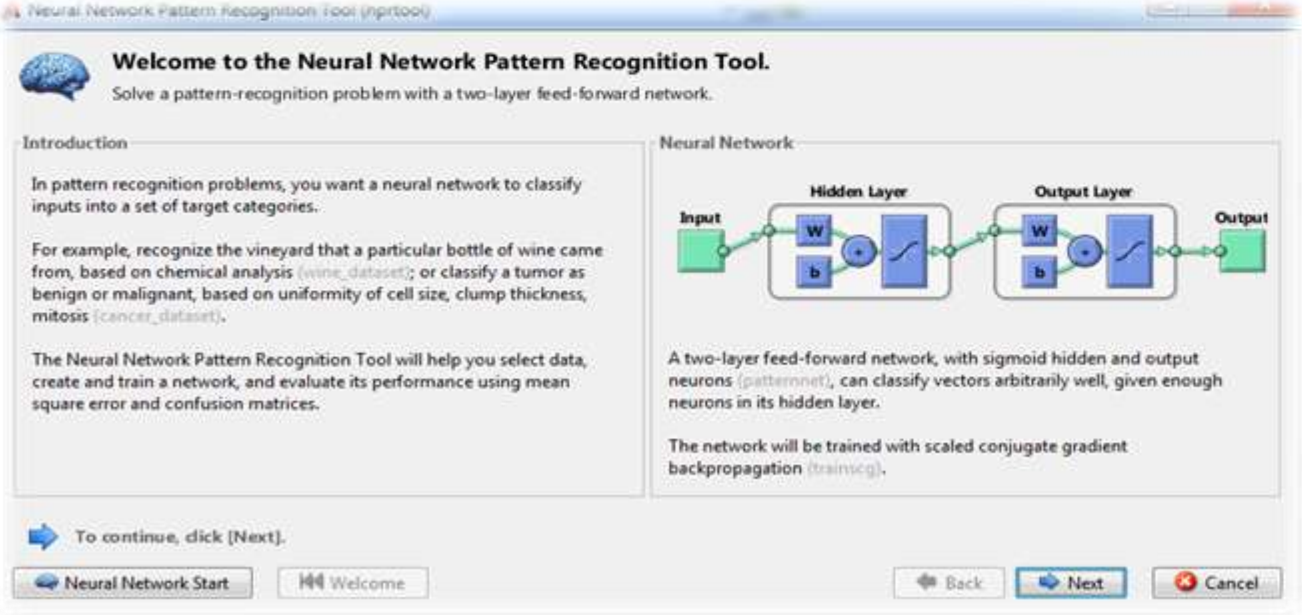
IN THE NAME OF ALLAH

Neural Networks

Neural Networks Using MATLAB



Neural Network Pattern Recognition Tool (nprtool)

Welcome to the Neural Network Pattern Recognition Tool.

Solve a pattern-recognition problem with a two-layer feed-forward network.

Introduction

In pattern recognition problems, you want a neural network to classify inputs into a set of target categories.

For example, recognize the vineyard that a particular bottle of wine came from, based on chemical analysis (wine_dataset); or classify a tumor as benign or malignant, based on uniformity of cell size, clump thickness, mitosis (cancer_dataset).

The Neural Network Pattern Recognition Tool will help you select data, create and train a network, and evaluate its performance using mean square error and confusion matrices.

Neural Network

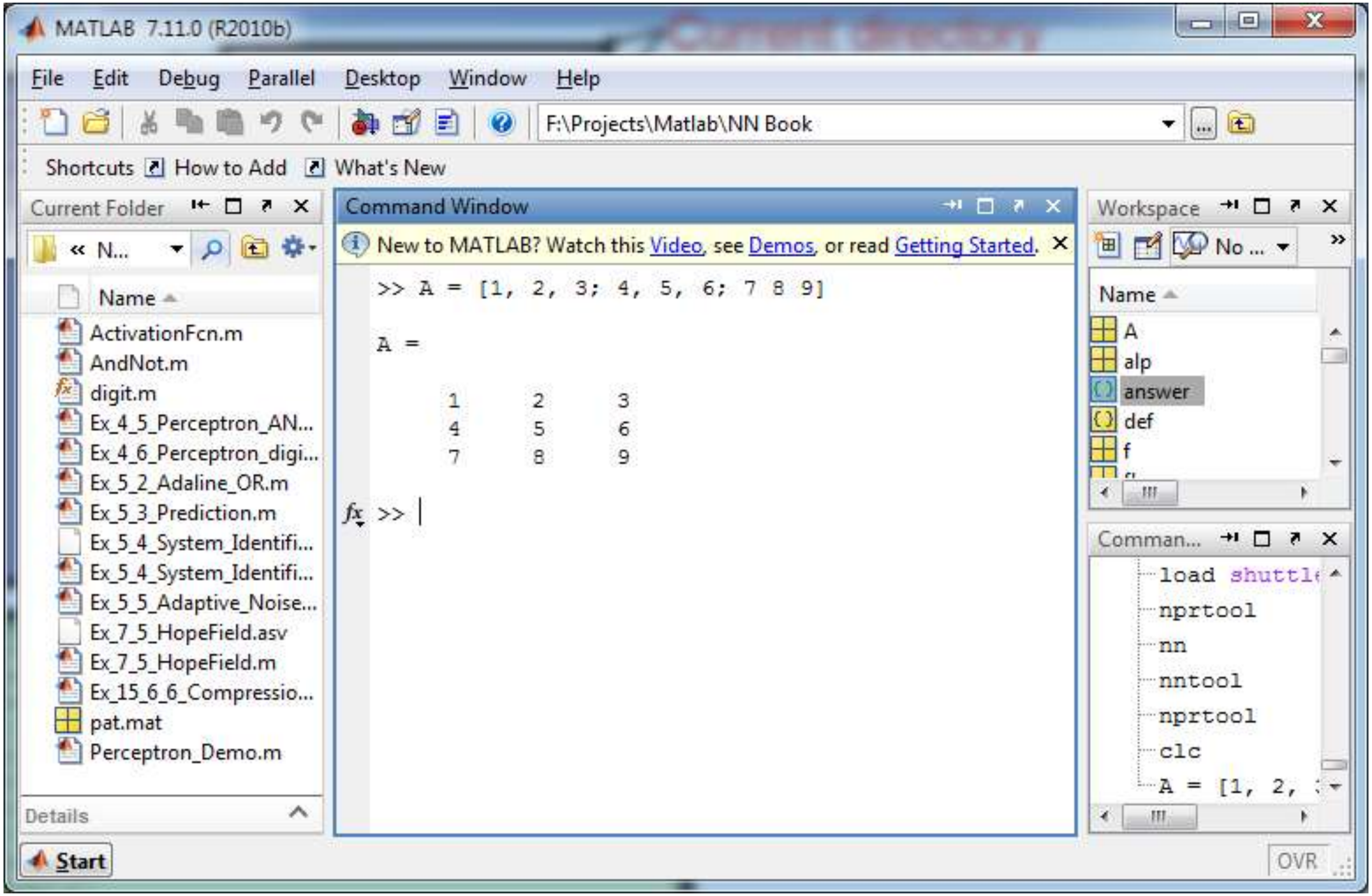
A two-layer feed-forward network, with sigmoid hidden and output neurons (patternnet), can classify vectors arbitrarily well, given enough neurons in its hidden layer.

The network will be trained with scaled conjugate gradient backpropagation (trainscg).

To continue, click [Next].

Shahrood University of Technology
Hossein Khosravi

MATLAB Environment



New to MATLAB? Watch this Video, see Demos, or read Getting Started.

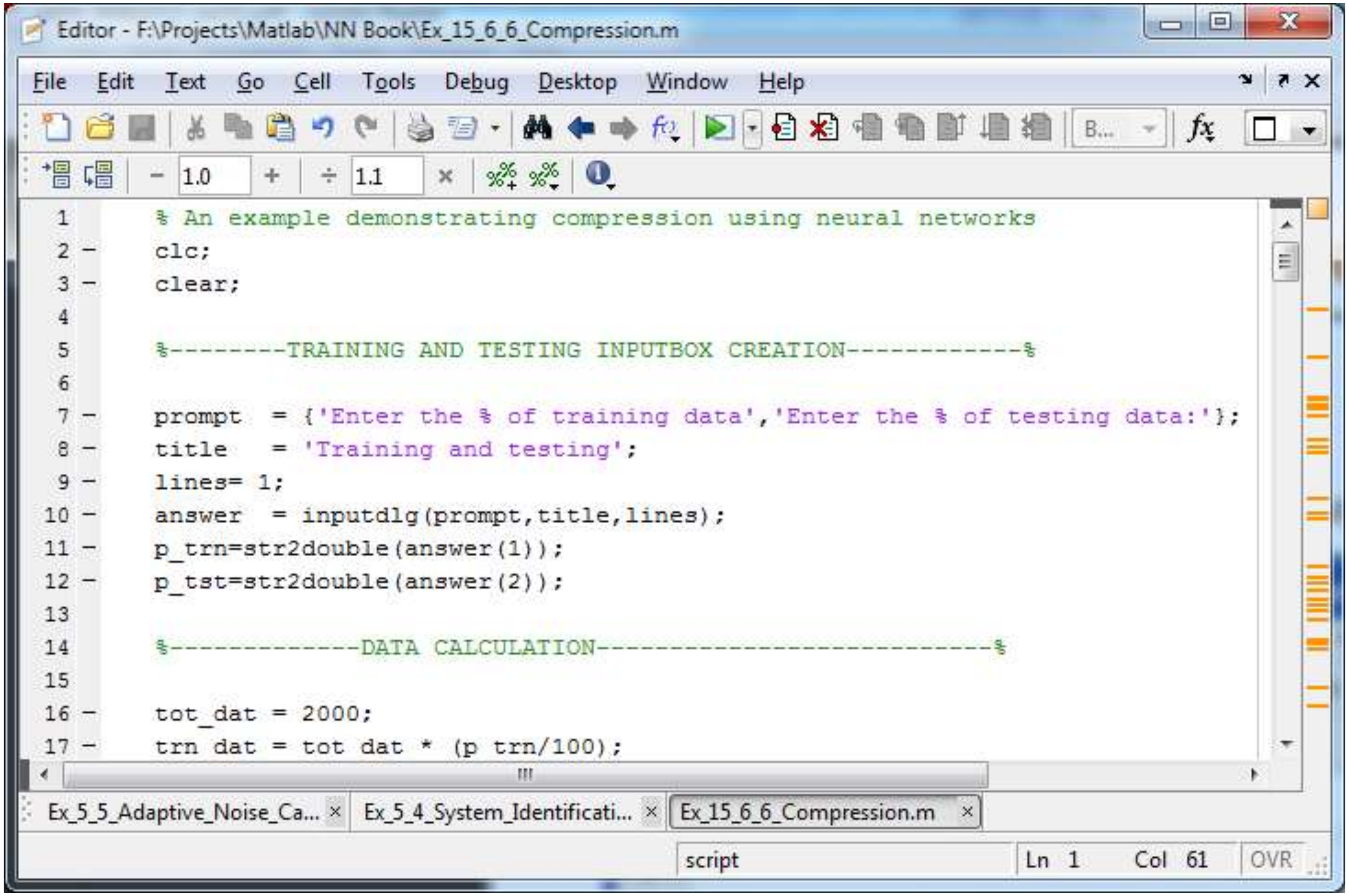
```
>> A = [1, 2, 3; 4, 5, 6; 7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9

>>
```

```
load shuttl
nprtool
nn
nntool
nprtool
clc
A = [1, 2, 
```

MATLAB Help

- help sin
  - Inline help
  - Concise
- doc sin
  - Opens help browser
  - Comprehensive
  - Several Examples

Writing scripts using editor


Editor - F:\Projects\Matlab\NN Book\Ex_15_6_6_Compression.m
File Edit Text Go Cell Tools Debug Desktop Window Help
B...
1.0
1.1
Ex_5_5_Adaptive_Noise_Ca...
Ex_5_4_System_Identificati...
Ex_15_6_6_Compression.m
script
Ln 1
Col 61
OVR

```
% An example demonstrating compression using neural networks
clc;
clear;

%--------TRAINING AND TESTING INPUTBOX CREATION------------%

prompt  = {'Enter the % of training data','Enter the % of testing data:'};
title   = 'Training and testing';
lines= 1;
answer  = inputdlg(prompt,title,lines);
p_trn=str2double(answer(1));
p_tst=str2double(answer(2));

%-------------DATA CALCULATION--------------------------%

tot_dat = 2000;
trn_dat = tot_dat * (p_trn/100);
```

Remarks

- **COMMENT!**
  - Anything following a **%** is seen as a comment
  - The first contiguous comment becomes the script's help file
  - Comment thoroughly to avoid wasting time later

- Note that scripts are somewhat static, since there is no input and no explicit output

- All variables created and modified in a script exist in the workspace even after it has stopped running

Variables

- MATLAB is a weakly typed language
  - No need to initialize variables!

- MATLAB supports various types; The most often used are
  - **» pi_num = 3.14**
    - 64-bit double (default)
  - **» a_char = ‘a’**
    - 16-bit char

- Most variables you’ll deal with will be **vectors** or **matrices** of doubles or chars

- Other types are also supported: complex, symbolic, 16-bit and 8 bit integers, etc.

Matrices: The most common type

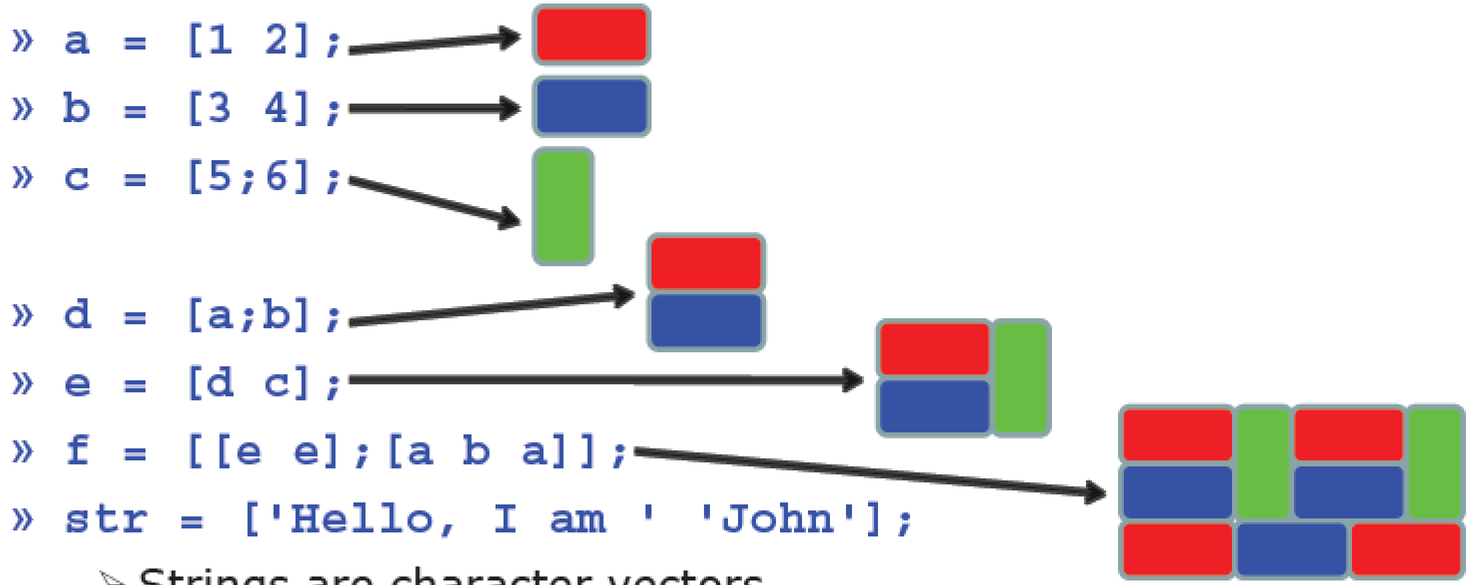
- Make matrices like vectors

- Element by element

```
» a= [1 2;3 4];
```

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

- By concatenating vectors or matrices (dimension matters)



```
» a = [1 2];
» b = [3 4];
» c = [5;6];

» d = [a;b];
» e = [d c];
» f = [[e e];[a b a]];
» str = ['Hello, I am ' 'John'];
```

➢ Strings are character vectors

Built in functions

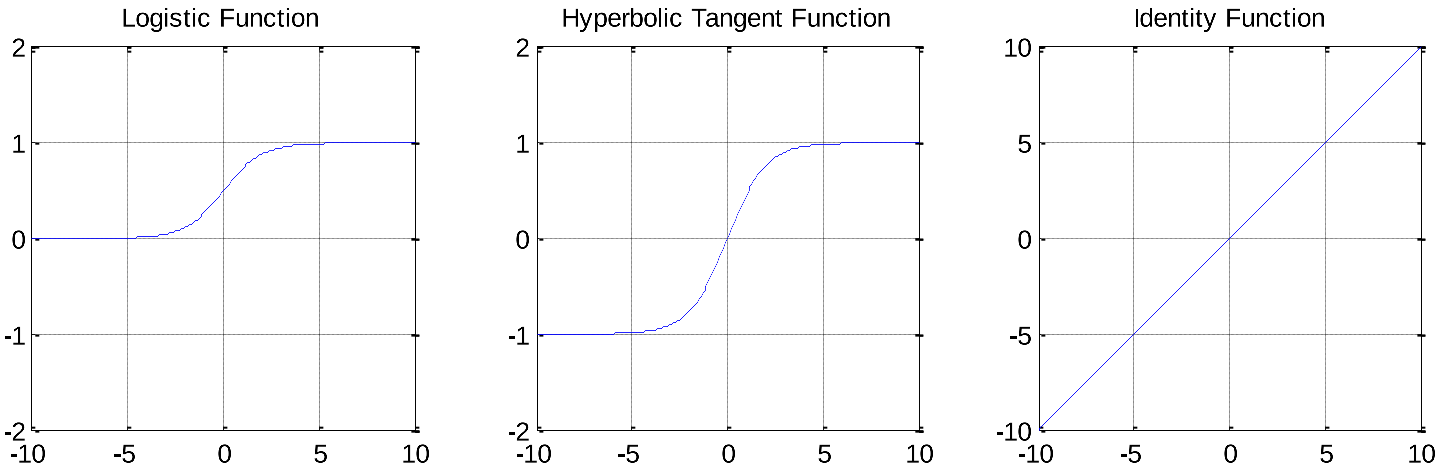
- MATLAB has an **enormous** library of built-in functions
- Almost any function you think, is available.

- Call using parentheses –passing parameter to function:
  - »`sqrt(2)`
  - »`log(2), log10(0.23)`
  - »`cos(1.2), atan(-.8)`
  - »`exp(2+4*i)`
  - »`round(1.4), floor(3.3), ceil(4.23)`
  - »`angle(1+2i); abs(1+2i);`

Example: Activation functions used in NN's

```
% Illustration of various activation functions used in NN's
x = -10:0.1:10;
tmp = exp(-x);
y1 = 1./(1+tmp);
y2 = (1-tmp)./(1+tmp);
y3 = x;
subplot(131); plot(x, y1); grid on;
title('Logistic Function');
subplot(132); plot(x, y2); grid on;
title('Hyperbolic Tangent Function');
subplot(133); plot(x, y3); grid on;
title('Identity Function');
```

Output


Logistic Function
2
1
0
-1
-2
-10
-5
0
5
10
Hyperbolic Tangent Function
2
1
0
-1
-2
-10
-5
0
5
10
Identity Function
10
5
0
-5
-10
-10
-5
0
5
10

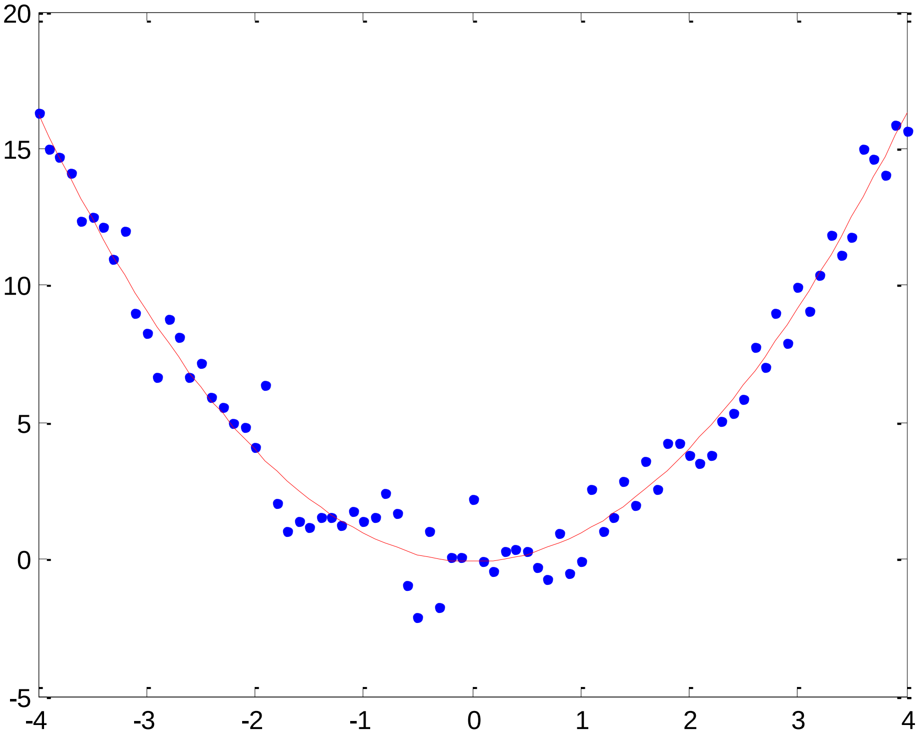
(a)

(b)

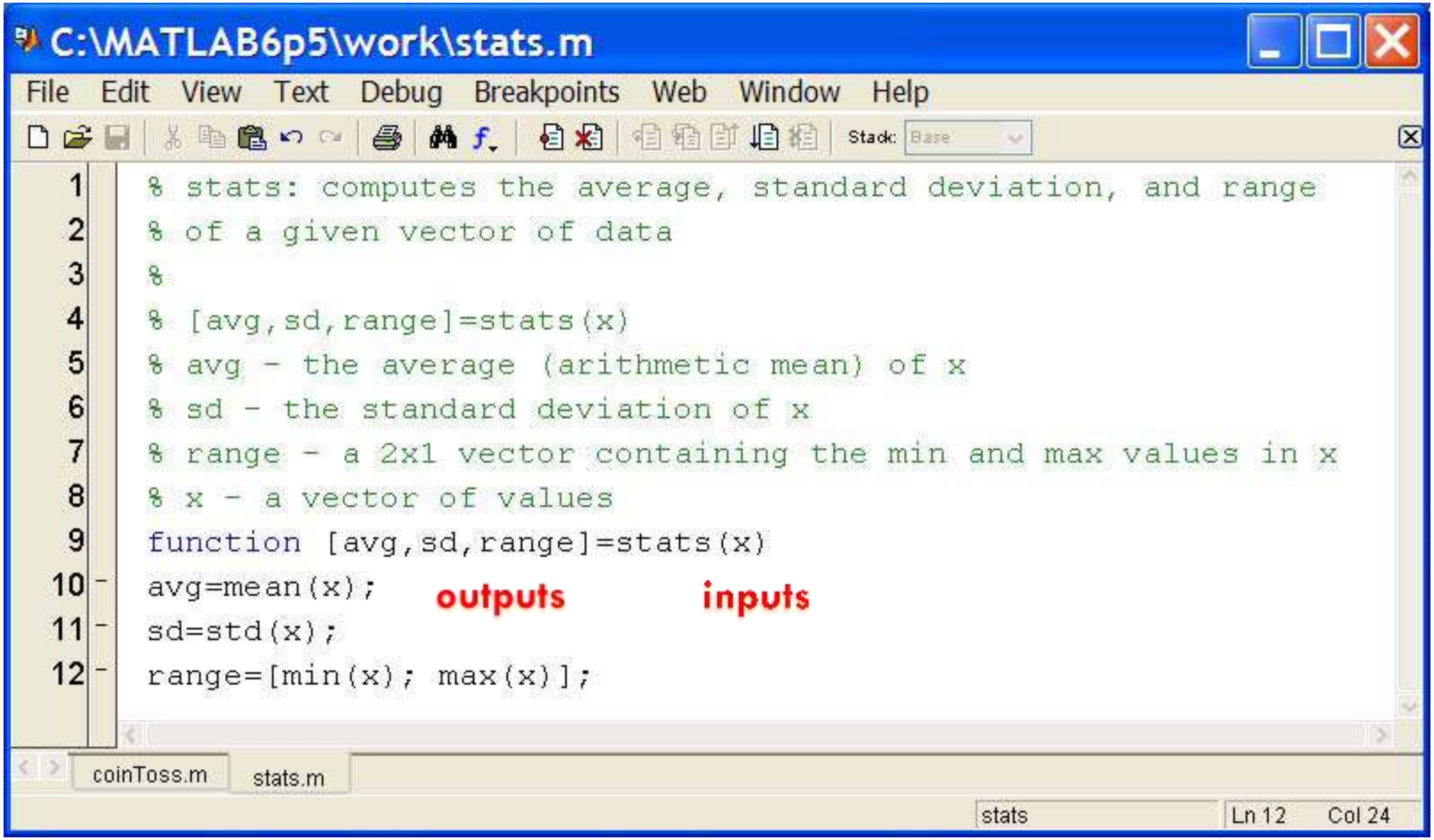
(c)

Example: Curve fitting

```
% Polynomial fit
x=-4:0.1:4;
y=x.^2;
y=y+randn(size(y));
plot(x,y,'.');
p = polyfit(x,y,2)
hold on;
plot(x,polyval(p,x),'r');
```



User functions



C:\MATLAB6p5\work\stats.m
File Edit View Text Debug Breakpoints Web Window Help
Stack: Base
coinToss.m
stats.m
stats
Ln 12
Col 24

```
% stats: computes the average, standard deviation, and range
% of a given vector of data
%
% [avg,sd,range]=stats(x)
% avg - the average (arithmetic mean) of x
% sd - the standard deviation of x
% range - a 2x1 vector containing the min and max values in x
% x - a vector of values
function [avg,sd,range]=stats(x)
avg=mean(x);
sd=std(x);
range=[min(x); max(x)];
```

outputs

inputs

User functions

- No need for return:
  - MATLAB 'returns' the variables whose names match those in the function declaration
- Variable scope:
  - Any variables created within the function but not returned disappear after the function stops running function

Relational Operators

- MATLAB uses mostly standard relational operators

Equal	==
Not equal	~=
greater than	>
less than	<
greater or equal	>=
less or equal	<=

Logical operators	element-wise	scalars
And	&	&&
Or	|	||
Not	~	
Xor	xor	
All true	all	
Any true	any	

- Boolean values: zero is false, nonzero is true
- See help . for a detailed list of operators

Code Efficiently

- Given x= sin(linspace(0,10*pi,100)), how many of the entries are positive?

Using a loop and if/else

```
count=0;
for n=1:length(x)
    if x(n)>0
        count=count+1;
    end
end
```

Being more clever

```
count=length(find(x>0));
```

length(x)	Loop time	Find time
100	0.01	0
10,000	0.1	0
100,000	0.22	0
1,000,000	1.5	0.04

- Avoid loops!
- Built-in functions will make it faster to write and execute

Avoiding Loops

- Avoid loops
  - This is referred to as vectorization
- Vectorized code is more efficient for MATLAB
- Use indexing and matrix operations to avoid loops
- For example, to sum up every two consecutive terms:

```
» a=rand(1,100);
» b=zeros(1,100);
» for n=1:100
»     if n==1
»         b(n)=a(n);
»     else
»         b(n)=a(n-1)+a(n);
»     end
» end
```

  - Slow and complicated

```
» a=rand(1,100);
» b=[0 a(1:end-1)]+a;
```

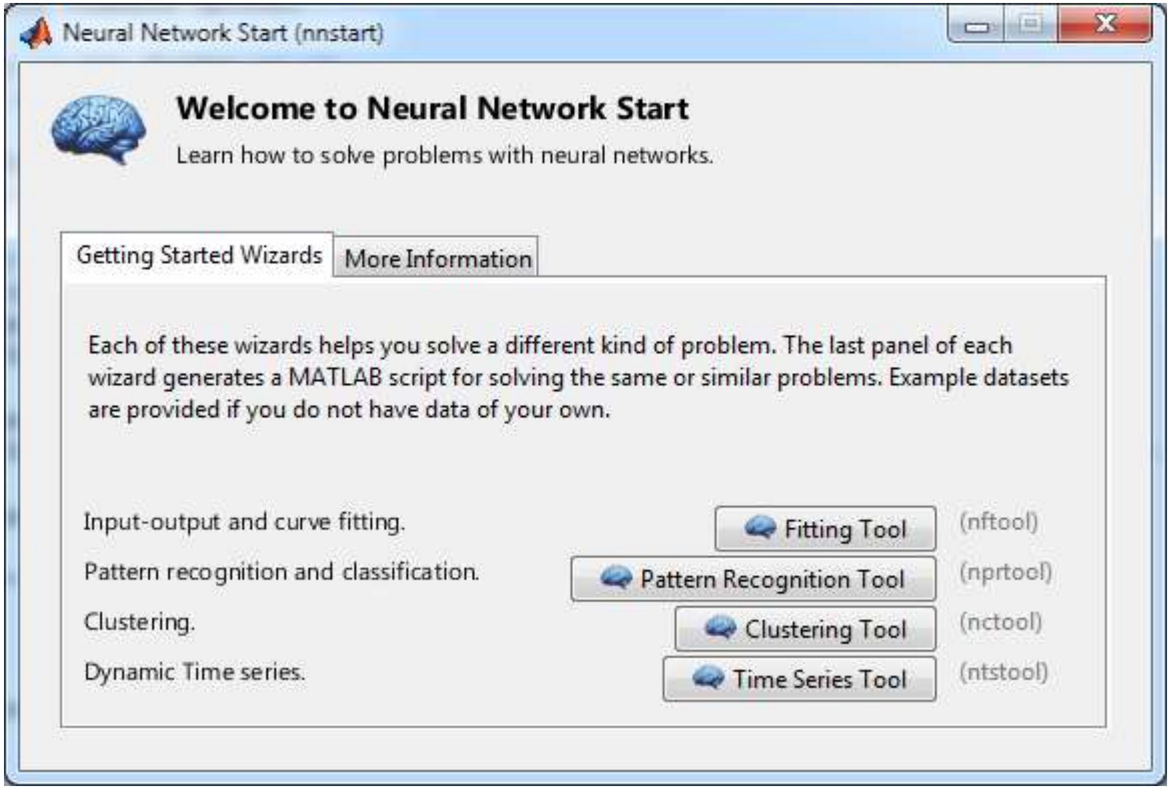
  - Efficient and clean. Can also do this using `conv`

Neural Networks

- nnstart
- nntool
- nftool
- nprtool
- nctool
- ntstool
- newp, newhop, newff, …

Neural Nets Using MATLAB

- nnstart: A good point to start with neural network toolbox



Welcome to Neural Network Start

Learn how to solve problems with neural networks.

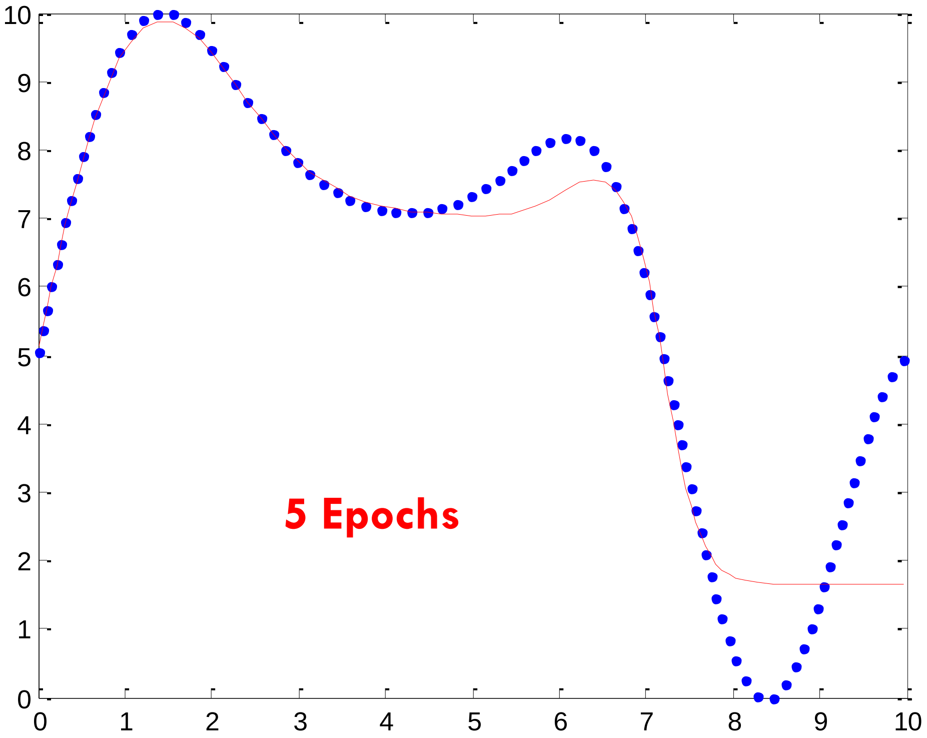
Each of these wizards helps you solve a different kind of problem. The last panel of each wizard generates a MATLAB script for solving the same or similar problems. Example datasets are provided if you do not have data of your own.

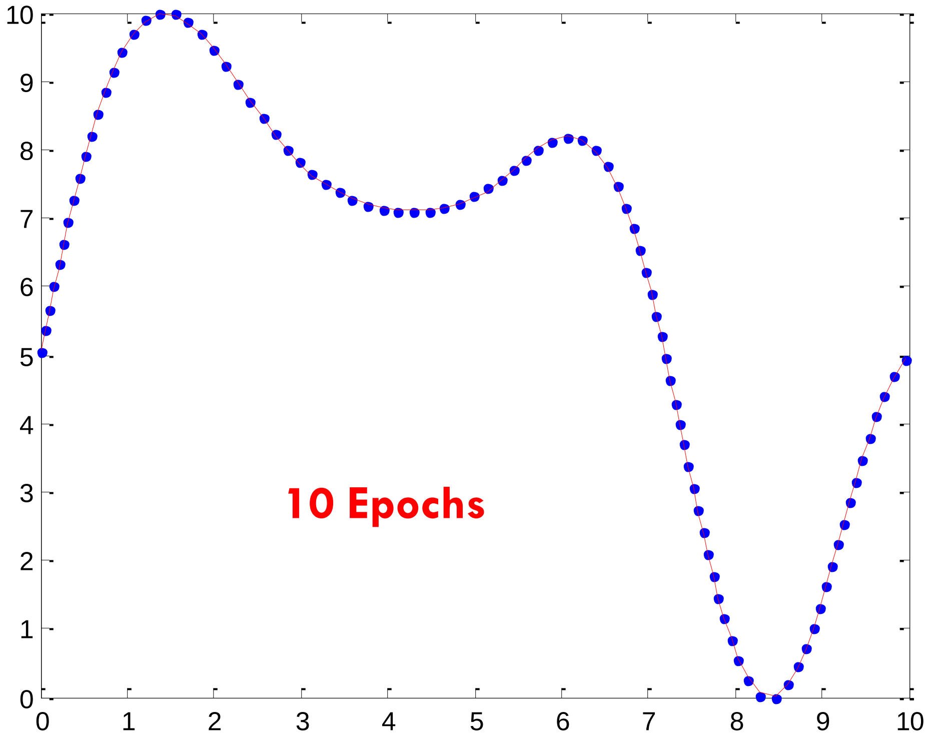
Input-output and curve fitting.	Fitting Tool	(nftool)
Pattern recognition and classification.	Pattern Recognition Tool	(nprtool)
Clustering.	Clustering Tool	(nctool)
Dynamic Time series.	Time Series Tool	(ntstool)

Example: fitnet

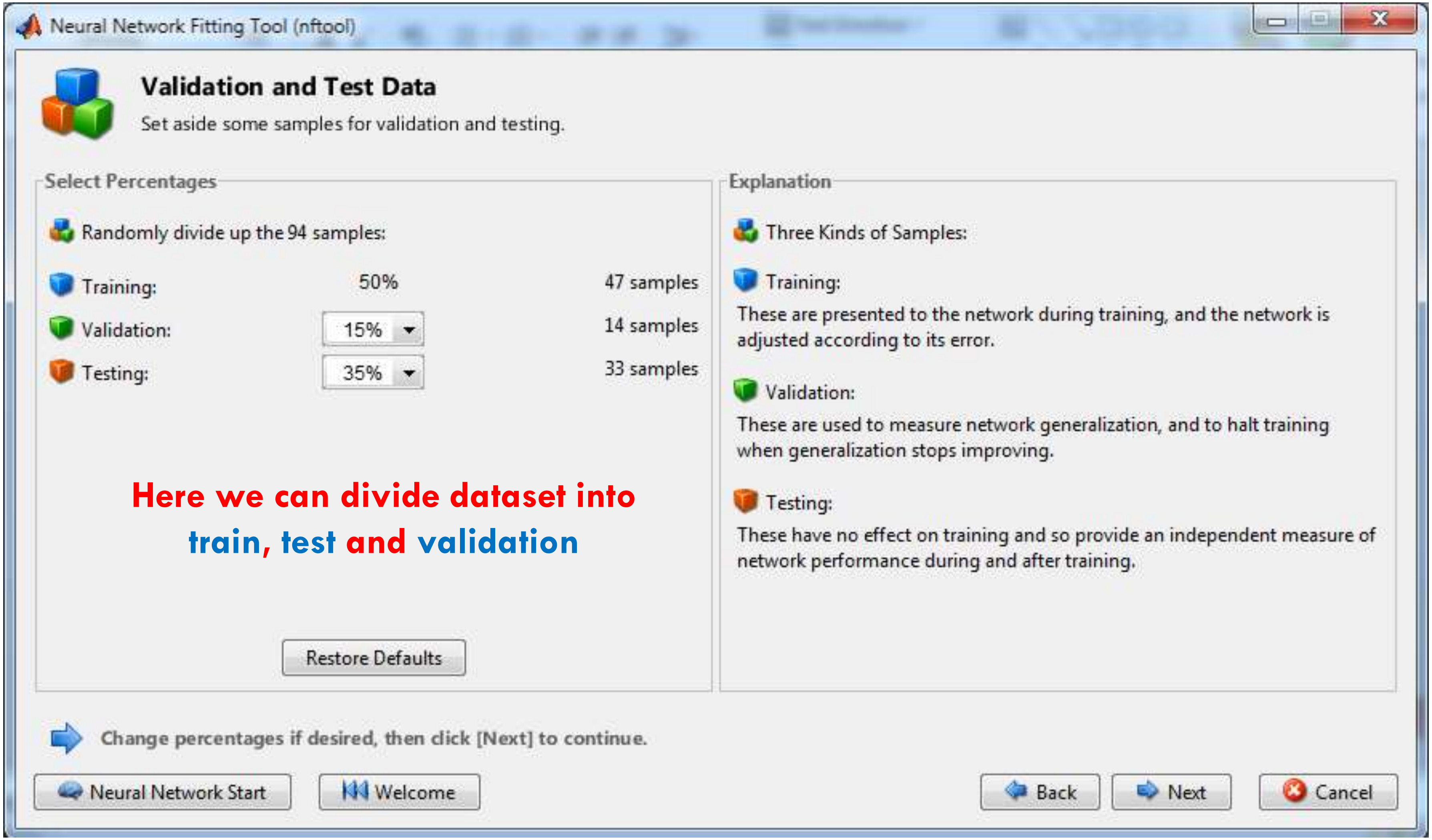
```
%fit net
%[x,t] = simplefit_dataset;
[x t] = simplefit_create;
%subplot(211)
plot(x,t,'.');
net = fitnet(5); % I told everything you think may be found
net.trainParam.epochs = 5;
net = train(net,x,t);
%view(net);
y = net(x);
%subplot(212)
hold on
plot(x,y,'r');
```

Result


5 Epochs
0
1
2
3
4
5
6
7
8
9
10
0
1
2
3
4
5
6
7
8
9
10


10 Epochs
0
1
2
3
4
5
6
7
8
9
10
0
1
2
3
4
5
6
7
8
9
10

Previous Example using Fitting Tool (nftool)



Validation and Test Data

Set aside some samples for validation and testing.

Select Percentages

Randomly divide up the 94 samples:

Training:	50%	47 samples
Validation:	15%	14 samples
Testing:	35%	33 samples

Here we can divide dataset into train, test and validation

Explanation

Three Kinds of Samples:

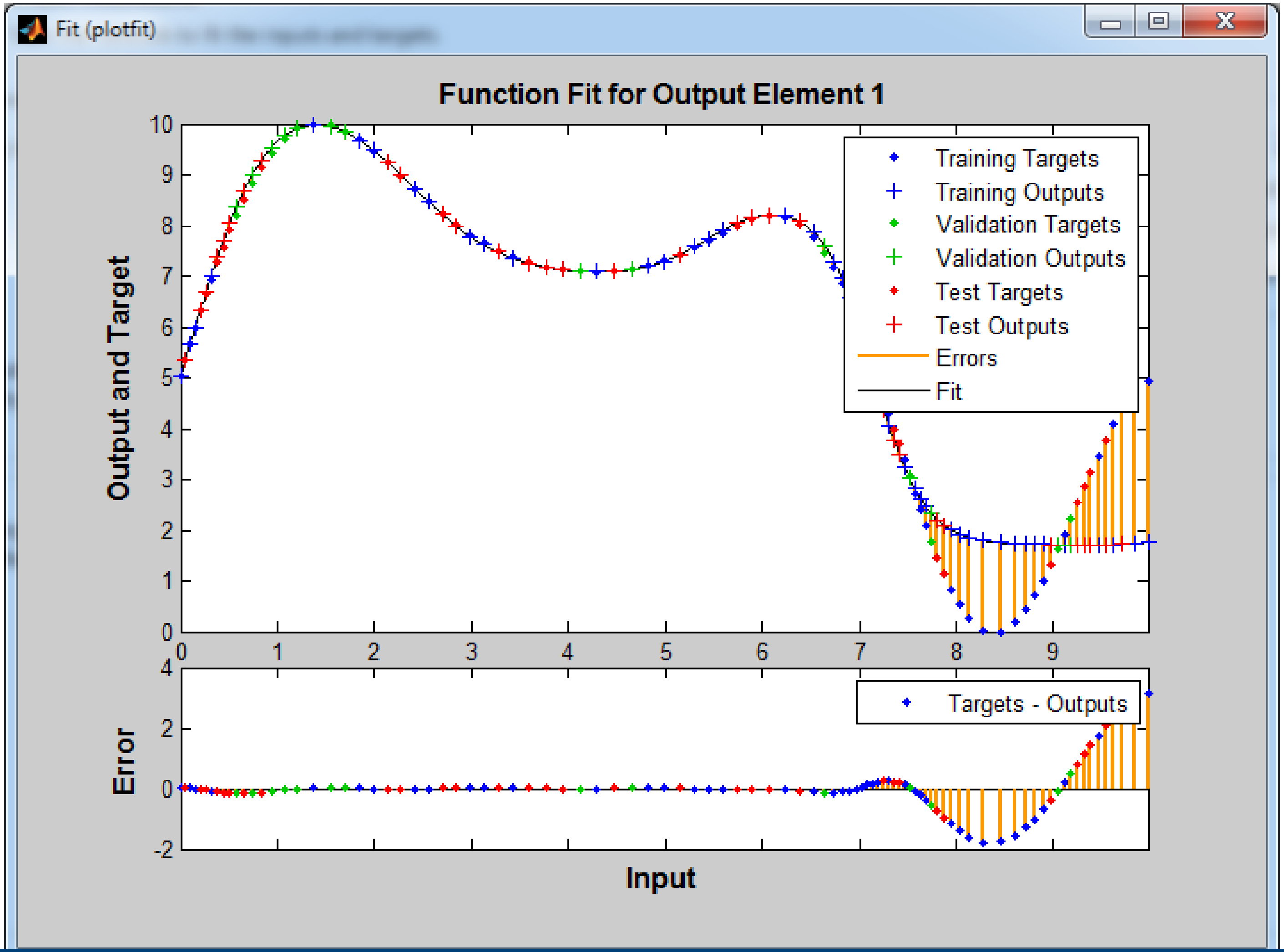
Training:
These are presented to the network during training, and the network is adjusted according to its error.

Validation:
These are used to measure network generalization, and to halt training when generalization stops improving.

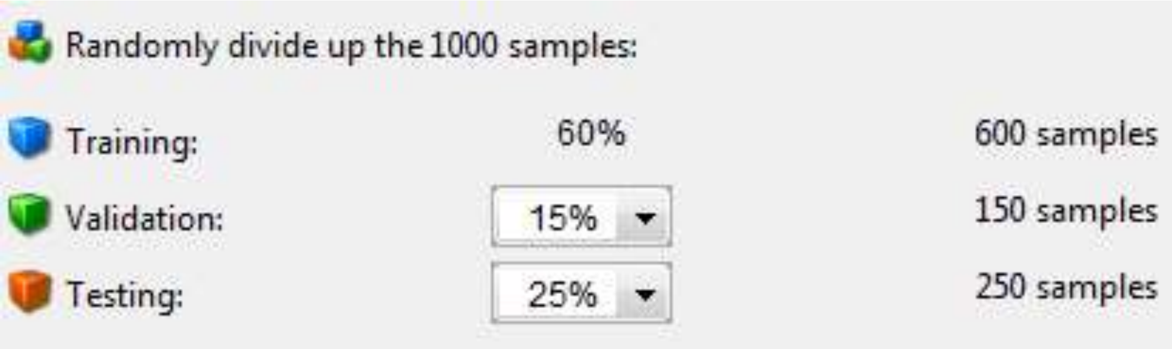
Testing:
These have no effect on training and so provide an independent measure of network performance during and after training.

Change percentages if desired, then click [Next] to continue.

Results

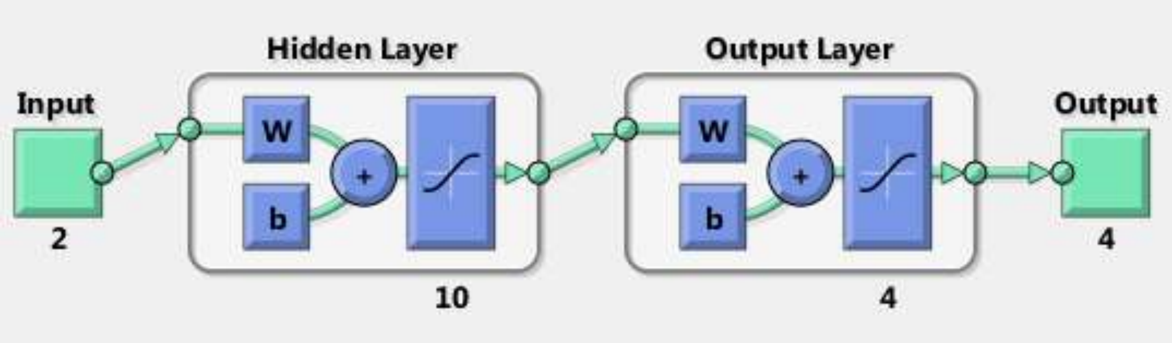

Fit (plotfit)
Function Fit for Output Element 1
Output and Target
10
9
8
7
6
5
4
3
2
1
0
0
1
2
3
4
5
6
7
8
9
Training Targets
Training Outputs
Validation Targets
Validation Outputs
Test Targets
Test Outputs
Errors
Fit
Error
4
2
0
-2
Targets - Outputs
Input

Pattern Recognition Tool (nprtool)

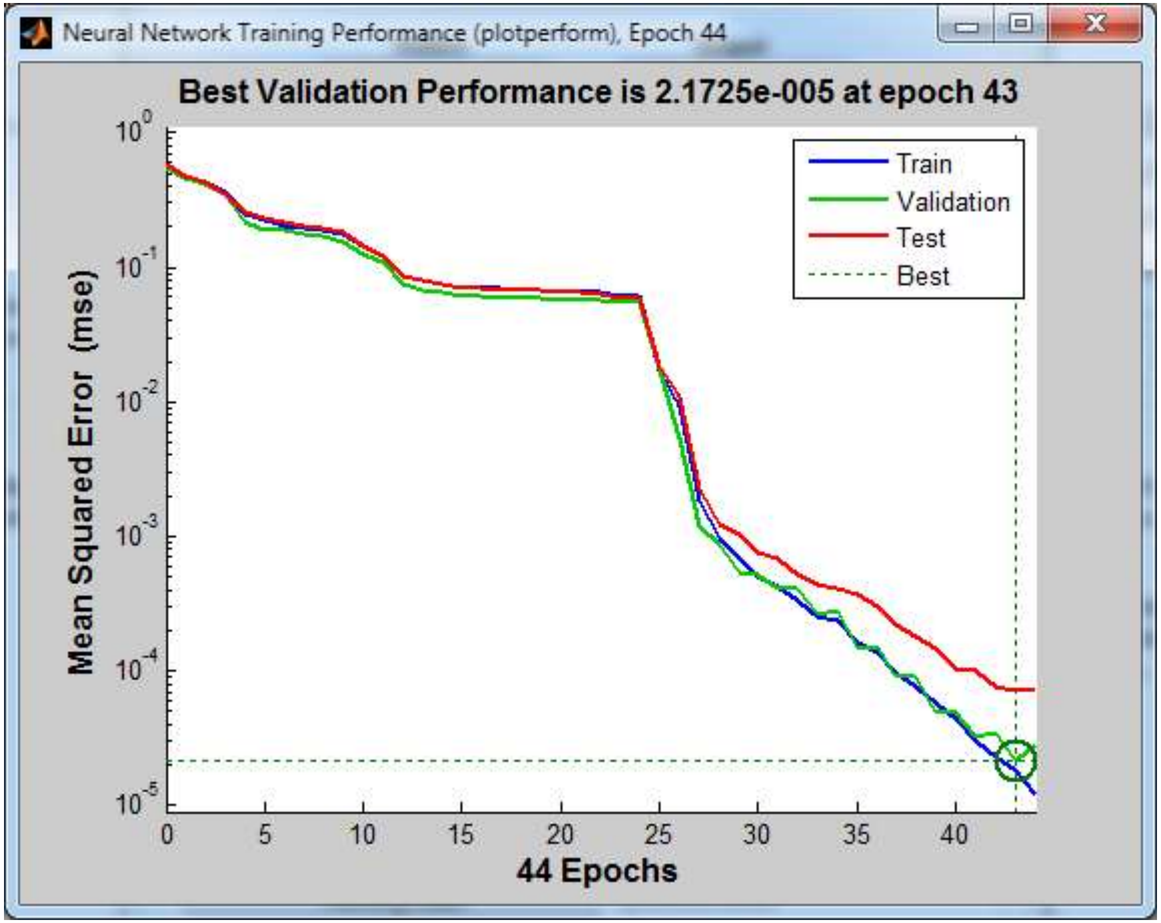


Randomly divide up the 1000 samples:

Training:	60%	600 samples
Validation:	15% ▾	150 samples
Testing:	25% ▾	250 samples



MSE Graph



Neural Network Training Performance (plotperform), Epoch 44
Best Validation Performance is 2.1725e-005 at epoch 43
Mean Squared Error (mse)
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
0
5
10
15
20
25
30
35
40
44 Epochs
Train
Validation
Test
Best

Perceptron

- newp: Create a perceptron.
- Obsoleted in R2010b NNET 7.0.
- Last used in R2010a NNET 6.0.4.

- Run digit recognition program: newp_digits.m

Adaptive Linear Network: Adaline

- newlin
- Or simple coding!
- Example: Prediction
    - Run program Adalline_Prediction.m
- Example: System Identification
    - Run program Adaline_System_Identification.m

MLP

- newff

- Example: Digit recognition
  - Run program ReadFeatures.m and train.m