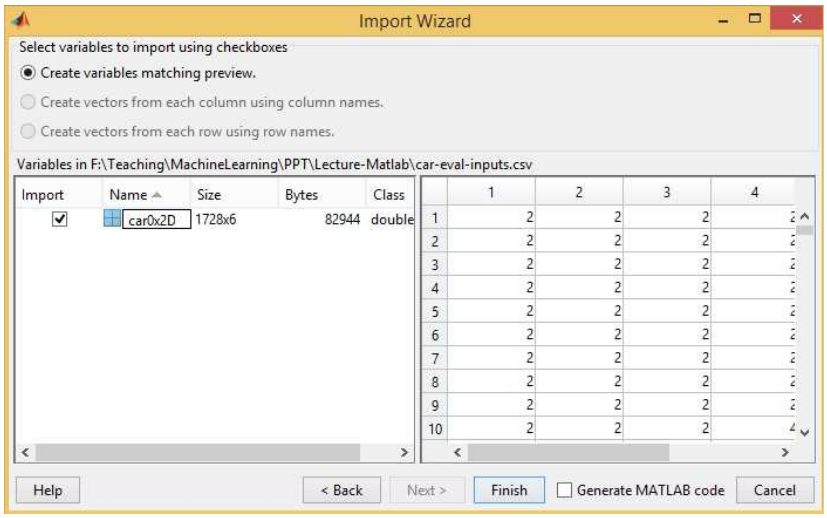
Loading Input Data

- Press the "…" button in front of inputs and load the "car-eval-inputs.csv" file (i.e. input feature values). After selection is made, the following window will appear. Press next…



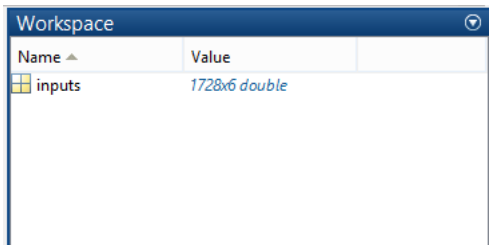
Renaming Input Data Array

- Matlab shows a preview of the array that is going to be imported. Right click on the Array name and rename it to "inputs"… then press Finish.



Inputs Array

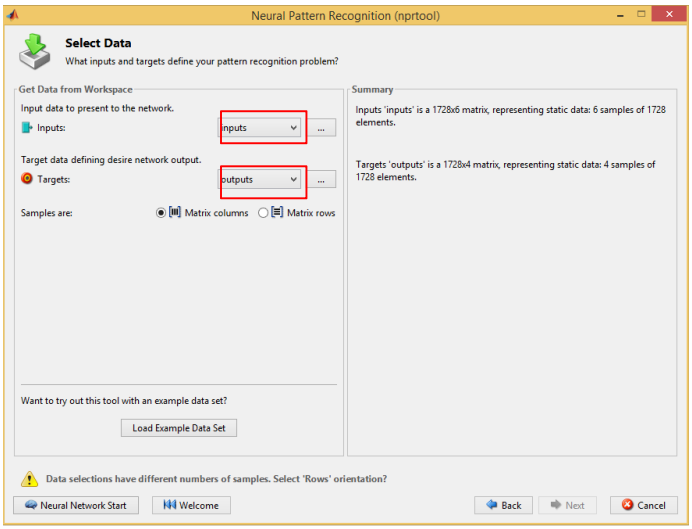
- If you check the main window of Matlab, in workspace section you will see that a 1728*6 array called "inputs" has been created.



- Now we should load the output array.

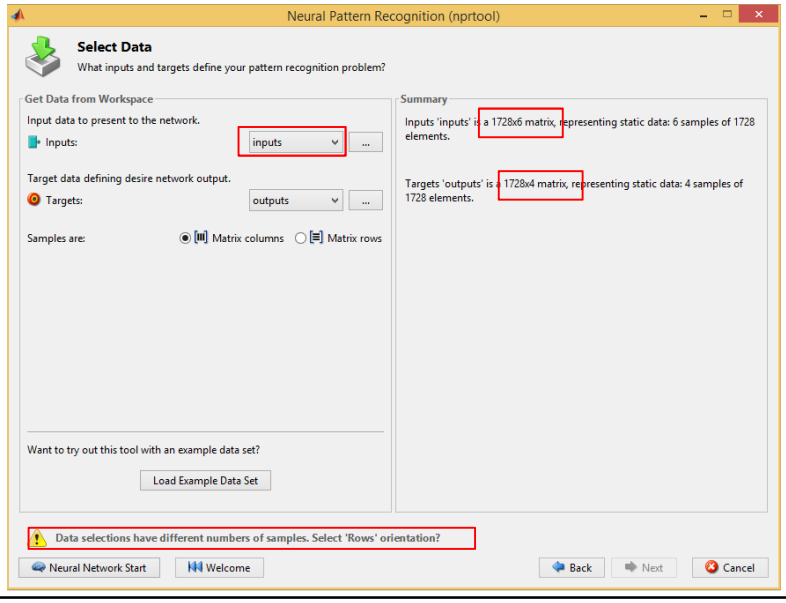
Importing Output Data

- Now press the "…" sign in front of Input data and load the 4 column output file we produced namely "car-eval-outputs.csv". After pressing next, rename the imported array as "outputs".
- Now you have two arrays in the memory of Matlab (workspace).
- Notice that if the format of output values file is not correct (should be only 1 and 0s), the array which is imported will not be usable. You will not see the name of array in front of Inputs field.



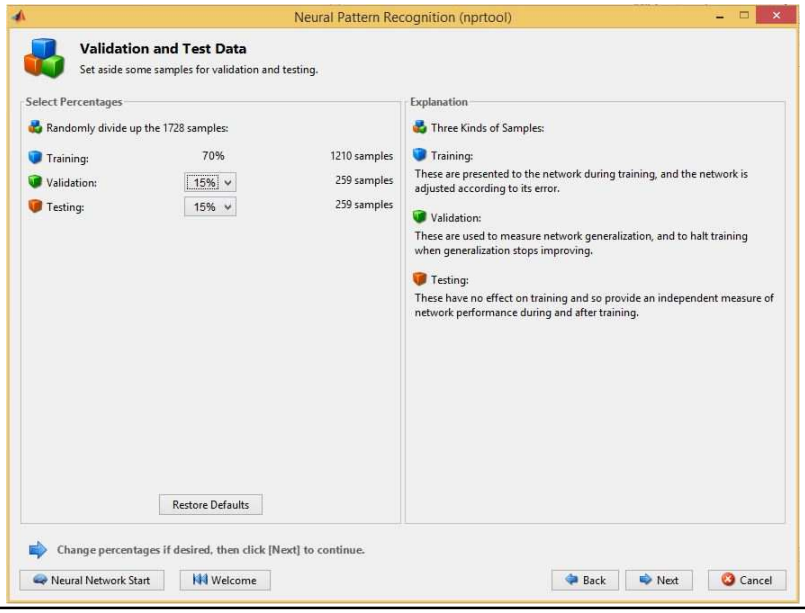
Adjusting Array Dimensions

- Matlab is now showing the dimensions of the both arrays, and also complains that they do not match!
- To fix the problem, in samples section select "Matrix Rows"…



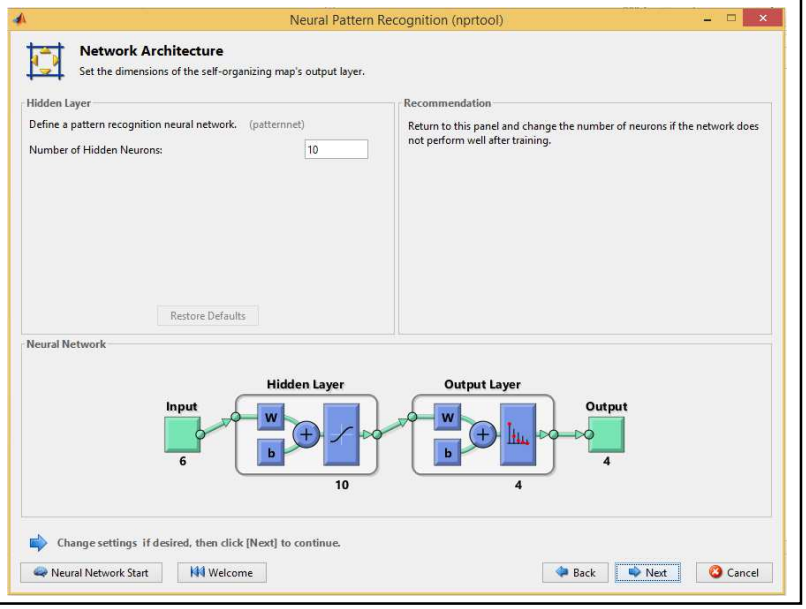
Selecting Training, Validation and Testing size

- In the next page you can select how much of the data you want to use for Training, Validation and Test datasets. Keep the suggested percentages and click Next.



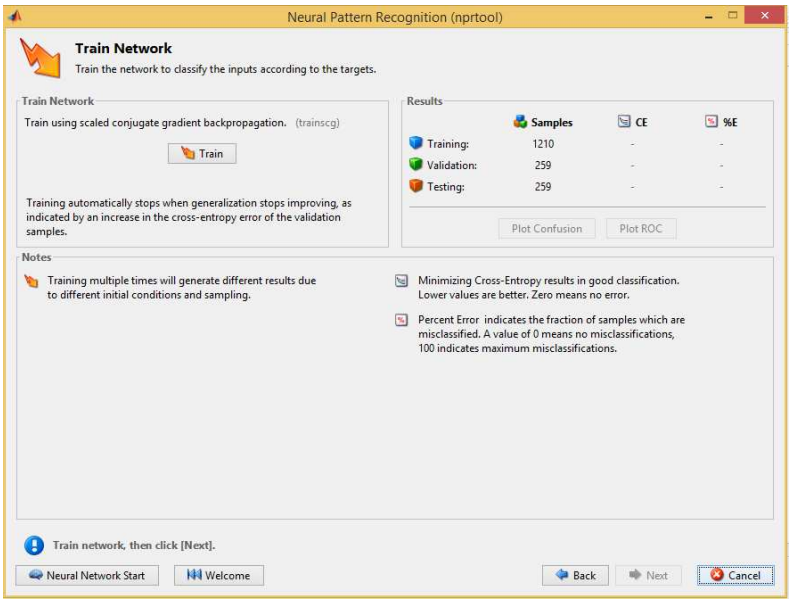
Network Architecture

- In this step you can select the number of hidden neurons. Your network will have 6 inputs (since we have 6 input value or features) and 4 outputs (because we have 4 classes and neuron values). You can keep the value but most of the time (not always) a number between the number of input neurons and output neurons produces good enough results.



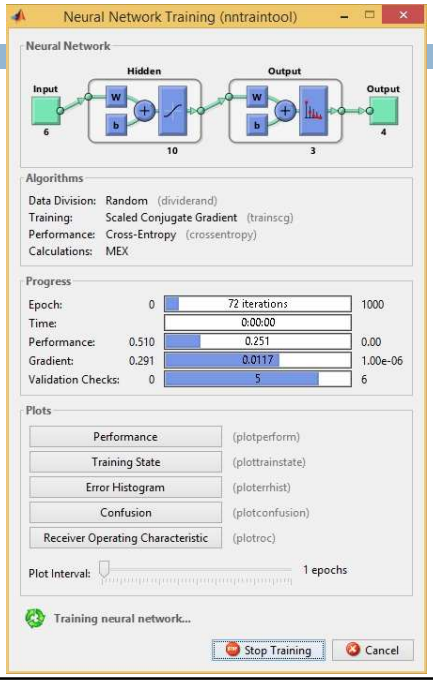
Training

- Now you see the training tab. Press the "Train" button to start training.



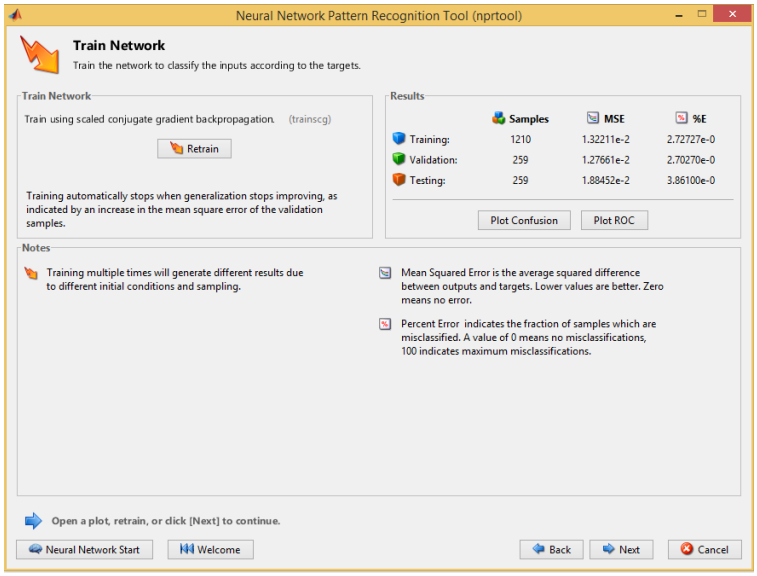
Training

- A training window will appear (and disappear after finishing the training phase).
- This will calculate the weights for the neurons' inputs.



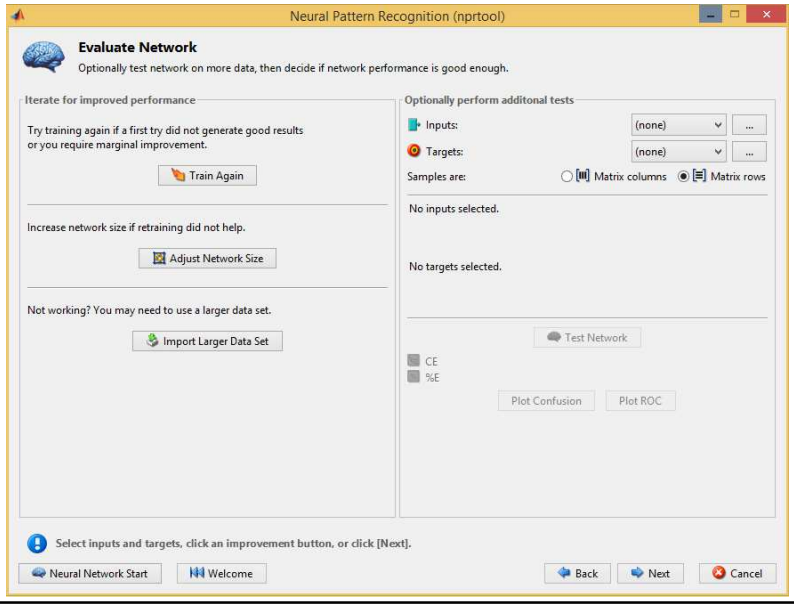
Results

- The results will appear in the window.
- The error for the training data is 2.72 %.
- The error for the test data is 3.86 %.
- This means a performance (correct classification rate) of 96.14% for the test data.
- Since the training is done based on random initial settings and random numbers, you may repeat the training until you are happy with the results.
- Press next



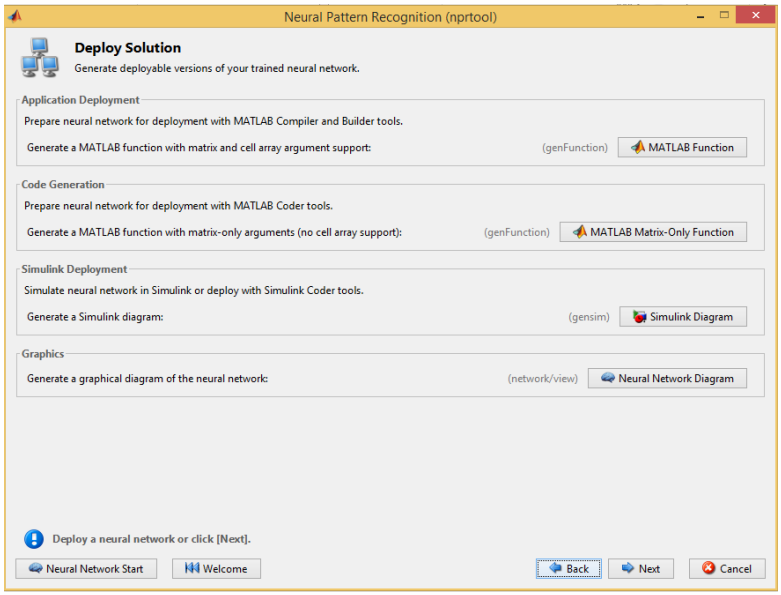
Final Adjustments

- In the next window, you get the last chance to modify or check your network.



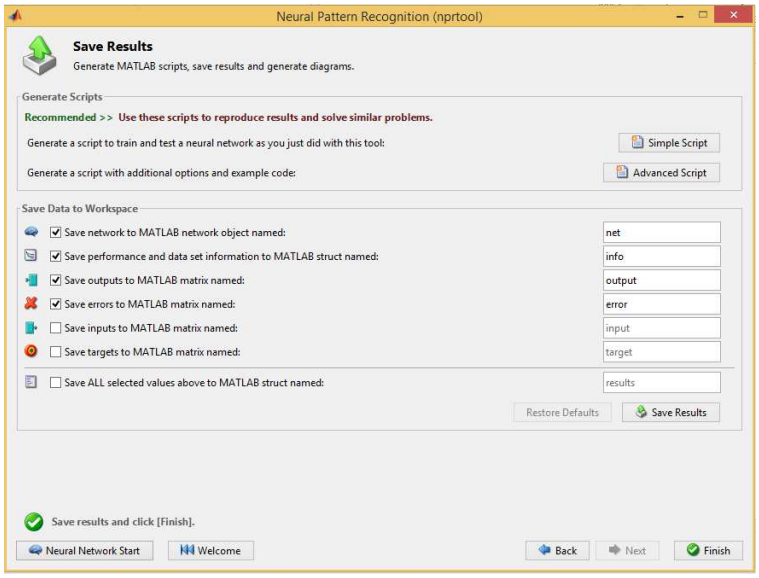
Exporting the Network

- In this step you can export the network to a function (first selection). If you press this button Matlab will build a function. The function has all the weights set in it, ready for use in your Matlab programs..
- For now, just press Next.



Saving the results

- In this page you can generate a script which will build the Matlab code which is equal to doing all these steps in command line.
- Pressing "save results" will save the network, errors and other data in Arrays with the shown names.
- Press Save Results …



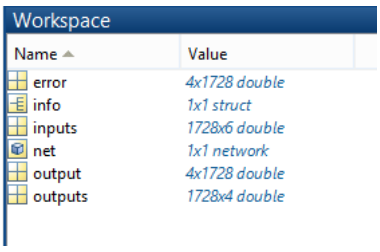
Adjusting Array Dimensions

- The following code shows the result of pressing "Simple Script" button:

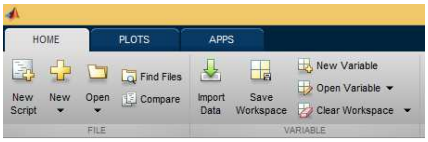
```
x = inputs';
t = outputs';

hiddenLayerSize = 10;
net = patternnet(hiddenLayerSize);

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

[net,tr] = train(net,x,t);

y = net(x);
e = gsubtract(t,y);
tind = vec2ind(t);
yind = vec2ind(y);
percentErrors = sum(tind ~= yind)/numel(tind);
performance = perform(net,t,y)
view(net)
```

Adjusting Array Dimensions

- Now look at the main window of Matlab (in the workspace). The network and other data has been saved in different variables (Arrays, etc.)

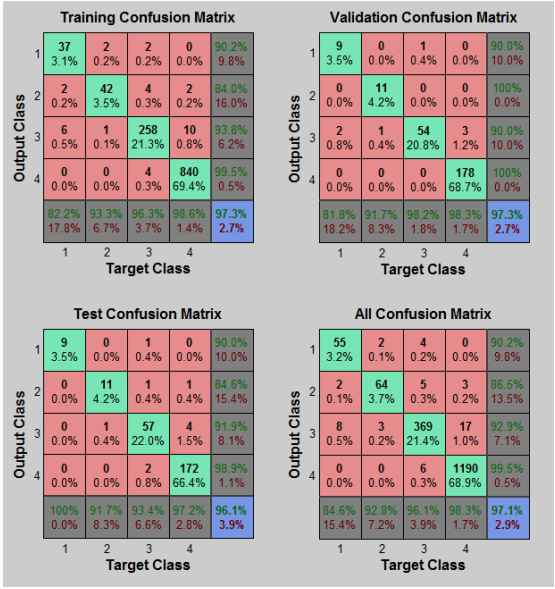


- If you like to keep all these variables, use Save Workspace to save all the variables that are in the memory:



Confusion Matrix

- One of the useful graphs that Matlab provides, is the confusion matrixes…
- Using this graph you can understand which classes produce more error and which ones are classified better in each of data sets.



Using the Network for Single Input vectors

- Now that your network is ready, you can use it to predict the class for any arbitrary new inputs (i.e. vector of 6 values). The input vector is 2,2,2,4,1,0 and the supposed output is 0.
- We first create a variable containing the input vector (notice that it is a column vector):
  a=[2;2;2;4;1;0]
- Now we give it to the network and get the output:
  y = net(a)
- The output shows the values of 4 output neurons:

  y =
  0.0000
  0.0000
  0.0000
  1.0000
- The above vector is equal to the row vector [0, 0, 0, 1] which selects the class 0 because the last element is higher (has received higher votes from the neurons) .

Using the Network for Single Input vectors

- Note that the outputs might not always be 1.0 and 0.0. For example another input vector produces the following outputs:

```
a=[2;2;2;2;2;2]
y = net(a)
```

- The output shows the values of 4 output neurons:

```
y =
0.0001
0.0001
0.0012
0.9963 -> highest value represents 1 , all others represent 0
```

- The above vector is again equal to the row vector [0, 0, 0, 1] which selects the class 0 (matches the data properly).

Using Sickit Neural Network

Using Sickit Neural Network

- **Note:** the library is already installed on the python I gave you… do not perform this step
- First we need to install Sickit Neural Network (may need disabling firewall)

  pip install scikit-neuralnetwork
- Now we can use the library to solve the classification problem using the library.

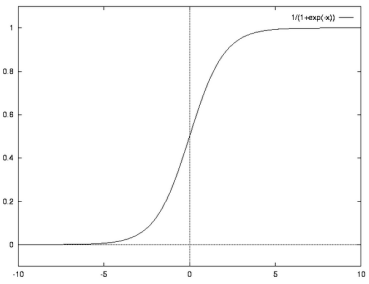
Using SkNN – Program 1 (Data from Array)

```
import numpy as np

features_train = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1],[3,3]])
labels_train   = np.array([1, 1, 1, 2, 2, 2])
features_test  = np.array([[-2, -2], [-2, -3], [2, 3], [1, 2]])
labels_test    = np.array([1, 1, 1, 2])

from sknn.mlp import Classifier, Layer

nn = Classifier(
    layers=[
        Layer("Sigmoid", units=100),
        Layer("Softmax")],
    learning_rate=0.001,
    n_iter=25)

nn.fit(features_train, labels_train)
pred = nn.predict(features_test)

print "Test labels:      ", labels_test
print "Predicted labels: ", pred

from sklearn.metrics import accuracy_score
print "Accuracy:         ", accuracy_score(pred, labels_test)

print "\nPredicted label for ", [-0.8, -1] ," is ", (nn.predict([[-0.8, -1]]))
```

Using SkNN – Program 1 (Data from Array)

- For hidden layers you normally use "Sigmoid" and "Tanh" (sometimes "Rectifier" or "ExpLin").
- For output units, You may typically use "Linear" or "Softmax".
- For classifiers we normally use Sigmoid for hidden layers and Softmax for output layer.

Softmax:



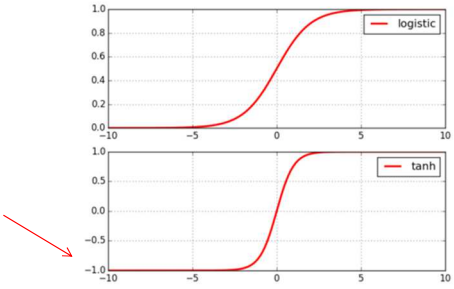
Sigmoid vs. Softmax:
All the Softmax units in a layer are constrained to add up to 1, whereas sigmoid units don't have this 'lateral' constraint. For example, in a sigmoid layer, all units could have the value 0.99 (though this is pretty unlikely), but this can't happen in a Softmax layer.

$$softmax_i(a) = \frac{\exp a_i}{\sum \exp a_i} \qquad \sigma(x) = \frac{1}{1+e^{-x}}$$

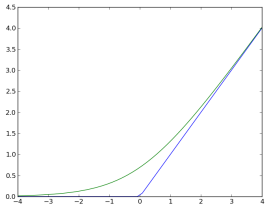
- For regression problems, use Rectifier for hidden and Linear for output.

Using SkNN – Program 1 (Data from Array)

- "Sigmoid" vs. "Tanh".



- ReLU: f(x)= max(0,x)



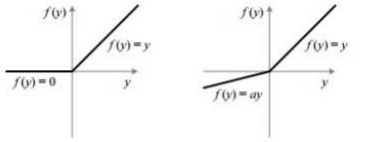


Figure 1. ReLU vs. PReLU. For PReLU, the coefficient of the negative part is not constant and is adaptively learned.